Universidad Nacional de Córdoba
Facultad de Matemática, Astronomía y Física

Ingeniería de Requisitos Web Orientada a Aspectos con Transformación de Modelos

Autor:
Martín F. Villalba
Supervisor:
Juan E. Durán

04 de Marzo de 2011

Resúmen

Los analistas usualmente describen requisitos usando notaciones involucrando conceptos técnicos que los clientes suelen desconocer. El expresar requisitos con notaciones legibles para los clientes (NLC) - es decir, que usan conceptos conocidos por ellos - les permite comprenderlos, validarlos, y colaborar en su mantenimiento. La minería de elementos respetando conceptos técnicos (necesarios para los desarrolladores) a partir de modelos NLC es una tarea importante. Actualmente existe falta de NLC apropiados para describir en detalle cómo las Cualidades No Funcionales (CNF) afectan globalmente al sistema; sólo hay algunas notaciones poco detalladas para ello, que únicamente permiten generar modelos de requisitos Orientados a Aspectos (OA) muy poco detallados. Por eso proponemos para el dominio de sistemas de información web una NLC que permite expresar cómo CNF afectan globalmente a la parte funcional del sistema. Además definimos una transformación de modelos usando ATL para mapear esos modelos NCU a dos notaciones conocidas de requisitos OA.

Palabras Clave

AOWE, Aplicaciones Web, Aspectos, ATL, Cualidades No Funcionales, Ingeniería de Requisitos, MDA, Modelos, NFR, Requisitos No Funcionales, SIG, Transformación de Modelos, UML

Clasificación

D.2.1
D.2.2

Este trabajo está dedicado
a Romina Altamirano por su ayuda,
a Juan Durán por su paciencia,
a Juan Montoya por su apoyo,
y a mi familia por su cariño.

Índice general

Índice general **7**

Índice de figuras **9**

1. Introducción **11**

2. Conceptos Básicos **13**
2.1. Orientación a Aspectos . . . 13
2.1.1. Terminología . . . 14
2.1.2. AOWE . . . 14
2.2. Requisitos No Funcionales . . . 14
2.3. XML . . . 15
2.3.1. DTD . . . 17
2.3.2. XQuery . . . 17
2.4. UML . . . 18
2.4.1. XMI y Ecore . . . 18
2.5. Transformación de Modelos . . . 19
2.5.1. MDA . . . 19
2.5.2. QVT, ATL, ECORE y Eclipse . . . 20
2.6. Web 1.0 y RIA . . . 21

3. Notación para Describir cómo Cualidades no Funcionales afectan al Sistema **25**
3.1. Motivación . . . 25
3.2. Delimitando el tipo de Metamodelo a usar . . . 25
3.3. Experimento con RICNF expresados en lenguaje natural y resultados obtenidos . . 26
3.4. Criterios de clasificación de intereses . . . 27
3.4.1. Criterio 1: Cuál es la parte afectada del sistema . . . 27
3.4.2. Criterio 2: Cómo impactan las cualidades no funcionales al sistema . . . 29
3.4.3. Criterio 3: Nivel de detalle del Interés No Funcional . . . 30
3.4.4. Criterio 4: Etapa del Desarrollo en la que se implementa el Interés No Funcional 31
3.5. Metamodelo MICNF . . . 32
3.5.1. Elementos de Modelado . . . 33
3.5.2. Combinaciones de Parcelas de Sistema . . . 40

4. Otros Modelos de Requisitos **41**
4.1. Framework NFR . . . 41
4.1.1. Conceptos . . . 42
4.1.2. Metamodelo . . . 42
4.2. Modelos de CU . . . 43
4.3. Descripciones de Casos de Uso . . . 44
4.4. Modelo de CU unificado y Tablas de composición . . . 45
4.4.1. ATL y XML . . . 46

4.4.2. Modelo de CU Unificado . . . 46
4.4.3. Tablas de Composición . . . 50

5. Proceso de Ingeniería de Requisitos 53
5.1. Descripción del proceso en general . . . 53
5.2. Traducción de RICNF en lenguaje natural a notación MICNF . . . 54
5.3. Obtención de SIGs de framework NFR . . . 55
5.4. Obtención de Modelo de Casos de Uso . . . 56
5.5. Obtención de Descripciones de Casos de Uso . . . 57
5.6. Transformación a Modelos Orientados a Aspectos . . . 58

6. Transformaciones 59
6.1. Mapa de Transformaciones . . . 59
6.1.1. Ejemplo . . . 59
6.2. Transformación 1 . . . 59
6.2.1. Especificación . . . 60
6.2.2. Decisiones de diseño . . . 60
6.2.3. Implementación . . . 61
6.2.4. Validación y Verificación . . . 62
6.2.5. Ejemplo . . . 65
6.3. Transformación 2 . . . 68
6.3.1. Especificación . . . 68
6.3.2. Decisiones de diseño . . . 69
6.3.3. Regla 2.1 . . . 69
6.3.4. Regla 2.2 . . . 73
6.4. Transformación 3a . . . 76
6.4.1. Especificación . . . 76
6.4.2. Decisiones de Diseño . . . 78
6.4.3. Implementación . . . 78
6.4.4. Validación y Verificación . . . 79
6.4.5. Ejemplo . . . 79
6.5. Transformación 3b . . . 79
6.5.1. Especificación . . . 80
6.5.2. Decisiones de Diseño . . . 80
6.5.3. Implementación . . . 80
6.5.4. Validación y Verificación . . . 81
6.5.5. Ejemplo . . . 81
6.6. Impresiones sobre la herramienta . . . 81

7. Trabajo Relacionado 85

8. Conclusiones y Trabajo Futuro 87

Bibliografía 89

Índice de figuras

2.1. Ejemplo de documento XML 16
2.2. Ejemplo de consulta XQuery y su resultado 17

3.1. Proceso MDA completo - Versión simplificada 26
3.2. Diagrama de Clasificación de Intereses No Funcionales 32
3.3. Modelo de Impacto de Cualidades No Funcionales 33
3.4. Representación de una instancia de la clase nfi 34
3.5. Representación de instancias de la clase parcela_sistema 36
3.6. Representación de una instancia de la clase req_icnf 39
3.7. Instancia de la clase req_icnf - Múltiples PS, un único NFI 39
3.8. Instancia de la clase req_icnf - Una única PS, múltiples NFI 39
3.9. Instancia de la clase req_icnf - Múltiples PS y NFI 40

4.1. Metamodelo para Softgoal Interdependency Graphs 43
4.2. Metamodelo para Casos de Uso 44
4.3. Metamodelo para Casos de Uso - versión de UML 44
4.4. DTD para Descripciones de Casos de Uso 45
4.5. Metamodelo de Casos de Uso Unificado 47
4.6. Metamodelo de Casos de Uso Unificado - versión simplificada para XML 47
4.7. Metamodelo de Casos de Uso Unificado - DTD de la versión simplificada 47
4.8. Representación de una instancia de la clase caso_uso 48
4.9. Representación de una instancia de la clase caso_uso_cc 49
4.10. Representación de una instancia de la clase actor 49
4.11. Metamodelo de Tablas de Composición 50
4.12. Metamodelo de Tablas de Composición - DTD 50

5.1. Proceso MDA completo - Versión completa 53
5.2. Traducción de RICNF en lenguaje natural a notación MICNF en el Proceso MDA 54
5.3. Obtención de SIGs de framework NFR en el Proceso MDA 55
5.4. SIG Usabilidad: resultado de la descomposición del Softgoal Help 56
5.5. Obtención de Modelo de Casos de Uso en el Proceso MDA 56
5.6. Ejemplo de obtención de Modelo de Casos de Uso y Descripciones de C.U. a partir de MICNF 57
5.7. Obtención de Descripciones de Casos de Uso en el Proceso MDA 58
5.8. Transformación a Modelos Orientados a Aspectos en el Proceso MDA 58

6.1. Transformaciones de Modelos en la Ingeniería de Requisitos 59
6.2. Diagrama de la Transformación 1 60
6.3. Regla de Transformación de una instancia de la clase parametro_impacto 61
6.4. Regla de Transformación de una instancia de la clase req_icnf 63
6.5. Notación BNF para Tópicos 64
6.6. Ejemplos de Requisitos expresados en MICNF 66
6.7. Ejemplo de Softgoal Interdependency Graph (SIG) 66

6.8. Resultado de la Transformación 1 67
6.9. Ejemplo de transformación para el caso Help 68
6.10. Diagrama de la Transformación 2 68
6.11. División de la Transformación 2 69
6.12. Metamodelo de Expansión 70
6.13. DTD del Metamodelo de Expansión 70
6.14. Consulta XQuery para roles de usuario 71
6.15. Descripción Textual de Caso de Uso "Inspeccionar Catálogo" 72
6.16. Modelo de Expansión para las MICNF del ejemplo 73
6.17. Código para obtener la Descomposición de una Parcela de Sistema 74
6.18. Código ATL para la intersección de parcelas 75
6.19. Código ATL para la unión de parcelas 76
6.20. Resultado de la Transformación 2.2 77
6.21. Diagrama de la Transformación 3a 77
6.22. Regla identidad en la Transformación 3a 78
6.23. Regla de mapeo de PS a CU en la Transformación 3a 79
6.24. Resultado de la Transformación 3a 79
6.25. Diagrama de la Transformación 3b 80
6.26. Regla de mapeo de PS a Tablas de Composición en la Transformación 3b 81
6.27. Reglas de consulta de XML para parámetros de Tabla de Composición 82
6.28. Resultado de la Transformación 3b 83

Capítulo 1

Introducción

Para el tratamiento requisitos no funcionales (RNF) durante la ingeniería de requisitos existen dos opciones: que el usuario ayude a recolectarlos, validarlos y mantenerlos; o que los desarrolladores usen su experiencia para definirlos sin la participación explícita del usuario. Para la segunda existen notaciones de requisitos basadas en metas (NFR[9], i*[45], KAOS[23], etc.) y notaciones de requisitos orientadas a aspectos (OA) - p.ej. AOSDUC[16], método de Sousa et. al.[38], ARCaDE[33],[27], etc. Para la primera los usuarios proporcionan requisitos en lenguaje natural o en alguna notación globalmente difundida (p.ej. Casos de Uso[25], documentos XML[42], etc.) y a partir de los mismos se identifican los conceptos necesarios para modelado basado en metas o modelado OA y se generan estos tipos de modelos. La diferencia entre las dos opciones es que en la primera están involucrados conceptos técnicos que los clientes o usuarios no conocen (p.ej. aspecto, pointcut, advice, contribución, correlación, crosscutting, etc.); dichos conceptos son útiles porque sirven para las fases posteriores de desarrollo (diseño, arquitectura, implementación, etc.)

En Ingeniería del Software son muy importantes las notaciones cercanas a los usuarios (NCU) - estas son notaciones que usan conceptos y vocabulario conocidos por los usuarios - (p.ej. modelos de CU, descripciones de escenarios, etc.) porque los usuarios pueden: comprenderlas fácilmente sin necesidad de un entrenamiento, validar los requisitos expresados usando estas notaciones, colaborar en el mantenimiento de los requisitos; además el mapeo de los requisitos de los usuarios a estas notaciones es más directo que el mapeo a notaciones técnicas. Con las NCU no alcanza, porque los desarrolladores necesitan trabajar con notaciones y paradigmas que conocen: OA, RNF, transformación de modelos, etc. Entonces la brecha entre NCU y las notaciones técnicas usadas por los desarrolladores necesita ser llenada. Por lo tanto las NCU necesitan ser manipuladas por herramientas de todo tipo, por ejemplo de transformación de modelos, de identificación de elementos de acuerdo conceptos técnicos etc. para producir así modelos basados en conceptos técnicos. Los usuarios suelen referirse a cualidades no funcionales (CNF) en sus requisitos en lenguaje natural ya sea expresando escenarios donde aparecen, o refiriéndose a ellas en forma global indicando cómo una CNF se impacta sobre funcionalidades del sistema. El primer caso es muy conocido y ha sido estudiado bastante; para el mismo hay notaciones ampliamente difundidas (descripciones de CU, BPMN, etc.); además se ha estudiado cómo a partir de estas notaciones e identificar conceptos como aspectos, pointcuts, relaciones de crosscutting, advices, etc. (p.ej. [14]) El tener que dar todos los escenarios que involucran RNF es costoso en tiempo y a este costo se le suma el de procesar dichos escenarios en forma semiautomática para obtener modelos OA o basados en metas. Por eso es interesante el no tener que incurrir en tantos costos y cobra interés el segundo caso donde los usuarios pueden definir en una sola oración cómo una CNF afecta globalmente al sistema; o sea, en una oración se puede plasmar lo que de otra manera involucraría muchos escenarios que muestran vínculos de CNF con requisitos funcionales.

Consideramos que para el segundo caso existe necesidad de NCU para describir en detalle cómo afectan globalmente las CNF al sistema; lo que si hay es algunas notaciones poco detalladas para ello que no permiten generar los requisitos técnicos en forma detallada (p.ej. [43, 41]), así se podrá acceder a los beneficios previamente mencionados.

Entonces los objetivos de este trabajo son dos: la definición de una NCU para describir cómo las CNF afectan globalmente al sistema y ver cómo sacarle el jugo a estas notaciones durante la Ingeniería de Requisitos; por ejemplo: generando automáticamente modelos OA, ayudando a definir modelos de requisitos funcionales y modelos de RNF, incluyendo la colaboración del usuario en actividades de validación, mantenimiento, etc.

En este trabajo como un primer estudio para alcanzar dichos objetivos proponemos para el dominio de sistemas de información web una notación de impacto de CNF en la parte funcional (metamodelo MICNF), que permite expresar cómo una CNF afecta globalmente a la parte funcional del sistema (Capítulo 2.6, página 23). Luego mediante el uso de transformación de modelos en ATL[17] planteamos mapear modelos MICNF a modelos de requisitos OA; proporcionamos para ello dos ejemplos: generación de modelos de CU unificados y generación de tablas de composición (ambas notaciones descritas en [38]); los cuales son de utilidad para los desarrolladores en etapas posteriores a requisitos (Capítulo 4.4.3, página 52); en particular en el método AOWE para desarrollo OA de aplicaciones web se usan esos modelos de requisitos OA ([5],[6]). Además planteamos un proceso de ingeniería de requisitos que integra la creación y generación de todos los modelos anteriores (Capítulo 3.5.2, página 40).

Capítulo 2

Conceptos Básicos

2.1. Orientación a Aspectos

Las disciplinas tradicionales de ingeniería manejaron la complejidad de los sistemas identificando y separando los intereses del sistema, y tratando cada uno de manera aislada; tal acercamiento, conocido como *Separación de Intereses* (SOC, del inglés *Separation of Concerns*), llevó a la creación de sistemas que serían más sencillos de implementar, verificar, evolucionar y entender. La disciplina del desarrollo de software adoptó rápidamente este acercamiento; los lenguajes de programación fueron así los precursores en el avance de la tecnología de desarrollo de software, proveyendo progresivamente abstracciones tales como funciones, procedimientos, tipos abstractos de datos y objetos, con los cuales ayudaron a obtener mayores niveles de SOC [36].

Siguiendo el desarrollo de lenguajes AO, las ideas de la Orientación a Aspectos se extendieron a todo el proceso de desarrollo, dando forma al campo del *Desarrollo de Software Orientado a Aspectos* (AOSD, del inglés *Aspect Oriented Software Development*).

Un concepto compartido por AOSD con otras metodologías es el concepto de separación de *intereses*, término ideado por Edsger Dijkstra en 1974 [12], el cual refleja el proceso de separar un sistema y/o proceso en características independientes con la menor superposición posible. Este problema es abarcado por todas las metodologías de manera distinta, aunque es en el contexto de AOSD donde surgen conceptos no tratados hasta el momento.

La separación de intereses es deseable tanto para lograr un razonamiento tanto modular (mantener los intereses separados, sin importar cómo afecten o influencien otros intereses del sistema, para poder razonar sobre cada interés de forma aislada) como composicional (entender las dependencias e interacciones entre intereses para razonar así sobre las propiedades globales y/o emergentes del sistema). Si bien existen diversas técnicas dentro de ingeniería de requisitos para lograrlo (Diagramas de Casos de Uso, Viewpoints, Metas, etc), estas metodologías no se enfocan de manera suficiente en el problema de intereses transversales (llamados intereses crosscutting, estos intereses atraviesan otros intereses - esto significa que se encuentran entremezclados con otros intereses y/o esparcidos a lo largo de varios intereses). En el tratamiento de los intereses crosscutting en los enfoques tradicionales para requisitos mencionados surgen dos problemas particulares: el *scattering* (intereses que se repiten a lo largo de diversas unidades del sistema) y el *tangling* (intereses que se entrecruzan entre sí en diversos módulos). La Ingeniería de Requisitos Orientada a Aspectos busca soluciones a estos problemas, brindando una separación de Intereses Crosscutting de las demás unidades de requisitos funcionales, lo cual permite simultáneamente eliminar el problema del *scattering* y reducir la complejidad. Esto nos brinda una mejor manera de entender el problema, presentando a la vez herramientas para razonar sobre él. [32].

El objetivo de AOSD es brindar apoyo al desarrollador mediante una separación clara de intereses y aspectos entre sí, proveyendo mecanismos que hagan posible abstraerlos y componerlos para producir el sistema completo [21]. La novedad que introduce el AOSD con respecto a otras metodologías es que, al reconocer que ciertas áreas de interés del sistema se encuentran diseminadas

a lo largo de múltiples otras áreas, brinda una mejor base para resolver ciertos problemas que otras metodologías no reconocen (siendo *tangling* y *scattering* los más prominentes).

2.1.1. Terminología

Concern Un concern (o interés, en castellano) es un área de interés o enfoque en un sistema; representa el criterio primario de descomposición de software en partes más pequeñas, manejables y comprensibles que tienen sentido para un ingeniero de software. Algunos ejemplos son los requerimientos, casos de uso, estructuras de datos y/o patrones. Existen múltiples maneras de formular los concerns de como unidades separadas bien identificadas, siendo los aspectos uno de tales mecanismos.

Crosscutting concern interés que no puede ser aislado dentro de un sólo módulo, dado que su misma naturaleza hace que su implementación deba estar diseminada a lo largo de varios módulos o compartiendo un mismo módulo con otros intereses.

Aspecto un tipo de concern en el desarrollo de software, es decir, un área de interés o enfoque en un sistema que se inserta en un punto de corte específico dentro de un sistema.

Scattering y Tangling Decimos que la representación de un interés está diseminada (del inglés, *scattered*) si está dispersa en vez de centralizada. La representación de concerns está enredada (del inglés, *tangled*) con un artefacto si están mezclados más que separados. A pesar de que son conceptos distintos, tanto *scattering* como *tangling* tienden a verse en conjunto.

Weaving El proceso de coordinar aspectos y no-aspectos. Puede hacerse de forma explícita (usando metodologías donde se detalla de qué manera se llevará a cabo el weaving) o implícita (donde únicamente se imponen restricciones sobre el resultado que debe cumplir el weaving), y puede hacerse en diversos puntos del proceso de desarrollo, desde un weaving manual cuando el código es escrito, antes o después de la compilación, o hasta el tiempo de ejecución. Proviene del inglés, *weave* (entretejer).

2.1.2. AOWE

Dentro de este marco surge *AOWE* (Aspect-Oriented Web Engineering), una metodología de desarrollo de aplicaciones web que contempla un proceso de desarrollo OA que comienza desde las primeras fases del ciclo de desarrollo de software, abarca la mayoría de las etapas del proceso de desarrollo, y que muestra una aceptable trazabilidad entre dichas etapas. AOWE está basada en UML, utiliza el framework NFR, y abarca las etapas de Ingeniería de Requisitos, Análisis y Diseño.

El presente proyecto toma como base varias ideas de AOWE, principalmente de su fase de Ingeniería de Requisitos, utilizando sus notaciones (y su ubicación general dentro del proceso), algunas de las cuales son formalizadas en este trabajo mediante metamodelos.

2.2. Requisitos No Funcionales

Los usuarios tienen expectativas no transmitidas de qué tan bien debe comportarse el software. Estas características incluyen qué tan fácil de usar es el software, qué tan velozmente se ejecuta, qué tan confiable es, y qué tan bien se comporta cuando surgen situaciones inesperadas. Los *Requisitos no Funcionales* (NFR, del inglés *Non-Functional Requirements*) definen estos aspectos del sistema y/o del proceso de desarrollo [40].

Los Requisitos no funcionales son entonces requisitos que imponen restricciones tanto en el producto a desarrollar (requisitos de producto), en el proceso de desarrollo (requisitos de proceso), o en las restricciones específicas que el producto/proceso debe cumplir (requisitos externos). El principal objetivo del Framework NFR es representar, organizar y analizar Requisitos No Funcionales (RNF). Este framework provee técnicas para justificar decisiones de diseño durante el proceso de

desarrollo; es también orientado a metas, dado que considera los RNF como metas a cumplir. Sin embargo, a diferencia de los enfoques tradicionales orientados a metas, este framework utiliza la noción de *softgoal*, término que representa una meta que no tiene un criterio concreto para determinar si ha sido satisfecho o no. Un softgoal se considera *satisfecho* cuando hay suficiente evidencia positiva en su favor, y poca o ninguna evidencia en contra [37]. Otra forma de trabajar con RNF es cuantificándolos, es decir, asegurando que los RNF provean mediciones específicas que el software debe cumplir. El número máximo de segundos que debe demorar una tarea, el tamaño máximo de espacio en disco de una base de datos, la cantidad de horas diarias que un sistema debe estar disponible y el número de usuarios concurrentes soportados son ejemplos de requisitos que el software debe implementar, pero que no cambia su comportamiento [40]. Esta definición de métricas para NFR, sin embargo, no será tenido en cuenta en el presente trabajo. El Framework NFR se encuadra dentro de una definición más general, que es la de *Modelos Basados en Metas*. Diremos que una *meta* (del inglés, *goal*) es un objetivo que se desea que el sistema (donde entendemos por sistema tanto al producto final como al ambiente en el que se desarrollará)cumpla. Una meta cubre tanto intereses funcionales como no funcionales. Las metas se formulan en distintos niveles de abstracción, lo cual permite en sus etapas más altas involucrar al cliente en el proceso de ingeniería de requisitos [23].

Dos de los modelos más populares para expresar metas son i* y KAOS. Algunas características de ellos, según [44], son:

- KAOS define RNF como metas funcionales; i* los define como softgoals que pueden contribuir positivamente o negativamente con otros softgoals.

- KAOS incluye el concepto de agente, el cual puede ser de software o humano. Son entidades atómicas, y pueden tener asociados requisitos y expectativas; i* usa el concepto más general de actor, que puede expresarse como un agente, un rol o una posición, lo cual permite una distinción más fina entre distintos tipos de actores y sus relaciones.

- KAOS provee una base detallada para la descripción de tareas (acciones), y ver cómo esa acción se relaciona con una restricción y/o un objeto (un agente, un evento, una entidad o una relación). En i*, una tarea se relaciona con una tarea, una meta, un softgoal o un recurso, con lo cual KAOS provee mejor soporte para una descripción detallada de tareas.

- KAOS provee un mecanismo más explícito para lidiar con análisis de riesgos, gracias al concepto de obstáculo. i* realiza este análisis en la forma de softgoals con aporte negativo sobre otros softgoals.

2.3. XML

XML (*Lenguaje de Marcado Extensible*, del inglés *Extensible Markup Language*)fue desarrollado por un grupo de trabajo conocido originalmente como el "SGML Editorial Review Board", formado bajo el auspicio del World Wide Web Consortium (W3C) en 1996. Derivado de SGML (Lenguaje de Marcado Generalizado, convertido en el estándar ISO-8879, del cual se deriva entre otros el lenguaje HTML), XML es un metalenguaje extensible basado en marcado (es decir, un sistema donde se introducen anotaciones o marcas, conocidas como *tags*, con información adicional a un texto, de modo tal que dichas marcas sean sintácticamente diferenciables del texto en sí), con el fin de convertirse en un lenguaje de intercambio de información estructurada entre plataformas y/o aplicaciones. Su aplicación actual se encuentra en áreas tan dispares como bases de datos, editores de texto y/u hojas de cálculo [4].

Las características de XML son:

- XML es un lenguaje basado en marcas

- XML fue diseñado para transportar, almacenar y compartir información entre aplicaciones

- XML no brinda marcas predefinidas, sino que brinda al usuario la tarea de definirlos según sus necesidades

- XML está diseñado para ser auto-descriptivo, es decir, para poder ser interpretado sin una guía de referencia (lo cual no ocurre, por ejemplo, con los archivos .doc de Microsoft Word). Al ser un formato basado en texto (en oposición a formatos binarios), permite a un usuario no sólo comprender su contenido sin la ayuda de ninguna herramienta, pero también generar nuevos documentos de manera sencilla

En un documento XML, un **elemento** es todo objeto representado dentro del documento; el contenido de un elemento se clasifica en *texto* (un elemento que contiene un texto concreto) o *elementos* (es decir, un elemento que contiene otros elementos dentro). El concepto de "elementos dentro de otros elementos" da a XML una estructura interna en forma de árbol, siendo el documento XML el elemento que contiene a todos los demás, e introduciendo el concepto de elemento **hijo** para un elemento contenido dentro de otro. Para delimitar un elemento se usan etiquetas(**tags**. Cada etiqueta se delimita entre los signos '<'y '>'. Las etiquetas se clasifican en **tag de apertura** (si marcan el inicio de un elemento), **tag de cierre** (si marcan el fin de un elemento) o **tag vacío** (tipo de etiqueta usada para elementos que no contienen sub-elementos).

Un elemento puede, además de contenido, tener propiedades adicionales; estas propiedades se conocen como **atributos**, los cuales brindan información adicional sobre el contenido del elemento. Se componen de pares nombre/valor, y se encuentran siempre en conjunto con la definición de un tag de apertura o vacío.

Un ejemplo de documento XML se presenta en la Figura 2.1, representando un esquema sencillo para preguntas de un examen:

```
<?xml version=''1.0'' ?>
<examen>
    <pregunta puntos=''2''>
        ¿Es P=NP? Brinde una demostración
    </pregunta>
    <pregunta puntos=''1''>
        ¿Es posible implementar un stack con un array?
    </pregunta>
</examen>
```

Figura 2.1: Ejemplo de documento XML

- *examen* es aquí el elemento principal, que a su vez contiene una lista de elementos *pregunta* (los cuales se consideran aquí elementos hijo del elemento examen); estos últimos son elementos con contenido de texto, por lo que no contienen otros elementos.

- cada elemento *pregunta* tiene a su vez un atributo llamado *puntos*. Este atributo nos brinda, en este caso concreto, información sobre el valor de cada pregunta del examen.

- *<examen>* es un tag de apertura del elemento examen, mientras *</examen>* es el tag de cierre.

Diseñado como una tecnología para el intercambio de información, XML es actualmente un estándar altamente usado como medio de almacenamiento y representación de información. En conjunto con los mecanismos de DTD y XQuery, XML establece un paralelo importante con los esfuerzos del OMG por diseñar una Arquitectura Orientada a Modelos.

2.3.1. DTD

DTD, siglas de *Definición de Tipo de Documento* (del inglés, *Data Type Document*) es un conjunto de declaraciones de elementos y atributos que define una sintaxis concreta acerca de cuáles elementos pueden aparecer en un tipo particular de documento XML. Los DTD se han establecido como uno de los estándares más utilizados para la definición de estructuras en documentos XML.

Con el fin de mantenerse a la vanguardia de los nuevos desarrollos (tales como la recomendación XML Namespace), XML se encuentra en proceso de reestructuración.

2.3.2. XQuery

XQuery es un lenguaje de consultas y programación funcional diseñado para realizar consultas sobre conjuntos de datos XML. Convertido en una recomendación del W3C en 2007, la misión del grupo de desarrollo de XQuery fue la de proveer facilidades flexibles para extraer datos de documentos reales y virtuales en la Web, proveyendo finalmente la interacción necesaria entre el mundo de la web y el mundo de las bases de datos. El objetivo final, entonces, sería el de acceder colecciones de archivos XML como si se trataran de bases de datos [34]. El lenguaje provee simultáneamente un mecanismo de consultas para la extracción y manipulación de datos (a través de las llamadas "expresiones FLWOR", acrónimo de *for, let, where, order by, return*) así como también para la creación de nuevos documentos XML:

For permite iterar sobre un conjunto de datos

Let declara variables dentro del scope de la consulta actual

Where utilizada para especificar criterios que filtren los datos iterados por los *for*.

Order by ordena los resultados de la consulta actual

Return para cada resultado de la consulta actual, determina cómo se debe devolver el valor obtenido

En la Figura 2.2 se muestra un ejemplo de consulta XQuery para encontrar los títulos de una colección de libros cuyo autor sea *Adams*, ordenados alfabéticamente por título de cada libro, y cuyo resultado (detallado debajo de la consulta) se presenta como una lista HTML.

```
<ul>
{
  for $x in doc("libros.xml")/catalogo/libro
  where $x/autor = "Adams"
  order by $x/titulo
  return <li>{$x/titulo}</li>
}
</ul>

<ul>
<li>Life, The Universe and Everything</li>
<li>So Long, and Thanks for All the Fish</li>
<li>The H.H. Guide to the Galaxy</li>
<li>The Restaurant at the End of the Universe</li>
</ul>
```

Figura 2.2: Ejemplo de consulta XQuery y su resultado

Al igual que se mencionara con XML, XQuery fue desarrollado al margen de los esfuerzos del OMG por crear un paradigma de desarrollo orientado por modelos; a pesar de esto, los objetivos

de XQuery de realizar transformaciones de documentos XML en otros documentos del mismo tipo trazan un paralelo importante con lo que sería luego MDA, coincidiendo ambas especificaciones en una gran cantidad de principios de diseño.

2.4. UML

UML, o Lenguaje Unificado de Modelado (del inglés, *Unified Modeling Language*) tiene como objetivo proveer a arquitectos de sistemas, ingenieros y desarrolladores de software herramientas para el análisis, implementación y diseño de sistemas basados en software así como también para el modelado de negocios y procesos similares. Uno de los objetivos primarios de UML es el avance del estado de la industria al permitir la interoperabilidad entre herramientas visuales de modelado. Dado que para permitir un intercambio importante de información entre herramientas es necesario un acuerdo sobre la semántica y notación, UML intenta proveer dicho acuerdo en conjunto con los siguientes requerimientos [31]:

- Una definición formal de un metamodelo basado en MOF que especifique la sintaxis abstracta del UML. Esta sintaxis abstracta define el conjunto de conceptos de modelado de UML, sus atributos, sus relaciones, y las reglas para combinar estos conceptos para la construcción de modelos UML.
- Una explicación detallada de la semántica de cada concepto de modelado de UML. La semántica define, de forma independiente a la tecnología, cómo los conceptos de UML deben implementarse en computadoras.
- Una especificación de los elementos de notación legibles por humanos para la representación de los conceptos individuales de modelado de UML, así como las reglas para combinarlos en una variedad de tipos de diagrama correspondientes a diversos aspectos de los sistemas modelados.
- Una definición detallada de formas en las cuales las herramientas UML pueden ser diseñadas para cumplir con esta especificación. Esto se logra mediante una especificación basada en XML de un formato de intercambio de modelos (XMI).

En el presente trabajo utilizaremos el concepto de Caso de Uso para expresar los requisitos funcionales del sistema, para lo cual deberemos utilizar los Diagramas de Caso de Uso definidos en UML.

2.4.1. XMI y Ecore

OMG crea en el año 2000 el *Formato de Intercambio de Metadatos XML*, conocido como XMI (del inglés, *XML Metadata Interchange Format*). El propósito principal de XMI es permitir un intercambio sencillo de metadatos (información utilizada para almacenar definiciones, estructura y administración de datos) entre herramientas de modelado (basadas en UML) y repositorios de metadatos (basados en MOF) en ambientes diversos y distribuidos. XMI integra cuatro estándares clave:

XML Lenguaje Extensible de Marcado, un estándar del W3C

UML Lenguaje Unificado de Modelado, un estándar de modelado del OMG

MOF Facilidad de Meta Objetos, un estándar de repositorios de metamodelado y metadatos del OMG

Mapeo de MOF a XMI[1] Reglas de mapeo de elementos MOF a XMI las cuales aseguran la interoperabilidad entre herramientas. Gracias a esto, podemos crear metaobjetos MOF en una determinada herramienta y compartirlos en la forma de documentos XMI, sin temor a incompatibilidades entre las herramientas.

La integración de estos tres estándares en XMI unifica lo mejor de las técnicas de metadatos y modelado del W3C y el OMG, permitiendo a los desarrolladores de sistemas distribuidos compartir objetos, modelos y otros tipos de metadatos a través de Internet [29].

La herramienta Eclipse, utilizada en este proyecto, almacena sus modelos y metamodelos utilizando este formato de archivo. Esto redunda en una especie de dualidad entre estos modelos y XML, ya que todo modelo UML generado por la herramienta puede interpretarse como un documento XML, y donde es posible generar documentos en XML que luego serán utilizados en ATL. Esta propiedad será utilizada en varias etapas del presente trabajo.

2.5. Transformación de Modelos

El concepto de Modelos es un concepto introducido por el Object Management Group (OMG) como respuesta a ciertos problemas surgidos a partir de la experiencia con la metodología de Desarrollo de Software Orientada a Objetos. El OMG, surgido como un consorcio para promover y estandarizar el uso de técnicas basadas en objetos, evolucionó eventualmente hacia la promoción de técnicas y estándares de modelados, creando para ello UML y los estándares relacionados MOF, XMI y QVT. Estos estándares, en conjunto, proveen la fundación para lo que sería la *Arquitectura Orientada a Modelos* (MDA, del inglés *Model-Driven Architecture*), siendo la *Transformación de Modelos* uno de los ejes principales de este nuevo enfoque.

Esencialmente, si consideramos que un *Modelo* es una abstracción de una cierta entidad o funcionalidad, una *Transformación de Modelos* será un proceso mediante el cual, tomando un modelo de origen perteneciente a un determinado *Metamodelo* (es decir, así como un documento XML obedece a una sintaxis, un modelo respeta las propiedades definidas por su metamodelo), podremos obtener un nuevo modelo, el cual se ajustará a un cierto Metamodelo (llamado *Metamodelo de salida*, que podrá o no ser igual al *Metamodelo de entrada*).

2.5.1. MDA

Creado originalmente en 1989 para fijar estándares para sistemas y metodologías orientadas a objetos, OMG propuso gradualmente a partir de 1996 un cambio de enfoque: de un enfoque donde "todo es un objeto", se propuso el cambio a un enfoque basado en el concepto de "todo es un modelo"; este nuevo enfoque pretende atacar ciertos problemas inherentes a la programación OO. Este nuevo paradigma, lanzado en 2001, es conocido como MDA, siglas de *Arquitectura Orientada a Modelos* (del inglés, *Model Driven Architecture*).

Un concepto importante es el concepto de **Metamodelado**: la construcción de una colección de "conceptos" (cosas, términos, etc) dentro de un cierto dominio. Un modelo es una abstracción de un sistema, su ambiente, o ambos; un **metamodelo** es una abstracción mayor, ilustrando propiedades del modelo en sí. Un modelo se ajusta a su metamodelo, del mismo modo que un lenguaje regular se ajusta a la gramática que lo define. En el contexto de MDA, diremos que un metamodelo define la estructura y reglas de buena formación del lenguaje en el cual los modelos son expresados.

MDA define una aproximación para la especificación de sistemas de IT que separa la especificación de la funcionalidad del sistema de la especificación de la implementación de dicha funcionalidad en una plataforma tecnológica específica. Ambas especificaciones se expresan como modelos [22]:

Modelos Independientes de Plataforma, o PIM especifican la estructura y funciones de un sistema sin dejar por ello de abstraer los detalles técnicos del mismo (del inglés, *Platform Independent Model*).

Modelos Específicos de Plataforma, o PSM se derivan de los PIM y especifican de qué manera debe implementarse la funcionalidad en una plataforma seleccionada (del inglés, *Platform Specific Model*).

MDA propone enérgicamente el concepto de generadores basados en modelos que acepten un PSM y generen artefactos de código a partir de ellos. Estos generadores son parte de un concepto más general, que es el concepto de **Transformación de Modelos**. Esencialmente, una transformación es la generación automática de un modelo de destino desde un modelo de origen, de acuerdo con una definición de la transformación [11]. Esta definición, a su vez, es un conjunto de **reglas de transformación** (descripciones de cómo una o más construcciones en el lenguaje de origen pueden ser expresadas en una o más construcciones en el lenguaje de destino) que, en conjunto, describe cómo un modelo en el lenguaje de origen puede ser transformado en un modelo en el lenguaje de destino.

2.5.2. QVT, ATL, ECORE y Eclipse

Dado que la transformación de modelos es un componente crítico de MDA, la OMG lanzó una solicitud de propuestas (RFP, del inglés *Request For Proposals*) en 2002 para definir un estándar, basado en MOF, para realizar transformaciones de modelos. Dicho estándar, adicionalmente, debía ser compatible con la suite de recomendaciones de MDA (UML, MOF, OCL, etc). Trabajando sobre las respuestas recibidas por compañías e instituciones, y tras tres años de trabajo, se aprobó un estándar conocido como **QVT** (acrónimo de Query/View/Transformation), el cual es soportado por distintas soluciones de software.

La sintaxis abstracta de QVT está definida como un metamodelo MOF 2.0. Este metamodelo define tres sub-lenguajes para transformar modelos, utilizando OCL 2.0 para realizar consultas. Los tres sub-lenguajes QVT forman colectivamente un lenguaje de transformaciones híbrido con construcciones declarativas e imperativas. Los lenguajes se llaman *Relations*, *Core* y *Operational Mappings*, y están organizados en una estructura de capas [24]:

Core es un lenguaje declarativo, más reducido que los otros dos. Las definiciones de las transformaciones tienden a ser más largas que sus equivalentes en Relations, y los enlaces de trazabilidad de elementos de modelos de origen a elementos de modelos de destino se manejan como elementos ordinarios del modelo, siendo responsabilidad del desarrollador su creación y uso. El propósito principal de Core es proveer la base para la semántica de Relations, lo cual resulta en una sintaxis más engorrosa y complicada. Su sintaxis es textual.

Relations provee de herramientas para especificar transformaciones como un conjunto de relaciones entre modelos. En Relations se definen patrones (conjuntos de atributos y propiedades que debe cumplir un modelo), los cuales se comparan luego contra modelos existentes. El resultado positivo de esta comparación se utilizará para modelar elementos en nuevos modelos y/o para hacer cambios en modelos existentes. A diferencia de Core, Relations posee tanto una sintaxis textual como una gráfica.

Operational Mappings extiende el lenguaje Relations, agregando construcciones imperativas (ciclos, condiciones, etc) y consultas OCL.

Una de las propuestas presentadas para el RFP es el Lenguaje de Transformaciones Atlas, conocido por sus siglas ATL. Este lenguaje fue desarrollado bajo la bandera del Eclipse Modeling Project, una comunidad de usuarios de la herramienta Eclipse cuyo objetivo es la evolución y promoción de tecnologías de desarrollo basadas en modelos, para lo cual proveen y desarrollan un conjunto de frameworks de desarrollo, herramientas e implementaciones de diversos estándares, todo sobre lo cual se apoya la etapa de implementación del presente proyecto. Dado que el estándar QVT aprobado y ATL se derivan de una misma especificación, las similitudes y coincidencias entre ambos son tales que es posible establecer una interoperabilidad entre ambos [18]. Siendo que *QVT Operational Mappings* es el lenguaje más completo de los tres, y dado que ATL implementa el estándar QVT completo, es común referirse a ATL como una implementación de *QVT Operational*.

Si bien ATL brinda facilidades de programación tanto funcionales como imperativas, estas últimas se encuentran en la actualidad desaconsejadas, bajo la filosofía "utiliza únicamente construcciones imperativas para la parte de una transformación que lo necesita, si es que alguna vez

lo necesita" - siendo las reglas de tipo *unique lazy* generalmente el mecanismo más adecuado para reemplazar ciclos *for*.

Una transformación ATL se compone de:

Módulos mecanismo para agrupar reglas ATL, según criterios del desarrollador. Cada archivo de código ATL contiene al menos un módulo. Es posible la inclusión de reglas de un módulo dentro de otro.

Reglas las reglas son el mecanismo principal de ATL. Cada regla toma al menos un modelo de entrada (conforme a alguno de los metamodelos de entrada) y devuelve un modelo de salida (conforme al metamodelo de salida); el cuerpo de la regla define cómo se hará la transformación, y se ejecuta sobre todos los modelos de entrada que pertenezcan al metamodelo adecuado. Existen tres tipos de reglas en ATL:

matched rule la regla más general, se aplica una vez sobre cada elemento del metamodelo de entrada que se encuentre en su dominio (es decir, todo elemento de un metamodelo que cumpla con ciertas condiciones, tales como pertenecer a una determinada metaclase), y genera un modelo de salida por cada aplicación. Estas reglas se aplican de forma automática.

lazy rule a diferencia de una matched rule, una lazy rule se aplica cuando es llamada explícitamente sobre un modelo de entrada (esta llamada tiene lugar en el cuerpo de otra regla). Si esta regla se aplica dos veces sobre un mismo modelo de entrada, el resultado serán dos modelos de salida con idénticas propiedades.

unique lazy rule similar a una lazy rule, pero al ser llamada dos o más veces sobre un mismo modelo de entrada no generará nuevos modelos de salida.

Cualquiera sea el tipo de regla, todas poseen la misma estructura, a saber:

- Un conjunto de modelos de entrada, cada uno de los cuales puede tener una condición auxiliar de aplicabilidad
- Definiciones locales opcionales (expresiones let, similares a variables, que faciliten la lectura del código ATL)
- Un modelo de salida[2], con reglas OCL y código ATL que indica de qué manera debe generarse este modelo
- Un bloque opcional imperativo que se ejecuta sobre el modelo de salida. Su principal utilización es para otorgar identificadores únicos a modelos que los requieran.

Helpers funciones auxiliares a las reglas. No son estrictamente necesarias, pero facilitan la modularización y facilidad de comprensión del código

En este contexto, diremos que una transformación en ATL se define por sus metamodelos de entrada y salida, y por un módulo principal, dentro del cual se aplicarán reglas en base a los criterios dados por el analista.

2.6. Web 1.0 y RIA

Desde un punto de vista estrictamente tecnológico (es decir, ignorando los aspectos sociales de la cuestión), existe actualmente una división entre la "vieja" internet, referida actualmente con el término "Web 1.0", y la "nueva" internet. Si bien en principio el término "Web 2.0" se utiliza frecuentemente para describir el estado actual de la Internet, este término se encuentra tan ligado con su connotación social que no resulta apropiado para describir de manera exclusiva

[2]Sintácticamente, una regla puede tener más de un modelo de salida, pero dado que esta característica es simplemente la agrupación de múltiples reglas en una sola mediante *syntactic sugar*, no se la incluirá dentro de la definición conceptual

los aspectos tecnológicos involucrados en la internet moderna; en su lugar, el término *Aplicación Rica de Internet*, o RIA (del inglés, *Rich Internet Application*), surge para explicar la plataforma tecnológica que da soporte a la llamada Web 2.0.

Comprender las diferencias entre ambos modelos de Internet es necesario para entender los alcances del presente proyecto; los términos que explicaremos a continuación nos ayudarán a enmarcar el presente proyecto en el contexto de la Internet actual.

Actualmente, se considera que el concepto de "Web 2.0" comenzó durante una sesión de *brainstorming* en una conferencia entre Tim O'Reilly y MediaLive International a principios de 2004. El grupo notó que, a pesar del crash de las empresas "punto-com", la web se había vuelto más importante que nunca. El grupo discutió cuáles consideraban que eran las competencias principales de las empresas en la nueva Web 2.0, e identificaron varios sitios y tecnologías que ejemplificaban esta nueva versión de la web, tanto a nivel tecnológico (con tecnologías tales como Flash o AJAX) como a nivel social (brindando características que convertían a los usuarios de espectadores a participantes, incluyendo blogs y redes sociales).

Todas las características tecnológicas de la Web 2.0 (es decir, eliminando el aspecto social) se han concentrado en lo que se conoce como RIA. Este término, creado por primera vez por la empresa Macromedia en 2002 [3], identifica así a aplicaciones web que comparten muchas de las características de las aplicaciones de escritorio.

Podemos definir la diferencia entre Web 1.0 y RIA de la siguiente manera: la Web 1.0 se caracteriza por

- Páginas estáticas (en HTML) y/o dinámicas (con ASP, PHP, Servlets, CGI, etc)
- Uso de tablas y marcos, reemplazados luego en parte por CSS
- Extensiones HTML propietarias
- Comunicación sincrónica entre el cliente y el servidor (p.ej. Información de un formulario HTML enviada a través de los mecanismos POST y GET)
- Sitios web que no mantienen estado

Las RIA, por otra parte,

- Corren sin modificaciones a lo largo de múltiples plataformas (incluyendo distintos navegadores y distintos sistemas operativos)
- Interfases con el usuario más ricas (es decir, que incorpora comportamiento antes restringido a aplicaciones de escritorio, tales como *drag & drop*), brindando facilidades de interfaz que antes no eran posibles (tales como modificación de interfaz sin necesidad de enviar datos al servidor o validación de datos en tiempo real, y mejorando la interacción con el usuario al incorporar respuesta inmediata a eventos), y con mayor respuesta (reduciendo los tiempos de espera hasta en un 80 %).
- En vez de recibir páginas completas, el cliente recibe información a partir de la cual construir la página final
- Comunicación asíncrona entre el cliente y el servidor, manteniendo un canal de comunicación constante con este último y evitando así recargar toda la página con cada evento del usuario

Un caso particular de tecnología RIA es Ajax (del inglés, *Asynchronous JavaScript and XML*), donde se utilizan

- Páginas dinámicas y Javascript
- HTML y CSS, separando la capa de presentación del contenido
- Mayor respeto por los estándares, los cuales han madurado de la mano de entidades tales como el W3C (evitando prácticas tales como la definición de nuevos elementos HTML, práctica común en los inicios de la Web 1.0)

- La información se envía y recibe de manera asíncrona, utilizando solicitudes *XMLHttpRequest*

Ajax logra una mejor interacción con el usuario utilizando DOM (*Document Object Model*, una API que proporciona un estándar de objetos para representar documentos HTML y XML) para modificar el contenido, estructura y contenido de los documentos HTML en tiempo real. El uso del objeto *XMLHttpRequest*, por otra parte, permite intercambiar información con el servidor sin la necesidad de recargar cada página, acelerando los tiempos de respuesta. Esta información se transmite en XML.

Debido a la existencia de estándares para HTML, XML, CSS y Javascript (los elementos constituyentes de AJAX), una aplicación Ajax puede correr sobre cualquier plataforma que implemente estos estándares. Estas características han permitido a Ajax posicionarse como un robusto estándar para la implementación de RIAs.

Si bien las RIA juegan un rol crucial en la mejora de los mecanismos de comunicación a través de la web (pasando de un modelo pasivo a uno donde el usuario juega un rol más activo), dichas aplicaciones se especializan en el intercambio de información en pequeños trozos; muchas de las aplicaciones Web, sin embargo, son *data-intensive* (del inglés, *intensivas en datos*); este término indica aplicaciones cuyo propósito principal es presentar grandes cantidades de datos a sus usuarios. La mayoría de los sitios para compraventa online o comercio electrónico son data-intensive, así como también lo son la mayoría de los sitios "institucionales" (tanto públicas como privadas) y las bibliotecas digitales [7]. Si bien variados sistemas de desarrollo web permiten el desarrollo rápido de aplicaciones data-intensive al proveer soporte para la publicación semi-automática de recursos de información, estos sistemas generaron una tendencia a un proceso de diseño poco estructurado y con pobre separación de intereses, en respuesta a lo cual surgieron diversas metodologías de desarrollo para este tipo de aplicaciones, como WebML [35].

Capítulo 3

Notación para Describir cómo Cualidades no Funcionales afectan al Sistema

3.1. Motivación

En un principio, el presente proyecto se enfocó en estudiar cómo incorporar Transformaciones de Modelos a la metodología AOWE. A medida que se avanzó en esta tarea, sin embargo, se volvió evidente que esta metodología carecía de los mecanismos necesarios para realizar un relevamiento exhaustivo de cómo las cualidades no funcionales presentes en los requisitos afectan al sistema en su totalidad.

Este cambio de enfoque en el objetivo del proyecto tuvo resultados colaterales evidentes:

1. Se debió definir una notación nueva para expresar cómo las cualidades no funcionales impactan al sistema (MICNF)

2. El estudio de cómo aprovechar la información brindada por estos modelos para producir modelos de requisitos funcionales y del framework NFR

3. Estudiar transformaciones de modelos no OA (de CU, de descripciones de CU, NFR, y MICNF) a modelos OA.

4. Definir metamodelos para las notaciones de AOWE que carecen de metamodelo.

Como una primera aproximación al proceso, la Figura 3.1 ilustra las distintas etapas del proceso de recolección de requisitos, tanto funcionales como no funcionales, su análisis y su eventual incorporación en modelos formales que permitan facilitar su análisis. En sucesivas secciones de este trabajo presentaremos con más detalle cada una de estas etapas.

3.2. Delimitando el tipo de Metamodelo a usar

Los usuarios suelen referirse a Cualidades no Funcionales (CNF) en sus requisitos en lenguaje natural ya sea expresando escenarios donde aparecen, o refiriéndose a ellas en forma global indicando cómo una CNF impacta sobre funcionalidades y otras CNF del sistema [13]. Los casos donde los usuarios pueden definir en una sola operación cómo una CNF afecta globalmente al sistema serán llamados *Requisitos de Impacto de Cualidades No Funcionales (RICNF)*.

En esta etapa del proceso, se busca obtener un procedimiento para realizar la minería de elementos de acuerdo a conceptos basados en OA a partir de requisitos ICNF brindados por los clientes. Dado que describir requisitos ICNF utilizando las notaciones basadas en metodologías OA

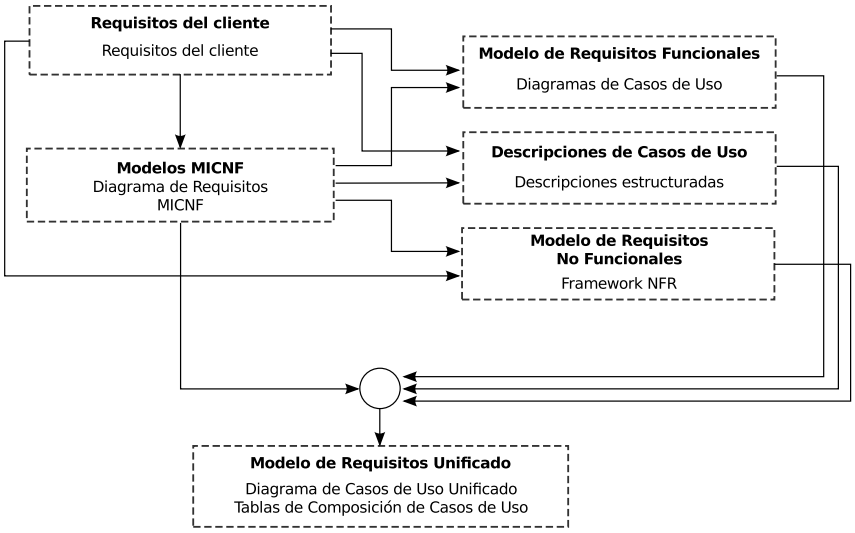

Figura 3.1: Proceso MDA completo - Versión simplificada

implica basarse en conceptos técnicos y estrategias para su construcción (ninguna de las cuales se espera que el cliente conozca), se hizo evidente la necesidad de plantear un metamodelo tal que fuera posible describir requisitos ICNF utilizando una notación legible por los clientes. Esto les brinda la posibilidad de comprender, validar y modificar en conjunto con el analista los requisitos a desarrollar; el hecho de que el mapeo de los requisitos de los clientes a notaciones ICNF legibles por ellos mismos sea mucho más directo que el mapeo a notaciones que involucran conceptos técnicos es otro fuerte indicador de que las necesidades que llevaron al planteo del nuevo metamodelo han sido satisfechas.

3.3. Experimento con RICNF expresados en lenguaje natural y resultados obtenidos

Como primer acercamiento al problema de desarrollar un metamodelo para requisitos RICNF, se llevó a cabo una recopilación y clasificación de requisitos ICNF a partir de diversas fuentes. En concreto, se recopiló una cantidad significativa de requisitos ICNF, expresados en lenguaje natural, y se realizó un estudio sintáctico y semántico de los mismos, con el objetivo de encontrar una estructura subyacente común a todos ellos, la cual abarcaría todos los requisitos que fuera posible.

El análisis de este conjunto de requisitos dio como resultado una primera clasificación de los requisitos ICNF basada en los siguientes 4 criterios:

1. **Cuál es la parte afectada del sistema:** consiste en determinar cómo afecta al sistema en su conjunto un nuevo Interés No Funcional (NFI, del inglés *Non-Functional Interest*), por ejemplo "¿Afecta a todas las operaciones?", "¿Debe aplicarse solamente en una unidad de requisitos funcionales particular?", etc.

2. **Cómo impactan las cualidades no funcionales al sistema:** debemos considerar de qué manera se insertará la nueva cualidad dentro del sistema (ya sea reemplazando a una operación por otra, insertándose en un punto determinado de un proceso, etc).

3. **Detalle del Interés No Funcional:** una vez que el cliente ha expresado el interés no funcional que desea incorporar, debemos preguntarnos si el detalle con el que lo ha descripto es suficiente, o si deberá ser refinado un poco más. No es lo mismo "el sistema debe permitir

el acceso únicamente a los usuarios autorizados" que "el sistema debe ser seguro", ya que implementar el segundo interés será notoriamente más trabajoso que el primero.

4. **Etapa de desarrollo en la que se implementa el Interés No Funcional:** como se detalla en secciones posteriores, los intereses de una cierta etapa de desarrollo tienden a compartir ciertas características distintivas (los requisitos sobre hardware, por ejemplo, tienden a pertenecer a la etapa de Despliegue, pero no a la de Testing). Dado que nuestro enfoque se centra en una etapa en particular, este criterio nos permitirá limitar nuestro enfoque a un subconjunto de las Cualidades no Funcionales del sistema.

3.4. Criterios de clasificación de intereses

Encontrar una estructura subyacente a los requisitos ICNF no fue una tarea sencilla; esta tarea demandó un análisis tanto semántico como sintáctico de un extenso conjunto de requisitos ICNF, el cual junto a un extenso conjunto de requisitos llevaría eventualmente a una clasificación de los mismos que sería de gran utilidad al momento de diseñar el metamodelo de RICNF.

En este capítulo se discutirá la estructuración en árbol de los criterios encontrados y sus sub-criterios. Es importante notar que algunos de los sub-criterios aquí detallados corresponden en realidad con situaciones imposibles de encontrar en una especificación realizada correctamente; dichos criterios se incluyen únicamente por completitud - si no incluyéramos dichos criterios, y luego nos encontráramos con un criterio no contemplado, sería imposible saber si dicha omisión ha sido deliberada o no. De este modo, si encontramos un RICNF que escapa a esta clasificación sabremos que se debe a una omisión no intencional, en base a la cual deberemos corregir nuestro modelo.

Un concepto que nos será de utilidad es el concepto de *recurso*[1]. En esencia, llamaremos así a los elementos de un sistema necesarios para una determinada unidad de requisitos funcionales: dada esta última, el término recurso incluye tanto a los elementos con los que estuviera relacionado (roles de usuario, otras unidades de requisitos funcionales, etc) como a aquellos componentes que no figurarían en un diagrama de requisitos funcionales (como podría ser un Diagrama de Casos de Uso), pero que sí estarán presente en el sistema final, tales como datos consultados en la unidad de requisitos funcionales, motor de base de datos, archivos leídos, memoria utilizada, etc.

En las siguientes sub-secciones expondremos los criterios hallados en detalle.

3.4.1. Criterio 1: Cuál es la parte afectada del sistema

Criterio 1.1: Rango del criterio

Afecta a todo el sistema ocurre al definir un interés no funcional sin detallar cuál de las *Unidades de Requisitos Funcionales* (URF) son las que deben cumplir con dicho requisito. Notemos que esto no significa que el interés afecte a todas las URF, ya que es posible que dicha relación pueda cumplirse aplicando el interés a un subconjunto de las URF. Por ejemplo, el criterio "rendimiento" puede cumplirse haciendo eficiente la URF "Búsqueda de material", sin necesidad de aplicar dicho criterio, por ejemplo, a la URF "Ingreso de datos de nuevo lector". Sin embargo, esta decisión de aplicar el requisito a un cierto subconjunto de URFs requiere una comprensión del sistema en su conjunto, por lo que es una situación de compleja resolución - en el caso presentado, es la pericia del analista la que determina que la eficiencia de un sistema puede mejorarse haciendo veloces las URF de búsqueda y lentas las de inserción de datos.

Afecta a una o más URF, pero no al sistema completo este es el caso que encontramos cuando se da un NFI que afecta a un subconjunto particular del sistema. Se diferencia del caso anterior en el sentido de que no ocurre nunca que el NFI afecte a todas las URF del sistema, sino que siempre tendremos la certeza de que afectará a un subconjunto de ellas.

[1] Este concepto está íntimamente ligado al concepto de "parcela de sistema", que expondremos más adelante

No afecta al sistema este es un caso que podemos encontrar, y que si bien no nos presenta mayor utilidad será bueno tener en cuenta. En ocasiones, es posible que se agregue un interés al sistema que, eventualmente, no afecte a ningún elemento del mismo. Este suele ser el caso de intereses cuya implementación no dependen de nuestro sistema en sí, sino de condiciones externas tales como hardware o políticas de empresa. Dichos intereses podrán ser descartados, o al menos destinados a futuras etapas del proceso de desarrollo.

Criterio 1.2: Referencia a las URF que afecta

Afecta a URF de manera explícita este es el caso cuando se nos dice exactamente a qué URF afecta un cierto interés no funcional.

Afecta a URF de manera implícita esta situación ocurre cuando se da un interés no funcional, pero no se explicita a qué URF debe aplicarse. Cuando esto ocurre, será necesario establecer a qué URF hace referencia, lo cual implicará un análisis del entorno en el cual se ubica el nuevo interés. Hay varias maneras en las que esto puede ocurrir.

Afecta a recursos utilizados por varias URF Cuando vinculamos un NFI con un recurso del sistema, estaremos estableciendo de manera implícita una relación entre el interés no funcional y las URF que hacen uso de dicho recurso. Un ejemplo concreto es el caso de aplicar seguridad, y más particularmente integridad, a un conjunto de datos. Si el sistema no nos provee un mecanismo para asegurar dicha propiedad, serán las URF que hagan uso de los datos las que deberán asegurar la integridad de los mismos. Los recursos utilizados por una URF se clasifican en:

- **Recursos de arquitectura** Estos recursos serán recursos físicos o lógicos del sistema sobre el cual se ejecutará la aplicación, tales como base de datos, ancho de banda o tiempos de procesamiento. El manejo de estos recursos siempre es administrado por el sistema operativo, el lenguaje de programación o el servidor web, por lo que un interés no funcional que afecte a uno de estos recursos deberá ser tenido en cuenta al momento de realizar el despliegue de la aplicación, puesto que tratarlo en alguna etapa anterior del proceso de desarrollo difícilmente será más eficiente (o con mejores resultados) que los sistemas ya implementados en un nivel más bajo de abstracción.
- **Información almacenada** A diferencia del caso anterior, los intereses sobre información almacenada varían sobre cómo deben implementarse - en ocasiones será tarea del sistema subyacente a la aplicación, pero también será normal encontrar casos donde dicho requisito deba ser implementado por la aplicación. Si este es el caso, debemos asumir simplemente que el interés se aplica a toda unidad de requisitos funcionales que utilice dicho recurso (en vez de con el recurso).
- **Información en pantalla** Cuando un interés afecte a un recurso de este tipo, se considerará que el interés afecta a las unidades de requisitos funcionales encargadas de mostrar, modificar o agregar a dicha información. Siendo que la aplicación es la capa que se encuentra entre la pantalla del usuario (donde el requisito debe cumplirse) y los datos del sistema (donde el requisito no puede cumplirse), es lógico que sea la aplicación la que se encargue de tomar los pasos necesarios para asegurar la realización del interés.

Afecta a un rol de usuario de los requisitos funcionales Si establecemos un interés no funcional sobre un rol de usuario del sistema, será preciso tener en cuenta algunas consideraciones. Un concepto básico a recordar es que un rol de usuario es un miembro del entorno del sistema, por lo que no nos es posible controlar su comportamiento, y como tal no tiene sentido aplicarle intereses restrictivos. Por ello, estableceremos dichos intereses sobre todas las URF con las que interactuará dicho rol de usuario, de modo tal que garantice el cumplimiento del interés. Hay varias maneras en las que un rol de usuario puede verse afectado.

- **Afecta al rol de usuario directamente** Este es el caso más sencillo, y es cuando establecemos de manera concreta un interés sobre un rol de usuario particular. Las consideraciones expuestas anteriormente serán más que necesarias para lidiar con este caso.
- **Afecta a un rol de usuario más general que el actual**[2] Si un rol de usuario más general que el actual es afectado por un interés entonces eso significará que el rol de usuario especializado también se verá afectado en las funcionalidades que comparte con el rol más general.
- **Afecta a un rol de usuario más particular que el actual** El rol de usuario actual también se verá afectado.

La clasificación anterior podría encuadrarse, a la vez, dentro de una segunda subclasificación, donde se evalúe si el rol de usuario afectado es un rol interno o externo (en la forma de aplicaciones externas). Si bien la interacción de ambos con el sistema es similar, la naturaleza de ambos es muy diferente entre sí, por lo que cada caso se tratará de forma individual, hablando de *rol de usuario* en el primer caso, y de *aplicación externa* en el segundo. Es esperado que esta diferencia se haga más profunda en etapas posteriores del proceso MDA.

3.4.2. Criterio 2: Cómo impactan las cualidades no funcionales al sistema

Otro aspecto a considerar es la manera en la que se verá impactada una unidad de requisitos funcionales por un interés no funcional. Este criterio tiene sentido solamente cuando pensamos en URFs que se describen mediante un conjunto de eventos, conjunto cuyos elementos se relacionan mediante un orden parcial (tal como ocurre en los CU, escenarios, procesos de negocios, etc.) y que poseen las siguientes características:

- Existen eventos iniciales (es decir, que no tienen eventos predecesores) y eventos finales (es decir, que no tienen eventos sucesores)
- Todo evento que no es ni inicial ni final tiene eventos predecesores y eventos sucesores
- Cada URF tiene asociada la idea de *ejecución* de las mismas, donde una *ejecución* consistirá de una secuencia de eventos tal que comienza con un evento inicial, termina en un evento final, y cada evento intermedio tiene un predecesor y un sucesor en la secuencia[3]

Tomando este marco de referencia, hay dos cuestiones a considerar: cómo se inserta el nfi dentro de la unidad de requisitos, y qué ocurre con la funcionalidad afectada. Existen tres formas posibles de que la implementación de un NFI se inserte: inmediatamente antes de que ocurre un evento de una URF (es decir, luego de los eventos predecesores en la URF a ese evento), en un evento de una URF (es decir, en el momento en que ocurre el evento), o inmediatamente después de un evento de una URF (o sea, antes de los eventos sucesores en la URF a ese evento); si nos preguntamos en cambio qué ocurre con la ejecución de la URF, esto se reduce a preguntarnos si la misma es interrumpida hasta que tiene lugar el nfi (es decir, se detiene la ejecución de la aplicación, se inserta la ejecución del nfi y una vez que esta termina se continua con la ejecución de la aplicación), ya sea de forma temporal o permanente (abandonando la ejecución de la URF), o si continúa mientras el NFI es considerado. Analizando cada caso por separado, el resultado es el siguiente:

- Si la URF *interrumpe su ejecución temporalmente*,

[2]Supongamos que existen dos roles de usuario, A y B, donde B es una generalización del rol A. En dicho caso, diremos que el rol B es más general que el rol A, y a la vez consideramos al rol A más particular que el rol B2

[3]Debe resultar evidente que dicha secuencia respeta también el orden parcial

inmediatamente antes del evento es equivalente a insertar la ejecución del NFI antes del evento, pero después de los eventos predecesores al evento. A este caso lo llamaremos **antes**.

en el evento este caso no tiene sentido porque el nfi se inserta luego que la URF interrumpió su ejecución y este momento no puede ser el momento en que ocurre un evento de la URF.

inmediatamente después del evento es equivalente a insertar la ejecución del NFI luego del evento pero antes de los eventos sucesores al evento. A este caso lo llamaremos **después**.

- Si la URF *interrumpe su ejecución permanentemente* esto implica que, a partir de un momento dado, la URF detiene su ejecución y se inserta la ejecución del nfi, tal que cuando el nfi termina de ejecutarse no se reanuda la ejecución de la URF (es decir, se renuncia a seguir ejecutando la URF). Teniendo esto en mente,

  inmediatamente antes del evento el evento seleccionado en la URF no se ejecutará; inmediatamente antes del mismo se habrá interrumpido la ejecución de la URF para siempre, y habrá comenzado la ejecución del NFI. A este caso lo llamamos **reemplazar**.

  en el evento este caso es igual al del ítem anterior.

  inmediatamente después del evento este caso implica que una vez que ocurrió el evento de la URF se interrumpe para siempre su ejecución y tiene lugar la ejecución del NFI. Este es también un caso de **reemplazar**.

- Si la URF afectada *continúa ejecutándose*, se considera que en el intervalo cuyos extremos son dos eventos de la URF (relacionados por la relación de orden parcial) tiene lugar la ejecución el nfi.

  inmediatamente antes del evento dado que ïnmediatamente antes de un evento.^{es} equivalente a "después de los antecesores de ese evento", este caso es el mismo que el siguiente.

  en el evento supongamos que nos referimos al evento e de una URF; entonces existe un evento e' tal que dentro del intervalo (e, e') tiene lugar el tratamiento del nfi en paralelo con la ejecución de la URF. A este caso lo llamaremos **durante**.

  inmediatamente después del evento como ïnmediatamente después de un evento.^{es} también .^{a}ntes de los sucesores de ese evento", este caso es el mismo que el anterior.

3.4.3. Criterio 3: Nivel de detalle del Interés No Funcional

Brinda el Interés de manera muy general este caso ocurre cuando el interés que se exige es dado de manera general, casi vaga, con criterios tales como "esta tarea debe realizarse de manera segura" o "el sistema debe ser portable". En estos casos, será tarea del analista desarrollar el NFI y deducir (probablemente con la ayuda del cliente) cual es la mejor manera de realizarlo. Este tipo de intereses, debido a su falta de especificidad, difícilmente nos serán de utilidad en su forma original.

Brinda un Interés más detallado Este caso es similar al anterior, pero ocurre cuando el cliente brinda información más concreta con respecto al interés, tal como "los usuarios deberán ser autenticados antes de ingresar" en lugar de "el sistema debe ser seguro". Dependiendo de qué tan general (o específico) sea este NFI, será mayor o menor el trabajo del analista al momento de establecer cómo aplicar este interés a las URF pertinentes.

Brinda una posible implementación del NFI Este tercer caso se aplica a las situaciones donde el cliente describe ya no un interés, sino las operaciones para implementarlo. Esto ocurrirá generalmente cuando se trate de operaciones conocidas, tales como los formularios de login y password o los sistemas de *captcha* para evitar comentarios automatizados (donde ambos son operacionalizaciones del interés no funcional "autenticación de usuarios"). Cuando esto ocurra, deberemos incorporar dichas operaciones sin mayor análisis.

3.4.4. Criterio 4: Etapa del Desarrollo en la que se implementa el Interés No Funcional

No todos los intereses se implementan de igual manera. Algunos de ellos deben ser tenidos en cuenta en etapas tempranas del desarrollo (seguridad vía autenticación de usuarios siendo el ejemplo clásico), mientras que otros pueden corresponderse con etapas posteriores del proceso MDA (la realización "implementar sistema de espejado de discos vía RAID" debe considerarse en la etapa de despliegue del sistema). Por ello, dividiremos los intereses en varias categorías, correspondientes a la etapa en la cual deben considerarse.

Un detalle adicional: si bien para esta etapa solamente debemos diferenciar entre los intereses que se consideran tempranamente y los que no, la clasificación que consideraremos es un poco más completa. Esto se debe a que esta información puede ser de utilidad en etapas posteriores, por lo que si no fueran consideradas en la fase de Ingeniería de Requisitos, deberían ser tenidas en cuenta igualmente en etapas futuras.

Ingeniería de requisitos Encontraremos este primer caso cuando la manera de implementar un cierto interés sea teniéndolo en cuenta durante la etapa de la especificación. Ejemplo: el interés no funcional "extensibilidad" (que indica el nivel de dificultad que tendrá la adición de funcionalidades al sistema una vez terminado su desarrollo) es un interés que, de no ser considerado durante la especificación, difícilmente será tenido en cuenta en etapas posteriores, por lo que obviarlo en esta etapa significará con seguridad que dicho interés no será implementado. Un detalle a tener en cuenta es que si al momento de realizar este análisis la especificación ya ha sido completada, que esta sub-clasificación funciona también como un chequeo de corrección de la misma - si encontramos un interés de este tipo, y la especificación misma no nos asegura su correcta implementación, entonces deberemos reconsiderar dicha especificación.

Desarrollo (análisis y diseño) Estos intereses harán referencia al sistema en sí, por lo que constituyen gran parte del trabajo de los desarrolladores. Un ejemplo simple será el caso "la búsqueda de libros debe ser veloz", tarea que significará para el desarrollador implementar búsquedas optimizadas, chequear restricciones de acceso de forma más eficiente o funcionalidad similar, la cual que se apreciará en el código mismo del sistema.

Despliegue Los intereses de esta categoría, por ser considerados únicamente en la etapa de despliegue, son requisitos relacionados tanto con el hardware sobre el cual se ejecutará el sistema como con el medio físico que lo rodea. El interés "acceso restringido al servidor", por ejemplo, parte del conjunto de intereses sobre seguridad, es un ejemplo de este caso.

Mantenimiento Llamamos mantenimiento a propiedades extra que podemos necesitar asegurar que se cumplirán en el sistema una vez que el mismo haya sido completado, pero que deben haber sido contemplados previamente. Un ejemplo práctico puede ser el RNF "debe ser posible extender la funcionalidad del sistema de manera sencilla", requisito que puede lograrse mediante la inclusión de un soporte para plugins. Así, si bien el sistema de plugins no influye significativamente en el sistema, es una característica que será necesaria en algún punto del futuro, cuando sea necesario añadir nuevas funcionalidades al sistema.

Externo al sistema Este caso se menciona únicamente por completitud, ya que en general no será de utilidad. Es posible que ciertos intereses no sean sobre el sistema, el proceso de desarrollo o el despliegue de la aplicación, sino sobre factores externos al sistema. Un ejemplo es el interés "productividad del equipo de desarrollo", que es un interés que referencia más al entorno de desarrollo del sistema que al sistema en sí.

Como hemos dicho anteriormente, bajo este esquema de clasificación encontramos ciertas combinaciones de los criterios anteriores para RICNF que son fácilmente descartables por corresponder a requisitos que no deben de estar presentes en una especificación realizada correctamente, por lo que de encontrarlos será preciso descartarlos en su forma actual y trabajar en una refinación de los

NFI del RICNF. Encontraremos también combinaciones que, si bien no son incorrectas, requerirán trabajo extra por parte del analista, ya que será él el que decida de qué manera se debe refinar el NFI para aplicarlo de forma correcta. También es muy posible que haya aspectos que aparezcan recién en diseño de bajo nivel o implementación.

La Figura 3.2 nos muestra de qué manera clasificar (y mas aún, descartar/refinar) RICNF. Las cruces muestran clasificaciones que deben ser descartadas, las tildes indican cuáles pueden ser aceptadas, y aquellas que permanecen sin marcar son aquellas que requerirán trabajo extra por parte del analista al momento de asociar los Intereses No Funcionales con las Unidades de Requisitos Funcionales.

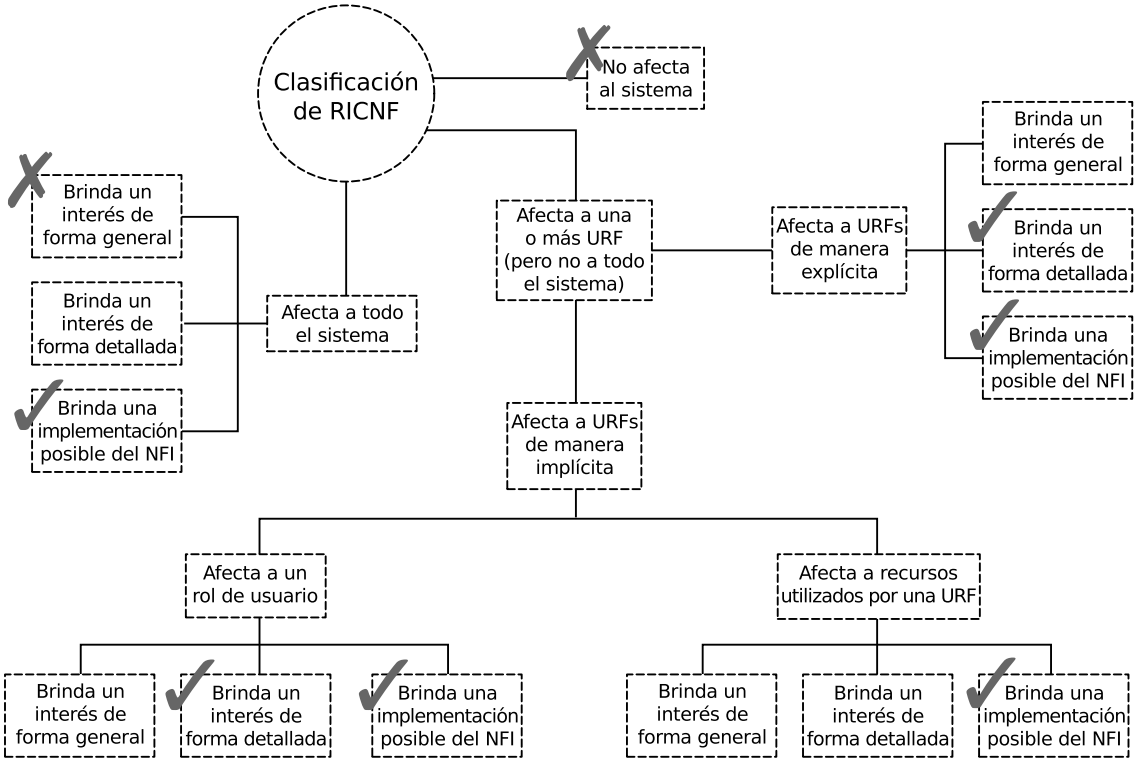

Figura 3.2: Diagrama de Clasificación de Intereses No Funcionales

Las sub-clasificaciones aceptadas son aquellas que proveen un criterio suficientemente específico; las clasificaciones rechazadas, en cambio, no poseen un único criterio de rechazo. En particular,

- Afecta a todo el sistema → Brinda el interés en forma general: Este requisito resulta demasiado vago como para ser implementado
- No afecta al sistema: Incluido por completitud, resulta trivial descartarlo

Hemos obviado en esta clasificación el Criterio 4, "Etapa del Desarrollo en la que se implementa el NFI", ya que aquellos que no pertenezcan a la etapa de Ingeniería de Requisitos serán simplemente ignorados.

3.5. Metamodelo MICNF

Una vez obtenidos los criterios mencionados en la sección anterior, los resultados obtenidos debieron ser plasmados en un metamodelo. Este metamodelo, llamado MICNF (siglas de *Modelo de Impacto de Cualidades No Funcionales*), se presenta en la figura 3.3

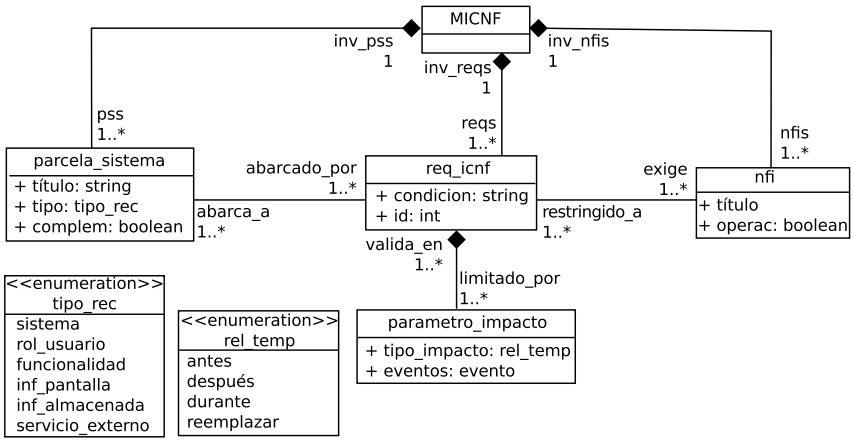

Figura 3.3: Modelo de Impacto de Cualidades No Funcionales

3.5.1. Elementos de Modelado

Clase nfi

Descripción La clase NFI (del inglés, *Non-Functional Item*) representará tanto cualidades no funcionales (las cuales se grafican con nubes de contorno fino como en el framework NFR) como soluciones concretas que permiten alcanzar cualidades no funcionales (las cuales se grafican con nubes de contorno grueso como en NFR) como operaciones decisiones de diseño, decisiones de arquitectura, etc..

Atributos

- **titulo:** `string`
  El título del NFI, indica el nombre de la acción que realiza y/o nombre de cualidad no funcional.
- **operac:** `bool`
  Indica si el ítem representa una operación (`operac=true`)

Asociaciones

- **restringido_a:** `req_icnf [1..*]`
  Requisito ICNF del cual forma parte la meta representada por el nfi actual.
- **inv_nfis:** `MICNF [1]`
  Objeto MICNF padre de todo el modelo.

Restricciones
Sin restricciones adicionales

Semántica Las relaciones entre la clase nfi y las demás clases (por intermedio de la clase `req_icnf`) representan la noción de que el nfi representado afecta a esos elementos en concreto. Por ejemplo, el caso más frecuente será el de asociación entre un Interés No Funcional (representado por un nfi) y una unidad de requisitos funcionales (instancia de la clase parcela_sistema con atributo *tipo=funcionalidad*), donde se establecen requisitos adicionales que debe cumplir una URF, generalmente en la forma de operaciones insertadas en algún paso concreto de un cierto escenario de una URF.

Notación

Figura 3.4: Representación de una instancia de la clase nfi

Cada instancia de la clase `nfi` se representará mediante una nube, de borde fino si proviene de un Interés No Funcional, o de borde grueso si se asocia con su realización, como se aprecia en la Figura 3.4.

Clase parcela_sistema

Descripción Diremos que una Parcela de Sistema (PS) es una parte de nuestro sistema sobre la cual podemos imponer RNF de algún tipo; una Parcela de Sistema debe ser interpretada como un conjunto de Unidades de Requisitos Funcionales (URF).

Atributos

- **tipo:** `tipo_rec`
  Indica qué tipo de recurso representa. `tipo_rec` es un tipo enumerado que adopta los siguientes valores: `sistema`, `rol_usuario`, `funcionalidad`, `inf_pantalla`, `inf_almacenada` y/o `servicio_externo` (ver semántica para más detalles)
- **titulo:** `string`
  Título de la unidad funcional. Dependiendo del atributo `tipo`,
  - cuando el tipo sea `sistema`, será el nombre que reciba el sistema
  - cuando el tipo sea `rol_usuario`, el título será el nombre de un rol al que representa
  - cuando el tipo sea `funcionalidad`, será el nombre de la URF que implementa dicha funcionalidad o el nombre de una funcionalidad que aparece en la descripción de una URF.
  - cuando el tipo sea `inf_pantalla` o `inf_almacenada`, el título describirá el recurso en cuestión
  - cuando el tipo sea `servicio_externo`, el título indicará el nombre de un servicio externo
- **complem:** `boolean`
  (`complem=true`) quiere decir que debe considerarse el complemento de la PS citada. La PS que es el complemento de una PS p está formada por todas las URF del sistema excepto las de la PS p.

Asociaciones

- **abarcado_por:** `req_icnf [1..*]`
  Requisito de Impacto de Cualidades no Funcionales del cual forma parte la PS actual.
- **inv_pss:** `MICNF [1]`
  Objeto MICNF padre de todo el modelo.

Restricciones
Sin restricciones adicionales

Semántica En el Modelo de Impacto de Cualidades No Funcionales, se busca representar de qué manera los intereses no funcionales (representados por los `nfi`) afectan a las diversas Unidades de Requisitos Funcionales del sistema. Para poder referirnos al conjunto de las Unidades de Requisitos Funcionales afectadas por un Interés No Funcional utilizamos el elemento `parcela_sistema`. Cada instancia de la clase `parcela_sistema` representa un subconjunto de todas las Unidades de Requisitos Funcionales del sistema.
Si analizamos qué parcela de sistema abarca cada tipo, obtendremos que:

- La PS `sistema` hace referencia a todas las Unidades de Requisitos Funcionales, sin excepción.
- La PS `funcionalidad`, se corresponde de manera directa con una URF o con una funcionalidad que aparece dentro de la descripción de una URF.
- La PS `rol_usuario` hace referencia al conjunto de URFs con las cuales interactúa el Rol de Usuario.

- Nos referimos por `inf_almacenada` a información persistente (p.ej. guardada en archivos, bases de datos, etc.). La PS `inf_almacenada` hace referencia así al conjunto de URFs del sistema en las cuales se acceda a la información almacenada mencionada en el título de la UF.
- Nos referimos por `inf_pantalla` a contenido que puede ser mostrado en pantalla por el sistema o valores (en algún dominio) que pueden ser ingresados por un rol de usuario. La PS `inf_pantalla` hace referencia así al conjunto de URFs del sistema dónde se cita la información en pantalla mencionada en el título de la PS.
- Finalmente, la PS `servicio_externo` consiste de todas las URF que requieren la ejecución de un servicio externo. Tradicionalmente, en un Modelo de Casos de Uso este tipo de sistemas tienden a identificarse como actores; en nuestro caso, sin embargo, hemos considerado más conveniente tratarlos como una entidad en sí mismo.

Notación La notación de cada instancia de esta clase depende directamente del atributo tipo. Dependiendo del valor del atributo, utilizaremos alguna de las notaciones descriptas en la Figura 3.5:

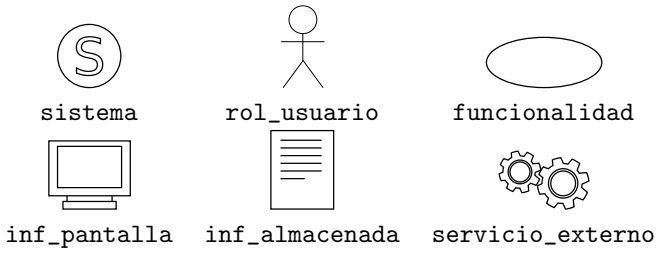

Figura 3.5: Representación de instancias de la clase parcela_sistema

Clase parametro_impacto

Descripción Al analizar el impacto de una Cualidad No Funcional es necesario especificar cuándo un NFI se va a cumplir (i.e. a partir de qué eventos de URFs funcionales) y de qué manera va afectar su cumplimiento al sistema (i.e. cómo afecta a los escenarios de las URFs funcionales). Para referirnos a ambas cosas introducimos el elemento clase `parametro_impacto`.

Atributos

- **tipo_impacto:** `rel_temp`
  Representa de qué manera el cumplimiento de un NFI afectará a escenarios de Unidades de Requisitos Funcionales. Puede tomar valores "*antes*", "*después*", "*durante*" o "*reemplazar*" (ver semántica)
- **eventos** `string[1..*]`
  Es una lista de eventos que deberían suceder en escenarios de alguna Unidad de Requisitos Funcionales

Asociaciones

- **válida_en:** `req_icnf [1]`
  Requisito de Impacto de Cualidades no Funcionales sobre el cual se aplicará la restricción.

Restricciones
En los casos en que tipo_impacto sea `antes` o `después`, la lista eventos será una lista de un único elemento. En los casos `durante` o `reemplazar` la lista podrá tener más de un elemento, pero los eventos deberán representar una secuencia continua de eventos (dentro de un escenario de alguna Unidad de Requisitos Funcionales). Ver semántica para más detalles

Semántica Para las instancias de una clase `parametro_impacto` pueden tenemos los siguientes casos:

- **Antes** e**:** en una URF donde ocurre el evento e cumplir con un NFI inmediatamente antes de que ocurra el evento e y luego del evento anterior a e en la URF.
- **Después** e**:** similar al anterior, en una URF donde ocurre el evento e cumplir con un NFI inmediatamente después de que ocurra el evento e y antes del evento siguiente a e en la URF.
- **Durante** e_1, e_2**:** el NFI se cumple durante cada intervalo dado por los eventos 1 y 2 (en ese orden) de la ejecución de la URF de la PS[4].
- **Reemplazar** e**:** el NFI se cumple justo luego del evento e, y a partir de ahí reemplaza a la ejecución URF que estaba en ejecución.

Dado que el caso *durante* abarca a más de un evento, en estas situaciones se entenderá que la lista de eventos representa una secuencia continua de eventos que ocurren dentro de algún escenario. Así, si `tipo_impacto=durante`, y eventos posee elementos (e_1, e_2), esto se interpretará como que el requisito de impacto del NFI se insertará en la URF en el intervalo de eventos descripto, es decir, luego del evento e_1 y antes del evento e_2.

Notación Dado que las instancias de esta clase se encuentran siempre asociadas a instancias de la clase `req_icnf` (cuya representación visual es una flecha), cada instancia de esta clase se representará mediante un reloj, acompañado de un operador de tipo `rel_temp` y la lista de eventos sobre los que aplica.

[4]En aplicaciones web esto puede lograrse de varias maneras: si bien el caso más sencillo es mediante un script de lado del cliente (Javascript siendo el más popular) que se ejecute en paralelo, también puede lograrse, por ejemplo, tomando información del sistema antes y después de la URF y evaluando dicha información (es decir, convirtiendo una ejecución en paralelo en una ejecución secuencial)

Clase req_icnf

Descripción Este elemento de modelado se va a usar para describir requisitos de impacto de intereses no funcionales sobre conjuntos de funcionalidades de la aplicación. Un requisito de impacto de intereses no funcionales sobre el sistema explica de qué modo uno o más NFI afectan a una o más PS. Una instancia de `req_icnf` encuentra sentido en sus relaciones con las clases `nfi`, `parcela_sistema`, una o más instancias de `parametro_impacto` y una condición. Una instancia de esta clase contiene toda la información sobre el impacto de las Cualidades No Funcionales.

Atributos

- **id:** `string`
  Un identificador interno de la clase. Útil para la implementación en código.
- **condicion:** `string`
  En qué estado tiene que estar el sistema para que puedan satisfacerse los intereses no funcionales del requisito ICNF.

Al momento de fijar una condición es importante recordar que dicha condición no debe ser del tipo "`rol_usuario` es el Rol de Usuario A", ya que esto debe expresarse agregando Parcelas de Sistema al requisito.

Asociaciones

- **exige:** `nfi [1..*]`
  Los NFI que intervienen en el requisito de impacto de CNF.
- **limitado_por:** `parametro_impacto [1..*]`
  Conjunto de parámetros de impacto necesarios para explicar de qué manera los NFI de un RICNF afectan a las Parcelas de Sistema del mismo.
- **abarca_a:** `parcela_sistema [1..*]`
  Las Parcelas de Sistema que intervienen en el RCC.
- **inv_reqs** `[1]`
  Objeto MICNF padre de todo el modelo.

Restricciones
No posee restricciones adicionales

Semántica Consideramos los siguientes casos para una instancia I de `req_icnf`:

- **Un parametro_impacto y una o más parcela_sistema:** Una instancia I de esta clase representa por medio de la instancia `parametro_impacto` de I la forma en que se entrelazará la ejecución de las operacionalizaciones, necesarias para satisfacer los NFI de I, con la ejecución de las URF que están en todas las parcelas de sistema de I a la vez y que contienen todos los eventos del `parametro_impacto`, siempre y cuando se cumpla el estado en que tiene que estar el sistema dado por la condición de I. Adicionalmente, si una de las PS es `rol_usuario` A, entonces las URF contempladas por I tienen que haber sido disparadas por el Rol de Usuario A.
- **Varios parametro_impacto y una o más parcelas de sistema:** Una instancia I de esta clase representa, para cada una de las instancias `parametro_impacto` de I, la forma en que se entrelazará la ejecución de la realización de los NFI (necesarias para satisfacer los NFI de I) con la ejecución de las URF que están en todas las PS de I a la vez y que contienen todos los eventos de esa instancia `parametro_impacto`, siempre y cuando se cumpla el estado en que tiene que estar el sistema dado por la condición de I.

Notación Dado que esta clase representa un vínculo entre diversos elementos, es lógico asumir como su representación la de una flecha. En este caso, la punta de la flecha apuntará a las PS, la cola a los NFI, y en el cuerpo de la misma se incluirán los parámetros de la relación.

Gráficamente, un ejemplo concreto es el de la Figura 3.6 (en donde también veremos cómo se representan la condición y los parámetros del requisito):

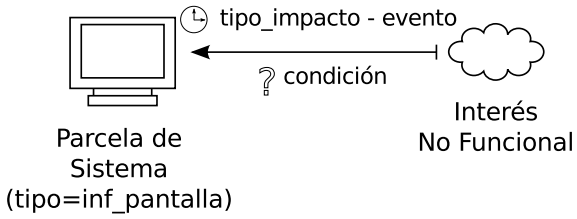

Figura 3.6: Representación de una instancia de la clase req_icnf

Hay varios casos particulares a considerar, que son los casos en los cuales se incluye más de un NFI o más de una unidad funcional. En estos casos, procederemos utilizando una flecha con múltiples cabezas y/o colas, como se aprecia en las Figuras 3.7 y 3.8.

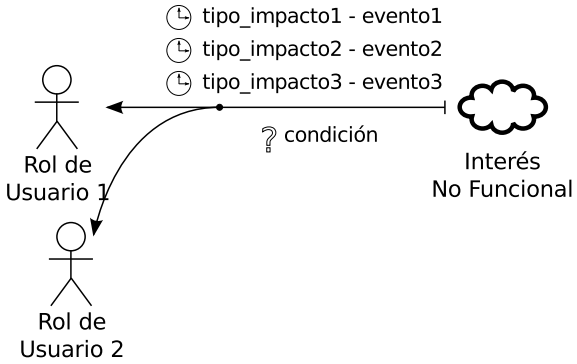

Figura 3.7: Instancia de la clase req_icnf - Múltiples PS, un único NFI

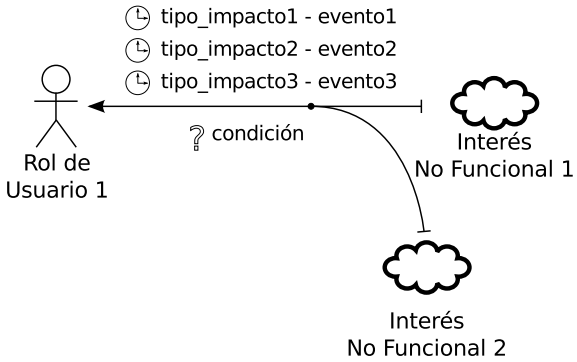

Figura 3.8: Instancia de la clase req_icnf - Una única PS, múltiples NFI

En el caso de que tengamos múltiples Parcelas de Sistema y NFIs involucrados, abandonaremos parcialmente esta notación para introducir un nuevo esquema, en el cual damos una caja

(representando al RICNF) que une mediante flechas a los diversos elementos de la condición, como se detalla en la Figura 3.9

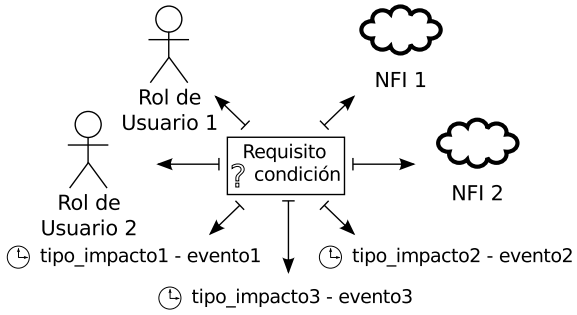

Figura 3.9: Instancia de la clase req_icnf - Múltiples PS y NFI

A modo de convención, siempre ubicaremos las Parcelas de Sistema a la izquierda y los NFI a la derecha, con los diversos tipos de parámetros y/o eventos en el medio

3.5.2. Combinaciones de Parcelas de Sistema

Si bien hasta el momento hemos mencionado que las diversas PS pueden combinarse, ciertas combinaciones no aportarán ningún tipo de información que no se pueda obtener de manera directa por una única PS. En particular,

- Si una de las PS es el sistema completo, entonces agregar otra PS hace innecesaria a la primera, ya que la intersección de ambas dará como resultado la segunda PS.
- Si el afectado es un Rol de Usuario, es importante notar que cualquier otra PS debe ser tal que sea este rol el que accede al recurso/realiza la acción. De otro modo, la condición implícita de que ocurran ambas cosas simultáneamente nunca será verdadera.
- De manera más general, siempre que se combinen dos recursos es preciso asegurarse de que los eventos a los cuales hacen referencia implícitamente (a través de las URF que los componen) posean una intersección no vacía.

Capítulo 4

Otros Modelos de Requisitos

4.1. Framework NFR

En la Ingeniería de Requisitos de Software, los Requisitos No Funcionales son requisitos de software que describen no lo que el software hará, sino cómo el software lo hará; ejemplos de esto son los requerimientos de rendimiento, requisitos de la interfaz externa del software, límites para el diseño y/o atributos de calidad. Los Requisitos No Funcionales son difíciles de evaluar, y por lo tanto, con frecuencia son evaluados de forma subjetiva.

En contraste a los acercamientos orientados por funcionalidad, el Framework NFR utiliza requisitos no funcionales tales como seguridad, rendimiento y costo para orientar el proceso de diseño. El framework apunta a incorporar los requisitos no funcionales en la mente de los desarrolladores. Los principales pasos en el proceso de diseño son:

- Adquirir conocimiento acerca del dominio particular del sistema, los requisitos funcionales del mismo, tipos particulares de RNF y técnicas de desarrollo asociadas
- Identificar RNF particulares para el dominio
- Descomponer los RNF
- Identificar "operacionalizaciones" (alternativas posibles de diseño para cumplir con los RNF del sistema objetivo)
- Tratar con ambigüedades, prioridades, concesiones, e interdependencias con otros RNF y operacionalizaciones
- Seleccionar operacionalizaciones
- Aprobar decisiones de diseño con racionalidad
- Evaluar el impacto de las decisiones

Buscando crear un mecanismo mediante el cual sea posible obtener información relevante a cada etapa del proceso de experiencias anteriores, surge el Framework NFR. Este Framework busca proveer una estructura para representar y almacenar el proceso de diseño en gráficos, llamados *Grafos de Interdependencia de software*, o SIG (del inglés, *Software Interdependency Graph*). El Framework también provee catalogación de conocimiento acerca de los RNF y su diseño, incluyendo técnicas de desarrollo.

La operación del Framework puede visualizarse en término de la construcción incremental e iterativa, elaboración, análisis y revisión de un SIG. los conceptos principales del Framework NFR aparecen en forma gráfica. Los **softgoals**, que son "suaves" por naturaleza, aparecen en forma de nubes. Los requisitos principales se muestran como Softgoals en la parte superior del gráfico, y

son conectados mediante **enlaces de interdependencia**, los cuales se muestran como líneas, y frecuentemente como flechas. Los Softgoals tienen asociados **etiquetas** (valores que representan el grado al cual una meta suave ha sido lograda), que se utilizan para proveer soporte al proceso de razonamiento durante el diseño. Las interdependencias muestras refinamientos de Softgoals "padre" bajando hacia otros Softgoals "hijos" más específicos. También muestran la **contribución** (impacto) de los hijos hacia los Softgoals superiores. Para determinar si un Softgoal ha sido cumplido, existe un procedimiento de evaluación, el cual toma en consideración etiquetas y contribuciones, pero también decisiones del desarrollador [9].

4.1.1. Conceptos

Softgoal Los *Softgoals* son la unidad básica para representar requisitos no funcionales. Ayudan al desarrollador a tratar con RNFs, los cuales pueden ser de naturaleza subjetiva, relativa y/o interactiva. Existen tres tipos de softgoals:

- **Softgoals NFR**, que representan Requisitos No Funcionales a satisfacer. Actúan en general como restricciones sobre el sistema, y son satisfechos por los softgoals operacionalización que representan componentes de diseño o implementación.
- **Softgoals Operacionalización**, que son técnicas de desarrollo que ayudan a satisfacer los RNFs. Son los únicos que aparecen de alguna forma (ya sea como una operación, representación de datos, asignación de un agente a una tarea, etc.) en el diseño o implementación de destino.
- **Softgoals Claim (aserción)**, los cuales ayudan a justificar decisiones. Proveen justificativos para las decisiones de diseño.

Cada softgoal tiene asociado un *tipo NFR* y uno o más *tópicos*. Si tomamos como ejemplo "*Precisión de una cuenta*", "*Precisión*.es el tipo y "*Cuentas*.es el tópico.

Interdependencias Los Softgoals se interrelacionan vía *interdependencias*. En dirección "descendente", un *refinamiento* toma un softgoal (el *padre*) y produce uno o más softgoals *descendientes*. En dirección .ªscendente", los softgoals descendientes hacen *contribuciones*, positivas o negativas, para satisfacer el softgoal padre.

- Existen tres tipos de **Refinamiento**: descomposición, operacionalización y argumentación. El primero refina un softgoals en otros del mismo tipo (es decir, en Softgoals NFR, Softgoals operacionalización y/o *Claims*); el segundo refine un Softgoal en Softgoals Operacionalización, mientras que el tercero incorpora notas sobre decisiones de diseño.
- Existen también distintos tipos de **Contribuciones**: *AND* y *OR* son dos contribuciones donde se determina que si todos los descendientes de un softgoal se satisfacen, también se satisface el padre (contribución *AND*) o bien donde el satisfacer uno solo de los descendientes es suficiente para satisfacer al padre (contribución *OR*); paralelamente a este tipo de contribución, que relaciona un grupo de descendientes con su padre, también existen otros tipos de contribución que relacionan a un sólo descendiente con su padre. Dependiendo de qué tan fuerte sea esta contribución, se clasifica como *MAKE* (contribución positiva fuerte), *HELP* (contribución positiva media), *SOME+* (contribución positiva débil), *SOME-* (contribución negativa débil), *HURT* (contribución negativa media) o *BREAK* (contribución negativa fuerte). Finalmente, si este dato se desconoce, existe la contribución neutra *UNKNOWN*.

4.1.2. Metamodelo

En [41], Suppakul, Chung y Mylopolous definen un metamodelo para el Framework NFR en la forma de un perfil UML. Siendo ellos co-autores de [9], en donde se introdujo por primera vez dicho Framework, la decisión de tomar este metamodelo como punto de partida se deriva de

manera natural. Ahora bien, a lo largo del desarrollo se ha optado por un modelo ligeramente simplificado, que respeta los lineamientos principales del perfil original de [41], pero que elimina ciertas características que, por ajustarse a lineamientos de estilo, resultan confusas y/o innecesariamente complicadas; adicionalmente, este perfil utiliza el mecanismo de clases de asociación, el cual no está definido en MOF, por lo cual debieron introducirse nuevas meta-asociaciones en su reemplazo. De este modo, el metamodelo implementado finalmente (y el utilizado a lo largo del presente proyecto) es el ilustrado en la Figura 4.1.

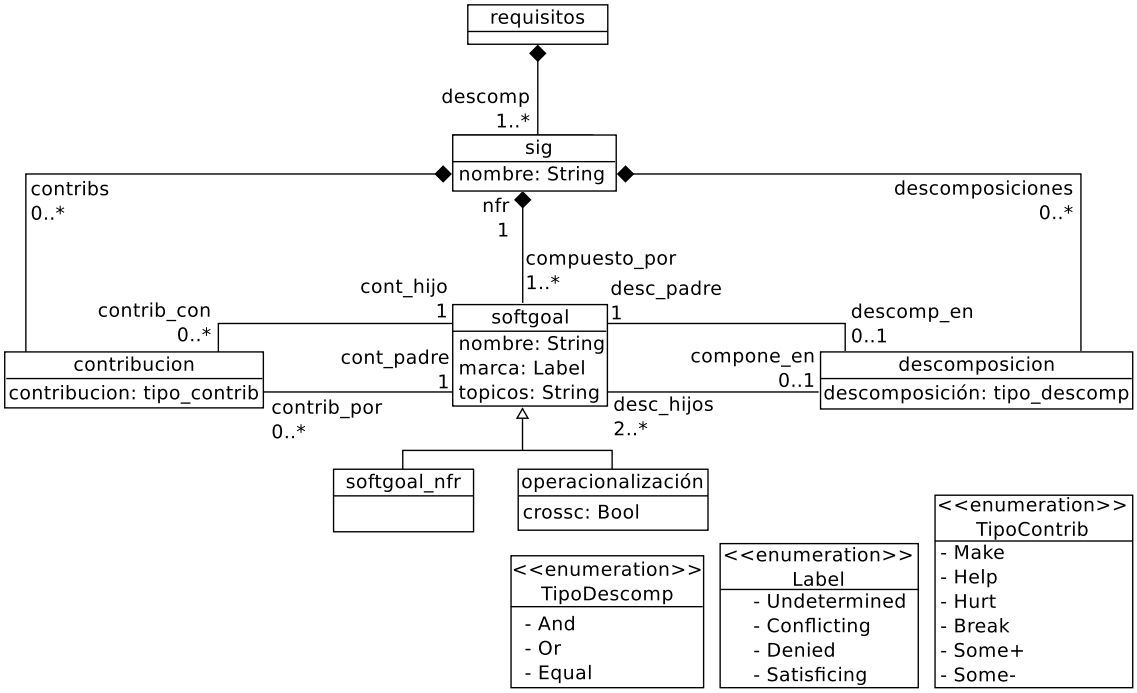

Figura 4.1: Metamodelo para Softgoal Interdependency Graphs

4.2. Modelos de CU

Uno de los diagramas más populares provistos por UML, y uno de los más sencillos, es el Diagrama de Casos de Uso. En este diagrama, se presenta una representación gráfica de la funcionalidad provista por un sistema en término de actores (los cuales pueden representar roles de usuario), sus objetivos (representados como casos de uso) y las dependencias entre estos elementos. El principal objetivo de este diagrama es ilustrar cuáles funcionalidades del sistema son realizadas por cuáles actores.

Dado que en el presente proyecto los Diagramas de Caso de Uso tendrán un rol importante, se consideró necesario implementar un Metamodelo para representarlos. Este metamodelo se presenta en la Figura 4.2, y su interpretación resultará sencilla para todo aquel que esté familiarizado con este tipo de diagramas. Es preciso aclarar, sin embargo, que el modelo implementado es una versión simplificada del Diagrama de Casos de Uso presentado en [30], siendo tal vez el detalle más sobresaliente el hecho de que las relaciones *extend*, *includes* y *realiza* no sean en nuestra versión elementos de modelado (lo cual involucraría la introducción de metaclases adicionales). Esto obedece al hecho de que nuestra representación actual simplifica de manera notable las expresiones XMI necesarias para identificar elementos, más sin perder por ello poder expresivo. A modo de comparativa, se incluye en la Figura 4.3 un metamodelo sin modificar, donde puede apreciarse un notable incremento tanto de las metaclases necesarias como de las meta-asociaciones

entre ellas.

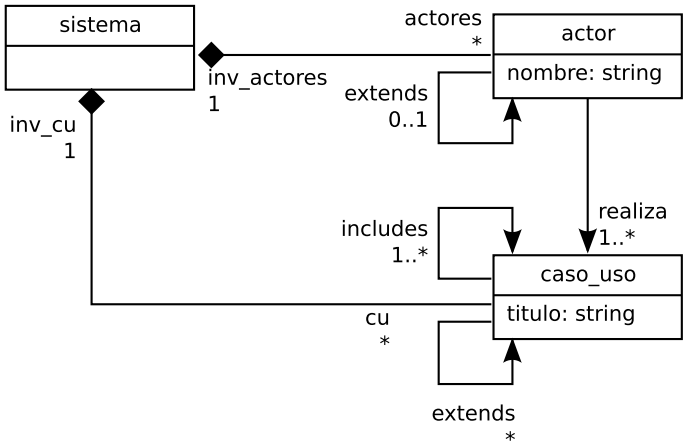

Figura 4.2: Metamodelo para Casos de Uso

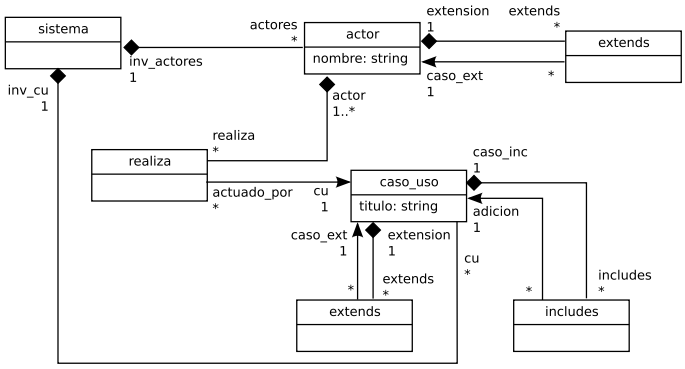

Figura 4.3: Metamodelo para Casos de Uso - versión de UML

4.3. Descripciones de Casos de Uso

Tradicionalmente, el mecanismo definido por UML para la captura de requisitos funcionales ha sido el Modelo de Casos de Uso (usados dentro de la metodología AOWE). Dichos CU debían ser descritos en detalle; para ello son ampliamente usadas las descripciones textuales de CU (proveyendo una descripción, entre otras, de sus escenarios). Es deseable expresar las descripciones de CU de manera textual en un modelo formal con su sintaxis; si no lo hiciéramos de esta manera, la automatización del proceso (recordemos que, según las ideas claves de MDA, es deseable que el proceso sea uno sólo, variando únicamente los modelos de entrada para obtener una salida distinta) sería más compleja (por ejemplo, involucraría procesamiento de lenguaje natural), tanto para el desarrollador como para los usuarios. Debido a la estrecha relación entre UML y XML, se optó por utilizar esta última herramienta para la formalización de la sintaxis de los requisitos, cuya sintaxis estará dada por el DTD de la Figura 4.4.

La sintaxis del DTD, como puede apreciarse, respeta fuertemente la sintaxis del MDCU (similar a la utilizada, entre otros, en [10]), pero incluye también ciertos campos con información extra que nos facilitará su posterior interpretación:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT CUs (descripcionCU+)>
  <!ELEMENT descripcionCU (base, descripcion, alternativa*)>
    <!ELEMENT descripcion (pasodesc+)>
      <!ELEMENT pasodesc (actor_desc, accion_desc, objeto_desc,
                          destinatario_desc*, condicion_desc*)>
        <!ATTLIST pasodesc numpasodesc CDATA #REQUIRED>
        <!ELEMENT actor_desc (#PCDATA)>
        <!ELEMENT accion_desc (#PCDATA)>
          <!ATTLIST accion_desc io_desc (entrada|salida|otro) #IMPLIED>
        <!ELEMENT objeto_desc (#PCDATA)>
        <!ELEMENT destinatario_desc (#PCDATA)>
        <!ELEMENT condicion_desc (#PCDATA)>

    <!ELEMENT base (nombre, resumen, actor+, pre*, post*)>
      <!ELEMENT nombre (#PCDATA)>
      <!ELEMENT resumen (#PCDATA)>
      <!ELEMENT actor (#PCDATA)>
      <!ELEMENT pre (#PCDATA)>
      <!ELEMENT post (#PCDATA)>
      <!ATTLIST base id CDATA #REQUIRED>

    <!ELEMENT alternativa (num_alt, condicion+, pasodesc+)>
      <!ELEMENT num_alt (#PCDATA)>
      <!ELEMENT condicion (#PCDATA)>
      <!ATTLIST alternativa viene_de CDATA #REQUIRED>
      <!ATTLIST alternativa sale_en CDATA #REQUIRED>
```

Figura 4.4: DTD para Descripciones de Casos de Uso

io_desc informa si la acción asociada a un paso es de presentación de datos de salida, ingreso de información, o ninguno de ellos.

objeto_desc objeto sobre el cual se realiza este paso del caso de uso

destinatario_desc si la acción tiene un destinatario (es decir, alguien que reciba el resultado producido por la misma), cuál es ese destinatario

Estos nuevos elementos fueron introducidos basándonos en las teorías de Gramática Generativa introducidas por Chomsky en 1965 (detalladas en [8]) y en sus posteriores extensiones. Bajo esta teoría, toda oración puede dividirse en un conjunto de elementos constitutivos (sujeto, predicado, objeto directo, verbo, etc). Siguiendo estas ideas, la sintaxis XML desarrollada permite consultar toda la información presente en la oración (sujeto que realiza la acción, objeto sobre el cual se realiza la misma, receptor de la acción, etc) de manera estructurada.

4.4. Modelo de CU unificado y Tablas de composición

El paradigma de orientación a aspectos se basa fuertemente en el paradigma orientado a objetos (OOP, del inglés *Object-Oriented Paradigm*) para validar la separación de intereses que OOP maneja pobremente. Así, en [15] se considera de utilidad adaptar los métodos y técnicas existentes en OOP para crear un nuevo acercamiento para impulsar el paradigma de desarrollo orientado a aspectos (AOSD, del inglés *Aspect-Oriented Software Development*). En este contexto, Sousa presenta en [15] diversas adaptaciones de OOP a AOSD desarrolladas con éxito, siendo dos de ellas el *Modelo de CU Unificado* y las *Tablas de Composición*.

Sousa propone una nueva forma de separar comportamiento crosscutting en CU:

1. El comportamiento crosscutting se ubica en un Caso de Uso separado, llamado *CU Crosscutting* (CUC)

2. El CUC se conectará con los casos de uso que afecta mediante un nuevo tipo de relación, llamada *crosscutts* (del inglés, *corte perpendicular*)

3. La información acerca de la composición entre un CUC y los CU que afecta se describirá de forma independiente, en una *Tabla de Composición.*

Para cada relación crosscutting, se especificará así una tabla de composición, la cual no solo proveerá mejor reutilización y comprensibilidad, sino que también simplificará el determinar el rango de CU que un CUC afecta y de qué manera lo hace.

Este acercamiento, utilizado por la Metodología AOWE entre otras, es la que hemos elegido para interconectar los Requisitos Funcionales con los No Funcionales. Ahora bien, dado que en [15] no se incluye un metamodelo acorde a UML sobre el cual desarrollar nuestras transformaciones, esta tarea quedó en nuestras manos.

4.4.1. ATL y XML

Un punto importante a considerar al momento de analizar los metamodelos desarrollados es el siguiente: dada la naturaleza de la información que se espera obtener de estos modelos, expresarlos en forma de un documento estructurado resulta la opción más natural: la utilización de una notación gráfica concreta resulta superflua, puesto que no aportaría ningún tipo de información relevante, y adicionalmente facilita su posterior consulta en posibles etapas futuras del proceso de desarrollo. Ahora bien, como se ha mencionado anteriormente, la etapa de implementación de las ideas aquí presentadas se ha realizado sobre la plataforma Eclipse y sus herramientas de modelado ATL, lo cual nos remite a la dualidad ATL-XML que hemos mencionado en la sección 2.4.1 (página 19). Por este motivo, dado que la única forma de obtener un documento XML es a través de su interpretación primero como modelo y luego como documento, hemos tenido que desarrollar modelos que, interpretados como XML, se adapten al esquema DTD buscado en un primer momento.

Por este motivo en las siguientes secciones, al presentar los metamodelos desarrollados, se expondrá tanto el modelo original UML como el DTD al cual se adapta su representación interna en XML.

4.4.2. Modelo de CU Unificado

El Modelo de CU Unificado se compone simplemente por un listado de CU, un segundo listado de CU Crosscutting (obtenidas a partir de un SIG como operacionalizaciones de un conjunto de RNFs), y relaciones croscutting entre CU y CUC. El metamodelo se detalla en la Figura 4.5

En el contexto de la implementación, en cambio, utilizar este metamodelo de forma directa no resulta lo más adecuado: dado que es preciso considerar en todo momento la dualidad ATL-XML que se ha mencionado, es deseable que todo modelo de este metamodelo, entendido como documento XML, sea sencillo de interpretar y consultar; esto no ocurre con esta implementación del Modelo de CU Unificado, debido al problema de las referencias: los documentos XML generados por Eclipse/ATL, con el (razonable, y hasta esperado) objetivo de no duplicar información, declaran cada elemento del modelo una sola vez, y luego hacen referencia a él mediante índices del tipo @ $<$ *clase_elemento* $>$. $<$ *indice* $>$. Este formato resulta extremadamente complejo de trabajar con consultas XQuery, dando como resultado modelos difícil de crear y de comprender, lo cual llama a buscar alternativas.

En este caso, la solución adoptada es un metamodelo simplificado, que facilita simultáneamente su creación, comprensión y consulta. Este metamodelo se detalla en la Figura 4.6:

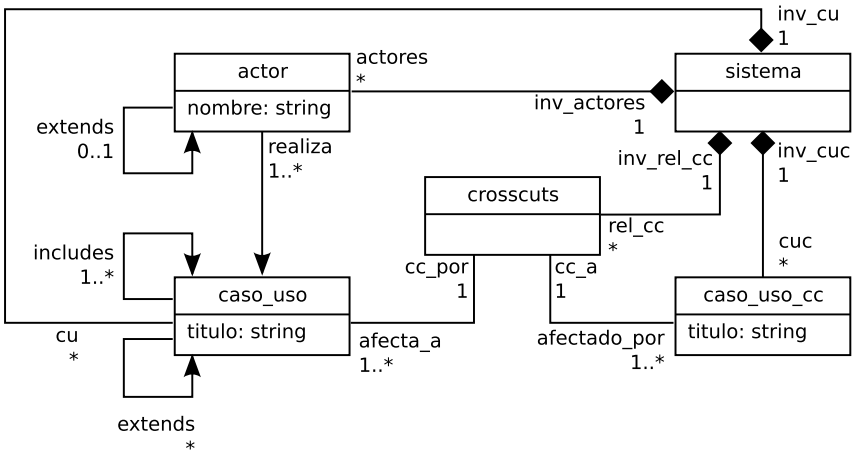

Figura 4.5: Metamodelo de Casos de Uso Unificado

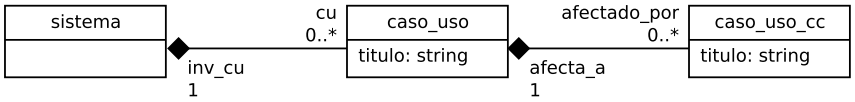

Figura 4.6: Metamodelo de Casos de Uso Unificado - versión simplificada para XML

DTD

El DTD que define la representación en XML del modelo simplificado se detalla en la Figura 4.7:

```
<!ELEMENT dcuc_simple:dcuc (cu*)>
  <!ELEMENT cu (afectado_por+)>
  <!ATTLIST cu titulo CDATA #REQUIRED>
    <!ELEMENT afectado_por EMPTY>
    <!ATTLIST afectado_por titulo CDATA #REQUIRED>
```

Figura 4.7: Metamodelo de Casos de Uso Unificado - DTD de la versión simplificada

Como se puede apreciar, este DTD es extremadamente sencillo e intuitivo: por cada Caso de Uso (representado por el elemento `cu`), se puede saber cuáles son los Casos de Uso Crosscutting que lo afectan consultando sus elementos `afectado_por` hijo. De todos modos, es importante recordar que esta definición es práctica para la etapa de implementación, pero resulta inadecuada para el proceso de Ingeniería de Requisitos.

Clase caso_uso

Descripción Esta clase representa una Unidad de Requisitos Funcionales correspondiente con un CU del MCU

Atributos

- **titulo:** `string`
  Título del CU representado

Asociaciones

- **afectado_por:** `caso_uso_cc[*]`
  Relación que indica cuáles operacionalizaciones operación afectan al CU representado
- **inc_cu:** `sistema [1]`
  Relación con el elemento padre del modelo
- **includes:** `caso_uso [1..*]`
  Relación que representa, a su vez, la relación *include* entre dos CU
- **extends:** `caso_uso [*]`
  Relación que representa, a su vez, la relación *extends* entre dos CU

Restricciones
Sin restricciones adicionales

Semántica Una instancia de la clase `caso_uso` se interpretará como la representación en este modelo de un CU Funcional, presente en un Modelo de Caso de Uso externo.

Notación
Por estar basado en el MCU, la notación para esta clase será la misma que se utiliza en

Figura 4.8: Representación de una instancia de la clase caso_uso

dicho modelo, es decir, un óvalo con el nombre del elemento, como puede apreciarse en la Figura 4.8. Su nombre puede encontrarse tanto debajo del elemento como dentro del mismo.

Clase caso_uso_cc

Descripción Esta clase representa un Caso de Uso Crosscutting, obtenido de una operacionalización operación de un softgoal

Atributos

- **titulo:** `string`
  Título de la operacionalización representada

Asociaciones

- **afecta_a:** `caso_uso [1..*]`
  Relación que indica cuáles Casos de Uso funcionales se verán afectados por el Caso de Uso Crosscutting representado
- **inc_cuc:** `sistema [1]`
  Relación con el elemento padre del modelo

Restricciones
Sin restricciones adicionales

Semántica Una instancia de la clase funcionalidad se corresponde con una operacionalización presente en la descomposición de un softgoal a través de los mecanismos previstos por el Framework NFR.

Notación
Similar a la notación de una instancia de la clase `caso_uso`, esta clase se representará mediante un óvalo relleno con un patrón lineal, como se detalla en la Figura 4.9.

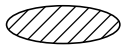

Figura 4.9: Representación de una instancia de la clase caso_uso_cc

Clase actor

Descripción Esta clase representa a un Actor, correspondiente con un actor de un Diagrama de Casos de Uso

Atributos

- **titulo:** `string`
  Nombre del actor representado

Asociaciones

- **extends:** `actor [0..1]`
  Relación que representa, a su vez, la relación extends entre dos actores representado
- **realiza:** `caso_uso [1]`
  Casos de Uso que son realizados por el actor
- **inv_actores:** `sistema [1]`
  Relación con el elemento padre del modelo

Restricciones
Sin restricciones adicionales

Semántica Una instancia de la clase actor se corresponde con un actor presente en un Diagrama de Casos de Uso.

Notación
Para esta clase se mantiene la notación definida por UML para los DCU: un dibujo simplificado de una persona, con el nombre del actor escrito en un costado, como se aprecia en la figura 4.10

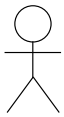

Figura 4.10: Representación de una instancia de la clase actor

Clase sistema

Descripción Clase contenedora de todos los demás elementos del modelo

Atributos
No posee atributos

Asociaciones

- **actores:** `actor [*]`
  Actores del modelo
- **cu:** `caso_uso [*]`
  Casos de Uso del modelo

- **cuc:** `caso_uso_cc [*]`
  Casos de Uso Crosscutting del modelo

Restricciones
Sólo puede haber una instancia de esta clase por modelo

Semántica Una instancia de esta clase se utilizará como punto central del cual derivar los demás elementos del modelo. Así, esta clase no cumple otro rol que el de contener a las demás instancias de cada clase. Si bien en modelos puramente teóricos esta clase es redundante, tiene utilidad en el marco de la herramienta ATL, donde es requerida.

Notación No posee notación, puesto que no se representa en el modelo

4.4.3. Tablas de Composición

El modelo de la Figura 4.11presenta un conjunto de Tablas de Composición, agrupadas por CU, y donde cada fila representa un requisito crosscutting.

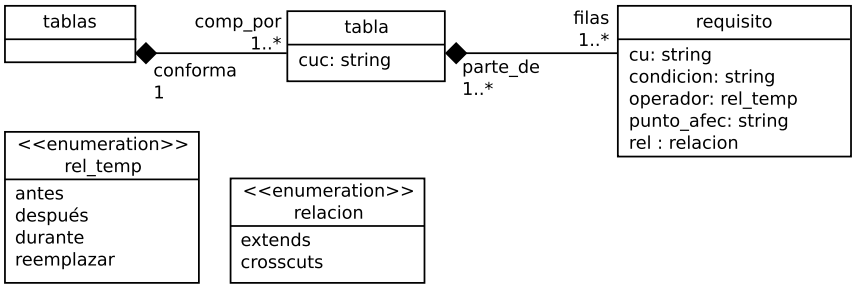

Figura 4.11: Metamodelo de Tablas de Composición

DTD

El DTD asociado a este metamodelo se presenta en la Figura 4.12

```
<!ELEMENT tabla_comp:tablas (comp_por+)>
  <!ELEMENT comp_por (filas+)>
  <!ATTLIST comp_por cu CDATA #REQUIRED>
    <!ELEMENT filas (punto_afec+)>
    <!ATTLIST filas cu CDATA #REQUIRED>
    <!ATTLIST filas condicion CDATA #REQUIRED>
    <!ATTLIST filas operador
              (antes|despues|durante|reemplazar) #REQUIRED>
    <!ATTLIST filas rel (extends|crosscuts) "crosscuts" #REQUIRED>
      <!ELEMENT punto_afec (#CDATA)>
```

Figura 4.12: Metamodelo de Tablas de Composición - DTD

Clase tablas

Descripción Clase base, de la cual se desprenden todos los demás elementos del modelo

Atributos
No posee atributos

Asociaciones

- **comp_por:** `tabla [1..*]`
  Relación que indica cuáles serán los Casos de Uso Crosscutting

Restricciones
Sólo puede haber una instancia de esta clase por modelo

Semántica Una instancia de esta clase se utilizará como punto central del cual derivar los demás elementos del modelo. Así, esta clase no cumple otro rol que el de contener a las demás instancias de cada clase. Si bien en modelos puramente teóricos esta clase es redundante, tiene utilidad en el marco de la herramienta ATL, donde es requerida.

Notación No tiene representación, dado que no representa a un elemento concreto del modelo

Clase tabla

Descripción Esta clase representa una instancia de una tabla de composición.

Atributos

- **titulo:** `string`
  Título del Caso de Uso Crosscutting representado en esta tabla

Asociaciones

- **conforma:** `tablas [1]`
  Elemento padre que contiene todas las instancias de esta clase
- **filas:** `requisito [1..*]`
  Relación que indica cuáles casos de uso se verán afectado por el CUC, y de qué modo

Restricciones
Sin restricciones adicionales

Semántica Cada instancia de esta clase representa una Tabla de Composición, habiendo una tabla por cada CUC obtenido del MICNF. Sus relaciones indicarán a cuáles requisitos funcionales afecta, y de qué modo.

Notación Dado que este metamodelo representa a una tabla, no posee una representación gráfica concreta. En el contexto de la tabla, sin embargo, se entenderá que cada instancia de este elemento es una tabla individual, cuyo título es el atributo `título` de la clase. Cada fila de la tabla, por otra parte, será una instancia de la clase `requisito` asociada mediante la relación `comp_por`.

Clase requisito

Descripción Esta clase se utilizará para representar una determinada funcionalidad y su relación con un Caso de Uso Crosscutting.

Atributos

- **cu:** `string`
  Título del CU Funcional al que representa esta instancia
- **condicion:** `string`
  Condición (si es que hubiera alguna distinta de `true`) bajo la cual se aplicará el CUC al CU funcional
- **operador:** `rel_temp`
  Operador que indica de qué manera se insertará el CUC dentro del CU con el cual es crosscutting

- **punto_afec:** `string [1..*]`
  Números de pasos de la descripción del CU en el cual se insertará el CUC
- **rel:** `relacion`
  Tipo de relación entre el CUC y el CU.

Asociaciones

- **parte_de:** `tabla [1..*]`
  Relación que indica cuál CUC será el que afecte al CU actual

Restricciones
Sin restricciones adicionales

Semántica Una instancia de esta clase posee toda la información relacionada a cómo se vincula una operacionalización obtenida del Framework NFR con un determinado CU obtenido del Modelo de Casos de Uso.

Notación Dado que este metamodelo representa a una tabla, no posee una representación gráfica concreta. En el contexto de la tabla, sin embargo, se entenderá que cada instancia de este elemento es una fila de una tabla de composición.

Capítulo 5

Proceso de Ingeniería de Requisitos

5.1. Descripción del proceso en general

A continuación daremos un breve resumen del proceso de Ingeniería de Requisitos adoptado en el presente trabajo; éste será examinado en mayor detalle en capítulos posteriores. El Proceso en su totalidad puede observarse en la Figura 5.1.

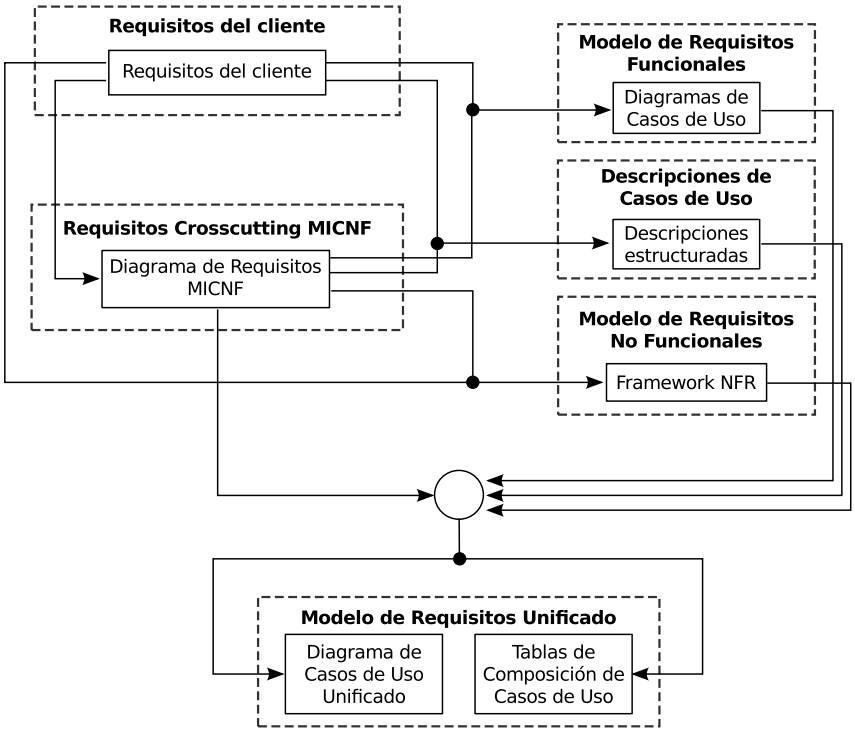

Figura 5.1: Proceso MDA completo - Versión completa

En una primera instancia, y partiendo de fuentes diversas tales como clientes, usuarios finales, o incluso documentación sobre proyectos similares, se obtienen *Requisitos de Impacto de Cualidades No Funcionales* (RICNF), término con el cual haremos referencia a requisitos sobre cómo debe impactar en el sistema una determinada Cualidad No Funcional. Estos RICNF estarán expresados

en lenguaje natural, por lo cual deberán ser traducidos primero a un modelo ICNF. Partiendo de este modelo, el analista podrá obtener Intereses No Funcionales (a los cuales nos referiremos como *Softgoals* en el entorno del Framework NFR), y se apoyará en ellas para definir un modelo NFR.

Adicionalmente, con el modelo ICNF el analista podrá obtener algunos CU y Actores que (a partir de parcelas de sistema de funcionalidad, roles de usuario y servicios externos) que le ayudarán, en conjunto con el *input* del cliente, a definir y/o mejorar un Modelo de Casos de Uso; paralelamente, el modelo ICNF le permitirá al analista obtener información mediante la cual elaborar descripciones de CU (funcionalidades, informaciones en pantalla, informaciones almacenadas y eventos, que serán usados en las descripciones de escenarios de CU). Recordemos que estas descripciones no contienen referencias a RNF, pues dicha información está en el modelo ICNF.

Habiendo hecho todo eso, se aplicará una transformación involucrando a todos estos modelos (a saber: modelos de requisitos ICNF, NFR, de CU y de descripciones de CU) que generará simultáneamente el modelo de CU unificado y las tablas de composición.

No podemos dejar de observar que los modelos ICNF, de CU y de descripciones de CU pueden ser evaluados por el cliente y usuarios finales; esto es posible por la amplia difusión de los modelos de CU, de las descripciones textuales de CU, de XML y por ser el modelo ICNF una notación legible por el cliente.

5.2. Traducción de RICNF en lenguaje natural a notación MICNF

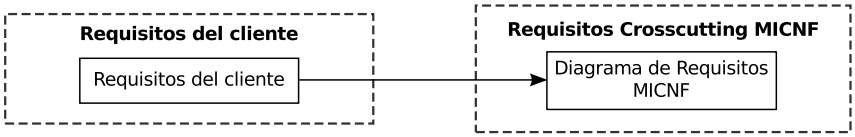

Figura 5.2: Traducción de RICNF en lenguaje natural a notación MICNF en el Proceso MDA

Analizar un texto en lenguaje natural, identificar requisitos y plasmarlos en una notación estructurada no es una tarea sencilla. Dado que el nacimiento de la notación MICNF responde precisamente a una solución a este problema, es conveniente hacer aquí un repaso histórico a su diseño, repaso que a su vez servirá también como fundamentación de los "cómo" y "porqués" de la notación MICNF. La primera gran tarea de nuestro proyecto fue determinar una cierta estructura subyacente a los requisitos ICNF expresados en lenguaje natural, tanto para identificarlos como para representarlos en un modelo estructurado. Esta tarea no resultó sencilla: se recopiló una gran cantidad de requisitos expresados con distintos grados de detalle (desde requisitos específicos, pensados por analistas, hasta requisitos vagos y/o poco concretos, más cercano a lo esperado de un cliente que no está familiarizado con la informática actual). Tras tabular y analizar todos estos requisitos, el resultado obtenido fue en primer lugar la clasificación según criterios dada en la Sección 3.3 (página 26), cuyos elementos de clasificación fueron eventualmente modelados de forma individual o grupal según el caso, desembocando en lo que conocemos como modelo MICNF. Este modelo, entonces, resulta adecuado por diversos motivos:

- Sus elementos proveen de un análisis de la estructura típica de los RICNF, garantizando que todas las clases allí definidas estén fuertemente basadas en situaciones reales

- La situación inversa también se aplica: todos los RICNF encontrados pueden descomponerse en las partes identificadas en un MICNF, por lo que todo RICNF podría expresarse en este modelo

- Dado que la notación gráfica contiene la misma información que el requisito original, resulta más sencillo de comprender para el cliente, facilitando el intercambio de ideas con quien, en definitiva, es el principal interesado en el desarrollo de nuestro sistema.

El proceso de traducción de RICNF en lenguaje natural a la notación MICNF puede resumirse, entonces, en los siguientes pasos:

1. Identificar oraciones que hagan referencia a RICNF
2. Clasificarlos de acuerdo al criterio ilustrado en el gráfico 3.2 (página 32)
3. Si el requisito es rechazado, o requiere refinamiento (y la experiencia del analista no es suficiente como para resolverlo), transmitir la inquietud al cliente.
4. Si el requisito es aceptado, identificar sus miembros constituyentes, a saber:
   - Parcelas de sistema que afecta
   - NFI deseado, y los parámetros de impacto
   - Condiciones de aplicabilidad
5. Crear una instancia de este requisito mediante la notación MICNF

Un ejemplo de este proceso puede apreciarse en el Capítulo 6.2.5, de la página 65.

5.3. Obtención de SIGs de framework NFR

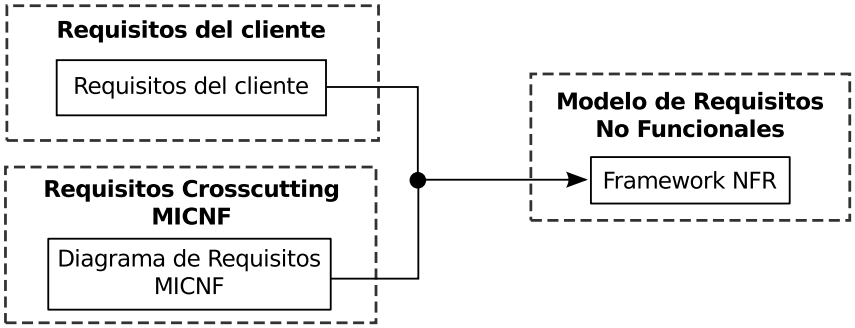

Figura 5.3: Obtención de SIGs de framework NFR en el Proceso MDA

Una vez relevados los requisitos del cliente y obtenido un MICNF, debemos utilizar esta información para la construcción de diversos modelos. Estos modelos utilizan ya conceptos técnicos y estrategias de construcción, ya que esta etapa del proceso es llevada adelante por el analista. El primer paso consiste en, partiendo de los NFI y Parcelas de Sistema de los RICNF, construir los modelos necesarios correspondientes al Framework NFR para obtener operacionalizaciones de Requisitos No Funcionales.

En este caso, nuestro punto de partida serán los NFI. Dado que estos intereses son no funcionales, lo que haremos es ubicarlos en el contexto de un SIG (Softgoal Interdependency Graph) como softgoals y luego, basándonos en las metodologías del Framework NFR, construir un SIG tal que nos permita obtener las operacionalizaciones necesarias para satisfacer los softgoals creados. Este SIG corresponderá con el metamodelo definido en la Figura 4.1 (página 43).

A partir del modelo MICNF el analista identifica softgoals y se apoya en ellos para definir el modelo NFR; para esto se usa la idea de que a partir de un RICNF r que tiene PS $p_1, p_2, \ldots, p_n$ y un NFI n, se puede generar el softgoal $n[p_1 \cap p_2 \cap \ldots \cap p_n]$ donde el tópico es la intersección

de los conjuntos de URFs de las PS p_i. Un softgoal $n[t]$ obtenido de RICNF puede refinarse a nuevos softgoals ya sea en tipo (considerando aspectos más específicos o detallados de n) o en tópico (considerando subconjuntos de t). Un ejemplo de esta descomposición en tópicos se puede apreciar en la Figura 6.7 (página 66), detallado en [13], donde se analiza un SIG de Usabilidad para un sistema de Comercio Electrónico.

Es posible también que al refinar los softgoals provenientes de NFIs aparezcan, en los softgoals descendientes, tópicos más refinados y distintos a los originales. Estos tópicos se caracterizarán por ser combinaciones de Parcelas de Sistema. Un ejemplo de este tipo de descomposición es el que se presenta en la Figura 5.4.

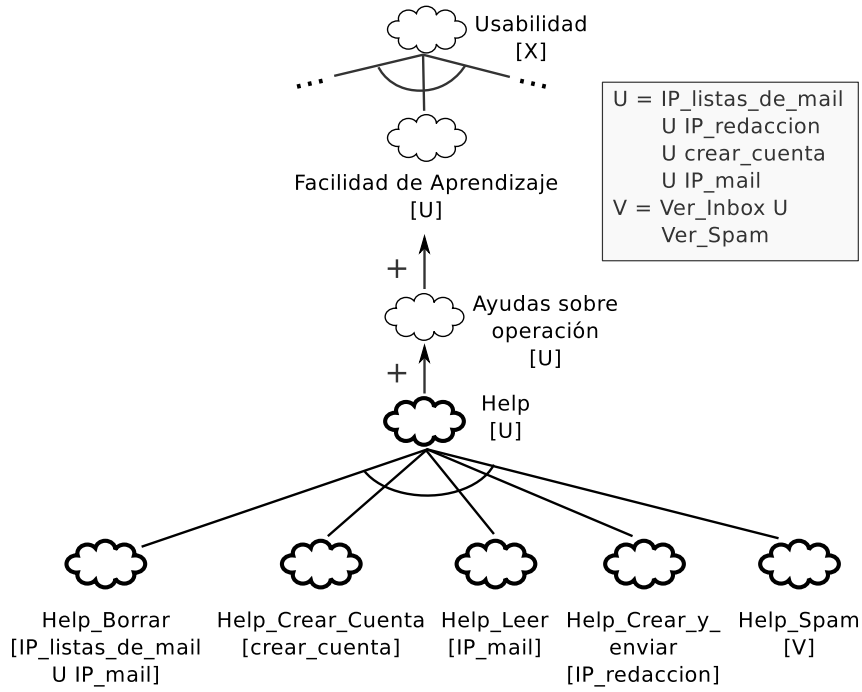

Figura 5.4: SIG Usabilidad: resultado de la descomposición del Softgoal Help

5.4. Obtención de Modelo de Casos de Uso

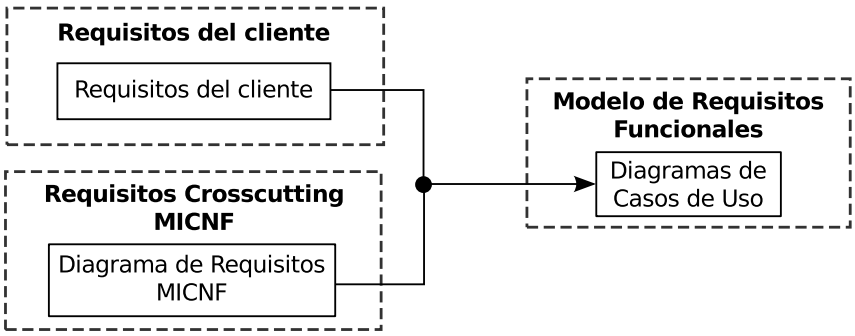

Figura 5.5: Obtención de Modelo de Casos de Uso en el Proceso MDA

Siguiendo una filosofía similar al caso anterior, en esta etapa del proceso MDA utilizaremos la información dada por el cliente en la construcción del Modelo de Impacto de Cualidades No Funcionales para derivar información sobre cómo construir un Diagrama de Casos de Uso, el cual convierta las Unidades de Requisitos Funcionales declaradas anteriormente en Casos de Uso y/o Actores en un nuevo diagrama.

Para esta etapa, deberemos prestar particular atención a las Parcelas de Sistema declaradas en el MICNF: si bien algunas parcelas tales como `funcionalidad` o `rol_usuario` tienen una traducción casi directa, otras (tales como recursos) no tendrán un único equivalente en el DCU, sino que servirán mas bien como guías de cómo crear el nuevo Diagrama. Un ejemplo de este proceso puede verse en la Figura 5.6. Como puede apreciarse, aquí las Parcelas de Sistema simbolizan Casos de Uso Funcionales, los Parámetros de Impacto sugieren pasos, etc. Siguiendo esta línea, podemos obtener un esqueleto general de los requisitos funcionales del sistema.

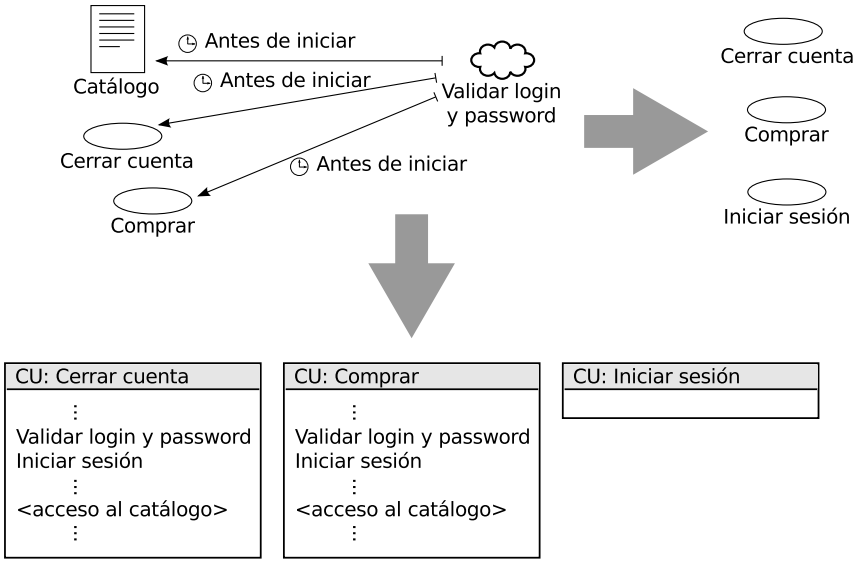

Figura 5.6: Ejemplo de obtención de Modelo de Casos de Uso y Descripciones de C.U. a partir de MICNF

Al igual que en el caso anterior, este Diagrama se ajustará a la definición MOF dada en la Figura 4.2 (página 44).

5.5. Obtención de Descripciones de Casos de Uso

Dado que en el paso anterior derivamos de forma exitosa un Diagrama de Casos de Uso, debemos ahora proceder al paso siguiente: dar una descripción estructurada de un Caso de Uso, sus escenarios, sus actores, sus ejecuciones alternativas, y demás información. Aquí los requisitos del cliente en su forma original serán de gran utilidad, dado que una gran parte de este trabajo surgirá de allí; a pesar de esto, el Diagrama de Requisitos Crosscutting MICNF será útil para la organización general del resultado, del mismo modo que lo fuera en el paso anterior para la creación del Diagrama de Casos de Uso.

A diferencia de pasos anteriores, las Descripciones Estructuradas de Casos de Uso no se expresarán en un modelo MOF, sino que se ha optado en su reemplazo por un documento XML, definido por el DTD dado en la Figura 4.4 (página 45). Esta decisión obedece a la naturaleza misma de las descripciones a crear: estas descripciones, si bien pueden modelarse sin inconveniente bajo un metamodelo MOF, son en esencia una variante de texto estructurado, tarea para la cual XML resulta mucho más adecuado.

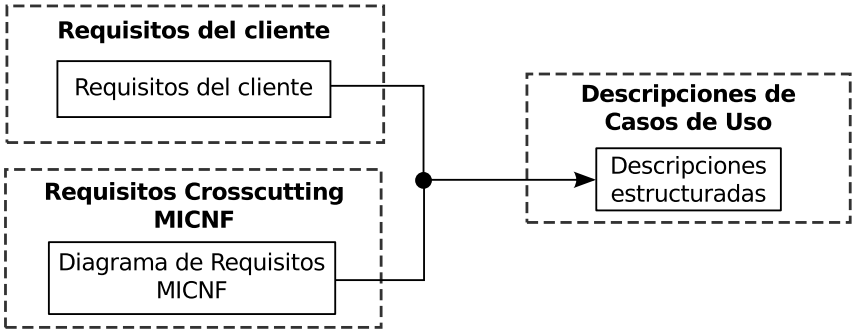

Figura 5.7: Obtención de Descripciones de Casos de Uso en el Proceso MDA

5.6. Transformación a Modelos Orientados a Aspectos

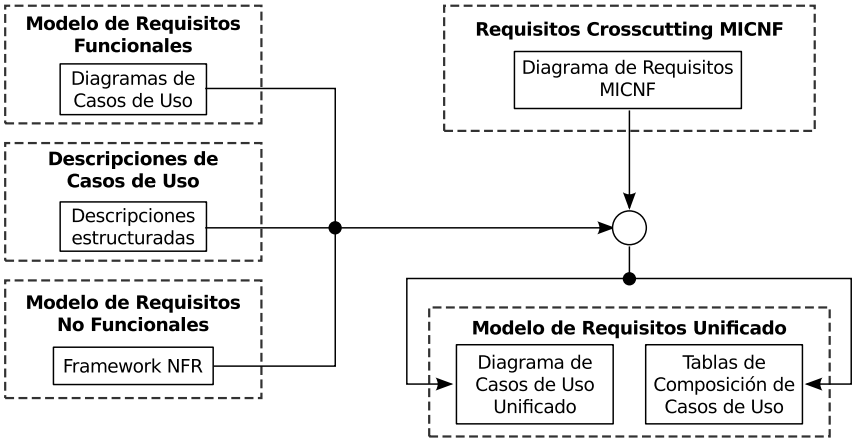

Figura 5.8: Transformación a Modelos Orientados a Aspectos en el Proceso MDA

El último paso del proceso MDA desarrollado en el presente proyecto es la obtención mediante transformaciones de los Diagramas de Casos de Uso unificado y el de Tablas de Composición de Casos de Uso. A diferencia de etapas anteriores, esta es la primera etapa del proceso donde el analista no es ya responsable de la generación de los nuevos modelos. Esta transformación, como puede apreciarse en el diagrama de la Figura 5.8, hará fuerte uso de todos los modelos creados en las etapas anteriores, lo cual no debe sorprendernos: dado que cada modelo buscaba modelar y/o formalizar una cierta visión de los requisitos dados por el usuario, resulta lógico que esta misma información estructurada se utilice ahora como entrada de una transformación.

Esta etapa será examinada en detalle en el capítulo siguiente.

Capítulo 6

Transformaciones

6.1. Mapa de Transformaciones

La Figura 6.1 ilustra el esquema de transformaciones del proceso de Ingeniería de Requisitos adoptado en el presente trabajo:

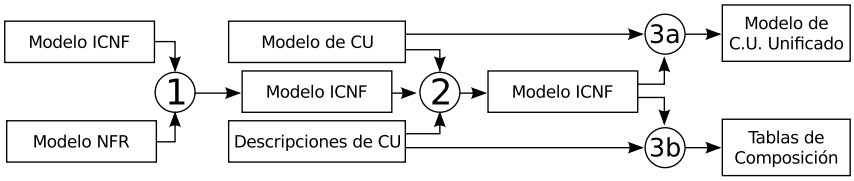

Figura 6.1: Transformaciones de Modelos en la Ingeniería de Requisitos

En líneas generales, las reglas se describen de la siguiente manera:

Regla 1 Obtiene información acerca de cuales softgoals deben seleccionarse como CUC

Regla 2 Obtiene información acerca de cuales CU se verán afectados por los CUC

Reglas 3a y 3b Mapean la información obtenida en etapas anteriores a los modelos OA definidos por Sousa

6.1.1. Ejemplo

A lo largo de este capítulo utilizaremos, a modo de ejemplo ilustrativo, un hipotético sistema de Comercio Electrónico. Si bien no se desarrollará aquí el caso en su totalidad, por ser demasiado extenso, se han seleccionado los fragmentos más representativos del proceso. Un último detalle a tener en cuenta es la cuestión de la etapa de implementación. Para realizar el presente proyecto, y como una forma de comprobar que la teoría desarrollada es aplicable de manera concreta, se ha desarrollado una implementación en ATL de los conceptos aquí explicados. Sólo se transcriben aquí los fragmentos de código indispensables para entender el punto que se esté exponiendo.

6.2. Transformación 1

En la primera transformación, ilustrada en la Figura 6.2, se reemplaza cada nfi cualidad no funcional del modelo MICNF de entrada por todas las operaciones del MSIG que son necesarias para satisfacer el softgoal de ese NFI. Se asume que antes se ha aplicado exitosamente el algoritmo de evaluación dado por Chung (i.e. se satisfacen todos los softgoals nfr) en [9].

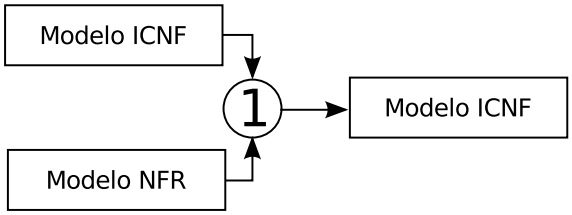

Figura 6.2: Diagrama de la Transformación 1

6.2.1. Especificación

Esta regla deberá cumplir con el siguiente requisitos: partiendo de un MICNF, cuyos nfis representan metas suaves (con mayor o menor especificidad, dependiendo de los requisitos originales del usuario), se reemplazará todos los nfis por otros que correspondan con las operaciones concretas necesarias para cumplir dichas metas. Dado que estas operacionalizaciones introducen tópicos, todos ellos deben estar representados de alguna manera mediante PS en el modelo MICNF.

Esta definición de la especificación requiere, sin embargo, un análisis más detallado sobre la naturaleza de los tópicos en un NFR. En particular, los tópicos son dominios de aplicación de la operacionalización, que limitan el rango de funcionalidades sobre las cuales puede aplicarse un softgoal. Debido a que esta definición de tópicos es muy similar a la definición de Parcela de Sistema, resulta lógico establecer un paralelo entre ambas definiciones. Esto, a su vez, introduce un nuevo problema no contemplado con anterioridad: dado que un tópico puede refinarse, lo mismo debería ocurrir con las Parcelas de Sistema.

La solución a este problema viene dada mediante la incorporación en esta especificación del refinamiento de Parcelas de Sistema: dado que una PS se entiende como un conjunto de funcionalidades, tanto el MICNF como el NFR deberán ajustarse a las siguientes restricciones:

- Todo tópico del NFR deberá corresponderse con una combinación (por medio de operaciones de conjuntos)de PS del metamodelo MICNF. Si bien esto restringe la capacidad expresiva de los tópicos, en la práctica no será un problema, ya que toda operacionalización operación debe, en última instancia, aplicarse sobre una o varias funcionalidades del MICNF, las cuales son representadas por las PS funcionalidad.

- Las PS del MICNF deberán poder refinarse: entendiendo una PS como un conjunto de (usualmente otras[1]) PS funcionalidad, deberá ser posible desplegar una PS en sus parcelas funcionalidad constituyentes

De estas dos restricciones, la primera se tratará en esta etapa. La segunda, en cambio, se analizará en la regla de transformación siguiente.

6.2.2. Decisiones de diseño

La primera decisión importante a considerar es el método para obtener, partiendo de un Interés No Funcional (el cual resulta por definición vago y/o inespecífico), el conjunto necesario de operaciones necesarias para lograr dicha meta. En este aspecto, dado que esta es la razón misma de ser del Framework NFR, utilizar los SIGs de este modelo resulta la decisión más natural.

La segunda decisión a considerar, derivada de la anterior, consiste en cómo lidiar con los tópicos, elemento presente en las descomposiciones típicas de un SIG. Luego de un análisis detallado tanto de los mecanismos disponibles en ATL como de las distintas descomposiciones posibles del proceso de transformación en general, se optó por lidiar con esta cuestión en esta misma transformación, en vez de incorporarla en una futura etapa - dado que el proceso de consultar el SIG es complejo,

[1] La PS funcionalidad siendo la excepción: una PS funcionalidad está compuesta por exactamente una PS funcionalidad, que es ella misma

resulta conveniente hacerlo una sola vez. Ahora bien, ¿cómo reflejar en el MICNF estos tópicos de manera semánticamente correcta?

Una observación particular nos brinda la respuesta a este interrogante: dado que los NFI establecen requisitos no funcionales sobre las parcelas de sistema, y dado que un tópico (en el marco de nuestro trabajo) no es otra cosa que una o más funcionalidades particulares de una o más parcelas del sistema (asociadas mediante los operadores de conjuntos típicos: unión, intersección y complemento), entonces resulta natural que cada tópico sea representado, en realidad, como un conjunto de parcelas de sistema, mediante la clase asociada `parcela\_sistema` (ya que una parcela de sistema es, precisamente, un aspecto particular del sistema sobre el cual se impone un requisito no funcional).

Finalmente, se ha optado por delimitar es la sintaxis de los tópicos: dado que los tópicos se tratan en conjunto, se decidió ajustar la notación de los tópicos a la Forma Normal Conjuntiva[2].

6.2.3. Implementación

Una implementación de esta transformación debe cumplir no solo los requisitos establecidos en la etapa de especificación, sino también con los criterios de evaluación detallados en la etapa de Validación y Verificación, dados más adelante. Habiendo definido estos aspectos (que podemos pensar como la Precondición y Postcondición de la transformación), nos hemos abocado a la tarea de lograr una implementación en ATL que nos permita realizarla.

En principio, para todos los elementos del MICNF que no deseamos modificar, tenemos reglas identidad que generan en el modelo de salida una instancia idéntica a la dada de entrada. En el caso de la clase `parametro_impacto`, por ejemplo, la regla que se aplica es la que se describe en la Figura 6.3

```
rule regla_pi {
    from
        s: MICNF!parametro_impacto
    to
        t: MICNF!parametro_impacto (
            tipo_impacto <- s.tipo_impacto,
            eventos <- s.eventos,
            valida_en <- s.valida_en
        )
}
```

Figura 6.3: Regla de Transformación de una instancia de la clase parametro_impacto

Como se puede apreciar, para cada propiedad del modelo de salida, el valor es copiado directamente de la misma propiedad en el modelo de entrada.

Lidiar con los elementos que sí debemos modificar, en cambio, es una tarea más compleja. Debido a las reglas de cardinalidad de las reglas ATL, donde una regla puede generar un único elemento de salida, no será posible plantear las reglas directamente sobre nfis y/o instancias `parcela_sistema`, sino que deberá hacerse de forma indirecta, tomando como entrada las instancias de la clase `req_icnf`. El argumento es el siguiente: tomemos como ejemplo la sub-regla "tomando un nfi que representa un softgoal, reemplazarlo por nfis que representen sus operacionalizaciones". Si bien el procedimiento más razonable parecería ser "diseñar una regla que, tomando como entrada un nfi (es decir, el softgoal), devuelva un conjunto de nfis (es decir, las operacionalizaciones)", la imposibilidad de devolver en ATL un conjunto como elemento de salida de una regla de alguna forma útil nos ha obligado a tomar un enfoque distinto, a saber: "tomando

[2]En lógica booleana, una fórmula está en forma normal conjuntiva (FNC) si corresponde a una conjunción de cláusulas, donde una cláusula es una disyunción de literales, donde un literal y su complemento no pueden aparecer en la misma cláusula. Toda fórmula proposicional puede convertirse en una fórmula equivalente que está en FNC.

como entrada un elemento `req_icnf`, reemplazar cada elemento de la relación exige (en ATL, una relación se interpreta como un conjunto de elementos) por un conjunto de nuevos elementos (sus operacionalizaciones), interpretados como elementos nfi del modelo de salida".

Veamos este ejemplo en código. En principio, la regla que toma como entrada un elemento `req_icnf` se detalla en la Figura 6.4.

En este ejemplo,

- La primera regla, `rule regla_icnf`, toma una instancia de la clase *req_icnf* y devuelve una nueva instancia de la clase, con las mismas propiedades y con las mismas relaciones, excepto para las relaciones *exige* y *abarca_a*, las cuales se modifican para incluir las nuevas parcelas de sistema y/o operacionalizaciones que correspondan al resultado esperado de la transformación.

- La regla `nuevos_ps` toma un *String*, representando al título de la parcela de sistema que se desea obtener, y devuelve una instancia de la clase *parcela_sistema* cuyo nombre es el mismo que el del parámetro de entrada, y donde sus relaciones se completan parcialmente con información del modelo de entrada, tal como la instancia de la clase *req_icnf* con la que debe asociarse - la información faltante será completada por la regla *regla_icnf* (notar que esta regla es *unique lazy*, por lo que sólo puede llamarse explícitamente por otra regla).

- La regla `nuevos_sg` es muy similar al punto anterior, pero convirtiendo operacionalizaciones de un SIG en elementos NFI de un MICNF.

La relación `exige`, continuando con el ejemplo, selecciona un conjunto de operacionalizaciones posibles (aquellas que están marcadas como satisfechas y no poseen sub-descomposición), y a cada una de ellas se las convertirá en un `nfi`, reemplazando así de forma efectiva al `nfi` presente originalmente en el modelo. Existe un requisito adicional necesario para esta etapa que no se detalla aquí (a saber, tomar solamente aquellas operacionalizaciones que corresponden a la descomposición del softgoal original) - básicamente, podemos decir que la operación *collect* será la que filtre cuáles de estos elementos son los que deben incluirse y/o descartarse, haciendo uso extensivo de helpers definidos con este propósito.

Un argumento similar se utiliza para determinar las nuevas parcelas de sistema a incluir en el MICNF de salida.

Definir la notación de los tópicos, por otra parte, fue una tarea directa y sin mayores complicaciones: dado que se tomó como base la Forma Normal Conjuntiva de notación de conjuntos, se decidió que los tópicos de una operacionalización deben expresarse de acuerdo con la notación BNF de la Figura 6.5.

6.2.4. Validación y Verificación

Hay tres aspectos principales a considerar para asegurarnos de que la implementación definida cumpla con nuestra especificación. En principio, podemos definir dicha especificación de la siguiente manera:

Sean

- r un RICNF que involucra un NFI n, y n es mapeado en el modelo MNFR a un softgoal $n[t]$

- $n_1[t_1], \ldots, n_m[t_m]$ en el modelo MNFR un conjunto de operacionalizaciones operación que permiten satisfacer $n[t]$

Entonces se reemplaza r de MICNF por nuevos RICNF como sigue:

- para cada t_i $(1 <= i <= m)$ se generan nuevos RICNF de la siguiente manera: asumimos que t_i es de la forma $p_1 \cup \ldots \cup p_s$ donde los p_j son de la forma $(q_j, 1) \cap \ldots \cap (q_j, n_j)$ tal que los (q_j, k) $(1 <= k <= n_j)$ son PS con los tipos de recurso encontrados en MICNF,

```
rule regla_icnf {
    from
        s: MICNF!req_icnf
    to
        t: MICNF!req_icnf (
            condicion <- s.condicion,
            id <- s.id,
            limitado_por <- s.limitado_por,
            exige <- MSIG!operacionalizacion.allInstances()
                    ->select(e | e.marca = #satisficing and
                            e.descomp_en.oclIsUndefined() and
                            e.contrib_por.size() = 0)
                    ->collect(e | thisModule.nuevos_sg(e)),
            abarca_a <- MSIG!operacionalizacion.allInstances()
                    ->select(e | e.marca = #satisficing and
                            e.descomp_en.oclIsUndefined() and
                            e.contrib_por.size() = 0)
                    ->collect(f | f.topicos->collect(h |
                            thisModule.nuevos_ps(h))),
            inv_reqs <- s.inv_reqs
    )
}

unique lazy rule nuevos_ps {
    from
        s:String
    to
        t:MICNF!parcela_sistema (
            titulo <- s,
            tipo <- #funcionalidad, -- No es relevante aun
            inv_pss <- MICNF!micnf.allInstances().first(),
            abarcado_por <- MICNF!req_icnf.allInstances().first()
    )
}

unique lazy rule nuevos_sg {
    from
        s: MSIG!operacionalizacion
            (s.marca = #satisficing and
            s.descomp_en.oclIsUndefined() and
            s.contrib_por.size() = 0)
to
            t: MICNF!nfi (
                titulo<-s.nombre,
                operac<-true,
                restringido_a<-thisModule.oraculo(s),
                inv_nfis<-MICNF!micnf.allInstances().first()
    )
}
```

Figura 6.4: Regla de Transformación de una instancia de la clase req_icnf

```
rule regla_icnf {
          <topico> ::= <funcionalidad>
                     | <union>
                     | <interseccion>
           <union> ::= <funcionalidad>
                     | <funcionalidad> U <union>
    <interseccion> ::= (<union>) INT <interseccion_suc>
<interseccion_suc> ::= (<union>)
                     | (<union>) INT <interseccion_suc>
```

Figura 6.5: Notación BNF para Tópicos

- entonces se agregan al modelo MICNF los requisitos r_j (con $1 <= j <= s$) compuestos por el NFI n_i, los parámetros de impacto de r y las PS $(q_j, 1), \ldots, (q_j, n_j)$.

Esta especificación se verifica en el cumplimiento de 3 reglas principales, a saber:

1. Los softgoals deben haber sido reemplazados por las operacionalizaciones necesarias para su correcta implementación
2. Todo tópico de un softgoal debe tener su equivalente en la Forma Normal Conjuntiva de Parcelas de Sistema
3. Los demás elementos del modelo MICNF deben permanecer ininmutables

El tercer paso es el más sencillo de verificar - simplemente, al definir reglas identidad sobre todas las clases, excepto la clase `nfi`, aseguramos que dichos elementos permanecen estables. De este modo, sólo debemos prestar atención a dos elementos del modelo: la clase `nfi` (junto con sus relaciones asociadas, `exige`, y su inversa, `restringido_a`), y la clase `parcela_sistema`.

Ahora bien, con respecto a estas clases, no sólo debemos asegurar que mantengan la integridad del modelo, sino que cumplan con los siguientes requisitos:

1. Ningún NFI original debe permanecer en el modelo de salida. En su lugar, cada nuevo nfi debe corresponder a la operacionalización del softgoal representado por el nfi original, a menos que dicho nfi represente una operacionalización operación.
2. La relación `exige`/`restringido_a` debe tener coherencia, es decir, eliminar toda relación con un `nfi` que desaparece, y vincular todo `nfi` nuevo con una clase `req_icnf`. Es importante notar que esto sólo es válido en el caso de que los tópicos de las operacionalizaciones operación que ayudan a alcanzar al NFI n sean iguales a los tópicos del softgoal asociado a n, pero si bien en caso contrario se requiere agregar nuevas PS, el paso siguiente será el encargado de asegurar esta coherencia.
3. Dado un conjunto c de tópicos obtenidos a partir de las operacionalizaciones de un requisito no funciona sobre una cierta Parcela de Sistema ps, dicha Parcela podrá reemplazarse (en el MICNF al que pertenecen tanto la PS ps como la operacionalización de la cual se deriva c) por c, pero solamente cuando ello no altere la semántica del requisito MICNF al que pertenecía ps.

La correcta definición del espacio de entrada y de salida de las reglas de transformación de nuestra implementación aseguran lo primero - creando reglas que tomen todo `nfi` como entrada y generen como salida nuevos nfis cuyas propiedades se obtienen exclusivamente a partir del SIG de entrada (asegurando que una de estas propiedades sea "el `nfi` de salida debe ser un descendiente del `nfi` de entrada, pero no su igual") podemos asegurar que el primer requisito se cumpla.

Similarmente, mantener la coherencia exigida en el punto 2 será sencillo: lo que debemos hacer es establecer que todo nuevo `nfi` mantenga las relaciones `restringido_a` del `nfi` de entrada (y, por

ende, la de su inversa): si consideramos que el `nfi` de entrada n hace referencia a un conjunto finito c de PS del `ricnf` en el que se encuentra, entonces cada tópico de cada operacionalización operación de n que haga referencia a una o más PS de dicho `ricnf` se verá limitada a los elementos de c (por ser la obtención de operacionalizaciones y tópicos un refinamiento). De este modo, incluso cuando el procedimiento genere nuevas PS o tópicos, dichos elementos serán siempre menos abarcativos que aquellos a los que reemplazan, por lo que evitaremos establecer vínculos previamente inexistentes, y garantizando así el cumplimiento del requisito 2.

El punto tres es el más complicado de verificar, ya que depende de factores externos al modelo; en concreto, en este paso deberemos confiar en la más importante de las condiciones de aplicabilidad: la coherencia de los modelos, no solo interna (es decir, modelos sintáctica y semánticamente correctos), sino la coherencia entre ellos. En la implementación actual asumiremos, por ejemplo, que el proceso de obtener las operacionalizaciones de un cierto softgoal se ha realizado de manera correcta, obteniendo tópicos coherentes con la parcela de sistema sobre la cual van a aplicarse - si esto no ocurre, estos pasos de la transformación se comportarán de forma errática, llevando a un modelo incorrecto. Asumiendo este postulado, en cambio, podemos verificar el siguiente razonamiento, expresado aquí en lenguaje natural: tomemos una parcela de sistema, afectada por un softgoal o meta suave. Este softgoal, a su vez, será descompuesto en operacionalizaciones, cada una de las cuales se aplica sobre un conjunto de tópicos. Ahora bien, dado que estos tópicos representan sub-parcelas de la parcela de sistema original, su unión estará incluida o igual en la parcela original (en el caso de la inclusión estricta, las sub-parcelas no incluidas serán aquellas que el requisito no funcional no afecta). Por lo tanto, reemplazar dicha parcela de sistema por aquellas representadas como tópicos no alterará el requisito MICNF original.

6.2.5. Ejemplo

Para esta primera transformación, los modelos de entrada serán un MICNF y una instancia del Modelo NFR. Para el caso del primero se presentan 8 ejemplos de MICNF[3], ilustrados en la Figura 6.6. Los mismos se describen textualmente de la siguiente manera:

(a) Brindar ayuda sobre operaciones tanto durante la compra de productos como durante la creación de una evaluación

(b) Loguear todas las operaciones de acceso a información almacenada luego de que éstas finalicen

(c) Luego de mostrar un producto a un cliente, mostrar también productos relacionados

(d) En todas las operaciones de consulta del Catálogo de Productos, brindar ayudas para la orientación del cliente en el sitio

(e) El cliente debe poder cancelar tanto las operaciones de Compra como las de Creación de una Evaluación

(f) Si al iniciar una sesión el sitio se encuentra sobrecargado, redirigir al cliente primero a un sitio espejo

(g) Antes de Acceder al Catálogo, Cerrar una cuenta o Comprar un producto, el cliente debe Validar su login y password

(h) Excepto al cerrar su sesión, el cliente debe poder obtener ayuda en todo momento

Cómo puede apreciarse, solamente dos de los MICNF presentados (los ejemplos *(a)* y *(d)*) hacen referencia a Requisitos No Funcionales. Es de esperarse que estos requisitos sufran alteraciones substanciales en comparación a los demás, por lo cual estos dos MICNF serán utilizados como ejemplos de aquí en más, en conjunto con los MICNF *(c)* y *(e)*.

Un segundo elemento necesario es un SIG del Framework NFR. Para el presente ejemplo, utilizaremos el SIG presentado en la Figura 6.7. En este ejemplo, la descomposición de *softgoals*

[3]Si bien es posible unificar los 8 requisitos en una única instancia del MICNF, se ha optado por mantenerlos separados por claridad

p

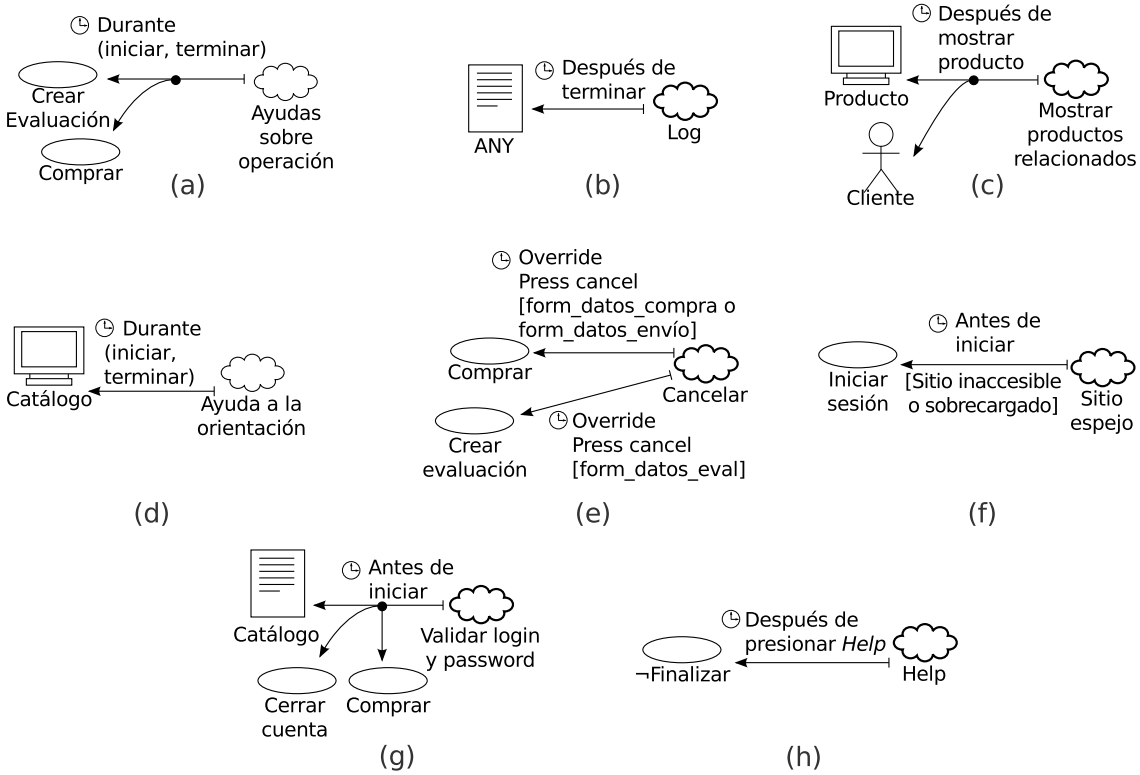

Figura 6.6: Ejemplos de Requisitos expresados en MICNF

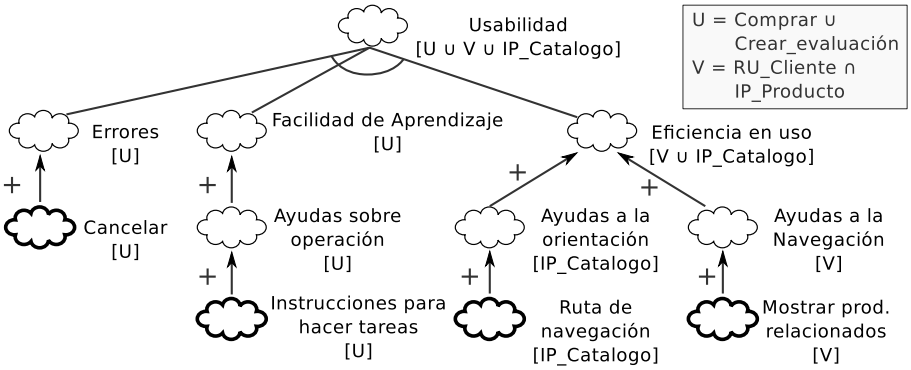

Figura 6.7: Ejemplo de Softgoal Interdependency Graph (SIG)

en *operacionalizaciones* se ha reducido a lo estrictamente necesario para brindar soporte a los ejemplos seleccionados. La referencia en el margen superior derecho es una guía a las abreviaturas utilizadas para hacer referencias a los tópicos de las descomposiciones: puede apreciarse en este ejemplo de qué manera los tópicos de la parcela de sistema *Eficiencia en uso* se descomponen en sub-tópicos en los softgoals hijos.

Tomando ambos como modelos de entrada de la primera Transformación, el resultado se detalla en la Figura 6.8.

Es importante notar en el ejemplo que, como se esperaba, los MICNF *(a)* y *(d)* han cambiado sus NFI, de softgoals a operacionalizaciones, siguiendo la descomposición delineada en el SIG. Los

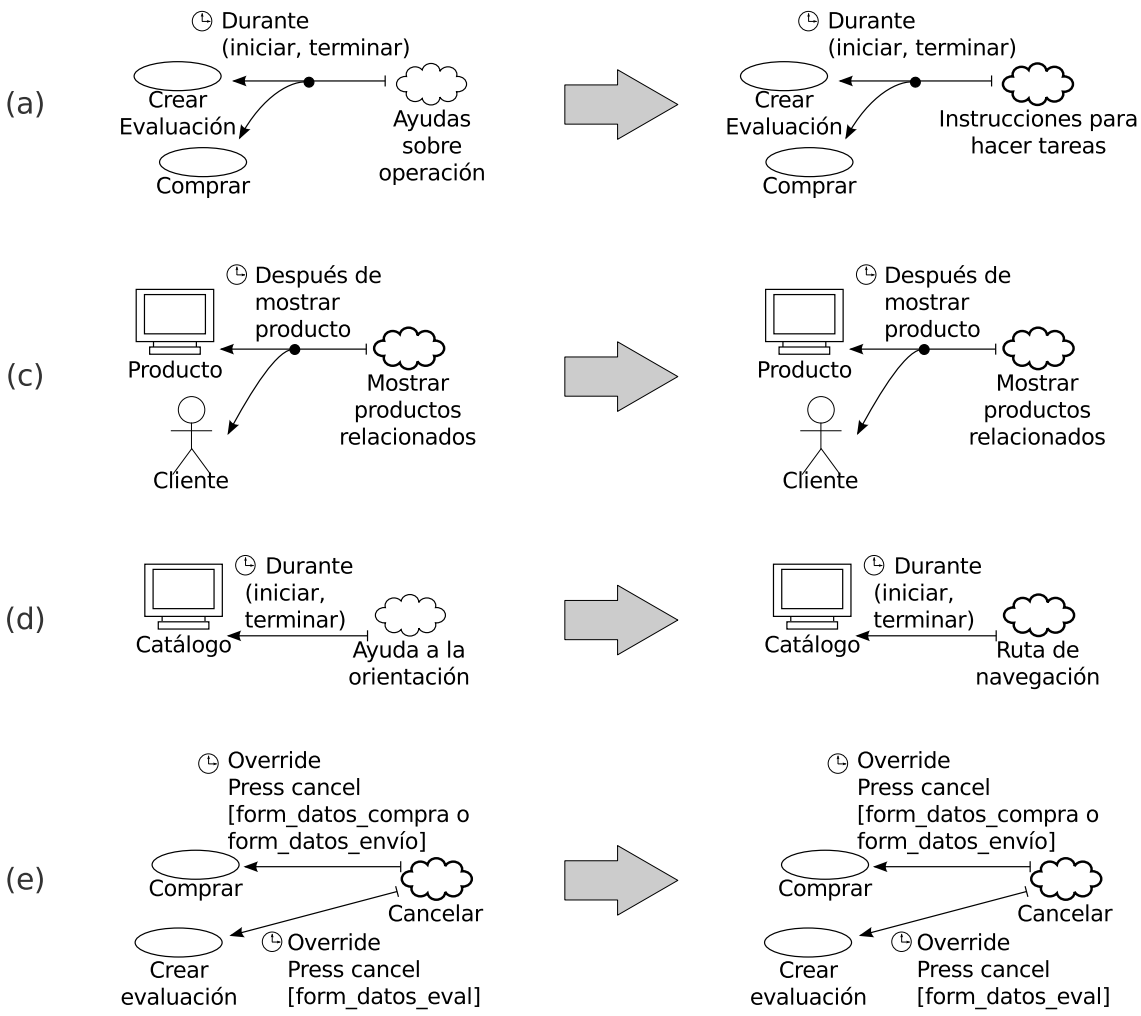

Figura 6.8: Resultado de la Transformación 1

MICNF *(c)* y *(e)*, en cambio, no han sufrido cambios aún.

Un segundo ejemplo del resultado de una Transformación es para el caso en el que existe una descomposición de tópicos dentro del SIG. Dicho SIG, presentado en la Figura 5.4 (página 56), se combina con un RICNF para obtener la transformación detallada en el ejemplo la Figura 6.9, donde vemos cómo tanto el NFI *Help* como la PS $\neg Finalizar$ se subdividen, el primero por efecto de la descomposición en Operacionalizaciones Operación del Softgoal representado por el NFI *Help* (se aprecia claramente que cada una de las 5 nuevas Operacionalizaciones Operación del SIG está presente en alguno de los modelos de salida), y el segundo por la introducción de nuevos tópicos durante la descomposición mencionada (para cada nueva Operacionalización, los tópicos sobre los cuales se aplica se traducen en una Parcela de Sistema); combinando ambas descomposiciones obtenemos 5 nuevos RICNF en reemplazo del original, aunque sin perder en ningún momento el requisito original: la combinación de estos nuevos RICNF daría como resultado un RICNF equivalente al de entrada.

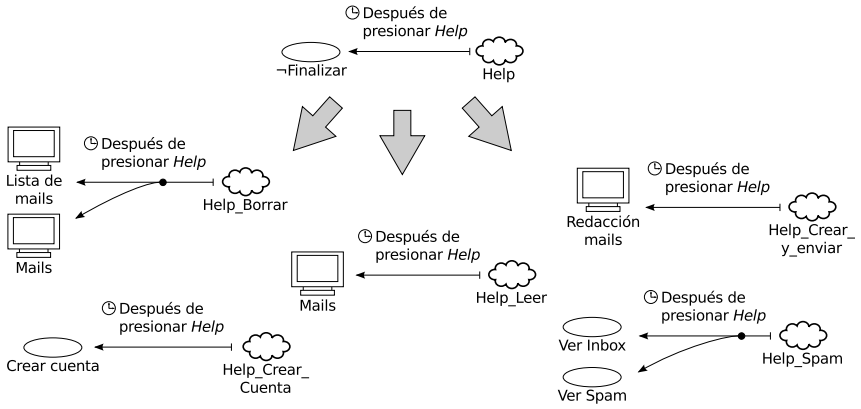

Figura 6.9: Ejemplo de transformación para el caso Help

6.3. Transformación 2

La segunda transformación, ilustrada en la Figura 6.10, combina la información obtenida en la transformación previa con las descripciones detalladas obtenidas de descripciones estructuradas de Casos de Uso. Ambas fuentes son entonces combinadas, y se vuelcan en la forma de nuevos RICNF, más detallados que los obtenidos en el paso anterior.

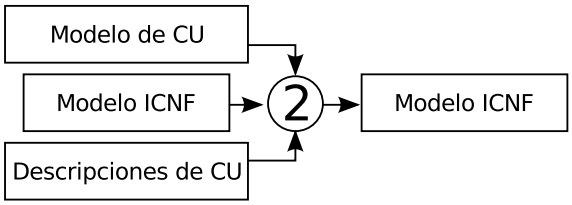

Figura 6.10: Diagrama de la Transformación 2

6.3.1. Especificación

Una característica particular de las parcelas de sistema de tipo `funcionalidad`, derivada del proceso de creación del metamodelo MICNF, es el hecho de que todas las demás parcelas pueden considerarse, en definitiva, como un conjunto de parcelas de sistema de este tipo. Teniendo esta consideración en mente, especificaremos la segunda transformación del presente trabajo como aquella en que, dado un requisito RICNF, se obtiene un conjunto de parcelas de sistema, todas ellas de tipo funcionalidad tales que, dada una parcela de sistema P, todas las parcelas de sistema de tipo funcionalidad $p_1, \ldots, p_n$ que la conforman se encuentran dentro de dicho conjunto de salida. Adicionalmente, exigiremos también que el nuevo MICNF mantenga la semántica del MICNF original.

Para cumplir con la especificación, todo RICNF R del MICNF de entrada de esta transformación se reemplaza por un RICNF R', de modo que R es igual a R' exceptuando las PS. Las PS de R', en cambio, se obtendrán de esta manera: primero se intersecarán todos los conjuntos de CU para las PS de R, y sobre esta intersección se descartarán aquellos CU que no contienen los eventos de ninguno de los parámetros de impacto de R.

6.3.2. Decisiones de diseño

Una de las decisiones más importantes a considerar para esta transformación es la cuestión de cómo obtener, partiendo de las Descripciones de CU, toda la información de parcelas de sistema de forma estructurada. Resulta claro que esta información es necesaria para establecer de qué manera se componen las distintas parcelas funcionalidad, pero decidir de qué modelos obtenerla es un proceso más complicado. Las facilidades de consulta XML de ATL no son lo suficientemente poderosas para obtener los CU a partir de las parcelas del sistema y de los parámetros de impacto, ya que las construcciones para realizar consultas son demasiado básicas, por lo cual realizar las consultas sobre estos modelos sería una tarea harto complicada. Se optó así por dividir esta etapa del proceso en dos etapas, tituladas como regla 2.1 y regla 2.2, las cuales se aprecian en la Figura 6.11.

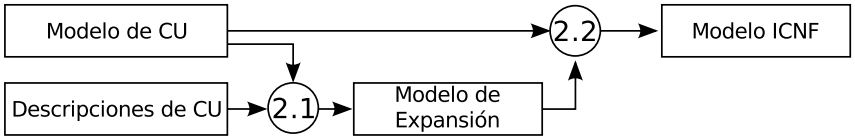

Figura 6.11: División de la Transformación 2

- La **regla 2.1** será la que realice las consultas del modelo de Descripciones de Casos de Uso (utilizando para ello XQuery) y generará un modelo extra, llamado *Modelo de Expansión*
- La **regla 2.2** tomará el modelo de expansión, interpretado como modelo de ATL, y lo utilizará para cumplir con la especificación general de la transformación, a saber, desplegar cada PS en las funcionalidades que lo componen, para luego descartar las combinaciones PS-NFI que no deban reflejarse en modelos posteriores.

De este modo, mientras la primera regla cumple el rol de facilitar la tarea de realizar consultas sobre el modelo (realizando todas las consultas necesarias y organizando sus resultados de modo tal que sean sencilla para consultar en ATL), la segunda es la que lleva el MICNF un paso más cerca del resultado buscado en esta etapa del proceso.

6.3.3. Regla 2.1

Especificación

Dado un Modelo de CU y las Descripciones de CU asociadas al mismo (es decir, todo CU del MCU de entrada se encuentra descripto en detalle en el documento XML que contiene las descripciones estructuradas), se obtendrá una instancia del Modelo de Expansión, el cual detallará de qué manera se despliega cada posible Parcela de Sistema de un MICNF en sus funcionalidades (en el contexto del MCU, una funcionalidad es igual a un Caso de Uso).

Decisiones de Diseño

Dado que los modelos de entrada para esta transformación ya están definidos, la única decisión pendiente es la de decidir cómo se obtendrá el modelo de salida, y en qué espacio tecnológico se reflejará (dado que la entrada consiste de un modelo y un texto estructurado, es preciso elegir uno de ellos como base para interpretar al otro). En este caso, dado que las consultas necesarias para obtener información de las Descripciones de CU son complejas, se optó por utilizar XQuery para realizarlas, inclinando la decisión hacia el espacio tecnológico de XML. Aún así, dado que la transformación siguiente debe realizarse en ATL, se decidió diseñar el modelo de salida de modo tal que pueda ser luego recogido en una transformación, utilizando la dualidad ATL-XML mencionada anteriormente (página 19); la alternativa de utilizar las facilidades de consulta XML

provistas por ATL fue descartada debido a la poca flexibilidad de dichas facilidades - el sistema de consulta de XML de ATL fue diseñado con el propósito de realizar consultas extremadamente simples, por lo que no están a la altura de la complejidad de esta etapa.

Implementación

La regla 2.1 es mayormente intuitiva, pero requiere conocer algunos detalles particulares acerca de los archivo XMI que utiliza ATL para almacenar modelos. Dado que XQuery permite la generación de archivos XML en base al resultado de consultas, entonces utilizando un DTD adecuado es posible crear un archivo XMI tal que, interpretado por Eclipse y ATL como un archivo .xmi, pueda ser importado como un modelo más y, por ende, sujeto al proceso de consulta y/o transformación que hemos desarrollado hasta este momento.

La clave, entonces, es la elaboración tanto de un DTD como de un modelo Ecore que compartan una sintaxis concreta, los cuales se presentan a continuación en la forma del llamado Metamodelo de Expansión. En principio, su metamodelo se define en la Figura 6.12.

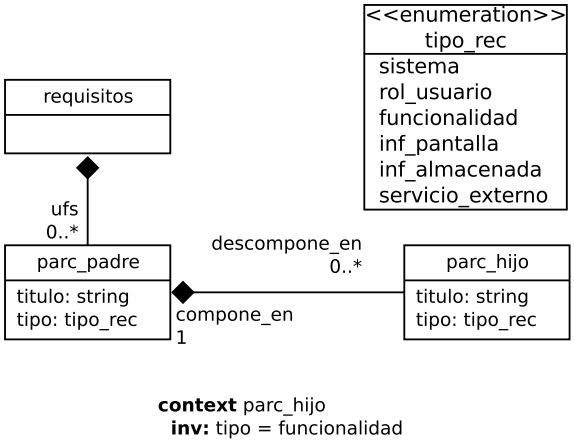

Figura 6.12: Metamodelo de Expansión

El DTD que lo define, por otra parte, se define en la figura 6.13

```
<!ELEMENT mmex:requisitos (parc_padre*)>
  <!ELEMENT parc_padre (parc_hijo*)>
  <!ATTLIST parc_padre titulo CDATA #REQUIRED>
  <!ATTLIST parc_padre tipo (funcionalidad|inf_pantalla|inf_almacenada
                      servicio_externo|sistema|rol_usuario) #REQUIRED>
    <!ELEMENT parc_hijo EMPTY>
    <!ATTLIST parc_hijo titulo CDATA #REQUIRED>
    <!ATTLIST parc_hijo tipo (funcionalidad|inf_pantalla|inf_almacenada
                   servicio_externo|sistema|rol_usuario) funcionalidad>
```

Figura 6.13: DTD del Metamodelo de Expansión

El proceso continúa, entonces, con la definición de las consultas correctas para obtener el Modelo de Expansión. Este modelo se caracteriza, simplemente, por estar formado por un primer conjunto de clases que representan a cada parcela de sistema, y un segundo conjunto de clases representando las parcelas de sistema funcionalidad de las que se compone cada elemento del primer conjunto. Dado que el orden de las consultas no afecta el resultado final (dado que no importa el orden en que se realicen las consultas), se optó por organizar las consultas XQuery por

tipo de Parcela de Sistema, consultando todas las expansiones de un mismo tipo antes de pasar al siguiente.

La expresión de la Figura 6.14, por ejemplo, obtiene todas las parcelas de sistema de tipo `rol_usuario`:

```
{
    for $x in doc("1.mrcc.xmi")//abarca_a
    where $x/@tipo="rol_usuario"
    return
        <parc_padre titulo="{data($x/@titulo)}" tipo="rol_usuario">
        {
            for $i in doc("2.descCU.xml")/CUs/descripcionCU
                where $i/base/actor = $x/@titulo
                return
                    <parc_hijo titulo="{data($i/base/nombre)}"
                    tipo="funcionalidad" />
        }
        </parc_padre>
}
```

Figura 6.14: Consulta XQuery para roles de usuario

Agregando declaraciones como esta, una tras otra, es posible extender la consulta indefinidamente, agregando un nuevo bloque para cada nuevo tipo de parcela de sistema.

Validación y Verificación

Para asegurar que el modelo de salida cumpla con los requisitos, debemos asegurar simplemente que el mismo expanda las PS del modo que corresponde, es decir, que el desplegado de una parcela corresponda con el resultado obtenido. Afortunadamente, la definición misma del documento XML nos facilitará este punto: seleccionando las propiedades correctas de cada entrada del documento, podemos seleccionar de manera precisa aquellas que nos interesan. Tomemos, a modo de ejemplo, el caso de la PS `rol_usuario`. Dado que un rol de usuario representa al rol que llama a la acción, resulta trivial establecer un paralelo con los actores del Diagrama de Casos de Uso y, por ende, con los actores presentes en la Descripción Estructurada. Así, para expandir una PS `rol_usuario`, todo lo que tenemos que hacer es seleccionar todas las funcionalidades que sean realizadas por el actor de mismo nombre que nuestra parcela.

Siguiendo similares argumentos para cada tipo de PS, obtendremos un Modelo de Expansión que cumple con nuestros requisitos, y dado que tanto las PS del MICNF como el MCU se basan en los mismos conceptos, establecer paralelos entre los elementos resulta posible en todos los casos.

Ejemplo

En la Figura 6.15 de la página 72 se describe un ejemplo simplificado de una Descripción de Caso de Uso, conteniendo únicamente la Descripción del Caso de Uso "Inspeccionar Catálogo".

Este documento será luego recogido por la consulta XQuery para obtener un Modelo de Expansión. Algunos de los análisis que se realizarán sobre este documento serán:

- El actor *cliente* está presente en la descripción del CU, por lo que la expansión de la PS Actor incluirá la PS Funcionalidad "Inspeccionar Catálogo"

- La acción *muestra* implica que se mostrará información en pantalla. Con un análisis más detallado, veremos que lo que se muestra es un listado de *productos*, por lo que la PS Información en Pantalla titulada *Producto* se expandirá también a "Inspeccionar Catálogo"

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE CUs PUBLIC "Caso de Uso" "ejemplo.dtd" >
<CUs>
  <descripcionCU>
    <base id="">
      <nombre>Inspeccionar Catálogo</nombre>
      <resumen>Mostrar el listado de productos disponibles
      en el Catálogo del Sistema</resumen>
      <actor>cliente</actor>
    </base>
    <descripcion>
      <pasodesc numpasodesc="1">
        <actor_desc>cliente</actor_desc>
        <accion_desc>selecciona</accion_desc>
        <objeto_desc>categoria</objeto_desc>
        <destinatario_desc>sistema</destinatario_desc>
      </pasodesc>
      <pasodesc numpasodesc="2">
       <actor_desc>sistema</actor_desc>
       <accion_desc>muestra</accion_desc>
       <objeto_desc>listado de productos</objeto_desc>
       <destinatario_desc>cliente</destinatario_desc>
      </pasodesc>
      <pasodesc numpasodesc="3">
       <actor_desc>cliente</actor_desc>
       <accion_desc>selecciona</accion_desc>
       <objeto_desc>producto</objeto_desc>
      </pasodesc>
      <pasodesc numpasodesc="4">
       <actor_desc>sistema</actor_desc>
       <accion_desc>redirige</accion_desc>
       <objeto_desc>compra</objeto_desc>
       <condicion_desc>actor ingresó previamente al sistema</condicion_desc>
      </pasodesc>
    </descripcion>
    <alternativa viene_de="4" sale_en="4">
     <num_alt>1</num_alt>
     <condicion>actor no ingresó al sistema</condicion>
     <pasodesc numpasodesc="1">
     <actor_desc>sistema</actor_desc>
     <accion_desc>muestra</accion_desc>
     <objeto_desc>mensaje de error</objeto_desc>
     <destinatario_desc>actor</destinatario_desc>
     </pasodesc>
    </alternativa>
  </descripcionCU>
</CUs>
```


Figura 6.15: Descripción Textual de Caso de Uso "Inspeccionar Catálogo"

- Trivialmente, la PS "Inspeccionar Catálogo" se expandirá a sí misma

Siguiendo razonamientos similares sobre las PS del ejemplo, y dado una descripción exhaustiva de los Casos de Uso presentes en el Modelo de Casos de Uso, obtendremos un Modelo de Expansión como el que se ilustra en la Figura 6.16[4].

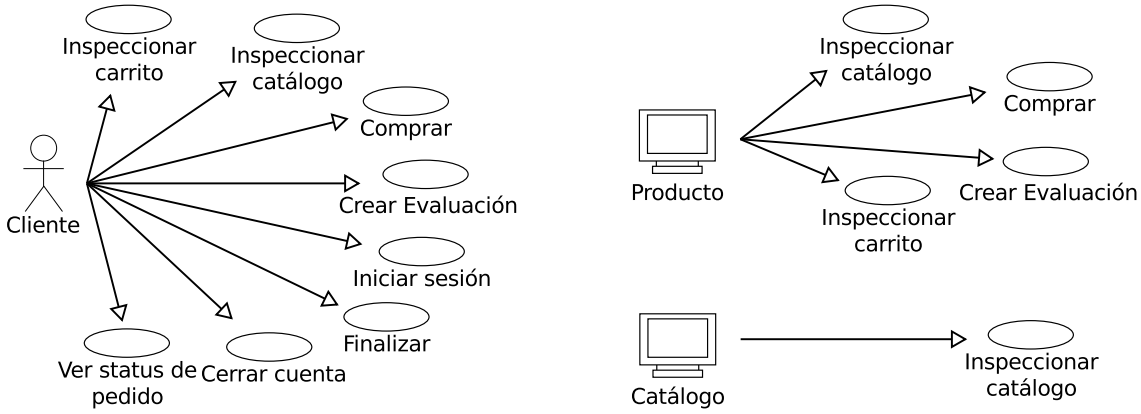

Figura 6.16: Modelo de Expansión para las MICNF del ejemplo

6.3.4. Regla 2.2

Especificación

Dado un MICNF de entrada y un Modelo de Expansión (el cual determina de qué manera descomponer una Parcela de Sistema en las funcionalidades que la componen), la regla 2.2 deberá realizar dos tareas:

- Reducir cada PS a sus funcionalidades mínimas (es decir, a únicamente aquellas que son afectadas por algún Interés No Funcional) e intersecarlas
- Eliminar RICNFs que sean imposibles de cumplir, incluyendo el descarte de funcionalidades que no contengan los eventos de ninguno de los parámetros de impacto en el RICNF.

Decisiones de Diseño

En cuanto al punto 2, "RICNFs que sean imposibles de cumplir", se ha optado por refinar esta sección de la especificación a "RICNFs incoherentes con la sintaxis". En cuanto a incoherencias *semánticas* (es decir, MICNFs que poseen una sintaxis correcta, pero donde un análisis de los elementos representados revela incompatibilidades entre los elementos, como puede ser una condición que nunca se cumplirá para un determinado rol de usuario), se ha decidido analizar las herramientas de la implementación para verificar la posibilidad (o imposibilidad) de tratar con estos casos. Esto incluye descartar funcionalidades de la intersección de las PS de un RICNF que no contienen los eventos necesarios de ninguno de los parámetros de impacto en dicho RICNF, así como también los casos donde una cierta combinación de elementos se interprete como un requisito de cumplimiento imposible.

Implementación

La implementación de la primera parte de la regla 2.2 (la expansión o desplegado de las PS) resulta simple y directa. Dado que ya sabemos de qué manera una PS se convierte en sus PS

[4]Los Modelos de Expansión no poseen representación gráfica - se los presenta aquí de esta manera por una cuestión de claridad

funcionalidad (gracias al Modelo de Expansión), el primer paso de la implementación consiste en ubicar la parcela equivalente en el Modelo de Expansión (es decir, aquella que tenga el mismo tipo y el mismo título), recorrer la relación `descompone_en`, y crear por cada uno de estos elementos una PS `funcionalidad` que se vincule con las mismas clases `req_icnf` que el MICNF original. El código ATL detallado en la Figura 6.17 es realiza precisamente esto: dada una PS, devuelve los títulos de las PS funcionalidad en las cuales se descompone

```
helper def : expande( s : String) : Sequence(String) =
  let titulo : String = s.regexReplaceAll('^.._','') in
    MMEX!parc_sistema.allInstances()
    ->select( e | e.compone_en->size()>0 and e.descompone_en->size()=0)
    ->select( e | e.compone_en->exists(g | g.titulo = titulo))
    ->collect( f | f.titulo).asSequence();
```

Figura 6.17: Código para obtener la Descomposición de una Parcela de Sistema

Ahora bien, no todas las parcelas nos serán de utilidad - en concreto, sólo nos interesan las PS `funcionalidad` que, a su vez, correspondan con algún tópico de algún NFI del RICNF (el cual se encuentra en un SIG donde aparece el NFI). Para realizar esta parte del proceso, las funcionalidades de trabajo con conjuntos de ATL (unión, intersección, extracción de elementos, etc.) resultan de gran utilidad, ya que será necesario utilizarlas para obtener la intersección de varias PS de un mismo RICNF. Así, dado que en un RICNF las PS están en un tópico de cualquiera de los NFI en un SIG apropiado, y ese tópico involucra los títulos de las PS `funcionalidad` expresados en Forma Normal Conjuntiva, lo que debemos hacer es:

- Separar el string en una lista de cadenas, en base al símbolo de intersección
- Separar, para cada subcadena de intersecciones, el título de cada PS involucrada
- Para cada cadena (la cual tiene la forma `<tipo de parcela>_<nombre>`), convertirla en una PS y calcular su expansión dentro del Modelo de Expansión
- Realizar la unión de cada resultado parcial
- Calcular la intersección con cada resultado

En conjunto con la regla dada anteriormente, dos de las principales funciones en este procedimiento serán las que separan cada cadena en sus elementos para luego calcular su intersección/unión, como se aprecia en las figuras 6.18 y 6.19.

Una segunda parte de esta regla, más particular al dominio de aplicación en el cual se aplique el MICNF, es la eliminación de RICNF imposibles. Esto ocurre, por ejemplo, cuando las funcionalidades de una PS y los tópicos que la afectan tienen una intersección vacía (implicando, por ende, que los requisitos crosscutting elegidos para implementar un determinado NFI no son los indicados para la PS seleccionada).

En la implementación concreta realizada para el presente proyecto, esta etapa no fue tenida en cuenta, no por cuestiones técnicas (las cuales fueron resueltas de forma efectiva), sino más bien por cuestiones semánticas: si bien un procedimiento que descarte situaciones que no puedan ocurrir es posible, dicho procedimiento debe basarse en un enfoque de *lista negra*, es decir, un enfoque donde se listan explícitamente las situaciones que deben descartarse. Este procedimiento, sin embargo, tiene una debilidad conocida: sin importar la extensión de la lista, siempre existirán elementos que, por tener incoherencias semánticas (el requisito no tiene sentido) en vez de sintácticas (el requisito no cumple con la sintaxis), pasarán este chequeo. Por este motivo, es preferible que sea el analista el que detecte el primer caso, en la Tabla de Composición, donde su detección será más sencilla, mientras que el segundo (requisito incoherente con el MICNF) será detectado y reportado por la herramienta, sin necesidad de intervención de nuestra parte.

```
-- Toma una colección de parcelas (donde cada elemento tienen la forma
-- "a U b U c" y devuelve la intersección de ellas, expandiendo en el
-- proceso los elementos
helper def :
calcula_intersec(conj_rest : Sequence(MICNF!parcela_sistema)) : Sequence(String) =
  if (conj_rest.size() = 0)
  then
    -- Conjunto vacio, no hay nada
    Sequence{}
  else
    if(conj_rest.size() = 1)
    then
      -- Conjunto con un único elemento - devolvemos ese mismo conjunto
      -- pero con sus elementos expandidos
      thisModule.calc_union(conj_rest->first().titulo)
    else
      -- Dos o mas elementos
      -- Calculamos la intersección recursiva entre el primer elemento
      -- (expandido) y el resto del conjunto
      let c1 : Sequence(String) =
               thisModule.calc_union(conj_rest->first().titulo) in
      let c2 : Sequence(String) =
               thisModule.calcula_intersec(
                      conj_rest->excluding(conj_rest->first())) in
      -- La intersección común no funciona aquí por (aparentemente)
      -- problemas con tipos, así que usamos nuestra implementación
      thisModule.intersecar(c1,c2)
    endif
  endif
;
```

Figura 6.18: Código ATL para la intersección de parcelas

Validación y Verificación

Afortunadamente, la verificación de este paso del modelo ya ha sido realizado por las etapas anteriores de la transformación. En concreto,

- Al seleccionar las parcelas dentro del Framework NFR, el analista ha dado explícitamente las PS mínimas que requiere el MICNF. Esto, intersecado con la descomposición de cada PS obtenida en la Regla 2.1, nos asegura que dispongamos de las mínimas parcelas necesarias para el cumplimiento del RICNF.

- El descarte de las parcelas con incoherencias sintácticas es realizado por la misma herramienta, por lo que esta parte de la especificación se cumplirá trivialmente. Las incoherencias semánticas, por otra parte, no son tratables con las herramientas provistas por ATL, con lo cual la especificación tal cual fue definida (es decir, donde la frase "RICNFs que sean imposibles de cumplir" se ha definido como "RICNFs incoherentes con la sintaxis") demuestra ser la definición más general posible.

Ejemplo

Tomando como base la expansión de los MICNF dada por el resultado de la Transformación anterior, podemos ver el resultado de aplicar la Transformación a nuestros modelos en la Figura

```
-- Toma un string representando un conjunto de parcelas con la forma
-- "a U b U c" y devuelve la unión de las mismas, previa expansión
helper def : calc_union(parc_rest : String) : Sequence(String) =
  if (parc_rest->size() = 0)
  then
    -- Cadena vacía, no hay nada
    Sequence{}
  else
    -- Buscar qué constante corresponde en vez de -1
    let pos : Integer = parc_rest->indexOf(' U ') in
    if(pos <= 0)
    then
      -- Conjunto con un único elemento - Devolvemos ese mismo conjunto
      -- pero con sus elementos expandidos
      thisModule.expande(parc_rest)
      else
        -- Dos o mas elementos
        -- Calculamos la unión recursiva entre el primer elemento
        -- (expandido) y el resto del conjunto
        thisModule.expande(parc_rest->substring(1,pos))->union(
        thisModule.calc_union(parc_rest->substring(
                                  pos + 4,parc_rest->size())))
      endif
  endif
;
```

Figura 6.19: Código ATL para la unión de parcelas

6.20.

En este caso deberemos centrar nuestra atención en los MICNF *(c)* y *(d)*:

- En el caso del MICNF *(c)*, vemos que sus PS han sufrido un cambio notable. Esto ocurre porque la expansión de la PS Cliente da como resultado todas (menos una) las PS Funcionalidad presentes en el Modelo de Casos de Uso en la forma de Casos de Uso. Al hacer la conjunción con las PS con información en pantalla sobre Productos, el resultado son las PS que se aprecian en el ejemplo
- A diferencia del caso anterior, aquí la PS Información en Pantalla relacionada con el Catálogo se expande a una única PS Funcionalidad, a saber, la PS "Inspeccionar Catálogo".

6.4. Transformación 3a

La última etapa del proceso de la transformación está compuesto por las reglas 3a (detallada en la Figura 6.21) y 3b, las cuales (a diferencia de reglas anteriores) son paralelas entre sí. En la regla 3a, por cada softgoal operación en el MICNF de entrada se crea un CU y ese CU se vincula con los CU que impacta (en el MICNF de entrada) ya sea usando una relación *crosscut* o una relación *extend* (de acuerdo con las reglas de Sousa definidas en [38]) para determinar el tipo de relación. Adicionalmente, el Modelo de CU Unificado contiene al modelo de CU en su totalidad.

6.4.1. Especificación

Dado que en este punto del proceso ya disponemos de toda la información necesaria para elaborar este modelo, todo lo que debe hacer esta regla es consultar y reordenar esta información

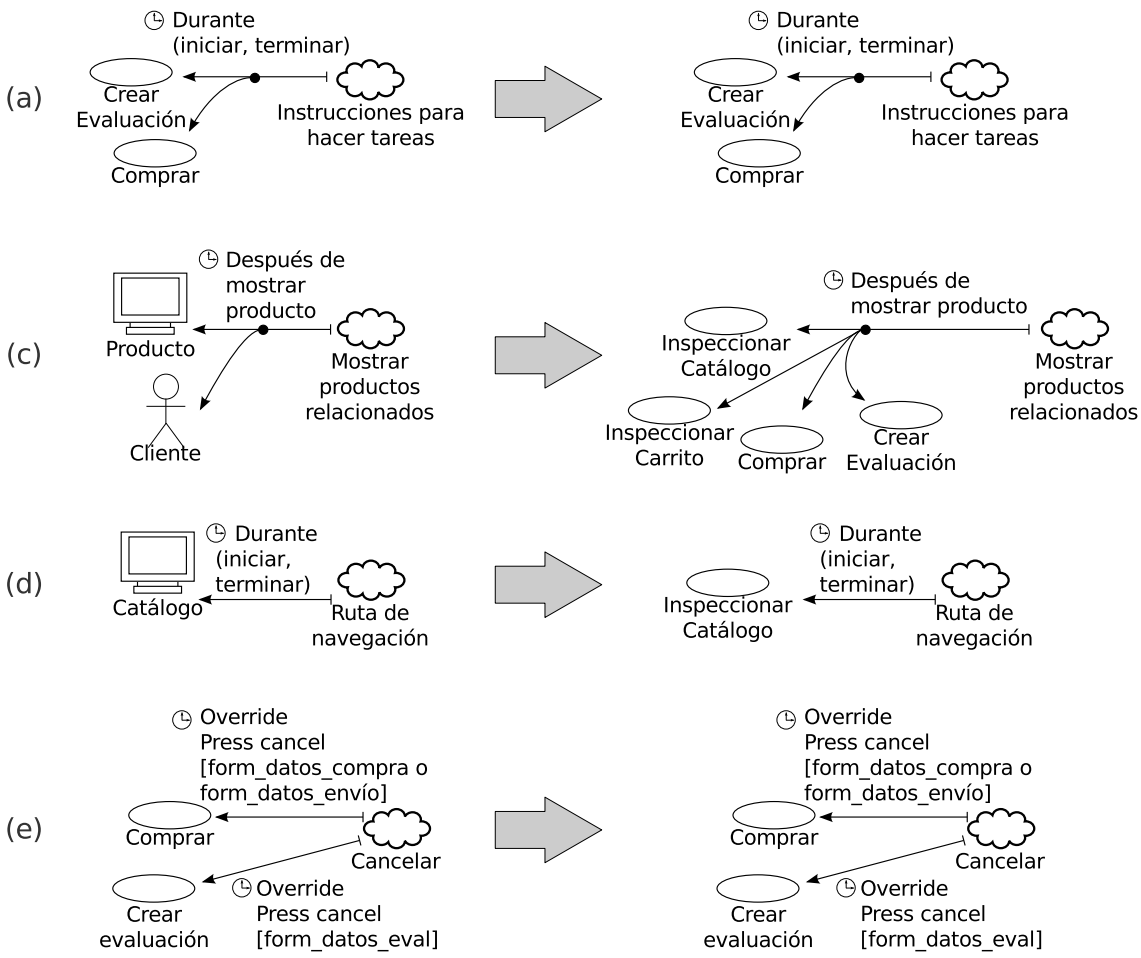

Figura 6.20: Resultado de la Transformación 2.2

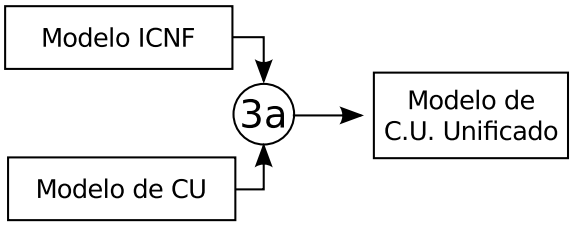

Figura 6.21: Diagrama de la Transformación 3a

en la forma de un Modelo de CU Unificado. Este modelo, basado en la metodología desarrollada por Sousa [38], determina de qué manera la implementación de los Intereses No Funcionales (representados por Casos de Uso Crosscutting) se relacionan con las funcionalidades del sistema (representadas por Casos de Uso). La especificación de esta regla, entonces, consiste en los siguientes ítems:

- El Modelo de CU Unificado contiene la totalidad de los elementos del Modelo de Casos de Uso
- Cada elemento del Modelo de Casos de Uso de entrada sobre el cual se fijó un interés no

funcional en alguna etapa anterior deberá estar relacionado con el Caso de Uso Crosscutting que corresponda, siguiendo los parámetros definidos por Sousa.

6.4.2. Decisiones de Diseño

Un detalle importante que puede notarse en el metamodelo es que la relación entre un CU y un CUC no lleva etiqueta. Esto obedece, en realidad, a una particularidad del dominio sobre el cual se ha realizado el análisis: si bien Sousa propone tres tipos de relaciones (*includes*, *extends* y *crosscuts*), sólo *crosscuts* es factible de ocurrir. Esto obedece al enfoque adoptado para la recolección de requisitos en etapas anteriores:

- El mecanismo adoptado que permite la refinación de parcelas de sistema acorde a las subdivisiones del Framework NFR implica que un CUC no puede representar un curso complejo y alternativo específico a un CU (dado que el CUC fue derivado sin tomar al CU como base), por lo que el caso *extends* no puede ocurrir.

- Por un argumento similar, el CUC no puede ser una parte esencial para cumplir el propósito primario de un CU, ya que un NFR no es (por definición misma) una funcionalidad y, por ende, no puede ser una parte esencial de ningún CU, por lo que la relación *includes* tampoco puede ocurrir.

6.4.3. Implementación

El primer requisito de la especificación resulta de implementación trivial en el modelo de salida: dado que el modelo debe copiarse de manera íntegra, las reglas ATL encargadas de este trabajo serán simples transformaciones identidad, es decir, reglas donde el modelo de entrada y el de salida poseen las mismas propiedades. El mapeo, entonces, se llevará a cabo con reglas similares a la ilustrada en la Figura 6.22.

```
unique lazy rule ps2cu {
    from
        s: MICNF!parcela_sistema
    to
        t:MCUC!caso_uso (
            titulo <- s.titulo
        )
}
```

Figura 6.22: Regla identidad en la Transformación 3a

El segundo punto, en cambio, resulta más interesante. Esencialmente, cada NFI del MICNF de entrada representa un Caso de Uso Crosscutting, por lo que cada NFI del MICNF será mapeado mediante una regla similar a la anterior, pero con una salvedad: toda PS con la cual esté relacionado el NFI será interpretada como una relación entre un CUC y un CU:

Este código recoge todas las PS con las que se relaciona el NFI y agrega, en el Modelo de Casos de Uso Unificado, una relación *crosscuts* entre ambos. En este sentido, una de las principales decisiones de implementación que se tuvo en cuenta es la cuestión de cuál debía ser el punto de referencia para la transformación, es decir si se debía completar los CU una vez tomado un CUC como referencia, o a la inversa. Debido a la simplicidad del modelo de salida, esta decisión no tiene mayor peso en el resultado (a diferencia de la regla 3b, donde el punto de referencia tiene influencia en el resultado).

```
rule nfi2cuc {
        from
                s:MICNF!nfi
        to
                t:MCUC!caso_uso_cc (
                        titulo <- s.titulo,
                        afecta_a <- s.restringido_a->collect(
                        f | f.abarca_a->collect(g | thisModule.ps2cu(g)))
                )
}
```

Figura 6.23: Regla de mapeo de PS a CU en la Transformación 3a

6.4.4. Validación y Verificación

Verificar que el Modelo de Casos de Uso se encuentre representado en su totalidad en el Modelo de Casos de Uso Unificado resulta sencillo - al ser las transformaciones reglas de tipo identidad, basta con asegurar que todo elemento del modelo de entrada se repita en el modelo de salida sin cambios. La observación realizada en la etapa de diseño con respecto a las relaciones *crosscuts* nos asegura, por otra parte, que todo lo que debemos hacer es vincular los nuevos CUC con las PS con las que se relacionaban en el MICNF de entrada, ya que es tarea de las reglas anteriores asegurar que la coherencia del MICNF se haya mantenido a lo largo de todo el proceso.

6.4.5. Ejemplo

Ahora que en los MICNF de ejemplo sólo hay PS Funcionalidad y NFIs que se corresponden con Operacionalizaciones, estamos en condiciones de combinar esta información en un Diagrama de Casos de Uso Unificado. El resultado de este proceso puede apreciarse en la Figura 6.24.

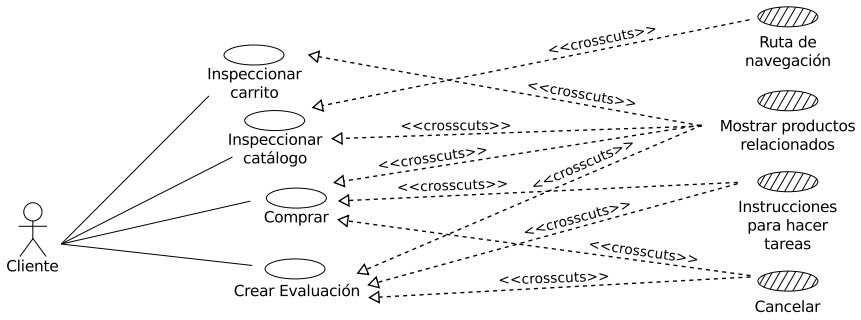

Figura 6.24: Resultado de la Transformación 3a

6.5. Transformación 3b

Para la última regla de transformación del proceso, por cada softgoal operación del MICNF de entrada se tiene una tabla; para cada CU impactado por el softgoal, se crean filas en esa tabla indicándose: la condición, tipo de impacto y pasos afectados del CU (obtenidos de averiguar en las descripción del CU los números de pasos donde ocurren los eventos de los parámetros de impacto). Todo el proceso se ilustra en la Figura 6.25

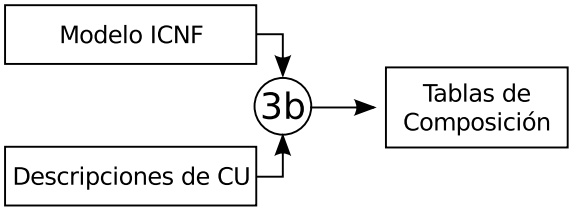

Figura 6.25: Diagrama de la Transformación 3b

6.5.1. Especificación

Esta regla es muy similar a la regla 3a, en cuanto únicamente debe funcionar como una regla de consulta sobre los modelos de entrada, con la diferencia de que el modelo de salida obtenido será más complejo, por poseer mayor información. En concreto, se espera que esta regla se ajuste a lo siguiente:

- Para cada Caso de Uso Crosscutting debe existir una tabla, donde se indique con cuáles CU se vincula y con cuáles parámetros
- Toda Parcela de Sistema del MICNF y todo NFI debe verse representada en el modelo de salida

6.5.2. Decisiones de Diseño

Debido a la similitud entre los modelos involucrados y el resultado esperado, las decisiones de diseño tomadas para la transformación 3a son también válidas aquí. La decisión que marca la diferencia entre ambas es el tipo de modelo esperado como salida: debido a que las Tablas de Composición son un tipo de texto estructurado, es deseable que el resultado de esta transformación sea un documento XML, y no un modelo. Para ello, una vez más se ha hecho uso de la dualidad ATL-XML. De este modo, el metamodelo de salida fue diseñado para que, interpretado como documento XML, su sintaxis sea clara y directa.

6.5.3. Implementación

La etapa de implementación de esta transformación resulta muy similar a la de la transformación 3a; en concreto, las consultas que deben realizarse son las mismas, pero tomando un punto de referencia distinto: para completar cada fila de la tabla, la regla de transformación analiza todos los pares (PS, NFI), toma aquellos que se encuentran relacionados por un `req_icnf`, y completa los datos de la tabla con dicha información. La regla se define en la Figura 6.26

Por otra parte, es también necesario consultar las Descripciones de CU para poder buscar correctamente los eventos en los CU. Debido a que esta es una consulta mucho más sencilla sobre un documento XML que las presentadas en la Regla 2.1, las funcionalidades básicas de consulta de Eclipse bastarán para la tarea. Un fragmento simplificado de las funciones utilizadas se detalla en la Figura 6.27:

El primer *helper*, `get_eventos_cu_durante`, es el que obtiene todos los eventos que ocurren desde que ocurre el evento a hasta que ocurre el evento b, mientras que el segundo *helper*. `get_nro_evento_cu`, obtiene los números de línea que corresponden a un determinado evento. En ambos casos, el código recorre las ramas del Documento Object Model (DOM) del documento XML ingresado (como el que se presenta en la Figura 6.15 de la página 72), descendiendo desde el elemento principal hacia los sub-elementos que cumplen con una cierta condición.

```
rule req2fila {
        from
                s:MICNF!parcela_sistema,
                t:MICNF!nfi
        to
                w:MTC!fila (
                        cuc<-t.titulo,
                        condicion<-t.restringido_a.first().condicion,
                        operador<-thisModule.comb_operador(
                                t.restringido_a.first().limitado_por),
                        punto_afec<-thisModule.comb_pasos(
                                t.restringido_a.first().limitado_por),
                        rel<-#crosscuts,
                        comp_a<-thisModule.ps2cu(s)
                )
}
```

Figura 6.26: Regla de mapeo de PS a Tablas de Composición en la Transformación 3b

6.5.4. Validación y Verificación

Al igual que ocurre con la regla 3a, el proceso de verificación del modelo de salida se limita a asegurar que todos los elementos se hayan mapeado de forma correcta, dado que cualquier otro tipo de tarea de verificación debió realizarse en etapas anteriores. En cuanto a los requerimientos de que toda PS y todo NFI se vean reflejados en el modelo de salida, la ausencia de reglas de tipo *lazy* (es decir, reglas que se llaman únicamente en ocasiones especiales) nos asegura que toda parcela de entrada de las clases `parcela_sistema` y `nfi` se verán reflejadas en el modelo de salida.

6.5.5. Ejemplo

La Figura 6.28 nos muestra la tabla resultante para los ejemplos mencionados a lo largo de este capítulo. Como puede apreciarse, los CU y los CUC que figuran en ella son los mismos que en la Transformación 3a, pero con mayor información sobre la relación *crosscutting* entre ambos.

6.6. Impresiones sobre la herramienta

La experiencia de trabajo con el lenguaje de transformaciones ATL sobre el framework Eclipse ha resultado ser mixta: si tuviéramos que dar una definición, podríamos decir que ATL es un lenguaje con una gran especificación, pero una implementación con amplio espacio para mejoras.

En concreto, las características del lenguaje ATL han demostrado ser muy robustas. Las construcciones declarativas dan como resultado reglas de transformación compactas (mas no por ello menos poderosas), las cuales combinadas con construcciones OCL dan como resultado un código claro y, en ocasiones, hasta intuitivo. La introducción de construcciones imperativas, por otra parte, y lejos de complicar la sintaxis, brindan un "toque" extra a la expresividad del lenguaje, permitiendo al desarrollador solucionar ciertos problemas particulares (iteraciones, mayormente) de forma directa y sencilla. Adicionalmente, los tres estilos de reglas en los que se subdivide el lenguaje (*matched rule*, *lazy* y *unique lazy*) han probado ser un acierto, ya que cada tipo de regla resulta indispensable para un cierto subdominio del problema; resulta difícil imaginar de qué manera agregar (o quitar) alguna de estas variedades de reglas pueda mejorar la experiencia del desarrollador.

Lamentablemente, y a pesar de ser un lenguaje tan bien concebido, la implementación oficial, realizada sobre el framework Eclipse, deja mucho que desear. Si bien el editor integrado cumple con las funciones básicas esperadas de un IDE, el proceso de *debug* es extremadamente frustrante:

```
helper def : get_eventos_cu_durante (a : String, b : String) : Sequence(String) =
  -- Obtiene el primer número de línea del rango,
  -- y lo guarda en c1
  let c1 : Integer =
    XML!Element.allInstancesFrom('CUs')
      ->select(e | e.name='descripcion')
        ->select(e | e.name='pasodesc' AND e->select(
          f | f.name='accion_desc' AND f.getAttrVal() = a)
        ->collect(e | f.getAttrVal('numpasodesc') in
  -- Obtiene el segundo número de línea del rango,
  -- y lo guarda en c2
    let c2 : Integer =
      XML!Element.allInstancesFrom('CUs')
        ->select(e | e.name='descripcion')
          ->select(e | e.name='pasodesc' AND e->select(
            f | f.name='accion_desc' AND f.getAttrVal() = b)
          ->collect(e | f.getAttrVal('numpasodesc') in
  -- Obtiene todos los eventos intermedios,
  -- y hace un conjunto con ellos
    let c2 : Integer =
      XML!Element.allInstancesFrom('CUs')
        ->select(e | e.name='descripcion')
          ->select(e | e.name='pasodesc' AND
                       e.getAttrVal('numpasodesc') >= c1 AND
                       e.getAttrVal('numpasodesc') <= c2)
            ->select(e | e.name='accion_desc')
            ->colect(e | e.getAttrVal());

helper def : get_nro_evento_cu( evento : String) : Sequence(String) =
  XML!Element.allInstancesFrom('CUs')
    ->select(e | e.name='descripcion' OR e.name='alternativa')
      ->select(e | e.name='pasodesc' AND e->select(
        f | f.name='accion_desc' AND f.getAttrVal() = evento )
      )->collect(e | e.getAttrVal('numpasodesc'));
```

Figura 6.27: Reglas de consulta de XML para parámetros de Tabla de Composición

por una parte, los mensajes de error en transformaciones son altamente crípticos, y con frecuencia la relación entre el mensaje y el error es difícil de ver (una asignación nula, por ejemplo, puede resultar en un volcado de excepciones Java, cuyo último mensaje es un error de instanciación de una clase desconocida). El *debugger* integrado, por otra parte, sufre con frecuencia problemas de descoordinación entre el código virtual y el que se muestra en pantalla, resultando en una interfaz donde la línea que se está ejecutando no corresponde con la mostrada, o un determinado *breakpoint* es ignorado, o puede simplemente ocurrir que el *debugger* se reuse a ejecutarse. Todo esto se ve complicado aún más por la presencia de dos máquinas virtuales para la ejecución del código ATL compilado: la primera tiende a reportar correctamente los errores (aunque peca en ocasiones de no ejecutar una regla sin dar detalles del porqué), pero no funciona correctamente en modo *debug*, mientras que la segunda funciona mejor en este modo, pero reporta errores en líneas sin relación con el problema real, o con errores distintos al verdadero; el hecho de que un mismo error pueda ser reportado como dos errores distintos según la máquina virtual que se utilice (o, peor aún, que el código corra con éxito en una pero falle en la otra) no hace sino agravar el problema.

Una característica que no se ha evaluado es la posibilidad de eliminar toda interacción de

Ruta de Navegación

CU Afectado	Condición	Operador de regla de composición	Punto Afectado
Inspeccionar catálogo	-	wrap	1 a 27

Mostrar Productos Relacionados

CU Afectado	Condición	Operador de regla de composición	Punto Afectado
Inspeccionar catálogo	actor=cliente	overlap.after	14
Inspeccionar carrito	actor=cliente	overlap.after	3
Comprar	actor=cliente	overlap.after	2
Crear evaluación	actor=cliente	overlap.after	9

Instrucciones para hacer tareas

CU Afectado	Condición	Operador de regla de composición	Punto Afectado
Comprar	-	wrap	1 a 4
Crear evaluación	-	wrap	1 a 29

Cancelar

CU Afectado	Condición	Operador de regla de composición	Punto Afectado
Comprar	se presiona el botón "Cancelar"	override	1 a 4
Crear evaluación	se presiona el botón "Cancelar"	override	11 a 29

Figura 6.28: Resultado de la Transformación 3b

usuario, ya sea definiendo las llamadas *ant tasks* (una herramienta que permite encadenar la salida de una transformación con la entrada de otro) o bien utilizando la interfaz Java de ATL (eliminando Eclipse de la ecuación, haciendo los llamados a los métodos de las clases internas que definen ATL). Esto obedece al hecho de que, dado que esta implementación dista de ser una herramienta de producción, el método actual (definiendo manualmente los modelos, y ejecutando las transformaciones de modo interactivo) enriquece la experiencia del desarrollo (permitiendo un mayor contacto con el código, su ejecución, y los pasos involucrados en cada etapa). Reconocer la existencia de estas funcionalidades puede resultar provechoso para un futuro trabajo sobre esta misma herramienta.

Capítulo 7

Trabajo Relacionado

Un tipo interesante de trabajos son aquellos que a partir de un modelo de requisitos no OA permiten identificar ya sea automática o semi-automáticamente aspectos (candidatos) y reflejar su comportamiento en modelos de requisitos OA.

Existen varios enfoques que parten de modelos i* [45] para identificar aspectos candidatos [39, 28, 2, 26]. El problema de partir de i* es que es un lenguaje que involucra numerosos conceptos técnicos, no captura los parámetros de impacto ni las parcelas de sistema de los RICNF y a partir de ellos solo se pueden obtener modelos de requisitos OA más pobres que los nuestros.

En [14] a partir de descripciones de CU, diccionarios de sinónimos, listas de palabras clave de RNF se identifican aspectos candidatos y se establecen relaciones crosscutting entre los aspectos candidatos y los CU donde fueron identificados. Nosotros en lugar de partir de descripciones de CU conteniendo referencias a RNF partimos de modelos ICNF y descripciones de CU sin referencias a RNF. Además producimos tablas de composición que son más detalladas.

En [43] se parte de matrices de dependencias entre unidades de software (que pueden ser URF) e intereses (que pueden ser CNF) y a partir de estas matrices se identifican aspectos candidatos y se obtiene la información de qué aspectos candidatos afectan a qué unidades de software. Otro tipo de aportes son los modelos de requisitos que permiten describir las relaciones entre RNF y requisitos funcionales. Existen varios enfoques de modelado basados en metas entre ellos: KAOS [23], NFR [9], i* [45], GRL [1, 19, 20], etc. Pero ellos consideran facilidades como los parámetros de impacto RICNF, las PS de los RICNF. Además involucran numerosos conceptos técnicos.

En [41] se propone un lenguaje de modelado integrado extendiendo UML para poder relacionar elementos de modelos de CU con softgoals del framework NFR; esto hace que se puedan asociar softgoals a: sistema (softgoals globales), a CU (softgoals específicos a CU), a actores (softgoals que restringen a actores), a relaciones entre actores y CU (softgoals que restringen la interfaz entre actores y el sistema). Entonces se tienen algunas PS, aunque son menos y menos detalladas que las de los RICNF (p.ej. no está información almacenada e información en pantalla); además en este enfoque no se consideran parámetros de impacto como en RICNF.

Existen numerosas notaciones de modelado de requisitos OA (p.ej. [16, 38, 27, 33]), pero estas suelen incluir conceptos técnicos de OA que los clientes suelen desconocer, además no contemplan PS definidas a través de recursos como en RICNF.

Capítulo 8

Conclusiones y Trabajo Futuro

En el presente proyecto hemos desarrollado, partiendo de numerosos ejemplos de Requisitos de Impacto de Cualidades No Funcionales, un Modelo de Impacto de Cualidades No Funcionales, de modo tal que el mapeo de los requisitos en lenguaje natural a modelos en MICNF sea lo más directo posible. Al estar enmarcada dentro de las Metodologías Orientadas a Aspectos, y gracias a que los intereses identificados en la etapa de requisitos son mantenidos a lo largo de las distintas etapas, se obtiene una adecuada separación de intereses, ajustándonos de este modo al espíritu de las metodologías OA.

Debido al enfoque empleado, nuestra metodología puede adaptarse a sistemas de información Web 1.0, tanto *thin client* como *fat client*. Esto ocurre principalmente por el tipo de Parcelas de Sistema que hemos contemplado, así como también contribuye a ello la forma en la que se describen los Casos de Uso en el DTD. El hecho de que los Modelos ICNF se enfoquen en cómo se debe impactar en las componentes funcionales del sistema, en donde las Unidades de Requisitos Funcionales que hemos considerado son Casos de Uso, no hace sino reforzar esta situación.

Finalmente, el hecho de que la notación misma de los MICNF resulte lo suficiente abstracta como para poder ser presentada al cliente, pero lo suficientemente concreta a la vez como para poder llevar adelante un proceso deductivo sobre sus elementos y generar así nuevos modelos, es un testimonio más de que la flexibilidad buscada para nuestros modelos es un objetivo que hemos alcanzado con creces.

Dado que el trabajo aquí presentado abarca distintas áreas de la Ingeniería del Software, las posibilidades de refinar los conceptos aquí desarrollados en futuros trabajos son múltiples. Tal vez el punto de extensión más interesante a explorar sobre esta metodología sea la posibilidad de expandirla a Familias de Aplicaciones y/o a RIAs. El esfuerzo necesario para ello no es menor, dado que a lo largo de todo nuestro desarrollo se han asumido ciertas características exclusivas de la Web 1.0 que, en otros contextos, resultan inválidos (ver, a modo de ejemplo, el análisis del Criterio 1.2 de Clasificación de NFI de la página 29). Esto extendería la metodología actual a nuevos campos de aplicación, excediendo incluso los límites de la Web.

Un segundo punto a considerar es la integración del trabajo realizado con otras etapas del proceso de desarrollo MDA. Todo el trabajo que hemos realizado aquí se encuadra dentro de lo que MDA define como Modelo Independiente de Plataforma (PIM), pero resta aún integrar las transformaciones de modelos en las restantes etapas MDA. Así como la metodología AOWE fue el punto de partida para el desarrollo de este trabajo, la incorporación de Transformaciones de Modelos en otras etapas de esta metodología abre un nuevo campo de estudio a considerar. Dentro de la misma etapa del proceso MDA, es importante notar que hemos estudiado el mapeo de modelos ICNF a modelos OA más bien sencillos, como lo son los Modelos de CU unificados y las Tablas de Composición. El mapeo de modelos ICNF a modelos OA más complicados, como ser aquellos que involucran *pointcuts*, parece ser en principio posible, por lo que verificar esta teoría resulta interesante.

Desde un punto de vista relacionado estrictamente con la metodología introducida, existen ciertas variaciones posibles a la misma cuyos efectos sería interesante explorar. La primera y la

más directa (más no por ello la más sencilla) consiste en introducir chequeos extra con respecto a los modelos involucrados en las transformaciones. De esta forma, ciertos chequeos de consistencia (tales como asegurar que las operacionalizaciones de un SIG, sus tópicos, los elementos de un MICNF y los Casos de Uso estén todos relacionados entre sí) que son en nuestra metodología responsabilidad del analista podrían, quizás, ser delegados (al menos parcialmente) a la implementación del proceso.

Una segunda alternativa, menos directa, consiste en limitar la descomposición del SIG a un Catálogo de Tipos (un concepto que no se ha tratado aquí, pero que ha sido tratado en detalle en [9]), y luego utilizar las PS como si de tópicos se tratase. La ventaja de esta alternativa radicaría, entonces, en que limitaría los puntos para inconsistencias (al eliminar la descomposición en tópicos de un SIG, esto remueve la posibilidad de inconsistencias entre el Framework NFR y los MICNF). Por supuesto, la implementación de esta alternativa requiere de un análisis más exhaustivo, por lo que será una tarea para futuros investigadores.

Finalmente, en lo que respecta a la implementación de las ideas introducidas en esta metodología, el hecho de que el trabajo haya sido realizado con un enfoque de prueba de concepto (es decir, probando que los conceptos presentados son viables), hay un amplio lugar para mejoras. Tal vez la más sobresaliente de estas posibilidades sea la de sobrepasar el framework Eclipse, incorporando todo el proceso en una herramienta integrada de software.

Bibliografía

[1] Akoumianakis, D., Katsis, A., and Vidakis, N. Non-functional user interface requirements notation (nfrn) for modeling the global execution context of tasks. In *Task Models and Diagrams for Users Interface Design*, K. Coninx, K. Luyten, and K. Schneider, Eds., vol. 4385 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2007, pp. 259–274.

[2] Alencar, F., Silva, C., Moreira, A., Araújo, J., and Castro, J. Identifying candidate aspects with i-star approach. In *Workshop Proceedings on Early Aspects at AOSD* (2006), pp. 4–10.

[3] Allaire, J. Macromedia flash mx - a next-generation rich client. Tech. rep., Adobe, March 2002.

[4] Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., and Yergeau, F. Extensible markup language (xml) 1.0 (fifth edition). World Wide Web Consortium, Recommendation REC-xml-20081126, November 2008.

[5] Casalánguida, H., and Durán, J. E. Aowe: una metodología orientada a aspectos para desarrollo de aplicaciones web. In *8th Argentinean Symposium on Software Engineering* (April 2007), pp. 90–104.

[6] Casalánguida, H., and Durán, J. E. Aspect oriented navigation modeling for web applications based on uml. In *Latin America Transactions, IEEE* (March 2009), Revista IEEE America Latina, IEEE Region 9, pp. 92–100.

[7] Ceri, S., Fraternali, P., and Matera, M. Conceptual modeling of data-intensive web applications. *IEEE Internet Computing 6* (July 2002), 20–30.

[8] Chomsky, N. *Aspects of the Theory of Syntax*, 1st paperback ed. ed. The MIT Press, March 1969.

[9] Chung, L., Nixon, B. A., Yu, E., and Mylopoulos, J. *Non-Functional Requirements in Software Engineering*, vol. 5 of *International Series in Software Engineering*. Springer, 1999.

[10] Cockburn, A. *Writing Effective Use Cases*. Addison-Wesley, January 2000.

[11] Czarnecki, K., and Helsen, S. Feature-based survey of model transformation approaches. *IBM Syst. J. 45* (July 2006), 621–645.

[12] Dijkstra, E. W. On the role of scientific thought (EWD447). In *Selected Writings on Computing: A Personal Perspective*. Springer-Verlag, 1982, pp. 60–66.

[13] Durán, J. E., and Villalba, M. F. Ingeniería de requisitos web orientada a aspectos con transformación de modelos. In *XXXVI Conferencia Latinoamericana de Informática* (Paraguay, October 2010), Centro Latinoamericano de Estudios en Informática.

[14] Haak, B., Díaz, M., Marcos, C., and Prior, J. Aspects extractor: Identificación de aspectos en la ingeniería de requerimientos. In *9no Workshop Iberoamericano de Ingeniería de Requisitos y Ambientes de Software* (2006), IDEAS 06.

[15] ITU. Telecommunication standardization sector draft specification of the goal-oriented requirement language. Tech. rep., ITU, September 2001.

[16] Jacobson, I., and Ng, P.-W. *Aspect-Oriented Software Development with Use Cases (Addison-Wesley Object Technology Series).* Addison-Wesley Professional, 2004.

[17] Jouault, F., Allilaire, F., Bézivin, J., and Kurtev, I. Atl: a model transformation tool. *Science of computer programming 72*, 1-2 (June 2008), 31–39.

[18] Jouault, F., and Kurtev, I. On the architectural alignment of atl and qvt. In *Proceedings of the 2006 ACM symposium on Applied computing* (New York, April 2006), ACM Press, pp. 1188–1195.

[19] Jureta, I. J., Faulkner, S., and yves Schobbens, P. P.y.: A more expressive softgoal conceptualization for quality requirements analysis. In *Proc. of IEEE Int. Conf. on Conceptual Modelling (RE06), 281-295. M. Kassab1, M. Daneva1, O. Ormandjieva2* (2006), Springer.

[20] Kassab, M., Ormandjieva, O., and Daneva, M. An ontology based approach to non-functional requirements conceptualization. In *ICSEA* (2009), K. Boness, J. M. Fernandes, J. G. Hall, R. J. Machado, and R. Oberhauser, Eds., IEEE Computer Society, pp. 299–308.

[21] Kiczales, G. Aspect-oriented programming. In *ICSE* (St. Louis, Missouri, USA, 2005), G.-C. Roman, W. G. Griswold, and B. Nuseibeh, Eds., ICSE 2005, ACM, p. 730.

[22] Kovse, J. Generic model-to-model transformations in mda: Why and how? In *OOPSLA 2002 Workshop on Generative Techniques in the context of Model-Driven Architecture* (Seattle, WA, USA, 2002).

[23] Lamsweerde, A. V. Goal-oriented requirements engineering: A guided tour. In *Proceedings of the 5th IEEE International Symposium on Requirements Engineering* (2001), vol. 249, Toronto, Canada, p. 263.

[24] Lengyel, L. An overview of qvt. Tech. rep., Budapest University of Technology and Economics - Department of Automation and Applied Informatics, July 2002.

[25] Miles, R., and Hamilton, K. *Learning UML 2.0.* O'Reilly Media, Inc., 2006.

[26] Monteiro, C., Ramos, R., Castro, J., Araújo, J., Moreira, A., and Alencar, F. The iaspectplugin to automatize the identification of crosscutting concerns on i*. In *I Latin-American Workshop on Aspect-Oriented Software Development* (2007), SBES 2007, pp. 105–116.

[27] Moreira, A., Araújo, J., and Rashid, A. A concern-oriented requirements engineering model. In *Proc. Int'l Conference on Advanced Information Systems Engineering (CAiSE) 2005, LNCS* (2005), Springer, pp. 293–308.

[28] Niu, N., and Easterbrook, S. Discovering aspects in requirements with repertory grid. In *Proceedings of the 2006 international workshop on Early aspects at ICSE* (New York, NY, USA, 2006), EA '06, ACM, pp. 35–42.

[29] OMG. Xml metadata interchange specification version 1.2. Tech. rep., Object Management Group, 2002.

[30] OMG. Omg unified modeling language supestructure version 2.2. Tech. Rep. formal/2010-05-03, Object Management Group, 2009.

[31] OMG. Omg unified modeling language infrastructure version 2.3. Tech. Rep. formal/2010-05-03, Object Management Group, 2010.

[32] Rashid, A. Aspect-oriented requirements engineering: An introduction. *Requirements Engineering, IEEE International Conference on 0* (2008), 306–309.

[33] Rashid, A., Moreira, A., and Araújo, J. Modularisation and composition of aspectual requirements. In *Proceedings of the 2nd international conference on Aspect-oriented software development* (New York, NY, USA, 2003), AOSD '03, ACM, pp. 11–20.

[34] Robie, J. Action item on syntax-based interoperability. W3C Mailing List, October 2003.

[35] Schauerhuber, A., Wimmer, M., Kapsammer, E., Schwinger, W., and Retschitzegger, W. Bridging webml to model-driven engineering: from document type definitions to meta object facility. *IET Software 1* (June 2007), 81–97.

[36] Shaker, P., and Peters, D. K. An introduction to aspect-oriented software development. In *Proc. Newfoundland Electrical and Computer Engineering Conference, IEEE* (November 2005).

[37] Sousa, G., Soares, S., Borba, P., and Castro, J. Xvii brazilian symposium on software engineering. In *XVII Brazilian Symposium on Software Engineering* (Manaus, Brasil, 2003), pp. 177–192.

[38] Sousa, G., Soares, S., Borba, P., and Castro, J. Separation of crosscutting concerns from requirements to design: Adapting an use case driven approach. In *In Proc. Early Aspects Workshop at AOSD* (2004).

[39] Spies, E., Rüger, J., and Moreira, A. Using i* to identify candidate aspects. In *Workshop on Aspect-Oriented Modeling at UML'04* (2004).

[40] Stellman, A., and Greene, J. *Applied Software Project Management*, 1 ed. O'Reilly Media, 2005.

[41] Supakkul, S., and Chung, L. A uml profile for goal-oriented and use case-driven representation of nfrs and frs. *Software Engineering Research, Management and Applications, ACIS International Conference on 0* (2005), 112–121.

[42] Team, W. A., Dix, C., Kovack, R., Rafter, J., Hunter, D., and Pinnock, J. *Beginning XML*, 2nd ed. Wrox Press Ltd., Birmingham, UK, UK, 2001.

[43] van den Berg, K. G., Conejero, J. M., and Hernández, J. Identification of crosscutting in software design. In *8th International Workshop on Aspect-Oriented Modeling* (March 2006), O. Aldawud, T. Elrad, J. Gray, J. Kienzle, and D. Stein, Eds., vol. 8 of *Aspect-Oriented Modeling*, University of Duisburg-Essen, pp. 1–7.

[44] Werneck, V. M. B., de Padua Albuquerque Oliveira, A., and do Prado Leite, J. C. S. Comparing gore frameworks: i-star and kaos. In *Ibero-American Workshop of Engineering of Requirements* (Val Paraiso, Chile, July 2009).

[45] Yu, E. Towards modeling and reasoning support for early-phase requirements engineering. In *Proceedings of the 3rd IEEE International Symposium on Requirements Engineering* (Washington, DC, USA, 1997), RE '97, IEEE Computer Society, pp. 226–.