Una Aproximación Semicualitativa al Tratamiento Automático de Requisitos de Calidad

Aplicación a la obtención automática de acuerdos de nivel de servicio en MOWS

Departamento de Lenguajes y Sistemas Informáticos
Universidad de Sevilla

Memoria de la tesis doctoral dirigida por don Rafael Corchuelo Gil y don Miguel Toro Bonilla, y desarrollada por don Antonio Ruiz Cortés para optar al grado de Doctor en Informática por la Universidad de Sevilla.

Sevilla, Mayo de 2.002

Este trabajo ha sido financiado parcialmente por los siguientes proyectos:

MENHIR: Modelos, Entornos y Nuevas Herramientas para la Ingeniería de Requisitos. Financiado por la Comisión Interministerial de Ciencia y Tecnología (TIC97–0593–C05–03).

GEOZOCO: Nuevos métodos y herramientas para la automatización del desarrollo de aplicaciones de comercio electrónico y su integración con sistemas de información geográfica. Financiado por la Comisión Interministerial de Ciencia y Tecnología (TIC2000–1106–C02–01).

WEST: Tecnología Software Orientada a Ambientes Web. Financiado por Programa Iberoamericano de Ciencia y Tecnología para el Desarrollo. Subprograma VII.18.

WEBPLUS (Nueva generación de servicios WEB). Financiado por el Ministerio de Ciencia y Tecnología (FIT-150100-2001-78) dentro del Programa de Fomento de la Investigación Técnica (PROFIT).

Yo, Antonio Ruiz Cortés, con Documento Nacional de Identidad numero 28.478.419-A,

DECLARO BAJO JURAMENTO

ser el autor del trabajo que se presenta en la memoria de esta tesis doctoral que tiene por título

Una Aproximación Semicualitativa al Tratamiento Automático de Requisitos de Calidad. Aplicación a la obtención automática de acuerdos de nivel de servicio en MOWS.

Fdo. Antonio Ruiz Cortés
Profesor Asociado del Dpto. de
Lenguajes y Sistemas Informáticos
de la Universidad de Sevilla
Área de Lenguajes y Sistemas Informáticos

Sevilla, Mayo de 2.002

UNIVERSIDAD de SEVILLA

RECTORADO

Sevilla, 25 de Setiembre de 2002
N/Ref.: Negociado de Tesis EL/Mar
Asunto: Enviando Tesis Doctorales Leídas

SR. DIRECTOR DE LA BIBLIOTECA DE
FACULTAD DE INFORMÁTICA Y
ESTADÍSTICA
UNIVERSIDAD DE SEVILLA

Adjunto le remito ejemplares de Tesis Doctorales leídas en Departamentos vinculados a esa Facultad a fin de que pasen a formar parte de fondos bibliográficos de consulta de ese Centro.

AUTORES DE LAS TESIS LEÍDAS

1. RUÍZ CORTÉS, ANTONIO
2. ORTEGA ALVARADO, LIDIA MARÍA

LA JEFA DE SECCIÓN DE DOCTORADO

Fdo.: Yolanda Díaz Rolando

Don Rafael Corchuelo y don Miguel Toro, profesores del Área de Lenguajes y Sistemas Informáticos,

HACEN CONSTAR

que don Antonio Ruiz Cortés, Licenciado en Informática por la Universidad de Sevilla, ha realizado bajo nuestra supervisión el trabajo de investigación titulado

Una Aproximación Semicualitativa al Tratamiento Automático de Requisitos de Calidad. Aplicación a la obtención automática de acuerdos de nivel de servicio en MOWS.

Una vez revisado, autorizamos el comienzo de los trámites para su presentación como Tesis Doctoral al tribunal que ha de juzgarlo.

Fdo. Rafael Corchuelo y Miguel Toro
Área de Lenguajes y Sistemas Informáticos

Sevilla, Mayo de 2.002

A Isabel, la persona que más desea el final de este trabajo, por su apoyo y, sobre todo, por su eterna paciencia.

A Miguel y Antonio, esos dos locos bajitos de los que tantas horas de compañía me ha privado esta tesis.

Prólogo

Ha de considerarse que no hay cosa más difícil de emprender, ni de resultado más dudoso, ni de más arriesgado manejo que ser el primero en introducir nuevas disposiciones, porque el introductor tiene por enemigos a todos los que se benefician de las instituciones viejas, y por tibios defensores, a todos aquéllos se beneficiarían de las nuevas; tibieza que procede en parte de la incredulidad de los hombres, quienes no creen en ninguna cosa nueva hasta que la ratifica una experiencia firme.

NICOLÁS MAQUIAVELO, EL PRÍNCIPE

Conoce el lector el referido como "síndrome de la farola"? B. Boehm lo presenta de manera muy amena en el prólogo de [Gilb 1988]:

> En una noche muy oscura, una persona un tanto bebida busca su cartera alrededor de una farola. Al cabo del tiempo, un transeúnte decide ayudarle. Después de un rato de infructuosa búsqueda el transeúnte pregunta: ¿aquí no hay rastro de la cartera, está seguro que se le cayó aquí? El borracho contesta: No. Se me cayó allí, más abajo, pero allí no hay luz.

Es evidente que todos padecemos este síndrome en mayor o menor medida. Existen zonas que podríamos calificar de bien iluminadas que reciben la atención de la mayoría de investigadores y en las que resulta más cómodo y rentable investigar. Existen otras zonas oscuras en las que pocos se adentran porque las expectativas de encontrar resultados que sean valorados por los investigadores de las zonas iluminadas son mínimas.

En nuestra opinión, la investigación en el contexto de una tesis doctoral en Ingeniería del Software debería llevarse a cabo en las numerosas

zonas oscuras, o cuando menos de penumbra, que existen en este área de conocimiento.

Evidentemente, definir el alcance y los objetivos concretos de una tesis doctoral no son tareas fáciles y muy especialmente en disciplinas inmaduras como es el caso de la Ingeniería del Software que ni siquiera tiene definido su *cuerpo de conocimiento* [Bourque *et al.* 1999]. Reflejo de esta situación son los numerosos debates sobre la orientación de la investigación en la Ingeniería del Software en general [Botella 2001], sobre lo que debe ser una tesis doctoral en Informática [Díaz 1999] y sobre los planes de estudio de Ingeniería Informática [Parnas 1999].

No es nuestra intención disertar sobre estos temas tan cruciales para el futuro de la disciplina en este prólogo, sí en cambio, deseamos hacer explícito el enfoque seguido desde el principio de este trabajo de investigación y que tan determinante ha resultado a la postre.

En primer lugar, aceptando la idea generalizada de que "los científicos investigan por *saber* y los ingenieros investigan por *resolver*", consideramos que el sesgo que debía tomar esta tesis doctoral que esta centrada en una disciplina con una clara orientación científico–técnica era un camino intermedio entre la investigación básica y la aplicada.

En segundo lugar, coincidimos con [Díaz 1999] en que toda tesis doctoral debe coincidir lo máximo posible con los perfiles formativos planificados en los diferentes planes de estudio. Así la tesis, se convierte en un medio para conseguir mejor docencia, evitando la maniquea disputa entre docencia versus investigación.

En tercer lugar, y coincidiendo de nuevo con [Díaz 1999], consideramos que la tesis no es un fin en sí misma, sino el medio para lograr unos objetivos superiores, de entre los que destaca el inculcar en el investigador el afán por aventurarse en caminos poco trillados y de adquirir amplios conocimientos sobre la disciplina.

Estas tres han sido las directrices que han guiado nuestro trabajo. Lo iniciamos en un terreno muy poco explorado: el prototipado arquitectónico de sistemas abiertos y distribuidos. De los muchos problemas identificados nos decantamos por los relacionados con la especificación y análisis de requisitos de calidad, convencidos de que los resultados de esta investigación tendrían aplicación en nuestra industria local a la vez que una positiva repercusión en la docencia.

Resumen

Hasta la fecha, el tratamiento sistemático de los requisitos de calidad ha sido investigado desde dos perspectivas opuestas: la cuantitativa y la cualitativa.

El principal inconveniente de los métodos cuantitativos es su dificultad para capturar la naturaleza subjetiva y flexible de los requisitos de calidad. Por su parte, el principal inconveniente de los métodos cualitativos es que hasta la fecha no se han definido técnicas de evaluación objetivas.

Por otra parte, existen aspectos para los que ninguno de estos enfoques resulta del todo satisfactorio, entre otros, la posibilidad de añadir consciencia temporal a los requisitos, definir cláusulas de negociación y demostrar automáticamente propiedades tales como la consistencia, la optimalidad y la conformidad.

En esta tesis, proponemos una tercera perspectiva que toma los mejores elementos de los dos enfoques tradicionales pero no sus limitaciones. Esta perspectiva la hemos denominado *semicualitativa* y se concreta en en:

- Una técnica para describir documentos de requisitos de calidad con rigor y precisión.
- Un marco formal de trabajo que permite demostrar propiedades sobre estos documentos.
- Un ejemplo de aplicación: la obtención automática de acuerdos de nivel de servicio en sistemas multiorganizacionales basados en servicios WEB.

Índice general

Índice de figuras

Índice de cuadros

Parte I

Introducción

Capítulo 1

Introducción

En su propio y auténtico sentido, ciencia es sólo investigación: plantearse problemas, trabajar en resolverlos y llegar a una solución [...] Investigar es descubrir una verdad o su inverso: demostrar un error.

ORTEGA Y GASSET, MISIÓN DE LA UNIVERSIDAD

El objetivo científico de una tesis doctoral no es otro que proporcionar contribuciones originales al conocimiento científico y técnico. En este primer capítulo trazaremos a grandes rasgos el camino que hemos seguido hasta obtener las contribuciones de esta tesis.

Comenzaremos comentando cómo se despertó nuestra curiosidad por investigar sobre la automatización del tratamiento de los requisitos de calidad. A continuación, ofreceremos una visión muy sintética de algunas tendencias relacionadas con este problema a fin de contextualizarlo, elaborar las hipótesis de partida y definir los objetivos de esta tesis. Finalmente, describiremos el contexto en el que hemos desarrollado este trabajo y comentaremos la estructura de esta memoria.

1.1. Motivación

Es comúnmente aceptado que los requisitos de calidad han recibido poca atención por parte de los investigadores en ingeniería del software y son muchas las razones que se han esgrimido para explicar este hecho. En nuestra opinión, la heterogeneidad junto con la la naturaleza subjetiva y relativa de los requisitos de calidad son los factores claves del origen y mantenimiento de esta situación.

Como no podía ser de otra manera, esta situación tiene su reflejo en el desarrollo de software. La figura §1.1, inspirada en las gráficas propuestas en [Osmond 1995][1], ilustra de manera muy plástica el resultado de esta situación.

La curva inferior ilustra el camino seguido con mayor frecuencia durante el desarrollo de un proyecto software. La principal preocupación de los responsables del proyecto es satisfacer tan rápido como sea posible los requisitos funcionales del sistema, a fin de que el cliente y los usuarios puedan dar su visto bueno lo antes posible. Durante la siempre difícil y estresante carrera por finalizar el proyecto, es frecuente dejar de lado algunos requisitos de calidad con la esperanza o la equivocada convicción, de que serán añadidos fácilmente una vez lograda la satisfacción de los requisitos funcionales. Por regla general, si se elige este camino, gran parte de los requisitos de calidad especificados inicialmente quedarán sin satisfacer, al menos dentro del plazo y presupuesto previstos.

Osmond mantiene (curva superior de la figura) que el uso de técnicas de desarrollo orientadas a objetos ayuda a mantener constante el nivel de calidad a lo largo del proyecto en todos sus aspectos incluyendo la funcionalidad. Un objetivo que persigue con esta estrategia es evitar que se dejen de satisfacer requisitos tan importantes como la facilidad de uso, fiabilidad o la facilidad de extensión a cambio de ofrecer una mayor funcionalidad, pues muy probablemente el sistema no será utilizado por ser poco fiable o difícil de usar.

Existen diversos trabajos que describen proyectos reales que han fracaso por un descuidado tratamiento de los requisitos de calidad. En [Breitman *et al.* 1999], los autores presentan un informe con el que justifican que algunos de los errores cometidos durante el desarrollo del sistema de gestión de avisos del servicio de ambulancias de Londres podrían haber

[1] Recogidas posteriormente en [Meyer 1997, p.12].

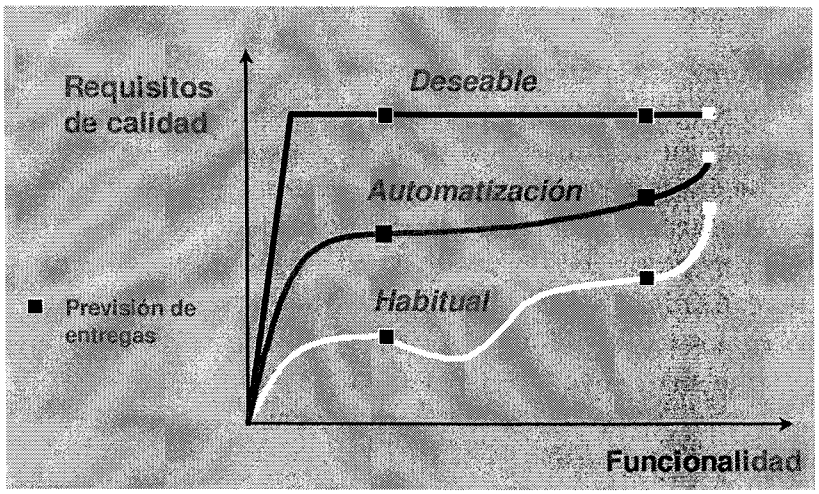

Figura 1.1: *Caminos posibles durante el desarrollo de un proyecto software.*

sido evitados o al menos aminorado sus efectos, de haber seguido algunas técnicas de análisis de requisitos y especialmente para los requisitos de calidad. Dicho sistema pretendía sustituir los procedimientos manuales de recepción de llamadas y posterior asignación de la ambulancia más cercana. Su puesta en marcha en 1992 fue desastrosa: ambulancias que llegaban a casa del paciente al mismo tiempo que la funeraria o pacientes que tras varias horas de espera decidían ir al hospital por sus propios medios.

El sistema era funcionalmente correcto, pero su fiabilidad y eficiencia eran pésimas. La primera y única actuación correctiva sobre el sistema fue dejar algunas de sus funciones bajo control manual. El resultado fue igualmente malo: el control manual se colapsó en 8 días. No obstante, al haber dejado accesible el informe de la investigación realizada al sistema [Finkelstein 1993] ha beneficiado la investigación posterior en este campo [Nuseibeh *et al.* 1994, Kramer y Wolf 1996, Finkelstein y Dowell 1996].

¿Cómo es posible que en los años 90 y en Inglaterra, un proyecto tan importante pudiera fracasar tan estrepitosamente? La respuesta ha dado lugar a muchos debates en diversas conferencias de ingeniería del software. En cualquier caso, es una muestra suficientemente significativa de la necesidad de profundizar en el conocimiento de los requisitos de cali-

dad y de mejorar las actuales técnicas de especificación, herramientas de desarrollo y entornos de ejecución para que sean "sensibles a la calidad"[2].

Por otra parte, el tratamiento automático de requisitos de calidad cobra cada día más importancia, pues son muchos los beneficios que de él se esperan. Una de las áreas más en auge es la relacionada con la elección de productos software (*software procurement*) [Finkelstein *et al.* 1996], donde son necesarias técnicas de especificación precisas y mecanismos de evaluación y comprobación de la satisfacción de requisitos. Algunas de las propuestas realizadas hasta la fecha han particularizado este problema en el contexto de los COTS [Dong *et al.* 1999, Iribarne y Vallecillo 2000], de los ERPs [Finkelstein *et al.* 1996, Franch 1998, Burgués *et al.* 2000], sitios WEB [Olsina *et al.* 1999, Olsina 1999] y servicios WEB [Ruiz-Cortés *et al.* 2000b, Ruiz-Cortés *et al.* 2001a].

Convencidos de que era posible mejorar el tratamiento que reciben actualmente los requisitos de calidad y de que éramos capaces de hacerlo nos decidimos a investigar sobre este tema. La aplicación directa de algunos de los primeros resultados sobre esta investigación en el terreno industrial y las expectativas de utilizarlos para mejorar el contenido de algunas asignaturas fueron suficiente acicate para iniciar este trabajo de tesis doctoral.

De manera muy general, podemos decir que el objetivo de esta tesis es aportar nuestro particular granito de arena a la disciplina, que se concreta en el desarrollo de una técnica de descripción formal que permite automatizar algunas de las actividades relacionadas con el tratamiento de los requisitos de calidad. Cumpliendo este objetivo esperamos reducir la distancia existente entre la actual forma de desarrollar software y la forma ideal (curva intermedia de la figura §1.1).

1.2. Tendencias en áreas relacionadas

La investigación sobre los requisitos de calidad no es patrimonio de los ingenieros de requisitos, también es un tema de interés para los arquitectos y diseñadores, los desarrolladores de *middlewares* y los responsables de la compra de componentes OTS (*Off–The–Self*) o del alquiler de servicios WEB, entre otros.

[2]Expresión tomada de [Frølund y Koistinen 1999] para referirse a las plataformas que garantizan la satisfacción de requisitos de calidad de servicio.

En esta sección, describimos algunas de las tendencias más significativas en relación a los objetivos de esta tesis de cada una de estas áreas. Se trata de una descripción general, quedando la descripción detallada para el capítulo §3 y para algunas otras secciones en las que se realizan comparativas de las técnicas desarrolladas en esta tesis con otras similares.

1.2.1. Ingeniería de requisitos

1.2.1.1. Requisitos de calidad

Los requisitos de calidad han sido y siguen siendo, probablemente por su heterogeneidad, los menos estudiados en ingeniería de requisitos, a pesar de que se empezó a hablar de ellos hace mucho tiempo. Hasta la fecha, la referencia más antigua que hemos encontrado que los aborda es [Yeh *et al.* 1984] en el contexto de la ingeniería de requisitos y [McCall *et al.* 1977] en un contexto más general.

Prueba de la insuficiente investigación realizada hasta la fecha es la falta de consenso, aún hoy, para referirse a este tipo de requisitos. En [Keller *et al.* 1990, Boehm y In 1996, Liu 1998] son denominados *requisitos de calidad*, en [L. Bass y Kazman 1998] *atributos de calidad*, en [Franch y Botella 1998, Robertson y Robertson 1999, Chung *et al.* 2000, Cysneiros *et al.* 2001] *requisitos no funcionales*, en [Roman 1985] *restricciones*, en [Mostow 1985] *objetivos* y en [Davis 1993] *non–behavioural requirements*.

Sin duda alguna, la expresión más utilizada es la de *requisito no funcional*, que en nuestra opinión no refleja de la manera más adecuada el significado del concepto que intentan referenciar, al menos en todos los escenarios posibles. En este sentido, estamos de acuerdo con la opinión expresada en [L. Bass y Kazman 1998, p. 76], sobre lo inadecuado del calificativo *no funcional* para referirse a todo aquello que no sea un requisito funcional. Por ejemplo, por regla general, la confidencialidad se entendería como un requisito no funcional, sin embargo, en un sistema de comercio electrónico se entendería como un requisito funcional, pues para satisfacer dicho requisito es necesario utilizar módulos de encriptación cuya funcionalidad debe ser definida con precisión.

En nuestra opinión, la expresión que mejor refleja el significado que para nosotros tiene un requisito de calidad es la utilizada en [Boehm y In 1996], donde son denominados *quality–attributes requirements* o simple-

mente *quality requirements*, refiriéndose a una condición que se establece sobre un atributo de calidad.

Requisitos blandos y duros

En [Liu 1998] se clasifican los requisitos de calidad en requisitos blandos (*soft*) y requisitos duros (*hard*). Un requisito se dice que es duro, si tiene que ser satisfecho completamente. Un requisito se dice que es blando, si puede ser satisfecho en un cierto grado. La propiedad que caracteriza a los requisitos blandos la denomina *elasticidad*.

Elasticidad

1.2.1.2. Tratamiento sistemático de requisitos de calidad

La investigación sobre el tratamiento sistemático de los requisitos de calidad puede ser clasificada atendiendo a dos criterios: su *objetivo* y la *técnica* empleada. Según el objetivo se distingue entre propuestas *orientadas al producto* y propuestas *orientadas al proceso* [Chung *et al.* 2000]. Las primeras se centran fundamentalmente en la descripción, evaluación y certificación de la calidad de un producto software [Keller *et al.* 1990, Finkelstein *et al.* 1996, Franch 1998, Frølund y Koistinen 1998b]. Las segundas, en cambio, se centran en el definición de métodos de desarrollo dirigidos por los requisitos de calidad [Mylopoulos *et al.* 1992, Boehm y In 1996, Chung *et al.* 2000].

Según la técnica empleada, se distingue entre propuestas *cuantitativas* y propuestas *cualitativas* [Liu 1998, Chung *et al.* 2000]. En las propuestas cuantitativas, un requisito de calidad se expresa como una condición o conjunto de condiciones sobre atributos de calidad. Por ejemplo, un requisito sobre la fiabilidad podría ser: "el tiempo entre fallos deberá ser superior a 15 días". Las propuestas cuantitativas suelen emplear la lógica de primer orden para expresar estas condiciones.

Por su parte, en las propuestas cualitativas, los requisitos se expresan haciendo uso de técnicas procedentes del razonamiento cualitativo. Por ejemplo, un requisito sobre la fiabilidad puede ser expresado como: "el tiempo entre fallos deberá ser alto".

Con frecuencia, las propuestas orientadas al producto son abordadas con técnicas cuantitativas y las orientadas al proceso con técnicas cualitativas, aunque también son posibles otras combinaciones [Chung *et al.* 2000].

1.2.1.3. Limitaciones de las actuales propuestas

Dificultad para capturar la elasticidad

En [Liu 1998] se discute sobre las limitaciones de la lógica de predicados para capturar la elasticidad de algunos requisitos de calidad. También afirma que esta limitación la sufren todas las propuestas cuantitativas pues todas usan la lógica de predicados como formalismo base. Incluso algunas cualitativas como es el caso de [Mylopoulos *et al.* 1992] también lo sufren, pues su mecanismo de razonamiento está basado en la lógica de predicados. Liu propone el uso de la lógica difusa como formalismo base para especificar requisitos ya que permite capturar fácilmente la elasticidad.

Detección y resolución de conflictos muy limitada

La negociación y resolución de conflictos entre requisitos de calidad, sobre todo, cuando participan varios participantes, es una de las áreas en las que más se está investigando. El interés se centra en lograr la máxima automatización. Uno de los proyectos de investigación que más resultados ha obtenido, y sigue obteniendo, sobre este tema, es el proyecto *Win–Win* (sección §3.3.4). Entre las aportaciones más importantes destacamos la técnica para semiautomatizar la detección de conflictos [Boehm y In 1996] y la técnica para sistematizar (que no automatizar) la resolución de conflictos [In *et al.* 2002]. No obstante, al igual que la mayoría de las propuestas, Win–Win también tiene su base en la lógica de primer orden [Lee 1996], por lo que tampoco captura la elasticidad de los requisitos.

Valoraciones de atributos poco expresivas

Los métodos cuantitativos utilizan métricas para medir o valorar atributos básicos de calidad y utilizan funciones de agregación para valorar atributos derivados (su valor depende de otros atributos). En ambos casos, la valoración se expresa con un único valor lo que supone una reducción importante de la información sobre el valor original del atributo. Por ejemplo, considerando que el atributo derivado RELIABILITY que representa a la fiabilidad de un servicio WEB se define a partir del tiempo entre fallos (TTF), el tiempo de recuperación (TTR) y que la reconexión tras un fallo sea o no sea transparente al usuario (REBIND), indicar que la fiabilidad es del 90 % ofrece mucha menos información que si decimos que el TTF es de 10 días, el TTR de 1 día y que la reconexión es transparente al usuario.

Además, si bien la definición de métricas para algunos atributos puede ser una tarea fácil, en general es una tarea difícil y laboriosa, sobre todo en lo referente a su proceso de validación. En [Olsina 1999] se propone una metodología cuantitativa para evaluar sitios WEB. Una de las actividades de dicha metodología es precisamente la definición de un marco de trabajo para validar métricas.

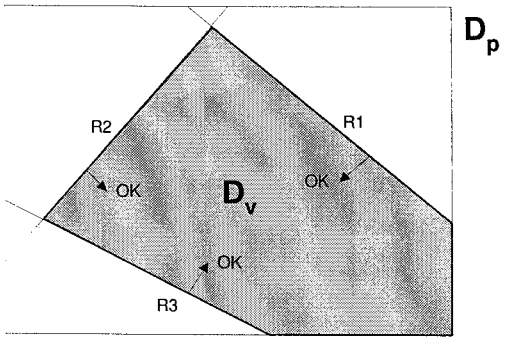

Figura 1.2: *Los requisitos de calidad como restricciones en el espacio de alternativas de diseño.*

1.2.2. Diseño arquitectónico

La satisfactibilidad de los requisitos definidos sobre atributos de calidad de alto nivel, también conocidos como *system-wide properties*, tales como la fiabilidad, disponibilidad, facilidad de mantenimiento, seguridad, facilidad de adaptación, interoperabilidad, escalabilidad, etcétera, son cruciales durante la explotación y mantenimiento de grandes sistemas.

En [Chung y Yu 1998] se discute sobre esta cuestión y se justifica la necesidad de que el diseño arquitectónico esté guiado por los requisitos de calidad, los cuales se interpretan como restricciones que determinan el espacio de alternativas arquitectónicas válidas (ver figura §1.2). Esta interpretación se corresponde con la que se hace en ingeniería de requisitos sobre requisitos en general [Durán 2000, Davis 1990].

Si D_p es el espacio de toda las posibles alternativas arquitectónicas para un sistema, cada requisito de calidad R_i establece una partición formada por D_i y $\bar{D}_i$, donde D_i es el conjunto de alternativas que satisfacen R_i y $\bar{D}_i$ es el conjunto de alternativas que no lo satisfacen.

En esta situación, D_v, el conjunto de las alternativas válidas, es decir, aquéllas que satisfacen todos los requisitos, sería la intersección de todos los conjuntos D_i. En otras palabras: $D_v = D_1 \cap D_2 \cap \ldots \cap D_n$

1.2.2.1. Problemas abiertos

En [Clements y Northrop 1996, L. Bass y Kazman 1998, Medvidovic y Taylor 2000] se muestran diferentes clasificaciones de los esfuerzos de investigación en el campo de las arquitecturas *software*. Tomando estas referencias como punto de partida, podemos clasificar los problemas relacionados con los requisitos de calidad en:

1. **De creación**. ¿Cómo obtener sistemáticamente la arquitectura de un sistema que satisface un conjunto de requisitos de calidad?

   El diseño arquitectónico es un proceso en el que predomina la intuición y las soluciones *ad hoc* sobre las soluciones sistematizadas, y donde el conocimiento y la experiencia de los diseñadores experimentados sigue sin ser transmitido de manera efectiva.

   Cuantificar el impacto de aplicar un patrón en el grado de satisfacción de los requisitos de calidad

   Los patrones de diseño [Gamma 1995] han supuesto un gran avance en el diseño de microarquitectura, pues su correcto uso dota al diseño de propiedades tales como la facilidad de extensión, facilidad de mantenimiento, etcétera. No obstante, el uso de patrones para diseñar arquitecturas [Buschmann *et al.* 1996] se enfrenta a un importante desafío: cuantificar el impacto que sobre las propiedades de calidad provoca la aplicación de un cierto patrón. Por ejemplo, aplicar el patrón "niveles" aumenta la interoperabilidad y la facilidad de modificación pero también reduce el rendimiento y aumenta el tamaño del producto. Mientras no sepamos cuantificar en qué medida la aplicación de un patrón aumenta o disminuye el grado de satisfacción de ciertas SWP, no podremos considerar segura su aplicación.

   En cuanto a la posibilidad de disponer de lenguajes de descripción de arquitecturas (LDAs) ejecutables, esto es, que generen código, en [Shaw y Garlan 1996, cap.7] se identifican dos estrategias: i) extender los actuales lenguajes de programación, que es el enfoque utilizado en [Bosch 1998] para generar automáticamente el código de "fragmentos arquitectónicos" y ii) extender los actuales LDAs para que generen código, que es el enfoque seguido en [Canal *et al.* 1999, Peña *et al.* 2000a].

2. **De representación** ¿Cómo comunicar una arquitectura con precisión, sin ambigüedades, de manera cómoda y abarcando todos sus aspectos?

No existe consenso en cuanto a la las vistas necesarias para describir la arquitectura software de un sistema, ni tampoco en cuanto al contenido de cada una. No obstante, se empieza a estar de acuerdo en la necesidad de incorporar información sobre los requisitos de calidad.

En [Medvidovic y Taylor 2000] se presenta una revisión muy completa de los actuales LDAs. Una de sus conclusiones señala la ausencia de un modelo de representación para los requisitos de calidad como una de sus limitaciones más importantes. En este sentido, los actuales LDAs deben considerarse incompletos.

En cuanto a los métodos de diseño orientados a objetos [Coleman *et al.* 1994, D. D'Souza and A. Wills 1999, Booch *et al.* 1999] la situación es similar ya que los requisitos de calidad no son elementos de primera clase dentro del modelo.

3. **De análisis**: ¿Cómo detectar y resolver conflictos entre requisitos?

   No resulta fácil diseñar una arquitectura que satisfaga todos los requisitos a la primera, sobre todo, si el conjunto de requisitos es grande y contiene algunos conflictivos: facilidad de extensión y rendimiento, rendimiento y portabilidad, disponibilidad y economía, etc. Detectar estos conflictos y proponer soluciones es una actividad para la que existen pocas propuestas y con muchas limitaciones.

   En [Bosch y Molin 1999] se dan las primeras ideas de una técnica para resolver conflictos entre requisitos. Esta técnica propone el uso de *transformaciones arquitectónicas* que modifican el grado de satisfacción de ciertos requisitos de calidad pero que mantienen invariable la funcionalidad.

4. **De evaluación**: ¿Cómo evaluar el grado de cumplimiento de los requisitos de calidad?

   En [Bosch y Molin 1999] se proponen cuatro técnicas para conseguirlo: la utilización de escenarios, la simulación, el modelado matemático y el razonamiento heurístico. Las técnicas más adecuadas para la evaluación de los llamados requisitos *operacionales*, es decir, aquellos que son visibles en tiempo de ejecución [L. Bass y Kazman 1998, Bosch y Molin 1999]: rendimiento, fiabilidad, robustez, tolerancia a fallos, etcétera, son las que utilizan técnicas de simulación. Por su parte, para los requisitos *no operacionales, no visibles en tiempo de ejecución o de desarrollo*: facilidad de modificación, grado de reutilización, etcétera, las técnicas más utilizadas son las basadas en escenarios [Kruchten 1995, Bosch y Molin 1999].

5. **De selección.** ¿Cómo elegir la mejor alternativa de diseño?

   Dada una especificación de requisitos, pueden existir varias alternativas de diseño que permitan construir un sistema que satisfaga dicha especificación. Cada alternativa presenta unas ventajas e inconvenientes diferentes que determinan el grado de adecuación a un problema concreto. Gran parte de estas ventajas e inconvenientes pueden ser expresadas sobre atributos de calidad por lo que disponer de una especificación precisa de los requisitos que satisface cada alternativa puede resultar muy útil.

1.2.3. Programación orientada a componentes

Desarrollar software de más calidad y en menos tiempo son los objetivos generales de cualquier paradigma. En el caso de la COP (*Component Oriented Programming*) estos objetivos se intentan lograr basando el desarrollo de aplicaciones en la composición de componentes COTS (*Commercial off–the–self*) [Szyperski 1998].

No obstante, todavía son varios los problemas que se necesitan resolver para que la COP logre desarrollarse en los términos en los que ha sido definida. Algunos de los más importantes relacionados con los atributos y requisitos de calidad son:

1. En la elección de COTS, los atributos de calidad tienen un papel muy importante. Cuando el número de atributos a valorar es muy elevado, el proceso de elección consume mucho tiempo y resulta muy laborioso. Si se desea realizar una elección automática es necesario especificar con precisión los requisitos exigidos a un componente y la calidad que éste proporciona [Iribarne y Vallecillo 2000].

2. La especificación de las interfaces de los COTS no suele ofrecer información sobre los requisitos de calidad que debe satisfacer el componente que proporciona el servicio y el componente que lo solicita (sección §7.1.2.2).

3. Las plataformas de ejecución deben proporcionar mecanismos para comprobar la validez de los contratos de los componentes y seleccionar el componente que implementa una interfaz satisfaciendo de manera óptima unos criterios determinados [Corchuelo *et al.* 2001].

Las posibilidades de las actuales plataformas de ejecución son muy reducidas en este sentido (sección §3.3.1).

4. Al tratarse de entidades binarias cuyo código no suele estar disponible, no podemos verificar su corrección, por lo que no es posible garantizar la corrección de ningún sistema que los incorpora [Voas 1998, Ruiz-Cortés *et al.* 2000b].

1.2.4. Programación orientada a servicios WEB

El éxito de INTERNET como plataforma de ejecución de aplicaciones distribuidas ha propiciado el nacimiento de una subindustria dedicada al desarrollo y explotación de **servicios WEB**, considerados por algunos como el siguiente paso en la evolución de la WWW [UDDI.ORG 2000, Microsoft 2001, Sun Microsystems, Inc 2001].

Desde el punto de vista del usuario, un servicio WEB es una entidad física autocontenida que implementa una o varias interfaces, contenida y administrada por un servidor WEB, y que está accesible mediante protocolos estándar de INTERNET. De este modo, la principal diferencia de un servicio WEB respecto a un componente es que para comunicarse con un servicio WEB basta con hablar el mismo protocolo independientemente del lenguaje, plataforma de ejecución o *middleware* utilizado.

El aumento de la demanda y complejidad de aplicaciones WEB, la presión por reducir los costes de desarrollo y explotación y las nuevas posibilidades que ofrecen los servicios WEB, son, a grandes rasgos, las principales razones que han motivado la propuesta de un nuevo paradigma de programación: la programación orientada a servicios WEB, o de manera más abreviada: WOP (*Web–Oriented Programming*) [Corchuelo *et al.* 2001].

La WOP ofrece respuestas a los principales problemas del desarrollo de sistemas multiorganizacionales basados en la WEB, o más abreviadamente MOWS (*Multi–Organisational Web–based Systems*) [Ruiz-Cortés *et al.* 2000b], que es como se conoce a aquellos sistemas cuya funcionalidad se obtiene subcontratando servicios WEB proporcionados por múltiples organizaciones y en los que el uso de los servicios del usuario final vienen regulados por acuerdos de nivel de servicio o SLAS *(Service Level Agreements)*.

En [Corchuelo *et al.* 2001] se identifican algunas utilidades que debería proporcionar las plataformas de ejecución de MOWS, a saber:

- **Consistencia**. La capacidad de determinar la ausencia de contradicciones en un conjunto de requisitos de calidad.
- **Conformidad**. La capacidad de determinar si los requisitos de calidad que satisface un servicio WEB satisfacen los requisitos de calidad exigidos por un determinado usuario.
- **Optimalidad**. La capacidad de seleccionar entre un conjunto de candidatos, el servicio WEB con el que se puede alcanzar el SLA que satisface de manera óptima unos determinados criterios.
- **Interoperabilidad**. La interoperabilidad se define como la capacidad de ofrecer la conformidad y la optimalidad cuando los requisitos que exige el usuario y los que satisface cada servicio WEB están definidos sobre diferentes catálogos de atributos.
- **Garantía**. La garantía se define como la capacidad de conocer si el uso de un servicio se ha realizado de acuerdo a las condiciones recogidas en el SLA. En el caso de haberse violado el SLA la plataforma tiene mecanismos para identificar el responsable de la violación del SLA. Esta propiedad permitirá que los proveedores de servicios facturen según la calidad que han ofrecido [Szyperski 1998, p.44].

Hasta la fecha la propuesta más avanzada que aborda la conformidad y la optimalidad es [Frølund y Koistinen 1998b, Koistinen y Seetharaman 1998] que como veremos en la sección §3.3.2, adolece de limitaciones muy importantes.

1.3. Hipótesis de trabajo

Como conclusión de la revisión resumida de algunas de las principales propuestas relacionadas con el tratamiento automático de los requisitos de calidad, planteamos la siguientes hipótesis de trabajo:

Sobre la elasticidad

Compartimos las limitaciones de las actuales propuestas para capturar la elasticidad de los requisitos de calidad que han sido identificadas en [Liu 1998]. Sin embargo, no compartimos la opinión de que la lógica difusa sea el formalismo más adecuado para dar soporte a un modelo que cubra todas las actividades relacionadas con el tratamiento automático de los requisitos de calidad.

Creemos que es posible capturar esa elasticidad de un modo más simple y con la posibilidad de añadirle consciencia temporal (sección §5.4). Nuestra propuesta es interpretar un requisito como una tupla (A, C, U, P), donde A es el conjunto de atributos sobre los que está referido el requisito, C es una restricción que representa a la condición que se establece sobre los atributos, U es un conjunto de funciones que definen la utilidad de cada atributo de A y P representa al período de tiempo en el que el requisito tiene validez.

Esta forma de modelar un requisito hace posible que varios problemas relacionados con el tratamiento automático de los requisitos pueden ser expresados como un CSP (*Constraint Satisfaction Problem*) [Marriot y Stuckey 1998], para los cuáles existen resolutores muy eficientes.

Sobre la detección y resolución de conflictos

Son pocas y muy limitadas las propuestas para automatizar la detección y la resolución de conflictos de requisitos de calidad. En el caso de Win–Win, la dificultad para lograr la automatización estriba en que las elementos básicos del modelo no disponen de un significado preciso ni de una semántica operativa. Creemos que es posible dotar de semántica a estas bases y de este modo lograr la automatización. Además, de este modo, extendemos Win–Win con posibilidades para negociar sobre los atributos de calidad y sobre el período de vigencia del requisito.

Por otra parte, hasta la fecha, no conocemos ninguna propuesta que contemple la posibilidad de que los márgenes de negociación de los participantes puedan ser conocidos total o parcialmente de antemano. Si se acepta esta posibilidad, existe la posibilidad de automatizar la resolución de conflictos, donde el grado de automatización es proporcional al conocimiento de los márgenes de negociación de los participantes.

Finalmente, hemos comprobado que algunos de los problemas existentes en la ingeniería de requisitos, en el diseño arquitectónico, en la COP y en la WOP, pueden ser definidos con un mismo marco conceptual y ser resueltos con las mismas técnicas. Es más, algunos de estos problemas pueden ser enunciados como un único CSP.

Sobre la forma de expresar atributos

La pérdida de información inherente al uso de métricas y el tiempo necesario para definirlas de tiempo necesario pueden evitarse si la valoración de los atributos se expresa como una restricción matemática.

1.4. Aportaciones

Las cosas no valen sino lo que se las hace valer.

MOLIÈRE

A partir de estas hipótesis de trabajo, hemos desarrollado un importante esfuerzo que ha dado lugar a un conjunto de ideas, técnicas y herramientas. A continuación resumimos las que consideramos mas relevantes:

- Hemos diseñado un lenguaje de especificación de requisitos que dispone de la mayor parte de características exigibles al lenguaje ideal. En adelante nos referiremos a este lenguaje como QRL (*Quality Requirements Language*), siendo sus principales características:

  1. Alto nivel de abstracción. El núcleo del lenguaje puede verse como un lenguaje ensamblador sobre el que se puede aplanar varios de los actuales QSLs.
  2. Expresividad. El uso de la programación con restricciones como base formal del lenguaje ha aumentado la capacidad expresiva del lenguaje, permitiendo la especificación de requisitos sobre múltiples atributos que puedan estar relacionados de manera no lineal y capturando la semántica que tienen los requisitos de los participantes proveedores.
  3. Consciencia temporal. Hemos extendido la actual noción de requisito con información sobre el período en el que está vigente el requisito. Esta extensión ha supuesto la extensión de las actuales nociones de consistencia, conformidad y optimalidad.
  4. Negociación. QRL ofrece soporte para detectar y resolver conflictos entre requisitos de calidad.
  5. Semicualitativo. QRL permite que las valoraciones de los atributos derivados se expresen como restricciones por lo que se evitan los problemas asociados a las métricas.
  6. Dualidad. QRL facilita la traducción de requisitos expresados en lenguaje natural estructurado a restricciones matemáticas.

7. Interoperabilidad. QRL ofrece un mecanismo básico para conseguir la interoperabilidad.
8. Viable. El lenguaje admite una compilación eficiente y dispone de muchas facilidades sintácticas, por lo que disfruta de las ventajas de los lenguajes formales, pero sin los inconvenientes clásicos de ineficiencia y dificultad de aprendizaje.

- Un marco de trabajo para comprobar automáticamente propiedades sobre documentos de requisitos de calidad. Concretamente la consistencia, la conformidad y la optimalidad.
- Hemos identificado varias propiedades que son de interés para las aplicaciones construidas en base a servicios WEB. Hemos propuesto las bases de un nuevo paradigma de programación: la WOP (*Web services Oriented Programming*). Su interés ha sobrepasado nuestra idea inicial de sistema "teórico" sobre el que tomasen cuerpo nuestras primeras soluciones, pues tiene una utilidad real y práctica a la hora de especificar, construir y explotar sistemas de comercio electrónico basados en servicios WEB [Pérez 2001, Corchuelo *et al.* 2001, Corchuelo y Ruiz-Cortés 2002].
- Hemos aplicado nuestros resultados al problema de la obtención automática de acuerdos de nivel de servicio. Problema que surge en el desarrollo y explotación de MOWS.
- Hemos desarrollado un modelo de referencia sobre el que se han valorado, comparado y organizado varias propuestas incluida la nuestra. Además hemos identificado las características que debería tener el lenguaje de requisitos de calidad ideal.

1.5. Contexto en el que se ha desarrollado este trabajo

1.5.1. Antecedentes

Nuestra tesis se inició con un problema excesivamente ambicioso, el prototipado de arquitecturas software. Empezamos investigando sobre el desarrollo de conectores para encapsular patrones de interacción entre múltiples componentes [Corchuelo *et al.* 1999]. En [Ruiz-Cortés *et al.* 1999]

presentamos la primera versión de CAL (*Coordination Aspect Language*), un lenguaje que permite especificar la coordinación de un sistema distribuido siguiendo el paradigma de la programación orientada a aspectos [Kiczales *et al.* 1997]. El desarrollo de este lenguaje en el contexto de los MOWS fue el tema principal de una tesis doctoral [Pérez 2001].

Al mismo tiempo, fruto de la colaboración con Amador Durán en el terreno de la ingeniería de requisitos [Durán *et al.* 1998, Durán *et al.* 1999b, Durán *et al.* 1999d, Durán *et al.* 1999c, Durán *et al.* 1999a], nos planteamos como objetivo construir un entorno de generación de prototipos de sistemas distribuidos a partir de modelos conceptuales. Este entorno a diferencia del construido por Rafael Corchuelo en su tesis doctoral [Corchuelo 1999] tendría en cuenta los requisitos de calidad para generar la arquitectura que resolvía de manera óptima el diseño del sistema.

En [Ruiz-Cortés *et al.* 2000a] presentamos una primera propuesta metodológica para el prototipado arquitectónico de sistemas distribuidos en general. Esta propuesta fue refinada y presentada en [Peña *et al.* 2000b] como una primera aproximación de lo que dimos en llamar *programación arquitectónica* (AP), e incluso realizamos un prototipo que daba soporte a alguna de las actividades que presentamos en [Peña *et al.* 2000a].

Estos primeros trabajos nos sirvieron para constatar que el desarrollo de este generador era una tarea complicada y un tanto desenfocada. Afortunadamente también obtuvimos una conclusión positiva: que la especificación de los aspectos de calidad era muy pobre, por no decir nula, en la mayoría de los lenguajes de descripción de arquitecturas. Llegados a ese punto decidimos centrar la investigación en el diseño de un lenguaje de especificación que nos permitiese expresar con rigor la vista de calidad de una arquitectura a la vez que realizar comprobaciones automáticas de algunas propiedades de interés.

1.5.2. Proyectos de investigación

Todo el trabajo que se describe en esta memoria ha surgido y se ha desarrollado en el contexto de varios proyectos de investigación, y en la experiencia que el autor posee en la empresa privada. Los resultados más influyentes de cada uno de estos proyectos han sido:

- SAD-CMA. En este proyecto se desarrolló el prototipo de los actuales sistemas de adquisición de datos (SADs) de la Consejería de Medio

Ambiente de la Junta de Andalucía, así como el sistema de comunicaciones entre las estaciones remotas de medida que albergan los SADs y los centros de control que reciben y explotan dicha información.

En este proyecto resultó especialmente complicado resolver los conflictos entre los diferentes participantes. Estos conflictos estaban relacionados con el sistema operativo de los sistemas de adquisición, el protocolo de comunicaciones, la portabilidad de las aplicaciones desarrolladas, la facilidad de uso y la facilidad para incorporar nuevas tarjetas de adquisición e instrumentos de medidas.

- RVCA. Redes de vigilancia de control atmosférico.
  Este proyecto permitió desarrollar un sistema de información que cubría todas las necesidades de un Sistema de Alerta MedioAmbiental típico [Ruiz-Cortés y Simón 1997]. Su subsistema de comunicaciones era bastante complejo, pues requería la recogida y entrega coordinada de información entre muchos agentes: estaciones de medida, fábricas involucradas en la emisión de fluidos, ayuntamientos, ministerio de medio ambiente, centros de protección civil, colegios, incluso ciudadanos con algún tipo de afección respiratoria.

  Con este proyecto comprobé lo complicado y costoso que resulta el desarrollo y el mantenimiento de protocolos de comunicación propietarios, sobre todo, por lo inadecuado que resulta el modelo cliente/servidor cuando se trata de coordinar a más de dos entidades. A la vez, también fue un proyecto donde la negociación de requisitos tuvo un importante papel.

  El sistema informático que da soporte a una RVCA es un claro ejemplo de MOWS. Además de tener que hacer frente a los problemas habituales en los sistemas federados, como era conseguir la interoperabilidad entre aplicaciones de diferentes Centros de Control, también es un tipo de sistema donde se utilizan los SLAs. Concretamente, para que la facturación de las empresas encargadas del mantenimiento de los sistemas informáticos que integran la red fuera acorde con la calidad de servicio proporcionada.

- MENHIR: Modelos, Entornos y Nuevas Herramientas para la Ingeniería de Requisitos.

  El objetivo de este proyecto fue investigar en el campo de la ingeniería de requisitos y en concreto acerca de las posibilidades que los

métodos formales pueden ofrecer en este campo y en el desarrollo de metodologías que los soporten. En el contexto de este proyecto se realizaron varias tesis doctorales. Nuestra trabajo de tesis, se apoya principalmente en dos de éstas: la de [Durán 2000] y la de [Corchuelo 1999].

- LEGO. En 1988 solicitamos un ayuda para un proyecto cofinanciado con fondos FEDER. El proyecto de título "LEGO: Entorno de desarrollo para aplicaciones distribuidas y heterogéneas basándose en componentes", recibió una calificación de 4 sobre 5, pero desafortunadamente no recibió financiación por falta de fondos.

  El principal objetivo del proyecto era rediseñar una aplicación de tratamiento y consulta de imágenes satélites que funcionaba en una plataforma UNIX con arquitectura monolítica. El rediseño proponía una arquitectura basada en componentes distribuidos usando CORBA como middleware. Aunque la ejecución del proyecto no se llevó a cabo, el estudio técnico y de viabilidad del proyecto nos proporcionó una muy buena visión de los beneficios y de los riesgos de las aplicaciones basadas en componentes distribuidos heterogéneos.

  Este proyecto fue solicitado junto con las empresas SADIEL e Informática el Corte Inglés.

- GEOZOCO: Métodos y Herramientas para la Automatización del Desarrollo de Aplicaciones de Comercio Electrónico y su Integración con Sistemas de Información Geográfica.

  El proyecto GEOZOCO tiene un presupuesto de 40.000€ y una duración prevista de tres años. Su objetivo principal es investigar en el desarrollo de una metodología y un conjunto de herramientas que permitan automatizar tanto como sea posible el desarrollo de aplicaciones de comercio electrónico y su integración con sistemas de información geográfica. Los objetivos concretos del proyecto consisten en el desarrollo de un marco metodológico equipado con las herramientas adecuadas para que el Ingeniero del *software* pueda elaborar sus aplicaciones de comercio electrónico y aprovechar en ellas las posibilidades que brindan los sistemas de información geográfica, pero siguiendo un riguroso proceso de producción guiado y automatizado en el que las modernas técnicas y tendencias en Ingeniería del *Software* tienen un papel protagonista.

Este proyecto fue solicitado con la Universidad de Castilla-La Mancha y la empresa SADIEL

- WEST: Tecnología Software Orientada a Ambientes WEB.

  El proyecto propuesto tiene como objetivo fundamental generar la tecnología necesaria para disponer de métodos de producción de software para aplicaciones de gestión en ambientes WEB, en el que confluyan los esfuerzos y resultados de los grupos participantes. Esa tecnología abordará desde una perspectiva unificada estrategias de desarrollo de aplicaciones hipermediales con estrategias convencionales, esforzándose en asegurar la calidad de los productos software finales obtenidos.

- WEBPLUS. Los primeros resultados de nuestra investigación sobre servicios WEB aplicados al comercio electrónico ha sido de interés para varias empresas. Este interés se ha concretado en la solicitud, por parte de la empresa Telvent (www.telvent.com), de ayuda para financiar un proyecto con cargo al Programa de Fomento de la Investigación Técnica (PROFIT).

  Este proyecto, titulado "WEBPLUS: Nueva generación de servicios WEB" y con un presupuesto total de 456770 € tiene como principal objetivo el desarrollo de un marco de trabajo para el desarrollo de aplicaciones basadas en servicios WEB, donde la colaboración con los servicios WEB está regulada por acuerdos de nivel de servicio. Este marco de trabajo está siendo validado con la migración de las actuales aplicaciones corporativas de Telvent y de algunas empresas del grupo ABENGOA (www.abengoa.es).

Durante el desarrollo de este trabajo, se ha consolidado el TDG (The Distributed Group, *http://tdg.lsi.us.es*), el grupo de sistemas distribuidos del Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Sevilla, siendo esta tesis la tercera tesis doctoral presentada en el seno de este grupo, y habiendo dado lugar a un buen número de las líneas de investigación en las que actualmente trabajan los miembros del mismo. Durante el desarrollo de esta tesis, se han establecido relaciones con grupos de otros países que trabajan en áreas relacionadas.

1.6. Estructura de esta memoria

La memoria de esta tesis doctoral ha sido organizada en cuatro grandes apartados. La primera parte está constituida por un único capítulo de introducción al problema general tratado en esta tesis y el contexto en el que se ha desarrollado.

La segunda parte, está constituida por los siguientes capítulos:

- El capítulo §2 presenta un modelo referencia que cubre tres procesos relacionados con el tratamiento de los requisitos de calidad: creación de catálogos de atributos, creación de documentos de requisitos de calidad, intermediación entre clientes y proveedores.
- El capítulo §3 ofrece una revisión de algunas propuestas relacionadas con los tres procesos de interés en este trabajo.
- El capítulo §4 describe las características del soporte lingüístico necesario para automatizar algunas de las actividades de los tres procesos identificados. También se indica en que medida las actuales propuestas ofrecen dichas características.

La tercera parte, está constituida por los siguientes capítulos:

- El capítulo §5 presenta de manera intuitiva el lenguaje de especificación que hemos desarrollado con objeto de cubrir el mayor número de características del soporte lingüístico ideal.
- El capítulo §6 presenta de manera rigurosa la semántica de QRL.

La última parte está constituida únicamente por el capítulo §7, en el que se discuten algunas otras aplicaciones de los resultados obtenidos en esta tesis y posibles líneas de trabajo futuro.

Parte II

Estado del arte

Capítulo 2

Modelo de referencia

Quality is a complex concept. Because it means different things to different people, it is highly context–dependent. Just as there is no one automobile to satisfy everyone's needs, so too there is no universal definition of quality.

BARBARA KITCHENHAM

La reflexión de Barbara Kitchenham refleja con bastante acierto la dificultad para dar una definición universalmente aceptada de calidad del software. Como vimos en el capítulo §1, la investigación en este campo ha sido abordada desde perspectivas y áreas muy diferentes, no en vano, es un aspecto presente durante todo el ciclo de vida de un producto software.

El que un tema sea estudiado desde muchas perspectivas tiene sus ventajas e inconvenientes. Uno de los inconvenientes con el que nos hemos encontrado en este caso, ha sido la falta de uniformidad en cuanto al léxico utilizado. Es por ello, que empezaremos el capítulo presentando un modelo de referencia que cubre todas las actividades relacionadas con los requisitos de calidad que tratamos en este trabajo.

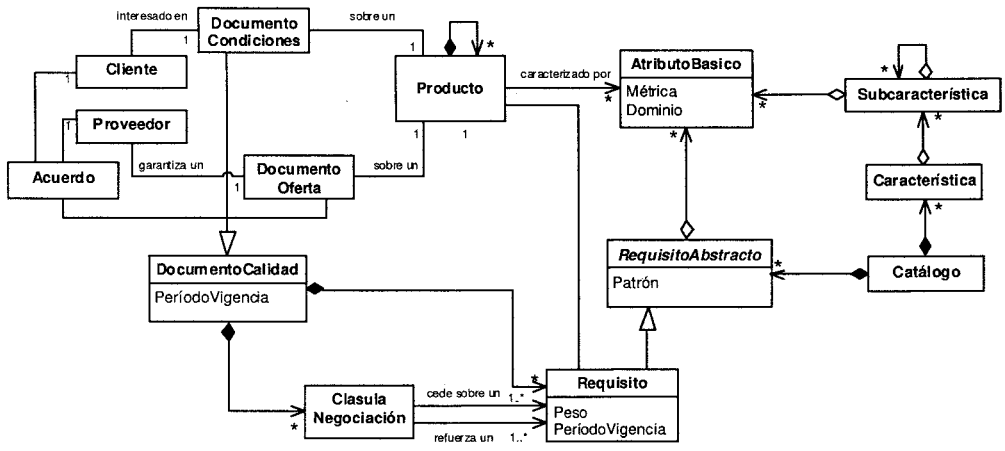

Figura 2.1: *Vista estática del modelo de calidad de referencia.*

Adicionalmente, este modelo de referencia ofrece un doble beneficio: establecer el vocabulario que vamos a utilizar en esta memoria y facilitar la comparativa de los trabajos revisados durante el transcurso de esta tesis. No obstante, conviene señalar, que el carácter de la presentación del modelo es intuitivo e introductorio, ya que sobre este modelo de referencia hemos basado la propuesta que presentamos de manera completa y rigurosa en los capítulos §5 y §6.

2.1. Vista estática

El modelo de referencia está organizado en dos vistas: una vista estática en la que definimos algunos conceptos y las relaciones que existen entre ellos, y una vista dinámica en la que definimos algunas actividades relacionadas con los requisitos de calidad.

El diagrama UML de la figura §2.1 representa la vista estática del modelo de referencia, cuyos principales elementos pasamos a detallar a continuación.

2.1.1. Productos

En los primeros trabajos sobre calidad, el concepto *producto* estaba asociado al de aplicación terminada. En nuestro modelo, un producto hace referencia a cualquier elemento software cuya calidad sea susceptible de ser caracterizada. Desde este punto de vista, un producto puede ser una aplicación completa, una biblioteca, un componente, una interfaz, un caso de uso, un servicio WEB o incluso una imagen para ser mostrada en una página WEB.

Es posible que los productos estén formados a su vez por otros productos que tengan interés desde el punto de vista de valorar la calidad. Por ejemplo, una aplicación puede verse como una agregación de módulos, un servicio WEB como una agregación de interfaces, una interfaz como una agregación de operaciones, etcétera. En todos estos casos, la valoración final del producto es una combinación de las valoraciones de los subproductos.

2.1.2. Clientes y proveedores

Un mismo producto puede estar proporcionado por diferentes *proveedores*. Un proveedor no tiene porque representar a una persona, puede ser un sitio WEB que ofrece diferentes servicios WEB en línea, o un sitio WEB en el que se pueden comprar y descargar automáticamente COTS, etcétera.

Sobre un mismo producto pueden estar interesados uno o varios *clientes*. Al igual que en el caso de los proveedores, un cliente no representa necesariamente a una persona, sino a cualquier entidad con la capacidad de solicitar un producto. Por ejemplo, una aplicación WEB que realiza una consulta a un repositorio compatible con UDDI, etcétera. Cuando no queramos diferenciar entre clientes y proveedores nos referiremos a ellos con el término *participante*.

2.1.3. Atributos

Un atributo de calidad es cualquier característica observable de un producto que puede influir sobre la percepción que un participante tienen del mismo. Para que esas características se puedan considerar realmente atributos y sean útiles, es necesario que se puedan definir con precisión, tanto

desde el punto de vista de su significado y su *dominio* (los valores que pueden tomar), como desde el punto de vista de las *métricas* necesarias para medirlas.

En general, una métrica es una función que transforma valores de un dominio empírico a un dominio formal. Esta función debe cumplir varias propiedades para garantizar que las conclusiones alcanzadas en el dominio formal sean válidas en el dominio empírico. En nuestro caso particular, la métrica de un atributo de calidad permite asociar un valor (ya sea cuantitativo o cualitativo) a un atributo.

2.1.4. Requisitos de calidad

Una *condición* sobre un atributo asociado a un producto siempre puede ser interpretada como un *requisito* de calidad. Por ejemplo, dado el producto P y el atributo a que toma valores en el dominio $D = [1..100]$, una condición definida por un cliente como $P.a > 5$ puede interpretarse como que el cliente se conforma con el producto P siempre y cuando el valor de a sea mayor que 5 (e implícitamente menor que 100).

Esta noción de requisito de calidad concuerda con lo que otros autores denominan *requisitos no funcionales*, pero no con otra noción muy extendida de requisito no funcional en la que se confunde con característica de calidad. Por ejemplo, en muchos textos se considera que la seguridad, la *fiabilidad*, etcétera, son requisitos no funcionales. En nuestro modelo, sin embargo, serían considerados únicamente como atributos.

Aunque todos los modelos de calidad interpretan los requisitos de calidad como restricciones, existen muy pocos modelos que especifican los requisitos como restricciones matemáticas. La mayoría de los modelos especifican los requisitos en lenguaje natural acompañados de un *criterio de aceptación* (*fit criteria*), el cual indica el procedimiento a seguir para comprobar que el requisito se cumple.

Algunos modelos permiten determinar el grado de satisfacción de un determinado requisito. Dicho cálculo se realiza evaluando una *función de satisfactibilidad* también conocida como de utilidad o de preferencia. En el caso de un requisito definido sobre varios atributos, la función de satisfactibilidad es una función multivariable (una variable por cada atributo que participa) que puede ser definida por el usuario o definida automáticamente a partir del valor y la importancia relativa o *peso* de cada atributo.

Los requisitos son entidades sensibles al tiempo, es decir, su *vigencia* o validez está referida a un período de tiempo determinado. Por ejemplo, la capacidad de un sitio WEB que ofrece información sobre el estado de las carreteras debe ser mucho mayor durante el día que durante la noche, y durante los días festivos y puentes vacacionales que en los días laborales. Esta característica, que en adelante denominaremos indistintamente *consciencia temporal* o *temporalidad*, cobra cada día más importancia y desde hace tiempo esta presente en la calidad del servicio y en la política de precios de servicios tales como la telefonía, la televisión por cable, el transporte de datos y el suministro eléctrico.

La incorporación de la temporalidad a los servicios, independientemente de que sean o no informáticos, beneficia tanto a usuarios como a proveedores. Los usuarios pueden adecuar el horario de uso del servicio para conseguir la máxima rentabilidad y los proveedores pueden hacer un uso más racional de los recursos del sistema. Por ejemplo, se puede obtener con antelación y mayor precisión la carga prevista [Hafid *et al.* 1998], lo que permite planificar con mayor acierto las operaciones de mantenimiento y de ampliación de capacidad.

2.1.5. Documentos de requisitos de calidad: condiciones, ofertas y acuerdos

El significado de un requisito, o para ser más precisos, de una condición sobre atributos de calidad, depende del participante que lo haya especificado. Por ejemplo, si la condición $\mathsf{P.a} > 5$ **and** $\mathsf{P.a} < 10$ ha sido especificada por un cliente, significa que el cliente se conforma con el producto P siempre y cuando a tome cualquier valor mayor que 5 y menor que 10. Por contra, si la condición la establece un proveedor, significa que el valor del atributo a del producto P siempre tomará un valor entre 5 y 10.

Los requisitos que exige un cliente sobre un determinado producto conforman un *documento de requisitos de calidad* al que nos referiremos como *documento de condiciones* o simplemente *condiciones*. Los requisitos que sobre un determinado producto son garantizados por un proveedor conforman un documento al que nos referiremos como documento de *oferta* o simplemente *oferta* [Ruiz-Cortés *et al.* 2001b].

Tanto los documentos de condiciones como los de ofertas pueden incluir *cláusulas de negociación*. Una cláusula de negociación indica en qué me-

dida un participante está dispuesto a ceder durante una negociación a fin de alcanzar un acuerdo.

Existe un tercer tipo de documento que recoge aquellos requisitos de calidad que tras un proceso de negociación recoge todos los requisitos que se comprometen a cumplir los participantes. Nos referiremos a dicho documento como *acuerdo de calidad* o simplemente *acuerdo*.

2.1.6. Catálogo de atributos

Generalmente, el número de atributos necesarios para caracterizar la calidad de un producto software de complejidad media, oscila entre los 40 y los 140 [Dujmovic 1996], por lo que casi todos los modelos ofrecen la posibilidad de organizarlos de forma jerárquica en forma de *catálogos* que agrupan conjuntos de atributos relacionados entre sí.

Los términos utilizados para referenciar a los diferentes niveles de la jerarquía son muy variados, aunque en los últimos años parece que se está extendiendo la terminología propuesta en la norma ISO 9126 [ISO 1991]: *característica, subcaracterística* y *atributo*. En el nivel más alto de la jerarquía se encuentran las características. Cada característica se refina en una o varias subcaracterísticas, las que a su vez, pueden refinarse de manera recurrente en nuevas subcaracterísticas y atributos. Estos últimos conforman el último nivel de la jerarquía.

Con frecuencia, las condiciones sobre atributos de calidad siguen un determinado *patrón*. Por ejemplo, sobre el atributo TTF que representa al tiempo entre dos fallos consecutivos, es probable que se definan requisitos conformes al patrón "P.TTF > n [unidad de tiempo]", donde P hace referencia a un producto y n a un valor numérico. Este patrón puede ser interpretado como una definición de un requisito abstracto a partir del cual se pueden derivar requisitos concretos.

Los principales objetivos que se persiguen al utilizar este tipo de patrones con condiciones sobre atributos de calidad, son los mismos que se persiguen con el uso de patrones lingüísticos durante la elicitación de requisitos: reducir el tiempo necesario para redactar el documento de requisitos a la vez que se aumenta la propia calidad del documento [Durán *et al.* 1999a].

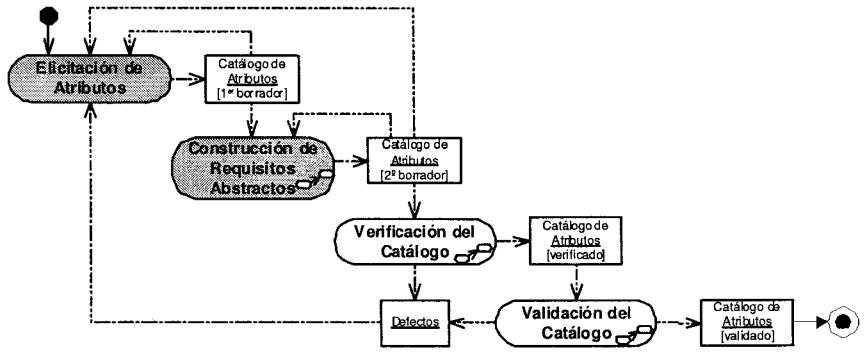

Figura 2.2: *Proceso de construcción de un catálogo de atributos de calidad.*

2.2. Vista dinámica

2.2.1. Construcción del catálogo de atributos de calidad

Las actividades del proceso de construcción del catálogo de atributos se muestran en el diagrama de actividades UML [OMG 2001] de la figura §2.2. El proceso se inicia con la *elicitación de atributos* de calidad que da como resultado una primera propuesta de catálogo de atributos que sólo contiene la definición de atributos. Esta propuesta se utiliza como base para *construir requisitos abstractos*, actividad puede llevarse a cabo como si de requisitos concretos se tratasen (ver siguiente sección).

Este segundo borrador del catálogo debe someterse a un proceso de *verificación* a fin de determinar si se está construyendo de acuerdo con la norma elegida para dicho catálogo. Una vez el documento ha sido verificado, es necesario que los responsables del catálogo *validen*, esto es, den el visto bueno al catálogo.

Hasta la fecha son pocas las propuestas que sistematizan la construcción del catálogo de atributos destacando el trabajo de [Franch 1998] y de [Olsina 1999]. En esta tesis abordaremos únicamente las dos primeras actividades.

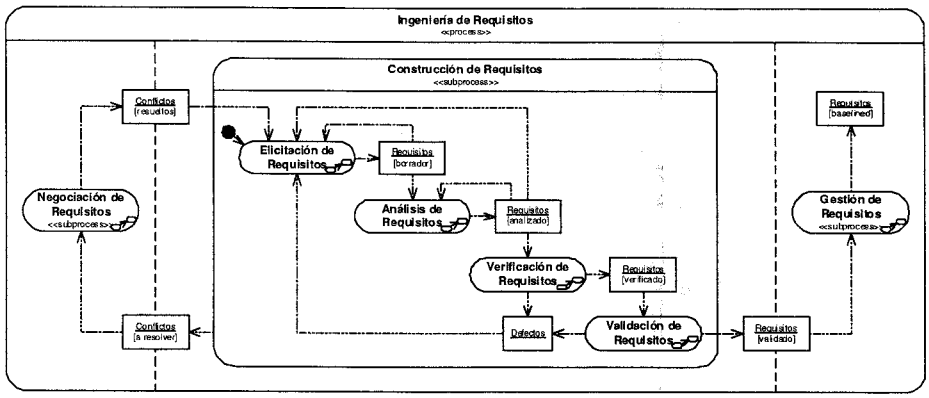

Figura 2.3: *Proceso de construcción de un documento de requisitos de calidad.*

2.2.2. Construcción de documentos de requisitos de calidad

Las actividades del proceso de construcción del requisitos de calidad se muestran en el diagrama de actividades UML de la figura §2.3. Dicho proceso comparte el mismo flujo que el proceso de ingeniería de requisitos propuesto en [Durán *et al.* 2002b].

El proceso se inicia con la *elicitación* de requisitos de calidad, utilizando como información de entrada el catálogo de atributos. Como resultado de la elicitación, se obtiene un borrador del documento de requisitos de calidad que se *analiza* a fin de detectar posibles inconsistencias entre las condiciones de los requisitos. Una vez comprobada la ausencia de inconsistencias, se *verifica* el documento a fin de comprobar que se están cumpliendo las normas de elaboración fijadas para el documento. Finalmente, con la *validación* del documento se comprueba que el documento de requisitos responde a las expectativas que sobre él tienen los responsables del documento. Si se detectan defectos en la especificación se iniciará de nuevo todo el proceso.

El proceso de construcción de un documento de requisitos de calidad guarda las mismas dependencias con los restantes procesos de ingeniería de requisitos. Si no es posible resolver todas las inconsistencias detectadas o bien algún o algunos participantes no está de acuerdo con el documento

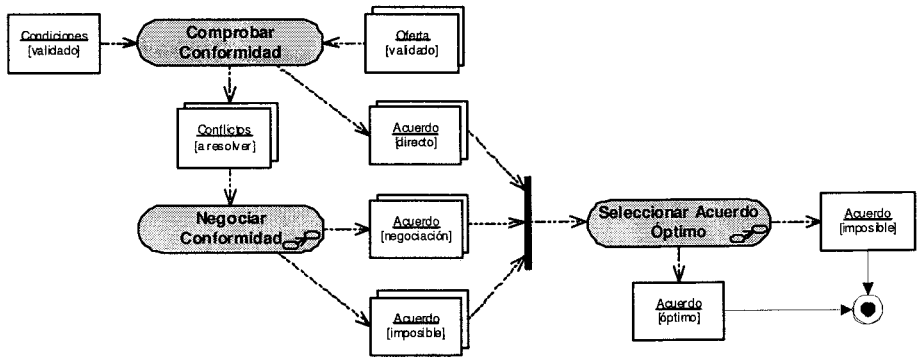

Figura 2.4: *Proceso básico de intermediación entre clientes y proveedores.*

final, es necesario iniciar un *proceso de negociación* sobre el documento de requisitos en cuestión. Por otra parte, una vez el documento de requisitos ha sido validado, debe continuar el *proceso de gestión.*

2.2.3. Intermediación entre clientes y proveedores

Las actividades del proceso de intermediación entre clientes y proveedores se muestran en el diagrama de actividades UML de la figura §2.4. El proceso de intermediación se inicia asumiendo que se dispone de todas las ofertas de todos los proveedores que proporcionan un determinado producto. Teniendo en cuenta esta consideración, el proceso se inicia *comprobando la conformidad* de las ofertas, a fin de conocer qué ofertas satisfacen el nivel de calidad requerido en las condiciones. A partir de estas ofertas es posible obtener directamente acuerdos de calidad. Con los proveedores de aquellas ofertas que no son conformes con las condiciones, se inicia un proceso de *negociación* a fin de lograr algún acuerdo, lo que no siempre es posible.

Una vez comprobada la conformidad de todas las ofertas y obtenidos todos los posibles acuerdos, bien directos, bien a través de un proceso de negociación, se procede a seleccionar el acuerdo óptimo. Si se está en un entorno competitivo, en el que los proveedores luchan por que su oferta sea aceptada, el acuerdo óptimo es aquél que maximiza la función de

satisfactibilidad asociada al documento de condiciones. Por contra, en un entorno cooperativo, en el que todos los proveedores y el cliente pertenecen a la misma organización, el acuerdo óptimo es aquel que tiene máxima satisfactibilidad.

Capítulo 3

Trabajo relacionado

Seis honrados servidores me enseñaron cuanto sé. Sus nombres son: cómo, cuándo, dónde, qué, quién y por qué.

RUDYARD KIPLING

En este capítulo revisamos algunas propuestas relacionadas con actividades de al menos uno de los tres procesos del modelo de referencia presentado en el anterior capítulo. Las propuestas se han organizado en tres grandes grupos o generaciones.

El criterio que hemos seguido para realizar la división en grupos está basado en las características del lenguaje de especificación de requisitos ideal que veremos en el siguiente capítulo. Finalizaremos el capítulo con un resumen sobre nuestra particular visión de la evolución y relación entre las diferentes propuestas analizadas.

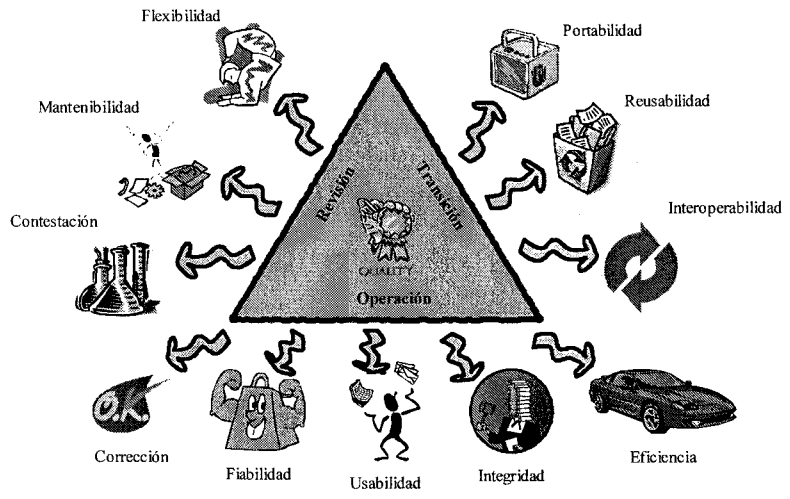

Figura 3.1: *Características del modelo de McCall.*

3.1. Modelos de primera generación

3.1.1. Modelos de McCall y Boehm

Desarrollado originalmente para evaluar la calidad del software del departamento de defensa de los EEUU [McCall *et al.* 1977], es sin duda alguna el modelo más referenciado. McCall propone un catálogo con 11 características y 25 atributos. La figura §3.1 muestra las 11 características del catálogo que propone McCall a las que se refiere como *factores*.

Obsérvese que el catálogo de atributos no dispone de subcaracterísticas. En nuestra opinión, esta circunstancia refleja el escaso conocimiento que sobre los factores de calidad se tenía en aquellos momentos. Como veremos más adelante, al aumentar el conocimiento sobre los factores de calidad, surge la necesidad de descomponer una características en varias subcaracterísticas.

El modelo de calidad de Boehm [Boehm *et al.* 1978] es muy similar al de McCall siendo la principal diferencia la estructura del catálogo, en este caso constituida por siete características y 12 atributos. Tanto en este modelo como en el de McCall, el grado en el que se satisface una característica

se obtiene como resultado de aplicar una función de agregación sobre las valoraciones de los atributos de las que depende dicha característica. Una de las funciones de agregación más comunes en los modelos de primera generación es

$$c(a_1, a_2, \ldots, a_n) = \sum_{i=1}^{n} w_i v_i(a_i)$$

en donde cada a_i representa el valor de un atributo, cada v_i representa la función utilizada para valorar el atributo y cada w_i su peso relativo con respecto a las demás.

Esta opinión se refuerza si tenemos en cuenta que la mayor parte de los atributos de McCall (a los que él se refiere como métricas) se valoran de un modo subjetivo.

3.1.2. Modelo FURPS

En los primeros catálogos de atributos era frecuente encontrar un mismo atributo asociado simultáneamente a varios atributos. Esta característica conocida como *solapamiento* ha sido señalada por algunos autores como poco deseable [Dromey 1995].

Uno de los primeros catálogos de atributos sin solapamiento fue propuesto por Hewlett Packard en [Grady y Caswell 1987]. Este catálogo al que denominaron con el acrónimo FURPS (*Funcionality, Usability, Reliability, Performance* y *Supportability*), proponía 5 características y 27 atributos.

3.1.3. ISO 9126

A finales de los 80, existían decenas de catálogos de atributos aunque ninguno de ellos podía ser considerado como un estándar de facto. No es necesario entrar en detalles para comprender que los problemas derivados de esta situación de ausencia de estándar llegaron a ser muy importantes. Tanto es así, que conscientes de la necesidad de definir normas internacionales sobre modelos de calidad de productos software, organismos tales como la ISO y el IEEE comenzaron a trabajar sobre el tema. El conjunto de normas que más repercusión ha tenido hasta la fecha, ha sido el de la ISO, más concretamente la serie ISO/IEC 9126 [ISO 1991].

La primera reunión del grupo de trabajo relacionada con esta norma fue celebrada en 1985 en Munich, en el contexto del proyecto 7.13 de título *"Software Product Quality Evaluation"*. Los objetivos iniciales del proyecto fueron definir un conjunto de características de calidad y definir un proceso de evaluación. Por razones fundamentalmente organizativas, la ejecución del proyecto duró más de seis años. No fue hasta 1991 cuando se obtuvo el primer resultado del proyecto, la norma que se publicó como ISO/IEC 9126 y que llevaba por título: *"Software Product Quality Characteristic and the Guide for their Use"*. La norma propone un catálogo organizado en 6 características y 21 subcaracteristicas con un solapamiento mínimo. Respecto, al proceso de intermediación, la norma sólo define un marco conceptual para la actividad de evaluación, pero no propone un método concreto.

3.1.4. Resumen

El interés por la calidad del software es tan antiguo como la propia historia de la informática, pero no fue hasta la década de los setenta cuando se empezaron a poner en marcha las primeras iniciativas dirigidas a describir la calidad y trabajar con los factores que influyen en ella de un modo más sistemático de lo que había sido hasta entonces.

En estos primeros años podemos destacar los modelos de MacCall, Boehm, Grady & Caswell y el propuesto por la ISO como cuatro de los más representativos de la primera generación. La profundización en el conocimiento de los factores de calidad más comunes, tuvo como fruto la definición de catálogos jerárquicos de atributos predefinidos y la sistematización del proceso de evaluación, aunque con un carácter fuertemente subjetivo.

3.2. Modelos de segunda generación

Los problemas derivados de la subjetividad de las métricas fueron identificados desde los primeros modelos de calidad. En [Cavano y McCall 1978] se discute sobre la necesidad de definir la calidad de la manera más exacta posible así como una manera de obtener medidas cuantitativas de la calidad del software a fin de poder realizar evaluaciones objetivas. No obstante, tuvieron que pasar más de 10 años hasta que surgieron los prime-

ros modelos de calidad que proponían el uso de métricas objetivas y cuantificables. En nuestra opinión, y coincidiendo con el criterio de [Pressman 1998, página 348], el interés por cuantificar es el rasgo que caracterizó a una segunda generación de modelos para valorar la calidad: los modelos *cuantitativos*. No obstante, y como veremos en la sección §3.2.7, existen otras características que marcan la diferencia entre los modelos de primera y segunda generación.

3.2.1. Modelo de Gilb

Tom Gilb fue uno de los primeros autores en ofrecer una propuesta para especificar atributos y requisitos de calidad objetivos y verificables [Gilb 1988]. Su propuesta se apoya en dos ideas principales: utilizar plantillas para especificar atributos y construir el catálogo de atributos con un enfoque ascendente y consensuado.

El enfoque propuesto por Gilb consiste en definir las características y subcaracterísticas del catálogo a partir de atributos que se pueden y saben medir de manera objetiva. De este modo, todos los requisitos que pueden definirse sobre los atributos básicos o derivados del catálogo son verificables.

Por ejemplo, un requisito expresado como "la fiabilidad del sistema deberá ser alta" es subjetivo y difícilmente verificable y no debería ser considerado como tal. Sin embargo, un requisito expresado como: "El tiempo máximo que puede estar fuera de servicio es de 30 minutos y las interrupciones y reanudaciones del servicio deben ser transparentes al usuario" es objetivo y más fácil de verificar.

Por otra parte, la definición del catálogo se lleva a cabo en reuniones en las que están representados todos los participantes en un proyecto. Con este enfoque, el catálogo de atributos deja de ser una información con estructura y contenido fijos para ser dependiente del dominio del problema y de las preferencias de los participantes en el proyecto.

Las plantillas propuestas por Gilb descansan sobre siete principios que deben ser respetados durante el proceso de construcción de atributos [Gilb 1988, pág. 134]. De ellos, sin duda alguna, el más importante es el principio de *cuantificabilidad* enunciado textualmente como:

FACILIDAD_USO:
SCALE = días laborables que tarda un empleado en aprender a utilizar el sistema
TEST = el 90 % de las funciones se llevan a cabo satisfactoriamente en un tiempo no superior al doble que necesitaría un usuario experimentado
WORST = entre uno y siete días
PLAN = menos de un día
BEST= menos de dos horas

Figura 3.2: *Ejemplo de uso de las plantillas de Gilb.*

> ***The measurebility principle**: All attributes can and should be made measurable in practice.*

Aunque el autor no ofrece ningún razonamiento formal para sustentar este principio, su discusión informal nos parece suficientemente razonable. Su argumento principal es que dado un atributo siempre es posible encontrar una escala para medirlo. Si esta escala no se encuentra es porque en realidad no se tratq de un atributo, sino de una subcaracterística que necesita descomponerse.

La figura §3.2 muestra un ejemplo de uso de su plantilla para definir un atributo de los que suelen definirse subjetivamente: la facilidad de uso. El significado de cada campo de la plantilla es el siguiente:

- SCALE. Es una descripción intuitiva y teórica del significado del atributo.
- TEST. Es una definición más práctica del atributo que indica cómo debe ser medido. Esta definición puede ser referida para un instante temporal concreto, p.e. TEST (1 mes), TEST(1 año), TEST (fase diseño).
- WORST. Indica el valor más desfavorable que puede tomar el atributo sin ser rechazado.
- PLAN. Indica el valor que se considera suficientemente bueno.
- BEST. Indica el mejor valor que se podría obtener. Tanto este campo como los dos anteriores pueden estar referidos a un instante de tiempo concreto.

3.2.2. Modelo de Keller

Pese a ser desarrollado hace más de 10 años, el modelo de Keller [Keller *et al.* 1990], es citado en [Chung *et al.* 2000] como la referencia al estado del arte de las propuestas cuantitativas orientadas al producto.

La estructura del catálogo es similar a la de los modelos de primera generación. En realidad, Keller no diseña ningún catálogo, sino que recomienda el uso del catálogo desarrollado en el RADC (*Rome Air Development Center*)[1]. Al igual que Boehm y McCall, este catálogo estaba diseñado especialmente para caracterizar grandes sistemas y aplicaciones, por lo que aconseja una tarea de adecuación del catálogo al contexto particular de cada caso.

Keller subraya una característica importante de su catálogo: la *comunicabilidad* (sección §4.6). La idea es definir las características con un lenguaje perfectamente entendible para los clientes (de hecho, el se refiere a éstas como *atributos orientados al cliente*) y las subcaracterísticas (a las que se refiere como *atributos orientados a los técnicos*) con un lenguaje especialmente claro y eficaz para los técnicos. Esta característica puede verse como una de las primeras propuestas de catálogos duales (secciones §4.6 y §5.1.1).

En nuestra opinión, la aportación del trabajo de Keller es doble. Por una parte, es de los primeros en separar claramente dos conceptos que aún hoy se no se definen separadamente y que resulta clave para reutilizar los catálogos de atributos. Por otra parte, presenta un marco de trabajo para definir métricas tanto para los atributos (métricas directas) como para las características y subcaracterísticas (métricas agregadas). Disponer de una métrica para cada elemento del catálogo facilita la construcción de requisitos cuantitativos y verificables.

No obstante, esta segunda aportación ya estaba presente casi con el mismo alcance en el trabajo de Gilb, si bien Keller no hace referencia a éste. De hecho, ambos autores coinciden al señalar como uno de los principales inconvenientes de sus propuestas, la dificultad para definir métricas válidas, pues es esta una actividad para la que la mayoría de participantes en proyectos software no han sido formados.

Respecto a las actividades relacionadas con la intermediación, se puede decir que el modelo de procesos propuesto por Keller contempla a grandes rasgos las mismas actividades que los modelos cuantitativos que se han

[1]El desarrollo de este catálogo duró más de diez años.

desarrollado posteriormente.

3.2.3. Normas ISO/IEC 9126 y 14598

Durante los cuatro años siguientes a la publicación de la primera versión de la ISO/IEC 9126, se propusieron continuas y profundas revisiones a la norma. Las reuniones de trabajo celebradas en Londres, Tokyo, Praga y finalmente Otawa dieron como resultado la creación de dos series de normas claramente diferenciadas, la serie 9126 dedicada a la definición de catálogos de atributos y métricas, y la serie 14598 dedicada a la evaluación de productos software. La razón de dicha división tal y como se discute en [Azuma 2002] se debió fundamentalmente a la necesidad de extender la primera versión para dar soporte a la construcción e intermediación (fundamentalmente evaluación) de documentos de requisitos de calidad.

Si bien se puede decir que la serie 9126 tuvo y sigue teniendo un impacto importante en la industria y en la academia, no se puede decir lo mismo de la 14598. En [Azuma 2002] se discuten algunas de las razones de este fracaso, apuntando como una de ellas el mantener separadas normas tan estrechamente relacionadas. De hecho, la ISO ya ha anunciado la unificación de ambas series en una única de nombre SQuARE o ISO 25000 [Azuma 2002], estructurada en cinco grandes divisiones dedicadas a: la construcción de catálogos de atributos (en terminología ISO se denominan modelos de calidad), la construcción de métricas, la construcción de requisitos, el proceso de evaluación y a guiar el proceso completo.

No obstante, nada parece apuntar a que SQuARE incorpore elementos que supongan un cambio cualitativo importante respecto a las normas actuales, al menos en la línea de las características que en nuestra opinión deben poseer los modelos de calidad para dar respuesta a las necesidades derivadas del desarrollo y ejecución de aplicaciones basadas en servicios WEB y en COTS.

3.2.4. Modelo de Dromey

El modelo de calidad de Dromey [Dromey 1995] fue propuesto inicialmente para evaluar la calidad de la implementación de productos software, si bien el autor afirma que también puede ser utilizado para evaluar documentos de análisis y de diseño. Al igual que Gilb varios años an-

tes, Dromey propone que la construcción del catálogo de atributos siga un modelo mixto arriba–abajo/abajo–arriba, pues de los contrario resulta muy difícil conseguir una estructura consistente y realmente útil. Uno de sus argumentos es considerar que es más fácil identificar la relación entre algo tangible, como son los atributos (que el denomina *quality–carrying properties*), hacia algo intangible como son las características.

El modelo de Dromey incorpora elementos originales relacionados con la evaluación de la calidad de una implementación que no aparecen en modelos donde lo que se intenta valorar es la calidad externa de aplicaciones y sistemas. No obstante, comparte algunos elementos e ideas con los modelos de Gilb y de Keller sobre todo en la necesidad de especificar los atributos de calidad de la manera más objetiva posible.

3.2.5. Modelo de Dujmovic

Aunque no es presentado como modelo de calidad[2], el método propuesto por Dujmovic en [Dujmovic 1996] para evaluar, comparar y seleccionar sistemas complejos, consigue el principal objetivo de un modelo de calidad: conocer el grado de satisfacción que sobre un conjunto de características posee un determinado producto. De hecho ha sido la base sobre la que se han definido modelos de calidad recientes, tales como el de Olsina [Olsina 1999, Olsina *et al.* 1999].

Catálogo de atributos. Dujmovic se refiere a los atributos de calidad como *variables de rendimiento*[3], y propone un catálogo de cuatro características (*hardware, software, rendimiento* y *soporte del vendedor* y 23 subcaracterísticas. Basándose en su experiencia, Dujmovic indica que el número de atributos necesarios para caracterizar sistemas software oscila entre 40 y 120, por lo que resulta casi obligatorio utiliza métodos cuantitativos para realizar una evaluación objetiva de la calidad de dichos sistemas.

Dujmovic asume implícitamente la existencia de una métrica para cada atributo, por lo que es posible asociar a cada atributo de calidad una función de satisfactibilidad (el la denomina función de *criterio elemental*).

[2]De hecho en el artículo donde se describe el método [Dujmovic 1996] el término *calidad* no aparece ni una sola vez.

[3]Imaginamos que por razones históricas puesto que estas variables no están referidas únicamente al rendimiento.

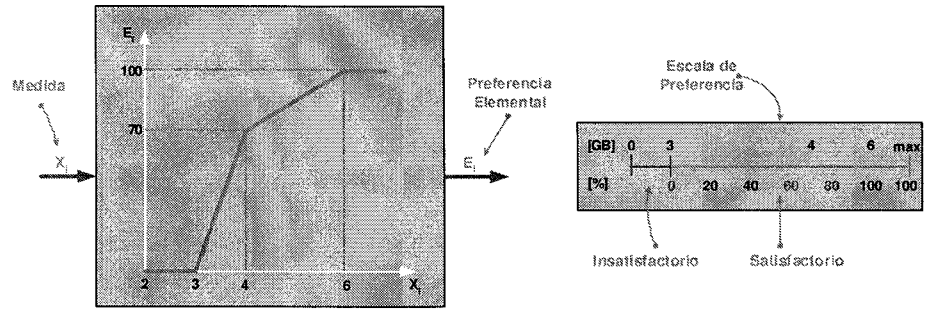

Figura 3.3: *Función y escala de satisfactibilidad elemental de Dujmovic.*

Por ejemplo, si C representa a la capacidad en GB de un disco duro cuyo rango de valores aceptables es $C^{min} < C \leq C^{max}$ entonces una posible definición de la función de satisfactibilidad, denotada como Ψ podría ser:

$$\Psi(C) = \begin{cases} 0 & \text{si } C \leq C^{min} \\ 0 < n < 100 & \text{si } C^{min} < C < C^{max} \\ 100 & \text{si } C \geq C^{max} \end{cases}$$

que también puede representarse gráficamente de la forma tradicional o mediante una escala de preferencia (ver figura §3.3).

Especificación de requisitos. Al igual que los restantes modelos de segunda generación, el modelo de Dujmovic sigue sin distinguir el concepto de atributo del concepto de requisito. No obstante, su idea de añadir una función de satisfactibilidad a los atributos es una característica bastante novedosa que será explotada por otros modelos tales como el de Olsina (ver sección siguiente) y que le diferencia claramente de los restantes modelos de su generación.

Otra característica interesante es que junto con la norma ISO 14598 es de los primeros en los que se distingue de manera clara entre documentos de condiciones y de ofertas, a las que el se refiere como *solicitudes de propuestas* y *propuestas* respectivamente.

El documento de condiciones que Dujmovic consta de una guía de pre-

paración de la propuesta, del sistema de evaluación a utilizar y en el caso de equipo informático de una guía de medición del rendimiento del equipo. La primera guía también incluye la lista de los atributos (junto con su función de satisfactibilidad) que se deben satisfacer y la definición de la función de agregación para valorar la calidad global.

Por su parte, en el documento de oferta, el proveedor del producto debe expresar las medidas de los atributos de acuerdo con la guía de preparación de propuestas asociada a las condiciones del cliente.

Proceso de intermediación. Otras de las aportaciones importantes del modelo de Dujmovic es la propuesta de un modelo de valoración de calidad alternativo al de los modelos clásicos procedentes de la teoría de decisión con múltiples criterios [Miller 1970]. Los modelos clásicos lineales–aditivos presentan varios inconvenientes en contextos donde se necesitan valorar muchas características y cuando se necesitan reflejar relaciones complejas entre ellas: obligatoriedad, opcionalidad, etcétera. Por ejemplo, si se dispone de 100 características a valorar, el valor medio del peso de cada atributo es sólo del 1 %, lo que plantea al menos dos inconvenientes [Dujmovic 1996][4]:

- No es posible modelar atributos que deben puntuar obligatoriamente, pues aunque el valor de dicho atributo es cero, la valoración global no tiene porqué ser cero, basta que la valoración de cualquier otro atributo sea distinta de cero.
- La contribución de un atributo a la puntuación global está limitada por el valor de su peso, el cual, en el caso de tomar valores tan pequeños resulta insignificante en la práctica.

El problema de modelar atributos obligatorios no se resuelve reemplazando la media ponderada por la media geométrica $c(a_1, a_2, \ldots, a_n) = \prod_{i=1}^{n} w_i \Psi(a_i)$, pues si la puntuación del atributo más insignificante es 0 la puntuación global también es cero, situación del todo inadmisible.

Estas limitaciones y otros problemas relacionados con las técnicas tradicionales de puntuación motivaron el desarrollo de un nuevo método de evaluación: el método LSP (*Logic Scoring Method*) [Dujmovic 1996]. La función de agregación que propone el método LSP tiene la forma:

[4]En [Olsina 1999, página 22] se comentan cuatro limitaciones adicionales.

$$c(e_1, e_2, \ldots, e_n) = \sum_{i=1}^{k} (w_i e_i^r)^{1/r}, \quad \sum_{i=1}^{k} w_i = 1, \quad w_i > 0, \quad i = 1, \ldots, k$$

donde $e_i = \Psi(a_i)$ y donde el exponente r es un valor real calculado para que la agregación consiga las propiedades lógicas deseadas. El cálculo del exponente se detalla en [Dujmovic y Elnicki 1982, Dujmovic 1991], no obstante, resumiendo podemos decir que el valor de r depende del grado de conjunción (c) o de disyunción (d) con el que queramos obtener la agregación. Esto es, si deseamos una posición conservadora en la que el resultado nunca sea mayor que el valor mínimo de las preferencias elementales entonces la conjunción debe ser máxima ($c = 1$). Por contra si deseamos una mantener una postura más optimista y considerar que el resultado se corresponde con el valor máximo de las preferencias elementales entonces la disyunción es máxima ($d = 1$ o $c = 0$). Es decir, para una agregación de dos preferencias elementales (e_1, e_2) se tiene que:

$$e_o = c\,\text{mín}(e_1, e_2) + d\,\text{máx}(e_1, e_2), \quad c + d = 1$$

En [Dujmovic 1996] se ofrecen los valores del exponente r para un total de 20 combinaciones de valores de conjunción y disyunción y también se indica que disponen de herramientas para calcular los valores óptimos de r, c y d.

El análisis de conformidad es extremadamente simple, el cliente define la valoración mínima admisible para la calidad global del producto $E^{\text{mín}}$ (normalmente el 67 %) y el precio máximo que se está dispuesto a pagar $C^{\text{máx}}$. Con esta información es posible determinar qué ofertas son conformes con los requisitos sobre la calidad y el precio. La figura §3.4 muestra la valoración de cinco ofertas, y cómo dos de ellas, la A y la B, no son conformes.

El proceso de elección de la oferta óptima consiste en determinar la oferta que maximiza una función sobre el nivel de calidad E y el precio C. En [Dujmovic y Elnicki 1982] se describen varias funciones habituales, cada una de las cuales destaca un aspecto diferente. En el ejemplo de la figura §3.4 resultaría elegida la oferta R por ser la que maximiza la expresión E/C.

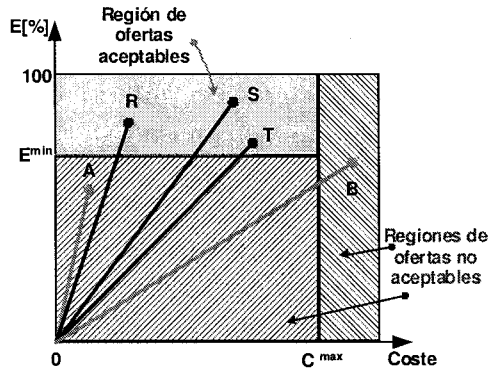

Figura 3.4: *Ejemplo de análisis coste/preferencia de Dujmovic.*

3.2.5.1. Discusión

El modelo de Dujmovic incorpora elementos muy útiles. No obstante, es posible realizar algunas extensiones para aumentar aún más sus posibilidades. Algunas de éstas son:

1. El proceso de elección de la oferta óptima, no es más que un caso particular de un problema de optimación más general, por lo que entendemos que se podría dejar libertad al usuario de decidir los atributos y la estructura que definen la función a maximizar y no reducirlo a la calidad global y al coste.

2. El modelo no contempla la posibilidad de que la oferta se especifique mediante condiciones libres y no únicamente como condiciones en igualdad. De este modo, aunque pudiéramos expresar medidas tales como: productividad $\in$ [10..12] unid/seg y consumo de memoria $>$ 2 MB no se sabría qué valor tomar para calcular la satisfactibilidad: ¿el más pequeño?, ¿el más grande?, ¿el punto medio?, etcétera.

3. Las limitaciones de los modelos de evaluación aditivos–lineales justifican el uso de modelos alternativos. No obstante, creemos que existen soluciones cuando menos igualmente eficaces.

   Por ejemplo, los problemas derivados de utilizar un número elevado de atributos se reducen en gran medida si éstos están organizados jerárquicamente, pues en tal caso la importancia relativa de cada

atributo debe ser tenida en cuenta únicamente con los atributos del mismo nivel (que por regla general nunca llegarán a más de 15 o 20), pero no con atributos de otros niveles.

Respecto a la función de agregación, entendemos que se pueden conseguir los mismos resultados haciendo uso de un modelo de especificación mucho más declarativo. Nos referimos a especificar los requisitos mediante restricciones matemáticas, con el que además es posible expresar relaciones aritméticas entre atributos.

3.2.6. Modelo de Olsina

En [Olsina 1999] se presenta una metodología cuantitativa para la evaluación y comparación de la calidad de sitios WEB. El modelo de calidad sobre el que subyace dicha metodología puede verse como una extensión de los modelos de Dujmovic y de Gilb y en nuestra opinión explota casi todas las posibilidades que pueden ofrecen los modelos cuantitativos puros que no tienen soporte lingüístico formal.

3.2.6.1. Catálogo de atributos

Olsina discute sobre los dos enfoques que, según [Fenton y Pfleeger 1996], pueden seguirse para definir el catálogo de atributos: el enfoque de *modelo fijo* en el que se toma un catálogos de atributos bien conocido (McCall, Boehm, ISO 9126) y el enfoque *a medida* en el que el catálogo de atributos se define consensuadamente entre los participantes sin necesidad de utilizar como base un catálogo publicado previamente.

Tras estudiar la idoneidad de estos dos enfoques para evaluar la calidad de sitios WEB, Olsina propone un tercer modelo fruto de la combinación de ambos: el *modelo mixto*. Dicho modelo propone partir de un catálogo base bien conocido (en su caso parte del de la ISO 9126), de manera que las características del nuevo catálogo sea un subconjunto de las del catálogo base. Por su parte, las subcaracterísticas, los atributos básicos y las relaciones entre ellos se definen por consenso entre todos los participantes. La figura §3.5 muestra el fragmento de un catálogo de atributos (Olsina los denomina *árbol de requerimientos*) correspondiente a la Confiabilidad, una de las seis características de la ISO 9126. Como puede comprobarse en la norma [ISO 1999], las subcaracterísticas de la Confiabilidad del catálogo de Olsina, nada tienen que ver con las de la norma.

3. Confiabilidad
  3.1. No deficiencia
    3.1.1 Errores de enlaces
      3.1.1.1 Enlaces rotos
      3.1.1.2 Enlaces inválidos
      3.1.1.3 Enlaces no implementados
    3.1.2. Errores o deficiencias varias
      3.1.2.1 Deficiencias o cualidades ausentes debido a diferentes *browsers*
      3.1.2.2 Resultados inesperados independientes del *browser* (p.e. errores de búsqueda imprevistos, deficiencias con marcos, etcétera)
      3.1.2.3 Nodos destinos (inesperadamente en construcción)
      3.1.2.4 Nodos WEB muertos (sin enlaces de retorno)

Figura 3.5: *Árbol de requerimientos de calidad de la* Confiabilidad.

La medición de atributos de calidad no es una tarea trivial (es uno de los problemas clásicos en el área de evaluación de la calidad), pues si bien existen algunos atributos fáciles de medir, por regla general no lo son. Para medir atributos, subcaracterísticas y características, Olsina propone el uso de métricas *válidas* en el sentido propuesto en [Fenton y Pfleeger 1996]. De este modo, la correspondencia entre el dominio empírico (objeto del mundo real) y el nuevo dominio numérico o simbólico (objeto del mundo formal) permita garantizar que las conclusiones alcanzadas en el dominio numérico se correspondan con las del mundo real. El campo *parámetros y variables disponibles* de la figura §3.8 define (por extensión) la métrica utilizada para el atributo Soporte a lenguaje extranjero.

Cómo puede comprobarse, asociado al atributo existen dos variables: una que representa el número de lenguajes soportados y otra el grado de soporte. Definir la métrica para la primera variable resulta trivial, pero en el caso de la segunda, que sigue siendo muy sencillo, es necesario tener cuidado para no violar la condición de representación. Por ejemplo, si se intercambian los valores de soporte S_2 y S_3 la función de satisfactibilidad tal y como está definida, asignaría una mayor satisfactibilidad a un sitio WEB que soporta inglés parcialmente que a un sitio que soporta inglés totalmente.

En [Olsina 1999] también se propone un marco conceptual para la validación de métricas tanto directas como indirectas (también conocidas co-

mo derivadas). Las métricas directas sólo pueden ser utilizadas para medir atributos y las métricas indirectas para medir características, subcaracterísticas y en ocasiones algunos atributos.

La especificación detallada de cada elemento del catálogo se realiza utilizando como modelo las plantillas de la figura §3.6. Cómo puede apreciarse, existen varios campos comunes a las tres tipos de elementos del catálogo (título, código, tipo, definición, comentarios, peso, ejemplos, valores de preferencia computados) y otros característicos de cada elemento. En la figura §3.7 se muestra la especificación del atributo Soporte a Lenguaje Extranjero siguiendo este modelo de plantillas. Como puede observarse, la función de satisfactibilidad de los atributos junto con otra información de interés, se detalla en otra plantilla aparte (ver figura §3.8).

3.2.6.2. Especificación de requisitos

En la propuesta de Olsina sigue sin existir una separación clara entre atributo y requisito. Esto es comprensible si tenemos en cuenta que entre los objetivos de su trabajo no se encuentra la especificación de requisitos, tan sólo la evaluación y comparación de la calidad de sitios WEB, la cual puede hacerse sin necesidad de especificar requisitos de calidad.

3.2.6.3. Método de evaluación

Como indicamos al principio de esta sección, el modelo de calidad de Olsina está muy influenciado por el de Dujmovic. De hecho, el método de evaluación que propone Olsina es muy similiar al propuesto Dujmovic. Una de las pocas diferencias, se encuentra en la caracterización que se realiza de la función de satisfactibilidad en tres regiones: insatisfactible (inferior al 40 %), marginal (entre el 40 % y el 60 %) y satisfactible (desde el 60 % al 100 %).

La figura §3.9 ilustra las transformaciones necesarias desde que se mide un atributo hasta que se obtiene la valoración global de la calidad de un productos.

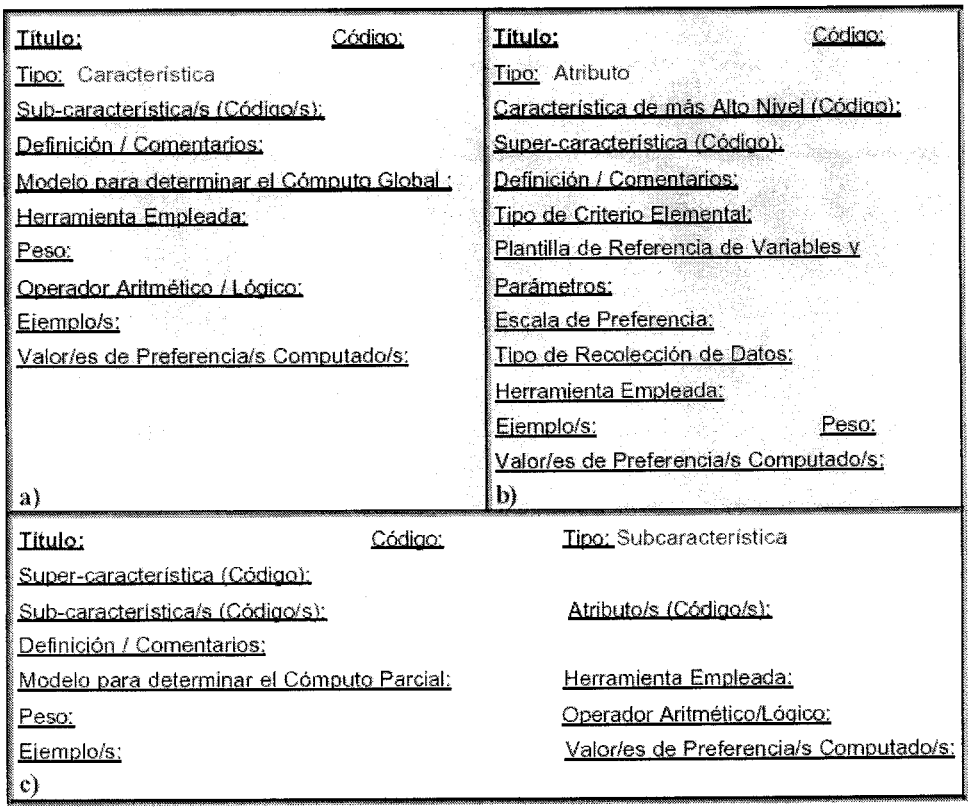

Figura 3.6: *Plantillas para especificar características, subcaracterísticas y atributos.*

Título: Soporte a Lenguaje Extranjero
Código: 1.4.1 Tipo: Atributo
Característica de más alto nivel: Usabilidad
Super-característica: Misceláneas
Definición / Comentarios: Este atributo modela el número de lenguajes extranjeros soportado por un sitio. Además, especifica el nivel de soporte para cada lenguaje, a saber: total (todas las páginas del sitio), parcial (sólo algunos subsitios), o mínimo (algunas páginas o documentos de algunos subsitios). El lenguaje nativo del sitio WEB no se computa como lenguaje extranjero.
Tipo de criterio elemental: Es un criterio multi--variable, continuo y absoluto.
Plantilla de referencia de variables y parámetros: Ver figura § 3.8.
Escala de preferencia: La escala por defecto.

Figura 3.7: *Especificación de un atributo de calidad con el modelo de Olsina.*

3.2.6.4. Valoración

Si bien el uso de métricas directas es imprescindible para medir atributos, entendemos que el uso de métricas indirectas puede ser sustituido por otro tipo de funciones que permitan medir características, subcaracterísticas y atributos. De este modo se consigue reducir el tiempo necesario para definir un atributo pues en general, comprobar la condición de representación de una métrica es una tarea que consume mucho tiempo.

La descripción informal del significado de los atributos resultando inadecuada para realizar actividades donde se requiere precisión, pues los problemas de ambigüedad inherentes al lenguaje natural puede llevar a interpretaciones y por tanto decisiones incorrectas: comprar un producto inadecuado, rechazar un producto adecuado.

Las medidas de los atributos no siempre pueden ser exactas. Por ejemplo, las magnitudes que dependen del tiempo, tales como el tiempo de respuesta, el tiempo de recuperación, la disponibilidad, incluso a veces el rendimiento, necesitan ser caracterizadas con indicadores estadísticos (media, varianza, percentiles, etcétera) o bien expresadas mediante rango de valores (valores máximo y mínimo). En otras ocasiones, puede que simplemente no sea posible conocer, por ejemplo, un proveedor de video bajo demanda puede garantizar que cada película está en uno de tres formatos conocidos, pero no puede conocer el formato concreto hasta que dispone

Aspecto	Descripción	Valores, Parámetros, Ejemplos
Código y Nombre del Atributo	El código debe ser único, en correspondencia con el árbol de requerimientos citado	1.4.1 (figura 9.1) Soporte a Lenguaje Extranjero
Definición	Este atributo modela el número de lenguajes naturales extranjeros soportados por sitios de museos típicos, y, además, el nivel de soporte para cada lenguaje. No se computa el lenguaje nativo del sitio web, como lenguaje extranjero.	Ejemplos: el sitio del museo Louvre (al Oct-1998) posee soporte parcial a tres lenguajes extranjeros (japonés, español e inglés). Por otra parte, el sitio del museo del Prado posee soporte total del lenguaje inglés
Fecha de Medición	Se especifica la fecha (o rango entre dos fechas), en la que se realizará la recolección de datos y se computará la variable de calidad para los parámetros planificados.	F=[dd-mm-aaaa] o desde F_1 a F_2
Parámetros *(y Variables) Disponibles*	Cantidad de Lenguajes Extranjeros, para el nivel de soporte Si Nivel de soporte, a saber: Total (en todo el sitio); Parcial (algunos subsitios del sitio); Mínimo (algunas páginas o documentos).	N_i S_i i =(1 .. n) para n=3 S_1= 0.2 -> soporte mínimo; S_2=1 -> soporte medio; S_3=2 -> soporte total.
Valores Planificados	Se indican algunos casos modelados, no de modo extensivo. Caso 1) Caso 2) Caso 3) Caso 4)	 $N_1 = 0$; $N_2 = 0$; $N_3 = 2$ y $S_3 = 2$ $N_1 = 0$; $N_2 = 1$; $N_3 = 1$ y $S_2 = 1$; $S_3 = 2$ $N_1 = 0$; $N_2 = 2$; $N_3 = 0$ y $S_2 = 1$ $N_1 = 5$; $N_2 = 2$; $N_3 = 0$ y $S_1 = .2$; $S_2 = 1$
Mayor grado de Satisfacción	El mayor valor de X, dentro de los números reales, obtenido de computar la función elemental y que se traducirá en una preferencia de calidad del 100%	Si X >= 100 $\rightarrow$ $IE_i = 100\%$
Función elemental	$X = 30 * \sum_i S_i * N_i$	Si X resulta mayor al valor 100, entonces se computa a X = 100
Referencias	Distintas fuentes de referencia	Ver la Plantilla del atributo del mismo código para el documento con el caso de estudio de museos en la Web (9.2.2)

Figura 3.8: *Ejemplo de plantilla de referencia de variables y parámetros para el atributo Soporte a Lenguaje Extranjero.*

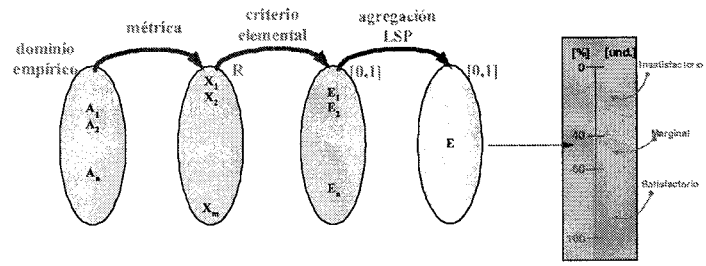

Figura 3.9: *Transformaciones entre dominios del modelo de evaluación de Olsina.*

de dicha película. De este modo, si un cliente desea contratar este servicio durante un período determinado no puede saber a priori que formato concreto tendrá.

3.2.7. Resumen

Hay una característica común a todos los modelos de segunda generación: la utilización de métricas para conseguir que la valoración de los atributos de calidad sea objetiva. Por otra parte, los últimos modelos incorporan la noción de función de satisfactibilidad de un atributo, de modo que a partir del valor medido se puede conocer objetivamente el grado de satisfactibilidad que dicho valor representa.

Respecto a la especificación de requisitos, se comienza a distinguir entre condiciones y ofertas. No obstante, los requisitos siguen sin ser elementos de primera clase, por lo que los documentos de requisitos son en realidad, documentos de atributos. La no separación de atributo y requisito reduce las probabilidades de reutilizar el catálogo de atributos, pues la función de satisfactibilidad de un atributo depende de los criterios de los participantes y también puede ser diferente para cada producto y cada una de sus partes. Pensemos en un atributo que tiene connotaciones diametralmente opuestas según el punto de vista desde el que lo estemos considerando. Por ejemplo, el valor *Java* puede ser ideal para el atributo *Lenguaje* desde el punto de vista de la portabilidad, pero poco adecuado desde el punto de vista de la eficiencia. Evidentemente, en este tipo de situaciones no es posible definir una función de satisfactibilidad válida, a

menos que esta se realice sobre un requisito y no sobre un atributo.

Por otra parte, las especificaciones de los catálogos y de los documentos de calidad se realizan de manera informal, en el mejor de los casos con plantillas, por lo que resulta complicado y arriesgado realizar con ellos operaciones que requieran precisión en el significado de los atributos y de los requisitos.

Respecto al proceso de intermediación, quizá lo más novedoso es la utilización de un modelo de valoración diferente al clásico aditivo–lineal y la utilización de análisis calidad/coste para seleccionar la oferta óptima.

Esta valoración de los modelos de segunda generación concuerda con el resultado de la encuesta publicada en [Kitchenham y Pfleeger 1996]. Ante la pregunta ¿qué problema considera es el más importante respecto a los atributos de calidad? El 28 % consideraba que era la especificación objetiva de requisitos de calidad, el 20 % consideraba que era el establecimiento *(setting–up)* de un sistema de gestión de calidad, el 18 % cumplir los requisitos de calidad, el 17 % la medición de la calidad y el 15 % restante la definición consensuada del catálogo de atributos.

3.3. Modelos de tercera generación

3.3.1. Servicio de intermediación de la OMG

El servicio de intermediación entre objetos distribuidos definido por la OMG [OMG 2000][5] y al que nos referiremos abreviadamente como *Trader*, tiene como principal objetivo encontrar el objeto que ofrece un determinado servicio y que mejor se adecúa a las preferencias del cliente que solicita la búsqueda. De este modo, la función que realiza el Trader puede interpretarse como un caso particular del problema de la búsqueda y selección óptima de productos.

[5]La versión de la especificación del *Trader* de 1996 fue aceptada por la ISO y la ITU-T para definir la función de intermediación de su modelo de referencia RM-ODP (*Reference Model–Open Distributed Processing*) [ITU/ISO 1995]

3.3.1.1. Catálogo de atributos

En el modelo de especificación del catálogo de atributos del *Trader*, se considera que los identificadores de los atributos (en su terminología *propiedades*) son autoexplicativos, universales y no necesitan descomponerse para poder obtener una escala de medida (ver sección §??). En otras palabras, la estructura del catálogo de atributos es plana.

Además del identificador y del dominio, cada atributo tiene asociado un *modo* que indica si es obligatorio asociarle un valor y si es posible modificar dicho valor posteriormente. Los cuatro valores que puede tomar son:

- *default*. La asignación es opcional y en caso de realizarse es posible la modificación posterior.
- *mandatory*. La asignación es obligatoria y es posible la modificación posterior.
- *readonly*. La asignación es opcional, pero no es posible la modificación posterior.
- *mandatory, readonly*. La asignación es obligatoria y no es posible la modificación posterior.

3.3.1.2. Especificación de requisitos

Los requisitos, en este caso condiciones, del objeto cliente se especifican con el lenguaje de restricciones de la OMG. Dicho lenguaje permite el uso de operadores aritméticos y relacionales, así como el uso de variables de tipo cadena, entero, real, enumerado y secuencia.

Merece la pena destacar, que el criterio de validez que establece la norma es sintáctico (que la restricción esté bien formada) permitiéndose situaciones potencialmente peligrosas. Por ejemplo, la restricción COSTE $\geq$ 5 **and** COSTE $\leq$ 4 es correcta desde un punto de vista sintáctico, pero no semántico, pues se trata de una restricción inconsistente ya que no existe ningún valor que pueda tomar la variable COSTE que satisfaga dicha restricción. De este modo, si un objeto cliente solicita la búsqueda de un objeto servidor que satisfaga la anterior condición, es evidente que el Trader nunca encontrará un objeto por más veces que realice la búsqueda.

Por su parte, los requisitos que satisface un objeto servidor, es decir su oferta, sólo pueden ser especificados mediante pares $(atributo, valor)$, es decir, restricciones en igualdad. Esto impide que se puedan especificar requisitos tan habituales como que la disponibilidad de un determinado servicio WEB estará entre el 95 % y el 99.99 %.

3.3.1.3. Evaluación y selección

La evaluación de un objeto distribuido se realiza en dos pasos. Primero se comprueba si el tipo[6] de dicho objeto es un *subtipo* del tipo del servicio solicitado por el cliente. La relación de subtipado se define de manera que sea posible la sustitución de cualquier objeto por un objeto cuyo tipo sea un subtipo del primero. Posteriormente se comprueba que los valores de las propiedades del objeto satisfacen la restricción correspondiente a las condiciones del cliente. La relación de subtipado se define como sigue: se dice que un tipo de servicio T_2 es un subtipo de un servicio T_1, si y sólo si:

1. El tipo de la interfaz de T_2 deriva o es del mismo tipo que el tipo de la interfaz de T_1.
2. Todas las propiedades de T_1 también están definidas en T_2.
3. Para cada una de las propiedades definidas en T_1 y en T_2 el modo de la propiedad en T_2 debe ser el mismo que, o *más fuerte que*, el modo de la propiedad en T_1.
4. Todas las propiedades de T_1 tienen un tipo valor que es un supertipo del tipo valor de la misma propiedad de T_2.

sabiendo que la relación *es más fuerte que*, que denotaremos por $>$, se define como:

$$mandatory, readonly > readonly = mandatory > default$$

[6]Según la norma, la información que caracteriza a un producto se denomina *tipo* y está compuesta por: i) los atributos que caracterizan los aspectos no funcionales o de calidad del producto, denominados *propiedades* y ii) la descripción en IDL de las operaciones de la interfaz del servicio, también conocida como el *tipo de la interfaz*. El tipo de la interfaz tiene asociado un identificador único dentro de un repositorio. Dicho identificador será utilizado por el proveedor (exportador) para indicar el tipo de interfaz (servicio) que está exportando y por el cliente (importador) para indicar el tipo de interfaz (servicio) que está buscando.

Dado que en dominio cabe la posibilidad de que sean varios los objetos que pasan satisfactoriamente el proceso de evaluación, es necesario disponer de algún mecanismo de selección. El *Trader* ofrece la posibilidad de devolver una lista de estos objetos ordenada de acuerdo con el criterio establecido por el cliente. Dicho criterio puede ser cualquiera de los siguientes:

- Un objeto O_1 es preferible a un objeto O_2 si el resultado de evaluar la expresión *exp* con los valores de las propiedades de O_1 es mayor que el obtenido al evaluar *exp* con los valores de las propiedades de O_2.
- Un objeto O_1 es preferible a un objeto O_2 si el resultado de evaluar la expresión *exp* con los valores de las propiedades de O_1 es menor que el obtenido al evaluar *exp* con los valores de las propiedades de O_2.
- Un objeto O_1 es preferible a un objeto O_2 si *exp* se evalúa a cierto con los valores de las propiedades de O_1 y se evalúa a falso con los valores de O_2.
- Un objeto O_1 es preferible a un objeto O_2 por azar.
- Un objeto O_1 es preferible a un objeto O_2 si O_1 fue encontrado antes que O_2.

Este modelo de selección y de evaluación asume un condicionante muy fuerte. Nos referimos a considerar que tanto el objeto servidor como el cliente utilizan el mismo catálogo de atributos de calidad. Suposición poco realista en el contexto de INTERNET, más que incluso asumir que comparten la jerarquía de tipos para las interfaces.

3.3.2. QML

QML (*Quality of Service Modeling Language*) es un lenguaje de especificación de requisitos de calidad de servicio desarrollado en los laboratorios de Hewlett Packard [Frølund y Koistinen 1998b]. QML está específicamente diseñado para facilitar la construcción de sistemas distribuidos sensibles a la calidad de servicio y más particularmente para aquéllos que utilizan IDL para describir las interfaces de los servicios que se requieren y proporcionan.

3.3.2.1. Catálogo de atributos

En QML los atributos de calidad se denominan *dimensiones*. Una atributo de calidad está asociado a un *tipo de contrato* que no es más que una facilidad sintáctica para agrupar atributos de calidad bajo el criterio que decida el diseñador. Las propiedades de un atributo de calidad son cinco, a saber:

- *Nombre*, identifica al atributo de manera única dentro de un tipo de contrato.
- *Dominio*, indica los valores que puede tomar el atributo: valores reales (**numeric**), valores enumerados (**enum**) o conjunto de valores enumerados (**set**). En el caso de los atributos de tipo enumerado y de tipo conjunto, la definición debe realizarse a la vez que la declaración, siendo posible definir una relación de orden existente entre estos valores.
- *Unidad*, indica la unidad en la que se expresa el valor del atributo.
- *Orden*, es una información utilizada para estudiar la conformidad entre requisitos. Puede tomar dos valores (**increasing**) y (**decreasing**). En el primer caso se suele hablar de dominios crecientes y en el segundo de dominios decrecientes.

La figura §3.10 muestra un catálogo de atributos especificado con QML que ha sido tomado de [Frølund y Koistinen 1998b]. Los atributos están agrupados en dos categorías o tipos de contratos: el tipo Rendimiento y el tipo Fiabilidad. Como puede observarse, la sintaxis concreta de QML es muy simple y el significado de cada atributo puede consultarse en la sección §5.2.

Por otra parte, la aportación original de su modelo está en ser de los primeros lenguajes en los que los catálogos de atributos no están vinculados a dominios específicos (sistemas en tiempo real, multimedia, etcétera), ni a categorías de requisitos de QoS (fiabilidad, rendimiento, etcétera), algo muy habitual hasta su aparición. Además, es independiente de los mecanismos utilizados para satisfacer los requisitos de QoS que especifica. Por ejemplo, permite expresar que no se admiten más de cinco fallos al año, pero no se indica qué mecanismo (replicación activa, primary–backup, etcétera) se utiliza para conseguir ese grado de fiabilidad.

```
type Fiabilidad = contract {
  numeroDeFallos : decreasing numeric no / year;
  tiempoReparacion : decreasing numeric sec;
  disponibilidad : increasing numeric %;
  erroresPosibles: increasing set {early, late, state, value, omission};
  falloServidor: increasing enum {halt, initialState, rollBack};
  semanticaOperacion: increasing enum {atLeastOnce, atMostOnce, once};
    with order { once < atLeastOnce, once < atMostOnce };
  reconexion: enum {true, false};
};
type Rendimiento= contract {
  retraso : decreasing numeric sec;
  productividad : increasing numeric mb / sec;
};
```

Figura 3.10: *Definición de un catálogo de atributos en* QML.

3.3.2.2. Especificación de requisitos

Los principales elementos para especificar requisitos en QML son dos: los *contratos* y los *perfiles*. Un contrato puede verse como un conjunto de condiciones predefinidas que pueden ser referidas de manera conjunta. Dichas condiciones siempre están referidas respecto a los atributos de un tipo de contrato. La figura §3.11 muestra la definición del contrato FiabilidadSistema, el cual establece condiciones sobre los atributos del tipo de contrato Fiabilidad especificado en la figura §3.10. Algunas de estas condiciones son: se requiere que no haya más de 10 errores al año, que la disponibilidad sea superior al 80 % y que al menos se controlen las situaciones en las que el servicio devuelva valores incorrectos (value) o realizar transiciones de estado incorrectas (state).

Como puede observarse, en el caso de atributos relacionados con magnitudes temporales, como es el caso del atributo tiempoReparacion, es posible establecer condiciones sobre medidas estadísticas relacionadas. Estas medidas estadísticas se denominan *aspectos* y hacen referencia a los percentiles, la media, la desviación y la frecuencia relativa. No obstante, los aspectos en QML no tienen semántica asociada, por lo que en realidad se tratan de identificadores carentes de significado. Por ejemplo, el lenguaje no comprueba que el percentil 50 sea menor que el percentil 80, o que la

```
interface ICuentaDeudora{
  bool aprobar (in string cod, in float cantidad );
  void reintegrar (in string cod, in float cantidad);
};
fiabilidadSistema = Fiabilidad contract {
  numeroDeFallos < 10 / year;
  tiempoReparacion { percentile 100 < 2000; mean < 500; variance < 10; };
  disponibilidad > 80;
  erroresPosibles ≥ {state,value};
  fallosServidor = initialState;
  semanticaOperacion > once;
  reconexion= true;
};
perfilCuentaDeudora for ICuentaDeudora = profile {
  require fiabilidadSistema;
  from aprobar require Rendimiento contract {
    retraso {
      percentile 50 < 10 msec; percentile 80 < 20 msec; percentile 100 < 40 msec;
      mean < 15 msec;
    };
  };
  from reintegrar require Rendimiento contract { retraso < 4000 msec; };
};
```

Figura 3.11: *Especificación de requisitos de calidad en* QML.

media sea inferior o igual al percentil 100.

La asociación entre contratos y productos se realiza mediante perfiles. La figura §3.11 muestra el perfil perfilCuentaDeudora definido para la interfaz ICuentaDeudora. Como puede observarse, en un mismo perfil es posible asociar varios contratos. La primera cláusula require del perfil indica que la implementación de esta interfaz debe satisfacer las condiciones del contrato fiabilidadSistema. La segunda cláusula require indica que la operación aprobar debe satisfacer condiciones especificadas en un contrato del tipo Rendimiento que es anónimo (no tiene nombre), sino que se define exclusivamente para dicha operación y por tanto no puede ser reutilizado. La última cláusula require indica que la operación reintegrar también debe satisfacer un contrato anónimo de tipo Rendimiento, aunque con unas condiciones concretas diferentes a las de la operación aprobar.

QML permite que los requisitos se puedan especificar sobre una interfaz en su conjunto y sobre cada operación individual, incluso sobre cada parámetro.

3.3.2.3. Evaluación y selección

QML puede ser utilizado en muchos dominios de aplicaciones y en diferentes fases del desarrollo. Los autores identifican dos posibles usos: la extensión de UML y la extensión de los middlewares utilizados en los sistemas sensibles a la calidad. En el primer caso, se propone el uso de perfiles (en el sentido de los profiles de QML que no de UML) en las relaciones de *dependencia* o *uso* y en las de implementación. En el caso de las relaciones de dependencia, la entidad origen podría expresar con un perfil la calidad que requiere de la entidad destino. En el caso de las relaciones de implementación, la entidad que implementa una interfaz podría indicar la calidad con la que se proporciona dicha interfaz.

La utilidad de esta extensión de UML para el diseño de sistemas que deben satisfacer requisitos de QoS es evidente, no obstante, los autores consideran que el mayor beneficio de QML está en su aplicación para extender las plataformas de ejecución de los actuales sistemas sensibles a la calidad de servicio. Según [Frølund y Koistinen 1999] un sistema sensible a la QoS (*QoS–aware system*) es aquel que conoce en cada momento la calidad de servicio que puede proporcionar a sus usuarios y la que requiere de su entorno de ejecución. Para construir este tipo de sistemas, es necesario que las especificaciones de QoS sean entidades de primera clase durante la ejecución del sistema, entendiendo por esto, que sea posible construir y comparar especificaciones de QoS en tiempo de ejecución. Esta última característica puede conseguirse en un nivel básico haciendo uso del servicio de intermediación de la OMG (sección §3.3.1), aunque no con todas las posibilidades que ofrece QML.

En cualquiera de los casos, sea en tiempo de diseño o en tiempo de ejecución, es necesario definir una relación que permita averiguar cuándo la calidad que ofrece una entidad es suficiente para satisfacer la calidad requerida por otra. QML denomina a esta relación *relación de conformidad* y la define de manera recurrente de la siguiente manera:

> *Un perfil P_1 es conforme con un perfil P_2 si los contratos asociados a P_1 son conformes con los contratos asociados a P_2. A su vez, un*

contrato C_1 es conforme con un contrato C_2 si todas los requisitos de C_1 son conformes a los requisitos de C_2.

Dado que un requisito se expresa como una restricción, la conformidad entre perfiles se reduce a un problema de conformidad entre restricciones.

La noción de conformidad entre dos restricciones propuestas por los autores de QML en [Frølund y Koistinen 1998b] depende del orden y del dominio de cada atributo. Para los dominios numéricos crecientes se dice que una restricción $c_1 \triangleq v\theta l_1$ es conforme con otra restricción $c_2 \triangleq v\theta l_2$, y se denota como $c_1 \rightarrow c_2$ sii $l_1 \geq l_2$, donde v identifica a un atributo, $\theta = \{>, \geq, =\}$ y l_1, l_2 son literales numéricos. Por ejemplo es fácil comprobar que si se satisface la restricción disponibilidad > 90 también se satisface la restricción disponibilidad > 85 y que también se verifica que $90 \geq 85$.

Para los dominios numéricos decrecientes sin embargo se tiene $c_1 \rightarrow c_2$ sii $l_1 \leq l_2$, donde $\theta = \{<, \leq, =\}$ y l_1, l_2 son literales numéricos. Por ejemplo, es fácil comprobar que si se satisface la restricción numeroFallos < 10 también se satisface la restricción numeroFallos < 20 y que también se verifica que $10 \leq 20$. Con un razonamiento análogo, los autores definen relaciones de conformidad para restricciones sobre atributos de tipo enumerado y conjunto, las cuales siguen dependiendo del dominio y del orden del atributo.

No obstante, y sin necesidad de mostrar en detalle la definición de conformidad en estas otras dos situaciones, es fácil comprobar que si bien QML permite especificar el mismo tipo de requisitos en el lado cliente que en el lado servidor, evitando una de las limitaciones de otras propuestas tales como el servicio de intermediación de la OMG (sección §3.3.1), lo hace a costa de sacrificar expresividad de dichas restricciones que han de tener siempre el formato variable operador literal. Esta falta de expresividad hace imposible especificar restricciones definidas sobre varios atributos (multidimensionales), o que expresan relaciones aritméticas o lógicas. Por ejemplo, ninguno de los requisitos del siguiente contrato serían válidos

coste $\geq$ 10 **and** coste $\leq$ 20
100-tiempoRecuperación/tiempoEntreFallos > 99.5
disponibilidad.mean – disponibilidad.variance > 95%
periodoUso $\geq$ Junio **and** periodoUso $\leq$ Septiembre

En el caso de los servicios WEB el uso de relaciones aritméticas resulta especialmente útil. Imaginemos un sistema que ha de decidir automáticamente entre dos servicios WEB que implementan la misma interfaz. El

primero oferta un tiempo medio de respuesta inferior a 50 segundos con una desviación típica de 10 segundos, es decir, que satisface las restricciones DELAY.mean $<$ 50 y DELAY.variance $<$ 10, y el segundo oferta un DELAY.mean $<$ 52 y DELAY.variance $<$ 2. Es evidente, que aunque el sistema seleccionaría el primer servicio, pues el segundo servicio no satisface las condiciones del cliente (su tiempo medio supera en dos segundos al máximo admitido), en la mayoría de las ocasiones el segundo servicio resultaría mucho más interesante, pues la variabilidad de su tiempo de respuesta sería significativamente menor.

Una forma de evitar situaciones como la anterior, en la que podemos dejar pasar ofertas interesantes, pasa por permitir restricciones que puedan utilizar expresiones aritméticas sobre los atributos. Si en el ejemplo anterior, el cliente hubiese expresado sus preferencias con la restricción (TTR.mean + TTR.variance) $<$ 55, hubiésemos elegido el segundo servicio en lugar del primero.

QML también restringe el operador que puede utilizarse para expresar la restricción. Si es un atributo creciente, no es posible utilizar los operadores $<$ y $\leq$; y si es decreciente no es posible utilizar los operadores $>$ y $\geq$. Por ejemplo, no se puede indicar un requisito del tipo numeroFallos $>$ 10, disponibilidad $<$ 90 %, que podría ser muy útil para el caso de realizar las pruebas de robustez de un determinado sistema.

Extender QML para solucionar las anteriores limitaciones expresivas supone una revisión profunda de la semántica operativa de la relación de conformidad, al menos tal y como está definida en [Frølund y Koistinen 1998a]. En nuestra opinión dicha revisión pasa necesariamente por interpretar la conformidad como un problema de satisfacción de restricciones.

3.3.2.4. Relaciones de refinamiento

QML permite establecer relaciones de refinamiento entre contratos y perfiles. La semántica de la relación de refinamiento es muy similar a la de relación de subtipado en el sentido de [Liskov y Wing 1994]. Un contrato B es un refinamiento de un contrato A si y solo si A y B son del mismo tipo y el contrato B es conforme con el contrato A. Por ejemplo, considerando la especificación de la figura §3.11 podemos definir un nuevo contrato refinando el contrato fiabilidadSistema como:

```
fiabilidadServidorNombres = fiabilidadSistema refined by
{
 numeroFallos < 8 / year;
 fallosServidor = rollBack;
 disponibilidad > 90;
}
```

En nuestra opinión este modo de ver el refinamiento limita innecesariamente posibilidades muy interesantes. Por ejemplo, no es posible que el contrato obtenido mediante refinamiento sea menos restrictivo que el contrato base. Los autores justifican su decisión indicando las ventajas que este tipo de refinamiento ofrece para realizar operaciones de sustitución de tipos, entendiendo por un tipo su aspecto funcional y su descripción de calidad. En nuestra opinión, esta justificación no resulta demasiado convincente, pues entendemos que las relaciones de refinamiento y de conformidad son ortogonales y por tanto ligar obligatoriamente el refinamiento con la conformidad limita innecesariamente las posibilidades de reutilización.

3.3.2.5. Herramientas de soporte

En [Frølund y Koistinen 1999] se presenta QRR (*QoS Runtime Representation*), una propuesta para manipular especificaciones de calidad de servicio en tiempo de ejecución. QRR proporciona una biblioteca de funciones C++ disponible como un servicio compatible CORBA para construir dichas especificaciones, las cuales se expresan en una notación híbrida IDL/C++. Además, la biblioteca proporciona funciones para comprobar la conformidad entre perfiles y un traductor de especificaciones QML. En nuestra opinión los principales inconvenientes de QRR son dos:

- Poca escalabilidad. QRR está diseñado para resolver restricciones con un formato muy simple, concretamente el formato de QML, por lo que no puede ser utilizado para trabajar con especificaciones de requisitos expresadas en otros lenguajes (NOFUN, QRL, etcétera), ni tan siquiera del servicio de intermediación de CORBA.
- Interoperabilidad nula. QRR considera que todas las especificaciones utilizan el mismo catálogo de atributos, al menos no proporciona ninguna función para adecuar especificaciones expresadas con catálogos de atributos diferentes.

3.3.2.6. Modelo de negociación

En [Koistinen y Seetharaman 1998] se propone un modelo de negociación de requisitos de calidad de servicio para dar respuesta a la reconfiguración y adaptación dinámica necesaria en aquellos sistemas donde la carga y los recursos varían con mucha frecuencia y casi siempre de manera inesperada. Una de las aportaciones originales y más importantes de este modelo, está en la posibilidad de realizar la negociación sobre documentos de requisitos cuyos atributos de calidad pueden estar organizados en múltiples categorías (tipos de contratos) independientes del dominio, pues hasta entonces todos los algoritmos de negociación se habían realizado asumiendo una única colección de atributos y centradas en el terreno de los sistemas multimedia distribuidos. A grandes rasgos, los principales pasos del algoritmo de negociación son:

1. El cliente le pide al proveedor de un servicio (en adelante servidor) los diferentes niveles de calidad (en adelante ofertas) con los que puede proporcionar dicho servicio. Esta petición puede ir acompañada de otras informaciones tales como la frecuencia de llamada de cada una de las operaciones del servicio.

2. El servidor devuelve las ofertas disponibles para dicho servicio que soportan la frecuencia de uso solicitada.

3. El cliente comprueba qué ofertas son conformes respecto a un nivel de calidad de referencia (en adelante condiciones).

4. El cliente elige la oferta óptima de entre todas las que son conformes, calculando para ello la preferencia global de cada oferta.

5. El cliente solicita la oferta óptima al servidor y éste comprueba si dicha oferta todavía sigue siendo válida.

6. En caso afirmativo, el servidor envía un mensaje de aceptación de contrato y se dará por finalizada la negociación. En caso contrario, se iniciará un nuevo ciclo de negociación, esto es, el servidor preparará una contraoferta y se la enviará al cliente, y la negociación seguirá en el paso 3.

El modelo propuesto para calcular la preferencia global de una oferta que proponen es muy similar a los utilizados en los modelos lineales

aditivos utilizados en los modelos de segunda generación. En este caso, la preferencia de la oferta se calcula a partir de la preferencia de los perfiles, que a su vez se calcula a partir de la preferencia de cada una de las operaciones de la interfaz asociada, que a su vez se calcula a partir de la preferencia de cada una de sus operaciones, que a su vez se calcula a partir de la preferencia de cada uno de los contratos asociados a cada operación, que a su vez se calcula a partir de la preferencia de cada requisito. El valor que representa a cada requisito en el computo de la preferencia se calcula como el valor medio de la función de preferencia en el intervalo de satisfacción del requisito. Es decir, si la función de preferencia para la disponibilidad viene dada por f entonces el valor que representa al requisito disponibilidad ≥ 90 sería $\frac{f(90)+f(100)}{2}$.

Las principales limitaciones de este modelo de negociación son:

- Sólo se contempla la negociación con un proveedor, lo que resulta poco práctico en sistemas abiertos como INTERNET en el que las negociaciones se suelen realizar sobre varios proveedores.
- Obliga a que todas las ofertas utilicen los mismos tipos de contratos y perfiles, lo que puede resultar especialmente difícil de conseguir cuando se trabaja con varios proveedores a la vez.
- Hasta la fecha, no se ha conseguido una implementación del protocolo que esté libre de bloqueos, incluso, los propios autores reconocen que es fácil que una negociación degenere en un ciclo infinito de intercambio de ofertas. Para evitar esta situación, los autores proponen modificar el algoritmo original, concretamente añadir información sobre: el número máximo de solicitudes y contraofertas que se permiten, el número máximo de solicitudes de ofertas que puede realizar el cliente, etcétera.
- El modelo de cálculo de la preferencia global es válido únicamente para requisitos representados por restricciones sobre un único atributo.
- El método de cálculo de la preferencia asume que el proveedor garantiza la uniformidad de los valores que satisfacen cada requisito de la oferta, y de ahí que se proponga el valor medio como valor representativo del requisito. En ocasiones esta consideración no es admisible, por lo que no se podría llevar a cabo la negociación. En la sección §5.1.2.4 se describe una posible solución para estos casos.

- Añadir la información sobre la frecuencia de uso separadamente de los requisitos resulta innecesario, pues no existe ninguna razón que impida contemplarla como un atributo más.

3.3.3. NoFun

NoFun (*NOn–FUNctional*) es un lenguaje formal de especificación de requisitos no funcionales diseñado por el grupo de investigación en ingeniería de requisitos de la Universidad Politécnica de Cataluña [Franch 1998, Burgués y Franch 2000, Botella *et al.* 2001]. NoFun ha sido el primer y único lenguaje formal orientado al producto desarrollado en España hasta la aparición de QRL [Ruiz-Cortés *et al.* 2001a]. En el ámbito internacional, no conocemos hasta la fecha ningún otro lenguaje que supere sus posibilidades para especificar catálogos de atributos de calidad.

La primera versión de NoFun fue presentada en la *International Conference on Requirements Engineering* de 1998 [Franch 1998]. Diseñado inicialmente para ser utilizado con tipos abstractos de datos de bibliotecas tipo LEDA y STL [Franch *et al.* 1997], las revisiones realizadas para ser utilizado en el contexto de la programación orientada a componentes [Franch *et al.* 1999, Franch y Pastor 2000] y su validación industrial en el contexto de los ERPs (*Enterprise Resource Planning*) [Burgués *et al.* 2000, Pastor *et al.* 2001] han dado origen a una nueva versión de NoFun [Burgués *et al.* 2000, Botella 2001].

En [Botella *et al.* 2001], NoFun es presentado como el elemento central de su estrategia para reducir los problemas derivados del uso del lenguaje natural en los actuales catálogos de atributos[7].

3.3.3.1. Catálogo de atributos

NoFun utiliza como referencia el catálogo de atributos propuesto en la norma ISO 9126, por lo que permite la especificación de atributos en tres niveles: características, subcaracterísticas y atributos, conocidos en NoFun como *entidades de calidad*. Los dominios de los atributos pueden ser especificados formalmente, la figura §3.12 muestra dos dominios definidos por usuario: el dominio AREAS_COMPAÑIA como una enumeración, y

[7]En realidad, los autores de NoFun se refieren a su catálogo de atributos como "modelo de calidad" al igual que ocurre en las normas ISO 9126 e ISO 14598.

el dominio ESCALA_SOBREDIMENSION como una enumeración con una relación de orden (**ordered**) incorporada, con el fin de que sus valores puedan ser comparados en relaciones *menor que* y *mayor que*.

NOFUN distingue entre *atributos básicos* y *atributos derivados*, aquéllos que se definen a partir de otros atributos. NOFUN proporciona tres constructores de dominios para facilitar la definición de atributos complejos: los constructores **function**, **tuple** y **set**. La figura §3.12 muestra varios ejemplos de atributos básicos y derivados, así como el uso de los tres tipos de constructores.

La especificación de características y subcaracterísticas también se realiza mediante módulos. La figura §3.13 muestra la especificación de la subcaracterística Exactitud[8] para la que se precisa importar varios módulos de atributos; y la característica Funcionalidad que precisa importar varios módulos de subcaracterísticas. Nótese que para simplificar la escritura de la especificación, se puede omitir la declaración explícita de características y subcaracterísticas, pues ésta puede ser inferida directamente a partir de la definición.

Descomposición estructural. NOFUN permite que la evaluación de un producto se obtenga a partir de las evaluaciones de los subproductos que conforman. La figura §3.14 muestra cómo definir el atributo que representa el grado de comprobación que admite un ERP como el mínimo del conjunto formado por los valores correspondientes al valor del grado de comprobación de cada una de las partes del producto.

Refinamiento del catálogo de atributos. NOFUN ofrece la posibilidad de definir módulos de características, subcaracterísticas, atributos y dominios mediante *refinamiento*. De este modo se facilita en gran medida la definición de nuevos catálogos. La figura §3.15 muestra varios ejemplos de uso del refinamiento en NOFUN. Por ejemplo, el dominio AreasCompañia es abstracto pues está definido en un módulo abstracto (obsérvese el uso de la palabra reservada **abstract**) y la definición de los valores no se proporciona sino que se delega a los módulos que refinen dicho módulo. En la figura §3.16, se proporciona una definición para el atributo AreasCompañia en el módulo SPA3_EXACTITUD.

[8]Este y todos los demás ejemplos de NOFUN utilizados en este capítulo han sido tomados de [Botella *et al.* 2001].

```
domain module AREAS_COMPAÑIA
 explanation Áreas o funciones de la compañía
 domain AreasCompañia
   defined as Comercial, Logística, Manufacturación, RecursosHumanos
end AREAS_COMPAÑIA

domain module ESCALA_SOBREDIMENSION
 explanation Escala de cinco valores de penalización por cobertura excesiva de
 características
 domain ordered EscalaSobredimension
   defined as Inexistente, Bajo, Excesivo, Medio, Alto
end ESCALA_SOBREDIMENSION

attribute module DATOS_ENTREGA
 explanation Fecha de entrega de los componentes
  attribute Mes declared as Integer[1..12]
  attribute Año declared as Integer[1970..]
  attribute Fecha derived
   declared as tuple (Integer[1..12], Integer[1970..])
   defined as (Mes, Año)
 explanation Nombre de la compañía que proporciona el producto.
  attribute Proveedor declared as string
end DATOS_ENTREGA

attribute module ORIENTACION
 imports AREAS_COMPAÑIA, ESCALA_SOBREDIMENSION
 attribute Cobertura
  explanation Grado en el que un ERP cubre las diferentes áreas de una compañía
  declared as function from AreasCompañia to EscalaSobredimension
                default Inexistente
 attribute ObjetivosPrincipales
  explanation Áreas de la compañía que son cubiertas adecuadamente por un ERP
  declared as set of AreasCompañia
  defined as set of a in AreasCompañia such that Cobertura(a)= Alto
end ORIENTACION
```

Figura 3.12: *Definición de dominios y de atributos en* NOFUN.

```
subcharacteristic module EXACTITUD
  imports ORIENTACION, ... // todos los módulos necesarios
  subcharacteristic Exactitud derived
    explanation Subcaracterística "Accuracy" de la ISO para el dominio de los ERPs
    defined as Tuple (DestinoPrincipal, ...)
end EXACTITUD

characteristic module FUNCIONALIDAD
  imports EXACTITUD, CONFORMIDAD, SEGURIDAD, INTEROPERABILIDAD, ...
  subcharacteristic Funcionalidad derived
    explanation Característica de funcionalidad de la ISO para el dominio de los ERPs
    defined as Tuple (..., Conformidad, Seguridad, Interoperabilidad, ...)
end FUNCIONALIDAD
```

Figura 3.13: *Especificación de características y subcaracterísticas de un ERP en* NoFun.

```
component hierarchy ESTRUCTURA
  ModuloERP part of ERP
end ESTRUCTURA

attribute module GRADO_COMPROBACION for ModuloERP
  explanation Grado de comprobación medido en la escala 0.0 .. 1.0
  attribute GradoComprobación declared as Real[0.0..1.0]
end GRADO_COMPROBACION

attribute module GRADO_COMPROBACION for ERP
  explanation ERP testing degree out of ModuloERP testing degree
  attribute GradoComprobacion derived
    declared as Real[0.0..1.0]
    defined as MIN m: m in ModuloERP: m.GradoComprobacion
end GRADO_COMPROBACION
```

Figura 3.14: *Usando la descomposición estructural para definir atributos en* NoFun.

abstract domain module AREAS_COMPAÑIA
domain AreasCompañia
explanation Áreas o funciones de la compañía
// La definición no se incluye porque es un dominio abstracto
end AREAS_COMPAÑIA

domain module ESCALA_SOBREDIMENSION
explanation Escala de cinco valores de penalización por cobertura excesiva de características
domain ordered EscalaSobredimension
defined as Inexistente, Bajo, Excesivo, Medio, Alto
end ESCALA_SOBREDIMENSION

abstract attribute module ORIENTACION
imports AREAS_COMPAÑIA,ESCALA_SOBREDIMENSION
attribute Cobertura
explanation Grado en el que un ERP cubre las diferentes áreas de una compañía
declared as function from AreasCompañía **to** EscalaSobredimension
default Inexistente
abstract attribute ObjetivosPrincipales **derived**
explanation Áreas de la compañía que son cubiertas adecuadamente por un ERP
declared as set elements AreasCompañia
end ORIENTACION

abstract subcharacteristic module GENERAL_EXACTITUD
imports ORIENTACION, ... // todos los módulos necesarios
subcharacteristic EXACTITUD **derived**
explanation Accuracy ISO/IEC subcharacteristic bound to ERP domain
defined as Tuple (DestinoPrincipal, ...)
end GENERAL_EXACTITUD

abstract subcharacteristic module EXACTITUD_COMPAÑIAS_PEQUEÑAS
refines GENERAL_EXACTITUD
domain AreasCompañia
defined as Comercial, Logística, Manufacturación, RecursosHumanos
end EXACTITUD_COMPAÑIAS_PEQUEÑAS

Figura 3.15: *Definición de paquetes abstractos en* NoFun.

```
subcharacteristic module ACME_EXACTITUD for ACME
  refines EXACTITUD_COMPAÑIAS_PEQUEÑAS
  attribute ObjetivosPrincipales derived
    defined as set of a in AreasCompañia such that Cobertura(a) ≥ Medio
end ACME_EXACTITUD

subcharacteristic module SPA3_EXACTITUD for SPA3
  refines EXACTITUD_COMPAÑIAS_PEQUEÑAS
  domain AreasCompañia
    defined as Comercial, Logística, Manufacturación, RecursosHumanos, Técnico
  attribute ObjetivosPrincipales derived
    defined as set of a in AreasCompañia such that Cobertura(a) = Alto
end SPA3_EXACTITUD
```

Figura 3.16: *Refinamiento de paquetes abstractos en* NOFUN.

3.3.3.2. Especificación de condiciones y ofertas

NOFUN interpreta y especifica los requisitos de calidad como restricciones sobre entidades de calidad. Además de su especificación formal, la definición de un requisito contempla su descripción informal y el grado de importancia durante el proceso de evaluación, que puede ser esencial, importante, aconsejable y marginal. Los requisitos también se organizan en módulos, la figura §3.17 muestra la especificación de dos módulos de requisitos.

En cuanto al modelo de especificación de las ofertas, NOFUN utiliza el mismo modelo que el *Trader* de la OMG (ver sección §3.3.1), esto es conjunto de pares (propiedades,valor) (no sería posible especificar cobertura $\in$ [Medio, Alto]. Esta asignaciones son descritas en los módulos de descripción (ver figura §3.17).

3.3.4. Modelo de negociación de requisitos Win–Win

La negociación de requisitos es una actividad que atrae la atención de muchos investigadores. Tal y como se destaca en [In *et al.* 2002], en las últimas ediciones de la ICSE (*International Conference on Software Engineering*) ha sido un tema referido por varios conferenciantes invitados (De Marco,

requirement module PROPIEDADES_ORIENTACION **on** ORIENTACION
explanation Propiedades generales de los atributos de ORIENTACION
definition
prop_1: **essential**
explanation Los productos ERP deben cubrir al menos un área de la compañia
defined as
exists a **in** AreasCompañia **such that** Cobertura(a) > Inexistente
end PROPIEDADES_ORIENTACION

requirement module RESTRICCIONES_FECHAS **on** DATOS_ENTREGA **for** ACME
explanation Requisito sobre la fecha de creación de los productos de ACME
definition
fechaEntregaACME: **advisable**
explanation La fecha de fabricación de los productos de ACME debe ser posterior a Abril de 1998, fecha en la que ACME comenzó a utilizar tecnologías OO
defined as
Proveedor = "ACME" **implies**
(Date.Año > 1998) **or** (Date.Año = 1998 **and** Date.Mes ≥ 4)
end RESTRICCIONES_FECHAS

description module FUN_PROD[ORIENTACION]
Cobertura(Comercial,Financiera, RecursosHumanos)= Alto
Cobertura(Logistica)= Medio
Cobertura(Produccion)= Medio
end FUN_PROD

Figura 3.17: *Definición de condiciones y ofertas en* NoFun.

Yourdon y Weiser entre otros). En [Boehm y In 1996] se hace referencia a varios proyectos reales que han fracasado principalmente por una pobre negociación de requisitos y muy particularmente de requisitos de calidad.

La negociación es una actividad común a la mayoría de procesos de ingeniería de requisitos [Boehm *et al.* 1994, Pohl 1997, Sawyer y Kontoya 1999, Durán 2000], aunque sin duda alguna es el modelo en espiral de Boehm [Boehm *et al.* 1994] el que dispone del modelo de negociación más refinado [Boehm *et al.* 1994, Boehm *et al.* 1995, Lee 1996, In *et al.* 2002], para el que incluso existe una herramienta CASE (QARCC, *Quality Attribute Risk and Conflict Consultant*) que automatiza la identificación de conflictos entre requisitos de calidad utilizando una base de conocimiento auxiliar [Boehm y In 1996].

El modelo de negociación Win–Win se fundamenta en la teoría *W* propuesta en [Boehm y Ross 1989], cuyo principio básico es: "*Make everyone a winner*", esto es, conseguir que el sistema desarrollado satisfaga las expectativas de todos sus participantes.

La figura §3.18(a) ofrece una visión general del proceso de negociación de Win–Win tal y como se propone en [Lee 1996]. En un desarrollo en el que participan múltiples participantes, es frecuente la existencia de *conflictos* entre las condiciones ganadoras (*win conditions*), i.e. los requisitos individuales, de cada participante. Por ejemplo, el plazo de ejecución y el presupuesto propuestos por el director del proyecto suele ser inferior a los que propone el arquitecto del sistema. Durante la negociación, estos conflictos se identifican y cada participante puede proponer una o varias *opciones* de solución para cada una de las condiciones ganadoras implicadas en el conflicto. Si se logra solucionar todos los conflictos, se dice que se ha alcanzado un *acuerdo* que se plasma en un documento que recoge una versión revisada de las condiciones ganadoras iniciales aceptada por todos los participantes.

A continuación describimos con un poco más de detalle los principales artefactos que se obtienen o que son utilizados durante la negociación, a saber: las *condiciones ganadoras*, los *conflictos*, las *opciones* y los *acuerdos*.

3.3.4.1. Condición ganadora

Una condición ganadora es cualquier requisito que un participante considera importante y beneficioso. Asumiendo la existencia de un espacio

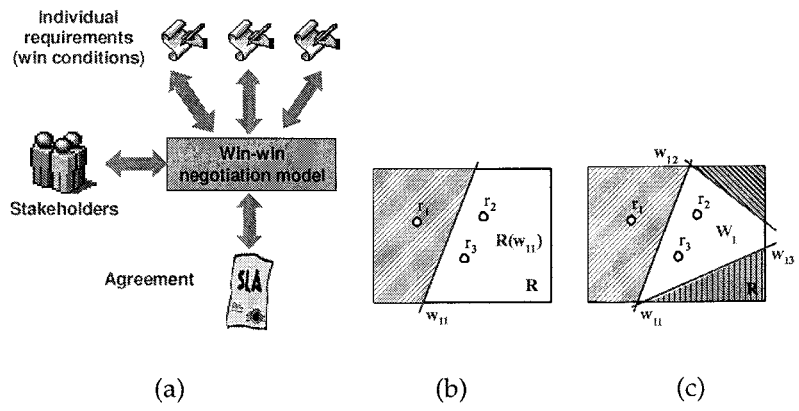

Figura 3.18: *a) Visión general del proceso de negociación de* Win-Win *b-c) Interpretación gráfica de una condición y de una región ganadora.*

R que contiene a todas las posibles especificaciones de requisitos (donde cada elemento de R es un conjunto de especificaciones funcionales, de rendimiento, de interfaces y de atributos), cualquier condición ganadora determina dos subconjuntos disyuntos de R, el conjunto de los requisitos que satisface la condición ganadora y el conjunto de requisitos que no la satisface. El conjunto de condiciones ganadoras de cada participante determina una región de R, denominada *región ganadora*. La figura §3.18(b) muestra la representación gráfica de una única condición ganadora y la figura §3.18(c) la representación de tres condiciones ganadoras que determinan la región ganadora del primer participante.

Estos conceptos se representan formalmente haciendo uso de la teoría de conjuntos. De este modo, la región ganadora del i-ésimo participante que se denota como W_i, se define como

$$W_i = \bigcap_{j=1}^{n} R(w_{ij})$$

donde $R(w_{ij})$ representa al conjunto de requisitos que satisfacen la condición ganadora j-ésima del i-ésimo participante, esto es

$$R(w_{ij}) = \{r \mid r \textbf{ satisface } w_{ij}\} \tag{3.1}$$

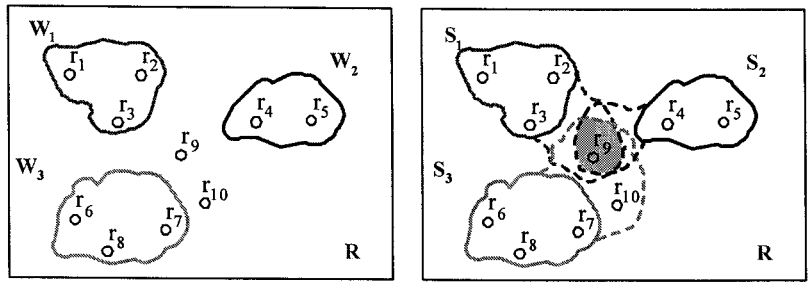

Figura 3.19: *a) Conflicto entre tres participantes b) Relajación de condiciones ganadoras para resolver un conflicto.*

Conviene hacer notar, que aunque los autores de Win–Win indican que las condiciones ganadoras son los requisitos de cada uno de los participantes, a juzgar por la estructura utilizada para definirlas (puede verse consultarse con detalle en [Lee 1996]), sería más conveniente denominarlas *objetivos* pues suelen expresarse con poca precisión y no suelen estar acompañados de criterios de aceptación con lo cual no son *verificables* [Davis 1993, IEEE 1993], propiedad muy deseable y que en nuestra opinión establece la diferencia entre requisito y objetivo.

3.3.4.2. Conflicto

Cuando las regiones ganadoras de al menos dos participantes son disyuntas, se dice que existe un conflicto entre ambos participantes. La figura §3.19(a) muestra una hipotética situación de conflicto entre tres participantes.

A partir de un conjunto de condiciones ($\{w_{ij}\}$) pueden surgir diferentes conflictos, cada uno de los cuales estará motivado por un subconjunto diferente (aunque no necesariamente disjuntos) de condiciones ganadoras. Cada uno de los conflictos existentes para un conjunto de condiciones dado se denota como I_k y la condición necesaria y suficiente que determina la existencia de cada conflicto I_k viene dada por:

$$\bigcap_{w_{ij} \in I_k} R(w_{ij}) = \emptyset \tag{3.2}$$

Por ejemplo, supongamos que las condiciones ganadoras del cliente son

w_{11} = (presupuesto, 6000€)
w_{22} = (interoperabilidad, {sockets, CORBA 2.2})

y las condiciones ganadoras del arquitecto son

w_{21} = (presupuesto, 7000€)
w_{22} = (interoperabilidad, {SOAP})

En este caso, tendríamos dos conflictos diferentes: $I_1 = \{w_{11}, w_{21}\}$ e $I_2 = \{w_{12}, w_{22}\}$.

3.3.4.3. Opciones

Cuando un conflicto es detectado, el conjunto de condiciones ganadoras involucradas es enviado (*issued*, de ahí la que los conflictos se denoten por I_k) a los participantes involucrados, a fin de que "se sienten a negociar" y encuentren una solución al conflicto. La solución de un conflicto requiere que al menos un participante *relaje* una o varias de sus condiciones ganadoras. La interpretación gráfica de la relajación de una condición se muestra en la figura §3.19(b). La región ganadora del participante que especificó la condición se aproxima a la de los demás participantes en conflicto, aumentando de este modo las posibilidades de encontrar un punto (en este caso requisito) en el que todos los participantes están de acuerdo.

Dada una condición ganadora w_{ij}, por regla general existen varias opciones para relajarla. Cada opción de relajación se denota como $\{\triangle w_{ij}\}_l$ y cada condición relajada como $\{w'_{ij}\}_l$, donde el subíndice l indica que se trata de la l-ésima relajación que puede aplicarse a la condición. Para el ejemplo visto anteriormente, una opción de relajación válida para la condición w_{11} del conflicto I_1 podría ser $\{\triangle w_{11}\}_1$ = (presupuesto, 1000€).

El proceso de acercamiento de posturas y de obtención de la opción es más complejo que el modelo que acabamos de describir y forma parte de la vista dinámica de Win–Win. No obstante, para los propósitos de nuestro trabajo no es necesario profundizar más.

3.3.4.4. Acuerdos

En general, las posibilidades de solucionar un conflicto dependen de la relajación que cada participante está dispuesto a realizar sobre sus condiciones ganadoras, pudiéndose dar el caso de que un mismo conflicto tenga varias soluciones. En cualquier caso, la condición que debe cumplirse para que un conjunto de condiciones relajadas constituyan un acuerdo de solución A_k es

$$\bigcap_{\{w'_{ij}\}_l \in A_k} R(\{w'_{ij}\}_l) \neq \emptyset \tag{3.3}$$

La unión de las todas las opciones que está dispuesto a aceptar un participante en un momento determinado se conoce como su *región de satisfactibilidad*, que se define como

$$S_i = \bigcap_{j=1}^{n_i} R(w'_{ij}) - W_i$$

donde

$$R(w'_{ij}) = R(w_{ij}) \cup R(\triangle w_{ij})$$

De este modo, la búsqueda de una opción de solución puede ser interpretada como una búsqueda sobre la *región de satisfactibilidad* de cada participante en conflicto.

3.3.4.5. Detección semiautomática de conflictos

En [Boehm y In 1996] se propone QARCC (*Quality Attribute Risk and Conflict Consultant*), una herramienta para identificar de un modo semi–automático conflictos potenciales entre requisitos de calidad. El elemento central de QARCC es una *base de conocimiento* (KB) que almacena información sobre los conflictos que pueden establecerse entre los atributos de calidad de una determinada taxonomía.

La tabla §3.1 muestra un fragmento de una hipotética base de conocimiento. Cada atributo de la taxonomía tiene una entrada en la KB. Como fruto de la experiencia, para cada atributo se conocen una o varias estrategias arquitectónicas para conseguir la propiedad asociada a dicho atributo y el tipo de impacto (positivo o negativo) que se ejerce sobre otros atributos. Por ejemplo, una estrategia seguida habitualmente para conseguir portabilidad es la división por niveles, la cual refuerza la facilidad de uso y

Atributo principal	Estrategia arquitectónica	Atributos reforzados	Atributos perjudicados
Seguridad	Comprobar entradas	Interoperabilidad, Facilidad de uso	Coste temporal y económico, rendimiento
	Redundancia		Coste temporal y económico, Rendimiento, Facilidad de uso, Facilidad de cambio
Portabilidad	División por niveles	Interoperabilidad, Facilidad de uso	Coste temporal y económico, rendimiento

Cuadro 3.1: *Ejemplo de base de conocimiento de QARCC tomada de [Boehm y In 1996].*

la interoperabilidad, a cambio de reducir rendimiento y aumentar el tiempo y el coste de desarrollo.

El proceso para identificar conflictos es muy simple: cada vez que se añade una condición ganadora se comprueba (automáticamente) si alguno de los atributos asociados a las condiciones ganadoras previamente registradas coincide con los atributos a los que perjudica la estrategia que debe aplicarse para satisfacer el atributo de la nueva condición. En caso afirmativo se habrá identificado la existencia, sólo probable, de un conflicto.

3.3.4.6. Selección sistemática de acuerdos

Dado que es posible obtener varios acuerdos para resolver un mismo conflicto, es necesario disponer de algún método que permita seleccionar el acuerdo más adecuado. En el modelo Win–Win original no se propone ningún método que permita sistematizar esta tarea, sin embargo en [In *et al.* 2002] se propone una extensión del modelo original que sí lo consigue.

La propuesta se basa en el uso de las técnicas de decisión multicriterio [Keeney y Raiffa 1976], más concretamente en el uso de funciones de eva-

luación lineales y aditivas del tipo $v = \sum_{i=1}^{n} w_i c_i$ donde v es la preferencia asociada a un conjunto de características cuyos valores individuales vienen dados por $\{c_i\}$ y donde la importancia de cada una en la preferencia global viene indicada por w_i. Para garantizar que la preferencia global es un valor real entre 0 y 1, se exige que $c_i \in [0, 1]$ y que $\sum_{i=1}^{n} w_i = 1$.

Por razones de espacio no describimos con detalle el método aunque si indicamos sus tres pasos principales. Inicialmente se calcula la preferencia que cada acuerdo A_k tiene para cada participante. Posteriormente se ordenan los acuerdos según el valor de su preferencia, si todos los participantes consideran como mejor valorado el mismo acuerdo, el método finaliza pues ya se conoce el mejor acuerdo, en caso contrario, se procede a un tercer paso en el que utilizando unas características y pesos consensuados entre todos los participantes se calcula la preferencia de cada acuerdo, la cual forzosamente determinará el mejor acuerdo.

3.3.5. Negociación automática en sistemas múltiagente

El interés por automatizar la negociación de requisitos de calidad no es exclusivo de los ingenieros de requisitos, sino que es compartido, al menos, por los diseñadores de sistemas distribuidos sensibles a la calidad [Koistinen y Seetharaman 1998], en particular para aplicaciones basadas en servicios WEB [Su *et al.* 2000, Corchuelo *et al.* 2001, Ruiz-Cortés *et al.* 2001a, Ruiz-Cortés *et al.* 2002a], y en nuestra opinión, también por los diseñadores de sistemas de múltiples agentes (MAS, *Multi–Agent System*) [Jennings *et al.* 2001].

Por regla general, las transacciones comerciales tradicionales (no electrónicas) entre clientes y empresas están precedidas de una negociación sobre el precio, el plazo de entrega, la calidad del producto y del servicio, etcétera. Dicha negociación suele ser llevada a cabo por las personas involucradas en la transacción o por sus representantes. En el terreno de los MAS y en particular en los sistemas de comercio electrónico, es deseable que la negociación pueda realizarse tan automáticamente como sea posible, reduciendo al mínimo la intervención humana.

En esencia, el proceso de negociación entre agentes, no difiere del proceso de negociación entre personas, por lo que cabe esperar que las soluciones existentes en el terreno de la negociación entre agentes puedan ser aplicadas, con todos los condicionantes necesarios, a la negociación entre personas. En este sentido, y de acuerdo con los trabajos de algunos de los

autores más reconocidos en el área de la negociación entre agentes [Sierra *et al.* 1997, Faratin *et al.* 2000, Jennings *et al.* 2001], la noción de negociación de referencia es la que se propuso hace más de 20 años en [Pruitt 1981], a saber:

> *"el proceso por el que se consigue llevar a cabo una acción conjunta entre dos o más participantes que no pudieron alcanzar un acuerdo con sus propuestas iniciales".*

Es evidente que se trata de una definición poco precisa, por lo que es posible derivar definiciones más concretas para diferentes contextos: análisis de requisitos, búsqueda de componentes y servicios WEB, tanto en tiempo de diseño como en tiempo de ejecución, etcétera. En [Sierra *et al.* 1997] se presenta un modelo de *negociación orientada al servicio* cuyo principal objetivo consiste en conseguir automáticamente el contrato que regula el uso que un agente cliente puede hacer de un servicio proporcionado por un agente proveedor. Es fácil comprobar que este objetivo coincide (al menos, en líneas generales) con el objetivo que persigue la noción de negociación adoptada en esta tesis.

El modelo de valoración de ofertas utilizado en la negociación orientado al servicio es muy similar al utilizado en [Koistinen y Seetharaman 1998] (sección §3.3.2.6), aunque no tan poderosa, pues ni las condiciones de los agentes clientes ni las ofertas de los agentes servidores pueden incluir restricciones intervalares.

La teoría que existe sobre negociación en agentes, la cual es inherentemente automática, puede ser utilizada para extender los actuales modelos de ingeniería de requisitos y las plataformas de sistemas distribuidos. En esta sección resumimos las principales tendencias que hay en este terreno. Estas tendencias estarán organizadas de acuerdo con las tres grandes áreas en las que según [Jennings *et al.* 2001] se puede clasificar la actual investigación sobre negociación automática entre agentes: los protocolos de negociación, los objetos de negociación y los modelos de decisión.

3.3.5.1. Protocolos de negociación

El protocolo de una negociación es el conjunto de reglas que dirigen la interacción entre los participantes. El alcance de estas reglas cubre los aspectos relacionados con los tipos de participantes (auténticos participan-

tes, representantes, representantes electrónicos, etcétera), los posibles estados de la negociación (esperando oferta, negociación finalizadas, etcétera), los eventos que provocan cambios de estado (oferta aceptada, nueva oferta, etcétera) y las acciones que puede realizar cada participante en cada estado (qué mensajes puede enviar, a quién, etcétera).

En [Su *et al.* 2000] se clasifican los protocolos de negociación en tres grandes grupos: de oferta (*bidding*), de subasta (*auction*) y de regateo (*bargaining*). El protocolo de oferta es el más simple: el comprador decide el producto que le interesa adquirir y solicita ofertas a un grupo de proveedores. De acuerdo a las ofertas recibidas, el comprador elige el proveedor que más le interesa. El protocolo de redes de contrato (*Contract Net Protocol*) [Smith 1980] es un buen ejemplo de este tipo de negociación.

En los protocolos de subasta, son los proveedores los que ofrecen una oferta y los consumidores los que pujan por ella siguiendo un determinado protocolo. Este modelo es seguido por varios sitios WEB y existen varias propuestas de arquitecturas de referencia para soportar la conexión de consumidores humanos y electrónicos (agentes), entre las que cabe destacar [Wurman y Wellman 1999].

En los protocolos de regateo, la negociación puede ser iniciada por los proveedores o por los compradores. En cualquier caso, una vez se conoce la oferta inicial (que no satisface a todos los participantes) se inicia un intercambio de ofertas y contraofertas hasta que se alcanza un acuerdo o se determina la imposibilidad de alcanzarlo. Evidentemente esta forma de negociar es la más compleja y es sobre la que más se está investigando. Además, coincide con el tipo de negociación realizada en sistemas distribuidos [Koistinen y Seetharaman 1998] y en sistemas de comercio electrónico [Su *et al.* 2000].

Independientemente del tipo de negociación utilizado, el protocolo de negociación puede ser *bilateral* o *multilateral*. Un protocolo es bilateral si sólo puede utilizarse entre dos participantes, es el caso de los protocolos propuestos en [Sierra *et al.* 1997, Faratin *et al.* 2000] y en [Koistinen y Seetharaman 1998] (sección §3.3.2.6). Un protocolo es multilateral si puede ser utilizado para negociar entre tres o más participantes. Hasta la fecha no conocemos ningún trabajo que trate la negociación multilateral de manera automática en el terreno de los agentes, aunque en el terreno de la ingeniería de requisitos, Win–Win si propone una semiautomatización de la detección y solución de conflictos entre múltiples participantes.

3.3.5.2. Objetos de negociación

Según [Jennings *et al.* 2001], los aspectos (*issues*) sobre los que hay alcanzar un acuerdo determinan lo que se conoce como el objeto de negociación. En este sentido se hablaría de objetos que comprenden un único aspecto, como era el caso de los primeros modelos de negociación, y de objetos que comprenden múltiples aspectos. Dado que un aspecto puede ser interpretado como un atributo de calidad de un objeto de negociación, en adelante nos referiremos a los aspectos como atributos. De este modo, pretendemos facilitar la comparación de la negociación en el terreno de los agentes y en la ingeniería de requisitos.

Según Jennings, de manera ortogonal a la estructura del acuerdo, el tipo de protocolo de negociación determina, al menos tres tipos de objetos de negociación, a saber:

- **Estáticos**. Su estructura y sus valores no pueden ser modificados. Es el tipo de objetos utilizado por los protocolos más antiguos.
- **Valores variables**. Su estructura no puede ser modificada, pero sus valores sí. Este tipo de objetos es estrictamente necesario en los protocolos de regateo.
- **Dinámicos**. Su estructura puede ser modificada, esto es, se pueden añadir o eliminar atributos. Por ejemplo, un agente que representa a un vendedor de coches puede añadir el airbag del copiloto si cree que esto convencerá al agente que representa al comprador.

Un modelo donde es necesario disponer de objetos de negociación con valores variables es el *modelo de negociación basado en compromisos* propuesto en [Faratin *et al.* 2000]. Dicho modelo extiende el modelo de negociación orientado al servicio propuesto en [Sierra *et al.* 1997]. El objetivo de la extensión es reducir el tiempo necesario para alcanzar un acuerdo cuando se trabaja en *ambientes no competitivos*, esto es, ambientes en el que los agentes no miran únicamente por sus intereses sino también por el de los demás agentes de su sociedad.

La idea básica es muy sencilla, se trata de que un agente pueda realizar una propuesta con la misma utilidad que la última propuesta que había hecho pero de mayor utilidad para el oponente en la negociación. El agente fija la utilidad mínima que debe ofrecer el acuerdo o contrato final alcanzado. Si tras varios intentos de negociación no ha sido posible alcanzar

un acuerdo que ofrezca una utilidad mayor o igual a la esperada, en lugar de considerar que la negociación no es posible, decide proponer compromisos al oponente. Dicho compromisos no modifican la utilidad global de la última oferta propuesta, pero si las condiciones locales sobre algunos de sus atributos. Por lo general, es posible encontrar múltiples acuerdos con la misma utilidad (conocidos como *acuerdos isonivel*). Para ello, basta reducir el valor de un atributo a cambio de aumentar el valor de otro, lo que a su vez requiere la posibilidad de modificar los valores de la oferta. La búsqueda de los acuerdos isonivel puede realizarse automáticamente, y para conocer el orden en el que deben proponerse dichos acuerdos también existen algoritmos (basados en lógica difusa).

3.3.5.3. Modelos de decisión

Los participantes en una negociación tienen que tomar decisiones sobre la aceptación o no de una oferta y sobre la siguiente oferta o contraoferta que tiene que proponer. Estas decisiones son tomadas siguiendo un modelo cuya complejidad depende del tipo de protocolo y objeto de negociación empleados. Nosotros proponemos una clasificación basada en los siguientes criterios:

- **Grado de automatización**. La toma de decisiones puede ser *automática*, en el caso de que los agentes dispongan de toda la información necesaria para tomar las decisiones y *semiautomatica*, cuando la toma de decisiones requiere la intervención humana.

  Si utilizamos la tecnología de agentes para automatizar el modelo de negociación Win–Win (sección §3.3.4) podemos identificar al menos tres actividades susceptibles de automatización: la detección de conflictos, la generación de acuerdos potenciales, la selección óptima de acuerdos. Estas mismas actividades pero con nombres diferentes son las mismas que se requieren en la negociación entre agentes.

- **Tipo de razonamiento**. Dependiendo del tipo de variables y operadores que se pueden utilizar para establecer las condiciones de los agentes distinguiremos entre modelos *cuantitativos*, *cualitativos* y *semicualitativos*. En los modelos cuantitativos las condiciones siguen el formato atributo = valor numérico. En los modelos semicualitativos (SC) el formato de las condiciones es más amplio atributo θ valor numérico, donde $\theta = \{<, \leq, >, \geq, \in\}$. Además en los modelos SC

es posible asociar una función de satisfactibilidad a cada atributo, de ese modo, es posible conocer el grado de satisfactibilidad de un requisito. En los modelos cualitativos es posible especificar condiciones sobre variables lingüísticas (aquéllas que pueden tomar por valor términos en lenguaje natural) y adverbios y adjetivos como operadores. Por ejemplo, *la disponibilidad deberá ser muy alta, pero con un coste razonable.*

- **Grado de conocimiento**. Dependiendo del grado de conocimiento que se tenga de los márgenes de negociación antes de que ésta se inicie se distinguen tres tipos de negociación: *cerrada*, cuando los márgenes de negociación son completamente desconocidos; *abierta*, cuando los márgenes con completamente conocidos, y *mixta*, cuando sólo se conoce parte de los márgenes de negociación.

  Por regla general, la negociación cerrada se da en ambientes competitivos en el que todos los agentes luchan por lograr su objetivo sin contemplar las preferencias de los demás agentes, y la negociación abierta en ambientes no competitivos. Aunque no siempre es así, por ejemplo, en [Faratin *et al.* 2000] se plantea un modelo de negociación cerrado basado en compromisos implícitos para un ambiente no competitivo.

  La negociación cerrada atrae la atención de los investigadores en inteligencia artificial distribuida y son numerosos los trabajos enfocados a aumentar la convergencia de los protocolos asumiendo esta falta de información, aplicando técnicas de lógica difusa [Faratin *et al.* 2000], computación evolutiva [Oliver 1996]. En [Faratin 2000, Jennings *et al.* 2001] se pueden encontrar referencias a numerosos trabajos en este campo.

- **Margen de negociación**. Cuando se negocia sobre un único atributo, todos los participantes (salvo que uno de ellos no esté dispuesto a ceder) ceden durante la negociación, por lo que la negociación se reduce a intentar perder lo mínimo posible. Por ejemplo si está negociando sobre el precio de un producto, cuyo precio inicial propuesto por el proveedor es de 125€ y donde el cliente está dispuesto a pagar 100€, lo normal es que el precio final sea una cantidad intermedia entre las dos iniciales. En cualquier caso, ambos participantes pierden respecto a sus expectativas iniciales, situación a la que en adelante nos referiremos como *margen de negociación sólo de pérdidas*.

  El margen de negociación sólo de pérdidas no es exclusivo de las negociación sobre un único aspecto. El modelo Win–Win (ver sección

§3.3.4) que soporta la negociación sobre múltiples aspectos, también tiene un margen de negociación de sólo perdidas, pues cada participante propone inicialmente las condiciones más favorables (condiciones ganadoras).

Cuando se utiliza una negociación basado en compromisos (ver sección §3.3.5.2), la negociación se realiza sobre varios atributos a la vez, de manera que se está dispuesto a ceder en algunos a cambio de ganar en otros, es decir, existe un margen de negociación y otro de pérdidas (la figura §5.14 de la página 147 ilustra esta idea). Dentro de los modelos de negociación por compromisos sobre múltiples atributos distinguimos entre los que tienen limitado el margen de negociación (los contratos isonivel del modelo de Faratin) y aquellos que no tienen limitación alguna, es decir, que tiene completa libertad en la negociación, como es el caso del modelo que proponemos en esta tesis.

3.3.6. NAFUR

De todos los modelos de calidad que hemos revisado en esta tesis, no existe ninguno en el que sea posible alcanzar automáticamente acuerdos cuando existen períodos de vigencia locales. Para soportar esta característica, es necesario extender las actuales nociones de consistencia y conformidad.

Tener consciencia o sensibilidad temporal conlleva dos beneficios muy importantes: i) aumenta la capacidad de negociación de los participantes, y ii) facilita el uso racional de los recursos del sistema [Hafid *et al.* 1998].

Por ejemplo, si el cliente de un sistema de video bajo demanda solicita ver una película a las 16:20 con las condiciones {COLOR= Yes, WINDOW_SIZE= LARGE} y el proveedor no puede garantizar dicho servicio a esa hora, puede devolver dos posibles propuestas: i) reproducir la película pero con una menor calidad (p.e, en blanco y negro y en una ventana pequeña) o ii) ver la película con la calidad deseada pero en un instante posterior al solicitado (p.e. t_1). Supongamos, que nuestro cliente elige la segunda propuesta y que al mismo tiempo, un segundo cliente solicita ver la misma película en el instante t_2 con $t_2 < t_1$. Supongamos que en ese momento el servidor si puede atender a la petición del segundo cliente, pero también desea sacar el máximo rendimiento a su sistema. Es evidente, que en tales circunstancias, resultaría ideal para el proveedor que el segundo

cliente aceptará ver la película en el instante t_2 (evidentemente a cambio pagaría menos por verla) pues de ese modo se optimizarían los recursos. En [Hafid *et al.* 1998] se presenta NAFUR (*Negotiation Approach with FUture Reservation*), un protocolo de negociación que permite llevar a cabo negociaciones como la descrita en este ejemplo.

La principal aportación de NAFUR es ser de los primeros protocolos que distinguen entre el instante de inicio del servicio y el instante en el que se realiza la petición (que no tiene porque ser la misma). Para que el proveedor pueda conocer por adelantado la carga de su sistema, NAFUR obliga a que los clientes indiquen en sus solicitud la duración prevista en el uso del servicio.

Para valorar adecuadamente la aportación de NAFUR, debemos destacar que hasta su aparición, y aún hoy día, los protocolos de negociación sólo determinan si es posible lograr un acuerdo, pero no informan de las razones por las que no es posible alcanzarlo. Otros protocolos más avanzados comienzan a dar esta información, pero sin tener en cuenta la componente temporal de los requisitos, asumiendo que los servicios se solicitan por períodos temporales indefinidos. No obstante, NAFUR comparte muchas de las limitaciones de otros modelos de calidad, concretamente:

- Considera que el período de vigencia de una petición de servicio puede ser representado con un único intervalo temporal, es decir con un par de valores (instante inicial, instante final). De este modo, no es posible establecer períodos de vigencia del tipo:

  desde las 8 horas hasta las 15 los lunes, miércoles y viernes y de 8 a 14 y de 16 a 20 horas los marte y jueves.

- Sólo permite modelar períodos de vigencia globales a la petición del cliente.

- Se trata de un modelo muy limitado pues: i) utiliza una función de valoración lineal aditiva clásica ii) únicamente permite el uso de restricciones en igualdad, incluso en las condiciones del cliente y iii) no puede definir requisitos sobre atributos de tipo conjunto.

3.3.7. Resumen

Formalidad es el término que mejor recoge la principal característica de los modelos de tercera generación. El desarrollo de los primeros len-

guajes formales de especificación de catálogos de atributos y requisitos de calidad facilitó los primeros pasos hacia la automatización de las tareas relacionadas con los requisitos de calidad. En adelante, nos referiremos a los lenguajes de especificación de atributos y requisitos de calidad con las siglas QSL (*Quality Specification Languages*).

Las primeras tareas que lograron automatizar los QSLs fueron la comprobación automática de la conformidad y la selección óptima de ofertas. No obstante, las posibilidades de los QSLs de tercera generación no son suficientes para soportar el desarrollo y explotación de MOWS y de los sistemas basados en componentes. En nuestro opinión, los aspectos cuyo soporte es inexistente o inadecuado son: la consciencia temporal, la capacidad de negociación, la interoperabilidad, la comunicabilidad y la viabilidad. Todos estos aspectos son tratados en el siguiente capítulo.

Capítulo 4

Lenguaje de especificación ideal

When we try to pick out anything by itself, we find it hitched to everything else in the Universe.

JOHN MUIR (1838-1914)

En este capítulo revisamos algunas propuestas relacionadas con los tres procesos del modelo de referencia vistos en el anterior capítulo. Como conclusión de esta revisión, también discutimos sobre las características que, en nuestra opinión, deben ser ofrecidas por el soporte lingüístico de nuestro modelo de referencia. Al mismo tiempo, indicaremos cuáles y en qué medida dichas características son ofrecidas por los trabajos presentados en este capítulo.

Identificar y definir con claridad y sin ambigüedades nuevos conceptos es una actividad sumamente compleja, pues tal y como pensaba John Muir, al intentar explicar algo por sí mismo, nos damos cuenta de que guarda relación con muchos otros conceptos. La caracterización de los modelos de cuarta generación que describimos en esta sección, ha utilizado como referencia las características de los modelos de tercera generación más algunas nuevas. Estas características serán presentadas suponiendo que existe un QSL ideal que las soporta todas.

De este modo, esta caracterización tiene un doble objetivo: por una parte, describir de la manera más clara y ordenada posible las principales características que deben soportar el QSL ideal y por otra, dar a conocer de la manera más objetiva posible en qué grado son soportadas dichas características por los actuales QSLs.

Con el ánimo de valorar de la manera más objetiva posible la distancia entre cada uno de los modelos de tercera generación analizados en el capítulo anterior y el QSL ideal, hemos identificado para cada una de estas características un conjunto de elementos a valorar. La discusión sobre la valoración de cada uno de estos elementos se ha omitido en aquellos casos en los que se trataba de elementos tratados en secciones anteriores de este capítulo.

Por último, indicar que en esta comparativa incluimos a QRL*, el QSL que presentaremos en el capítulo §5. Con ello pretendemos dar a conocer de manera muy general, pero completa, todas las aportaciones que* QRL *realiza frente a las actuales propuestas.*

4.1. Formalidad

La valoración del termino formal ha ido cambiando a lo largo de la historia de la informática, pasando por momentos en los que este calificativo era sinónimo de excelencia y momentos en los que era claramente denostado. Quizá porque se ha asociado tradicionalmente con los métodos y los lenguajes de especificación formal, que aunque han tenido una gran acogida en el mundo académico no han llegado a calar profundamente en el mundo de la industria [Ciapessoni *et al.* 1999]. Hoy en día, la historia reciente nos proporciona una buena perspectiva sobre los métodos y los lenguajes formales y creemos que una de las principales características de un modelo de cuarta generación debe ser precisamente esta la formalidad, pero entendida desde los siguientes puntos de vista:

- La formalidad debe aportar al modelo una descripción rigurosa y completa de todos sus elementos.
- El grado de formalidad debe alcanzar un equilibrio razonable entre la expresividad del modelo y el grado de automatización que se puede conseguir con él.
- La formalidad debe implicar la posibilidad de automatizar la com-

probación de algunas propiedades de interés en el ámbito de aplicación del modelo.

Por lo tanto, desde nuestro punto de vista, la formalidad del modelo debe entenderse como la base para poder implementar herramientas que nos permitan trabajar con nuestros documentos de calidad y demostrar algunas propiedades. Entre las propiedades más importantes que todo modelo debería soportar se encuentran: la consistencia, la conformidad y la optimalidad.

Además de la capacidad para demostrar estas tres propiedades, la valoración del grado de formalidad (ver tabla §4.1) también tiene en cuenta otros criterios:

- **Semántica y sintaxis bien definidas**. La primera característica que debe tener un lenguaje para demostrar automáticamente propiedades es disponer de una sintaxis y una semántica bien definidas.
- **Sensible a la indecidibilidad**. La semántica formal de los QSLs revisados asume que el mecanismo de resolución de restricciones es completo, esto es, es capaz de resolver todas las restricciones asociadas a un requisito y comprobar la consistencia, la conformidad y la optimalidad. Esta consideración pone en peligro la viabilidad del lenguaje, pues a poco que se utilicen restricciones no lineales, disyunciones y restricciones que involucren a muchos atributos, surgen problemas de decidibilidad. En nuestra opinión, sería más adecuado definir una semántica estratificada en la que se definiera de manera separada la semántica del lenguaje y la semántica del resolutor. Este es el esquema seguido en la definición de la semántica formal de QRL.
- **Facilidad de uso**. Creemos que una medida objetiva de la facilidad de uso de un formalismo, puede ser la pertenencia o no de dicho formalismo a los planes de estudio de ingeniería informática. Por ejemplo, la teoría de ecuaciones e inecuaciones tanto lineales como no lineales, la programación lineal y la programación con restricciones, son buenos ejemplos de formalismos fáciles de usar y bien conocidos, al menos en el nivel que es necesario utilizar para definir la semántica de los QSL.

Debe tenerse en cuenta, que si bien casi todos los QSLs de tercera generación soportan la demostración de propiedades, la complejidad de es-

Criterio	OMG	QML	NoFun	MAS	QRL
Semántica y sintaxis bien definidas	X	X	X	X	X
Sensible a la indecidibilidad					X
Demostración automática de la consistencia	X	X		X	X
Demostración automática de la conformidad	X	X		X	X
Demostración automática de la optimalidad	X	X		X	X
Formalismo de fácil uso y muy extendido	X	X	X	X	X

Cuadro 4.1: *Valoración de la formalidad.*

ta demostración depende del grado en el se soportan las restantes características. Como veremos al describir la semántica de QRL, la comprobación de la conformidad y de la consistencia cuando se dispone de consciencia temporal y de cláusulas de negociación, no se puede llevar a cabo con los modelos conceptuales de los actuales QSLs.

4.2. Expresividad

La expresividad de un modelo es un concepto altamente subjetivo y dependiente del contexto en el que nos movamos. No obstante creemos que es posible aplicar el calificativo expresivo a todos aquellos modelos en los que el conjunto de elementos de modelado que ofrecen es lo suficientemente rico como para poder expresar los problemas de mayor interés dentro del dominio para el que están pensados sin que sea necesario recurrir a artificios. En definitiva, un modelo será tanto más expresivo cuanto menor sea la distancia semántica entre las herramientas de modelado que ofrece y la solución a los problemas en los que estamos interesados.

Por ejemplo, en QML no es posible expresar requisitos que incluyan una relación no lineal entre dos atributos de calidad, pero no por ello se

puede decir que es poco expresivo, pues puede que para el dominio del problema que sus diseñadores han tomado de referencia, la necesidad de expresar este tipo de requisitos sólo se da en un número muy reducido de casos.

Los principales elementos utilizados en la especificación de la calidad de un producto son dos: el catálogo de atributos y los requisitos de calidad. Las características que en nuestra opinión determinan la expresividad de un catálogo de atributos son:

- **La definición modular**. Esto es, la posibilidad de definir e importar módulos o catálogos de atributos.
- **La definición por refinamiento**. Esto es, la posibilidad de definir nuevos atributos refinando la definición de otros ya existentes.
- **Constructores complejos**. Esto es, la posibilidad de definir nuevos atributos utilizando, cuando menos, constructores de conjuntos, de tuplas y de funciones.

Estas características se han definido tomando como referencia a NoFun, el cual las soporta muy adecuadamente. QRL también las soporta, aunque no proporciona las facilidades sintácticas que proporciona NoFun.

Por su parte entendemos que las características que determinan la expresividad de un modelo de especificación de requisitos son:

- **Tipo de restricción que se puede especificar**. Si bien todos los modelos de calidad admiten la interpretación de un requisito como una restricción en el espacio de todos los productos posibles, no todos permiten expresar el mismo tipo de restricciones. Nos referimos a la imposibilidad que algunos QSL tienen de especificar (con una semántica operacional precisa) restricciones sobre varios atributos, restricciones intervalares en los requisitos de las ofertas y restricciones no lineales.
- **Tipo de refinamiento utilizado**. Cuando se definen requisitos sobre atributos que poseen una cierta complejidad, resulta muy útil disponer de mecanismos que faciliten el refinamiento de requisitos. Hasta la fecha, los dos QSLs que soportan refinamiento son QML y NoFun.

QML ha sido diseñado para ser utilizado por las plataformas de objetos distribuidos, donde el reemplazamiento automático de componentes juega un papel crucial [Vallecillo 1999]. El reemplazamiento automático requiere no sólo el uso de mecanismos que garanticen que la interoperabilidad funcional con el resto del sistema no se ponga en peligro [Vallecillo *et al.* 2000], también es necesario garantizar la interoperabilidad de los aspectos no funcionales o de calidad [Ruiz-Cortés *et al.* 2000b]. Por esta razón, QML considera el refinamiento en el sentido de [Liskov y Wing 1994], esto es, los requisitos refinados siempre son más restrictivos que el requisito base.

NOFUN sin embargo, no tiene como principal objetivo el ser un lenguaje ejecutable, sino un lenguaje de modelado de la calidad fácil y cómodo de utilizar, por lo que un refinamiento que sigue el esquema de la herencia de implementación habitual (donde los requisitos refinados no tienen porqué ser más restrictivos que el requisito base) resulta más adecuado.

En el caso del servicio de intermediación de la OMG, existe una noción de subtipado que extiende la de Liskov, pero que no tiene el alcance del subtipado en QML. Nos referimos a que en el aspecto de la calidad sólo comprueba que todos los atributos del supertipo, están en el subtipo y que los tipos (dominios) de los atributos del subtipo son un subtipo de los atributos del supertipo; pero no se realiza ninguna comprobación sobre la restricción que se impone sobre un atributo.

- **Distinción de requisitos**. Algunos lenguajes como NOFUN permiten indicar la importancia de un requisito, concretamente distingue entre requisitos esenciales, importantes, aconsejables y marginales. Dicha información es utilizada posteriormente durante el proceso de evaluación. Evidentemente, la distinción de requisitos sólo tiene sentido en los documentos de condiciones, al menos nosotros no hemos encontrado utilidad a utilizarlas en las ofertas.

- **Satisfactibilidad implícita y explícita**. Hasta la fecha, los modelos que ofrecen mecanismos para modelar la satisfactibilidad de un requisito lo hacen de la misma manera. A partir de los valores de los atributos evalúan una determinada función de agregación que tiene la misma forma para todos los requisitos. Sería interesante que ésta se pudiera definir de manera explícita e individual para cada requisito.

Criterio Subcriterio	OMG	QML	NoFun	WinWin	MAS	QRL
Sobre el catálogo de atributos						
Definición modular			X			X
Definición por refinamiento			X			X
Constructores complejos			X			X
Sobre la especificación de requisitos						
Restricciones multidimensionales		X				X
Restricciones no lineales		X				X
Restricciones intervalares en la oferta		X				X
Refinamiento: subtipado(S), herencia(H)	S(1)	S	H			S,H
Distinción de requisitos			X			X
Función de utilidad implícita	X	X		X	X	X
Función de utilidad explícita	X			X		X

Cuadro 4.2: *Valoración de la expresividad.*

El servicio de intermediación de la la OMG dispone de una característica muy cercana a la noción de satisfactibilidad de QRL. Nos referimos a la política de selección de ofertas. Una de las posibilidades pasa por definir expresiones para evaluar la proximidad de cada oferta a las condiciones del cliente. No obstante, debe tenerse en cuenta que esta característica está soportada porque en la oferta no se admiten restricciones intervalares, y por ende, la evaluación de la función es trivial. En el caso de admitir restricciones intervalares en la oferta la evaluación se complica, pues se hace necesario calcular el mínimo de una restricción.

Por su parte, en [In *et al.* 2002] se proponen varios métodos para definir funciones de satisfactibilidad dependiendo de la naturaleza de los atributos que intervienen en el requisito.

Creemos que es interesante resaltar que el hecho de utilizar el mis-

mo formalismo no implica necesariamente disponer de la misma capacidad expresiva, pues se pueden utilizar diferentes modelos semánticos. Por ejemplo, QML y el Trader de OMG usan el mismo formalismo, pero la semántica de los requisitos de la oferta son diferentes. En QML son restricciones libres y en el Trader de la OMG son restricciones en igualdad.

Existen otras características que determinan las posibilidades expresivas de un QSL de cuarta generación: la capacidad de abstracción, la temporalidad y la capacidad de negociación. No obstante, hemos decidido tratarlas separadamente por la importancia que están cobrando recientemente y por la escasa atención que han recibido hasta ahora.

4.3. Abstracción

En numerosas ocasiones, es necesario evaluar y contratar diferentes tipos de productos: aplicaciones, subsistemas, elementos de una descripción arquitectónica, interfaces, COTS, etcétera. Para estas ocasiones, sería muy deseable poder especificar los requisitos de todos los productos con el mismo lenguaje.

De los lenguajes de tercera generación, el único lenguaje que puede ser utilizado con independencia del producto utilizado es NOFUN, pues QML y OMG sólo pueden ser utilizados sobre interfaces que han de estar necesariamente descritas en IDL de CORBA.

Salvando las diferencias, esta situación de dependencia del producto que sufren los actuales QSL, recuerda la dependencia que tenían los primeros lenguajes de programación respecto de la máquina física. En aquel momento la dependencia fue superada haciendo uso de la *abstracción*. En este momento, vuelve a ser necesaria para conseguir la independencia del producto.

Un elemento a tener en cuenta en la valoración de la abstracción es la posibilidad de definir requisitos referidos tanto al producto en su conjunto como a cada uno de los elementos o partes que lo componen. Gracias a esta característica, es posible especificar requisitos de calidad sobre el comportamiento global de un caso de uso y sobre cada una de sus acciones (sección §5.8.1) y sobre una interfaz y sobre cada una de sus operaciones (sección §5.3.3). Esta característica puede generalizarse y permitir la especificación sobre productos con estructuras de múltiples niveles, por

Criterio	OMG	QML	NoFun	WinWin	MAS	QRL
Independencia del tipo de producto			X	X		X
Descomposición estructural (1 nivel)		X	X	X		X
Descomposición estructural multinivel			X			X

Cuadro 4.3: *Valoración de la abstracción.*

ejemplo, la que se necesitaría para especificar requisitos en Java, sobre un paquete, sus clases y los métodos de éstas.

4.4. Temporalidad

Las características que determinan el grado de consciencia temporal son:

- **Requisitos individuales**. Dar la posibilidad de que a cada requisito tenga su propio período de vigencia.
- **Períodos definidos por múltiples intervalos**. La forma más simple de representar un período de vigencia es con un par de atributos que representen al instante inicial y final del período. De este modo, la expresión FECHA > MAY 2000 **and** FECHA < JUN 2003 (que no todos los QSLs admiten expresar en un documento de oferta) representa a un periodo de vigencia compuesto por un único intervalo de tiempo.

  Para definir períodos más complejos (p.e., la disponibilidad de un servicio debe ser del 99 % de lunes a viernes desde las 7:00 hasta las 15:00 y los martes y jueves también desde las 17:00 a las 20:00, y del 90 % en el resto del tiempo) necesitamos utilizar múltiples intervalos, lo que resulta mucho más incómodo de especificar con los actuales ya que sería necesario definir numerosos atributos.

  Hasta la fecha no conocemos ningún QSL que permita expresar este tipo de períodos. La situación se agrava aún más cuando no es posible encontrar ningún patrón de periodicidad en los intervalos.

Criterio	OMG, QML, NOFUN, Win–Win, MAS	QRL
Requisitos individuales		X
Período definido por múltiples intervalos		X
Atributos temporales		

Cuadro 4.4: *Valoración de la temporalidad.*

- **Atributos temporales**. Existen requisitos cuya forma natural de especificación implica el uso de atributos temporales. Por ejemplo, la forma más directa de especificar formalmente el requisito la fecha de caducidad del producto deberá ser de al menos 30 días a partir de la fecha actual sería con la restricción FEC_CAD $\geq$ now + 30, donde FEC_CAD sería un atributo de tipo Fecha.

Evidentemente, cualquier QSL que permita la definición de atributos derivados y funciones sobre estos atributos (p.e. NOFUN) puede definir atributos derivados para representar fechas e instantes temporales, incluso definir operaciones relacionadas. Sin embargo, tal y como veremos en el capítulo §6, aquellos QSLs que representan los requisitos como restricciones sobre atributos de calidad necesitan revisar su semántica, pues la semántica del tiempo es diferente a la de los restantes atributos de calidad, exactamente la contraria para ser más precisos.

4.5. Negociación

El modelo de negociación de los modelos de cuarta generación debe extender los actuales modelos de ingeniería de requisitos y las plataformas de sistemas distribuidos. Nosotros proponemos que esta extensión se realize teniendo en cuenta las posibilidades de los actuales modelos de negociación en sistemas multiagentes que vimos en la sección §3.3.5 y cuyas características son recogidas en la tabla §4.5.

Criterio Subcriterio	OMG, NOFUN	QML	WinWin	MAS	QRL
Protocolo					
Oferta (O), Subasta (S), Regateo (R)		R	R	O,S,R	O,S
Bilateral(B), Multilateral(M)			M	B	M
Objeto negociación					
Con múltiples atributos		X		X	X
Valores variables		X	X	X	X
Dinámicos			X (2)	?	X
Modelo de decisión					
Semiautomático (S), Automático (A)		A	S	A	A
Cuantitativo (C), Semicualitativo (S), Cualitativo (CL)		S(3)	?	C(3)	S
Abierto(A), Cerrado(C), Mixta(M)		C	C	C	A, M, C(1)
Margen de negociación: sólo pérdidas (P), isonivel (I), libre (L)		P	P	I	L
(1) No se tiene implementado el algoritmo para la negociación cerrada					
(2) Sólo manualmente					
(3) Pero sin soportar restricciones intervalares					

Cuadro 4.5: *Valoración de la negociación.*

Criterio	OMG, QML, NOFUN, Win–Win, MAS	QRL
De lenguaje natural estructurado a notación formal		X
De notación formal a lenguaje natural estructurado		X
De lenguaje natural a notación formal		

Cuadro 4.6: *Valoración de la dualidad.*

4.6. Dualidad

La necesidad de que los documentos de requisitos sean comprensibles por clientes y usuarios a la vez que útiles para los desarrolladores (incluso tratables por las plataformas de desarrollo y ejecución), conduce inevitablemente a la utilización de una doble notación [IEEE 1993]. En [Wieringa 1996] y en [Durán 2000, pág.17] se considera la dualidad junto con la no ambigüedad, como una de las propiedades deseables que debe tener una especificación de requisitos: la *comunicabilidad*.

En nuestro contexto, la comunicabilidad sigue siendo muy importante y necesaria, pero siempre y cuando ésta no merme la capacidad de tratamiento automático. Pues bien, a la capacidad de poder traducir de manera automática un documento de requisitos entre dos notaciones, donde estas dos notaciones consiguen que el documento sea comunicable la denominaremos en adelante como *dualidad*.

Hasta la fecha, no conocemos ningún lenguaje que pueda ser considerado dual tal y como acabamos de definir, siendo QRL el único que hasta la fecha proporciona esta posibilidad. No obstante, QRL sólo proporciona la dualidad desde un lenguaje natural estructurado, es decir, un lenguaje del que conocemos todas las posibles frases y su interpretación, en este caso en términos de restricciones matemáticas.

4.7. Interoperabilidad

Existe una característica común a todas las propuestas revisadas en nuestro estudio, que pone en serio peligro su utilidad práctica y real. Nos referimos a la consideración de que todos los participantes utilizan los mismos catálogos de atributos, esto es, que utilizan la misma ontología de referencia.

A pesar de los numerosos catálogos de atributos de calidad que se han definido, no se puede hablar de una ontología común, al menos de facto. Incluso, definiendo una ontología de referencia común basada en el modelo de calidad de la ISO 9126, tan pronto como los participantes refinan estas ontologías (lo que a veces es estrictamente necesario) comienzan a surgir problemas de interoperabilidad semántica.

La interoperabilidad semántica es un problema que está recibiendo mucha atención en el área de los agentes. Existen al menos dos grandes aproximaciones al problema dependiendo de si se asume que las ontologías locales tienen una ontología base común o no la tienen. En el primer caso, existen trabajos que permiten identificar automáticamente si es posible la comunicación entre agentes que utilizan estas ontologías. El segundo caso, existen trabajos que proponen técnicas que logran semiautomatizar dicho proceso siempre y cuando las ontologías sean reducidas y la distancia semántica sea pequeña. Entre ellos cabe destacar las ontologías de consenso propuestas en [Stephens y Huhns 2001].

También cabe la posibilidad de abordar este problema de manera más rudimentaria aunque no por ello menos efectiva. Nos referimos a establecer manualmente la correspondencia entre las ontologías. En nuestra opinión, ésta es una aproximación que resulta eficaz cuando el número de ontologías no es demasiado elevado y se combina con el uso de procedimientos automáticos de extracción de información. En la sección §5.7 se indica como se definen correspondenc proponemos un pequeño lenguaje para definir este tipo de correspondencias entre diferentes ontologías. La tabla §4.7 recoge las diferentes soluciones con las que se puede lograr la interoperabilidad semántica.

Hasta la fecha no hemos encontrado ningún modelo de calidad que se preocupe de los problemas de interoperabilidad semántica, aunque existen herramientas comerciales como Biztalk de Microsoft que facilitan la definición de la correspondencia entre ontologías, incluso generan de un modo automático el código que permite pasar un documento definido en

Criterio	OMG, QML, NOFUN, Win–Win, MAS	QRL
Definiendo una correspondencia *ad–hoc*		X
Considerando un ontología base común		
Estableciendo una ontología de consenso		

Cuadro 4.7: *Valoración de la interoperabilidad.*

una ontología a otro documento equivalente pero definido con otra ontología.

4.8. Viabilidad

Son muchas las características que puede reunir un lenguaje de especificación de requisitos, aunque no siempre es posible conseguir modelos de ejecución con una eficiencia "razonable" que las soporte a todas, lo que reduce en gran medida la probabilidad de que sea utilizado para resolver problemas reales y que sean la base de herramientas comerciales. Una correcta valoración de esta probabilidad de uso, en adelante *viabilidad*, debe contemplar tanto aspectos comerciales como técnicos.

En nuestra opinión, existen al menos dos aspectos técnicos medibles objetivamente, a saber: el número de características que soporta (temporalidad, negociación, guía de uso, existencia de compiladores, etcétera) y el coste computacional de cada una de ellas. El primero de los aspectos ha sido tratado a lo largo de esta sección. En cuanto al coste computacional de las algoritmos que dan soporte a los diferentes modelos, debemos señalar que no existe mucha información al respecto, por lo que no hemos desarrollado más este punto.

4.9. Evolución de modelos de calidad

A continuación resumimos las características diferenciadoras de las cuatro generaciones de modelos de calidad identificadas. Las características están organizadas en torno a los tres procesos del modelo de referencia presentado al principio de este capítulo: construcción del catálogo de atributos, construcción de requisitos de calidad e intermediación entre clientes y proveedores. La tabla §4.8 resume estas características y la figura §4.1 refleja las dependencias entre los diferentes modelos.

4.9.1. Construcción del catálogo de atributos

La primera generación de modelos aportó diferentes catálogos de atributos de calidad. Dichos catálogos fueron diseñados pensando en las características de grandes sistemas software y hardware, aunque la primera versión de la ISO 9126 tuvo en cuenta esta situación en el diseño de su catálogo.

En 1988, Tomas Gilb fue de los primeros en proponer que el catálogo de atributos se construyera de manera consensuada entre los participantes del proyecto, y que la definición de cada atributo incluyera una escala de medida y un rango de satisfactibilidad. Posteriormente, en 1990, Keller extendió la idea de Gilb y definió un marco de trabajo para definir métricas no solo de los atributos (métricas simples), sino también sobre las subcaracterísticas y características (métricas complejas). En 1999, Olsina propuso una metodología para definir de manera sistemática el catálogo de atributos, incluyendo la definición de métricas.

En todos estos modelos, la semántica de los atributos de calidad se expresaba en lenguaje natural lo que dificulta gravemente aquellas operaciones que requieren disponer de un significado preciso de éstos. En 1997 y en contextos completamente diferentes, se presentan dos lenguajes que proponen el uso de notaciones formales para definir los catálogos, QML en el contexto de los middlewares para sistemas distribuidos y NOFUN para artefactos software en general.

Finalmente, las posibilidades de los modelos de cuarta generación hacen necesarios lenguajes que permitan definir catálogos de atributos que incluyan requisitos abstractos y dominios temporales.

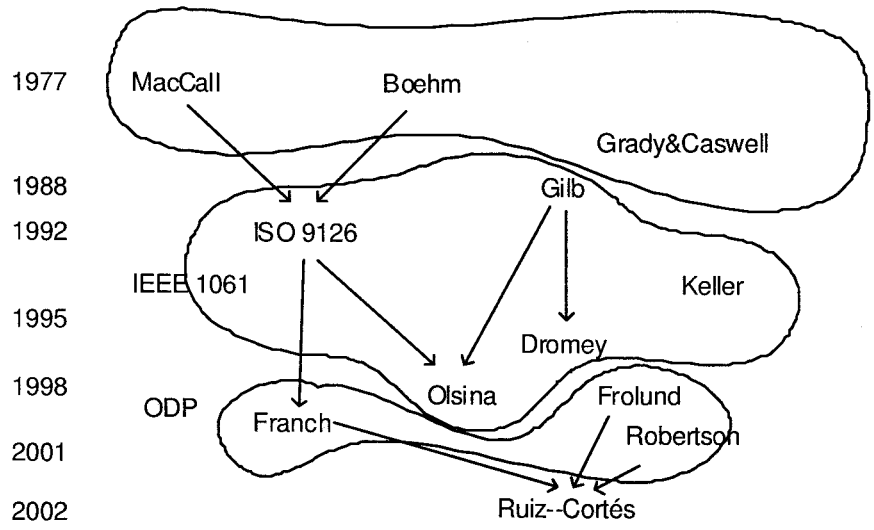

Figura 4.1: *Evolución de los modelos de calidad.*

4.9.2. Construcción de requisitos de calidad

Los modelos de calidad de primera generación no contemplaban la especificación de requisitos de calidad. En los modelos de segunda generación se empieza hablar de requisito de calidad, pero no se considera un elemento de primera clase dentro del modelo. Hay que esperar a los modelos de tercera generación para que se comience a distinguir claramente entre atributos y requisitos.

En los modelos de segunda generación, concretamente en el de Dujmovic, se comienza a distinguir entre documentos de condiciones y de oferta, aunque los requisitos de las ofertas siempre se expresan como pares (propiedad, valor) y no como condiciones libres sobre atributos. El uso de condiciones en desigualdad no será posible hasta la aparición de QRL aunque con graves limitaciones expresivas.

4.10. Resumen

En este capítulo hemos presentado las posibilidades y limitaciones de diversas propuestas que abordan gran parte de los problemas relaciona-

1ª Generación	2ª Generación	3ª Generación	4ª Generación
Catálogo Atributos			
Estándar, orientado a aplicaciones completas	Consensuado, métricas simples para atributos y compuestas para características y subcaracterísticas	Formal, con posibilidades de refinar e importar	Inclusión de requisitos abstractos y dominios temporales
Construcción Requisitos			
	Especificación de requisitos, condiciones y ofertas (propiedad, valor)	Requisitos como elementos de primer orden, consciencia temporal básica, ofertas con condiciones en desigualdad.	Expresiva, abstracta y dual
Intermediación			
Valoración subjetiva, valoración media ponderada	Métricas objetivas, valoración lógica por preferencias	Se automatiza la comprobación y la negociación de la conformidad, elección de la oferta óptima	Interoperabilidad semántica, QSLs compilables

Cuadro 4.8: *Evolución de los modelos de calidad orientados al producto.*

dos con el tratamiento automático de requisitos de calidad. En nuestra opinión, una solución adecuada y eficaz de estos problemas puede y debe ser abordada en el contexto de un lenguaje de especificación de documentos de requisitos de calidad. Con tal propósito, hemos identificado las principales características que dicho lenguaje debería poseer y hemos señalado en que medida la soportan algunos de los modelos de calidad estudiados.

Utilizando los tres planos de Broy [Broy 2001]: metodológico, descriptivo y formal, podemos concluir que hasta la fecha los modelos de calidad se han centrado en los dos primeros, pero han dejado de lado el plano formal. Entendemos, que es necesario y como veremos en el capítulo §6 es posible cubrir este plano.

Parte III

Nuestra propuesta

Capítulo 5

El lenguaje QRL

We must recognize the strong and undeniable influence that our language exerts on our ways of thinking and, in fact, delimits the abstract space in which we can formulate and give form to our thoughts.

NIKLAUS WIRTH

En este capítulo describiremos de una manera informal e intuitiva las principales características del soporte lingüístico, que no metodológico, que hemos diseñado para soportar algunas de las características que debe ofrecer un modelo de calidad orientado al producto de cuarta generación. De ahora en adelante haremos referencia a dicho soporte como QRL *que son las siglas de* "Quality Requirements Language".

Comenzaremos el capítulo presentando la base formal sobre la que se sustenta QRL. *Continuaremos describiendo el soporte que ofrece* QRL *para construir catálogos de atributos, documentos de requisitos y para intermediar entre clientes y proveedores.*

5.1. Base formal de QRL

5.1.1. Conjetura de dualidad "requisito–restricción"

> *I often say that when you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meagre and unsatisfactory kind.*
>
> KELVIN

La principal hipótesis de trabajo sobre la que se apoya QRL se sintetiza en la siguiente conjetura:

> *Todo requisito de calidad puede ser expresado como una frase en lenguaje natural y como una restricción en lenguaje matemático*

Por ejemplo, el requisito "El tiempo entre dos fallos consecutivos será de al menos 6 minutos", puede expresarse como la restricción "TTF $\geq$ 6", donde TTF (*Time To Failures*) representa el tiempo, expresado en minutos, entre dos fallos consecutivos.

Esta conjetura se apoya en las ideas de [Gilb 1988, Robertson y Robertson 1999] (ver sección §3.2.1) para especificar documentos de calidad *verificables*. En esencia, estas ideas vienen a decir que para cualquier atributo de calidad, siempre es posible encontrar una escala para medirlo, por tanto, siempre es posible establecer un requisito como una condición sobre los valores que puede tomar una medida de dicho atributo.

No obstante, con independencia de si es o no universalmente válida nuestra conjetura, aceptar su validez en nuestro contexto de trabajo permite garantizar que los acuerdos de calidad alcanzados sean *verificables*. Entendiendo que un acuerdo de calidad es verificable si y sólo si todo requisito contenido en él es verificable, es decir, existe un proceso finito y de coste razonable por el que una persona o una máquina puede comprobar que el objeto del acuerdo satisface todos los requisitos del acuerdo [Davis 1993, IEEE 1993].

Una condición absolutamente necesaria para que un requisito sea verificable es que la frase empleada para definirlo sea *no ambigua* y que se defina de forma *medible*, ya que no puede comprobarse algo que no se puede medir ni definir de forma precisa [Durán 2000, pág. 20]. Nuestra propuesta para definir de manera medible es utilizar restricciones matemáticas sobre los atributos de calidad de los que depende dicho requisito.

Obsérvese que la conjetura no dice en ningún momento que la restricción matemática equivalente a la frase que describe un requisito de calidad se derive automáticamente a partir de ésta[1], tan sólo señala que esta equivalencia siempre debe existir o por el contrario la frase en lenguaje natural será vaga y no podremos considerarla un requisito. Será un objetivo vagamente definido que habrá que refinar hasta dotarlo de una semántica precisa.

5.1.2. Interpretando requisitos como restricciones

La principal ventaja de interpretar los requisitos de calidad como restricciones está en la posibilidad de automatizar algunas de las actividades relacionadas con ellos. La clave de esta automatización es plantear dichas actividades como problemas de satisfacción de restricciones o CSP (*Constraint Satisfaction Problem*). Los CSPs han sido estudiados desde hace varias décadas y actualmente existen numerosas y variadas técnicas para solucionarlos [Hentenryck y et al. 1996, Marriot y Stuckey 1998, Freuder y Wallace 2000].

A continuación introducimos de manera intuitiva la versión más básica de los CSPs correspondientes a la consistencia, la conformidad y al optimalidad. En otras secciones de este mismo capítulo se describirán versiones mas refinadas y en el capítulo §6 se describirán rigurosamente.

5.1.2.1. Consistencia de un requisito

La propiedad más básica sobre un requisito que podemos interpretar como un CSP es la *consistencia*. Un requisito es consistente si su descripción no contiene ninguna contradicción.

[1]No obstante, si esta equivalencia se indica de manera explícita sí es posible encontrar técnicas que facilitan la traducción automática.

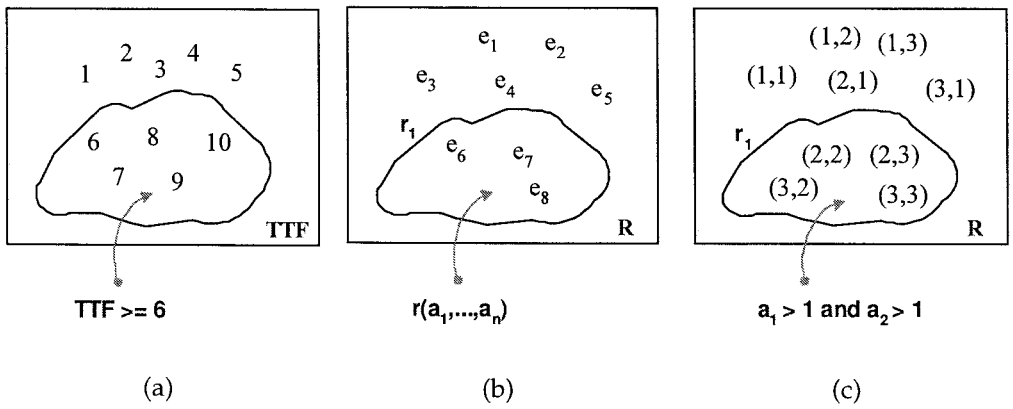

Figura 5.1: *Interpretación gráfica de la satisfactibilidad de una restricción.*

Por ejemplo, en el caso de un documento de condiciones, es evidente que el requisito "El precio de uso del servicio no podrá ser superior a 100€ ni inferior a 200€" es inconsistente o contradictorio pues no existe una cantidad que sea superior a 200€ que sea a la vez inferior a 100€. Esta contradicción se ve más fácilmente al expresar el requisito como una restricción, pues salta a la vista que la restricción Precio $\leq$ 100 **and** Precio $\geq$ 200 no tiene solución.

De un modo general, siempre es posible determinar la consistencia de un requisito estudiando la satisfactibilidad de su restricción asociada. La figura §5.1(a) ilustra de un modo intuitivo el concepto de satisfactibilidad. Para la restricción TTF $\geq$ 6 (considerando que el dominio de TTF es [1..10]) es posible encontrar cinco valores del dominio del atributo TTF que en el caso de sustituir al atributo en la restricción consiguen que la expresión resultante se evalúe a cierto.

Por ejemplo, la sustitución de TTF por el valor 9, que en adelante denotaremos como $\{ \text{TTF} \mapsto 9\}$, obtiene como resultado la expresión $9 \geq 6$ la cual se evalúa a cierto. En el capítulo §6 se define con rigor el significado de la satisfactibilidad de una restricción. Por ahora, es suficiente con esta idea intuitiva.

Esta noción de satisfactibilidad es igualmente válida para restricciones

sobre varios atributos y por tanto para un requisito definido sobre varios atributos. Si consideramos la existencia de un conjunto R que contiene a todas las valoraciones que se pueden realizar sobre un conjunto de atributos $A = a_1, \ldots, a_n$. Una restricción $\phi(a1, \ldots, a_n)$ siempre define dos conjuntos disyuntos sobre R, el conjunto constituido por aquellas valoraciones $(e_i, \ldots, e_j)$ que la satisfacen y su complementario. La figura §5.1(b) ilustra gráficamente esta idea y la figura §5.1(c) muestra un ejemplo concreto de una restricción satisfactible sobre dos atributos.

5.1.2.2. Consistencia de un conjunto de requisitos

Comprobar la consistencia de un conjunto de requisitos es una generalización del problema de la comprobación de un conjunto de requisitos. Se trata de identificar si existe contradicción en la especificación de un conjunto de requisitos.

Dado que un conjunto de requisitos puede verse como un único requisito resultado de yuxtaponer todos los requisitos de dicho conjunto, la restricción asociada del requisito resultado de la yuxtaposición es el resultado de la conjunción de todas las restricciones asociadas.

Por ejemplo, sea el conjunto de requisitos formados por el requisito "El precio de uso del servicio no podrá ser superior a 100€" y por el requisito "El precio de uso del servicio no podrá ser inferior a 200€". El requisito asociado al conjunto será "El precio de uso del servicio no podrá ser superior a 100€ y el precio de uso del servicio no podrá ser inferior a 200€" y su restricción asociada Precio $\leq$ 100 **and** Precio $\geq$ 200.

Por tanto, la operativa para comprobar la consistencia de un conjunto de requisitos no es más que una generalización de comprobar la consistencia de un único requisito, pues como acabamos de ver la operativa para determinar la satisfactibilidad de una restricción es la misma independientemente del número de atributos que intervienen en la restricción.

De este modo, en adelante consideraremos que un conjunto de requisitos es consistente si y sólo si la conjunción de sus restricciones asociadas es satisfactible.

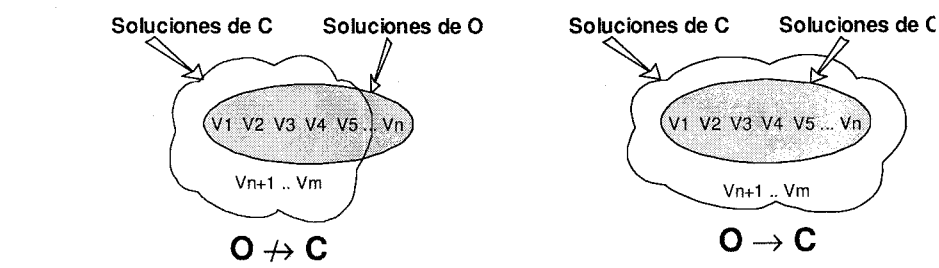

Figura 5.2: *Interpretando la conformidad como un CSP.*

5.1.2.3. Conformidad entre requisitos

Se dice que un documento de oferta satisface o *es conforme con* un documento de condiciones si el nivel de calidad indicado en la oferta satisface el nivel de calidad indicado en las condiciones.

Por ejemplo, si el único requisito de las condiciones es TTF $\geq$ 6 y el único requisito de oferta es TTF $\geq$ 8 es fácil comprobar que la oferta es conforme con las condiciones ya que el nivel de calidad de la oferta satisface el nivel de calidad indicado en las condiciones y por tanto que existe conformidad. Sin embargo, si la oferta es TTF $\in$ [5..8] no se puede garantizar la conformidad pues el significado de ese requisito en un proveedor es que se compromete a un TTF entre 5 y 8 pero no dice nada sobre que valor es el más probable (igual siempre ofrece un TTF de 8 que un TTF de 5). Por tanto, es posible que el nivel de calidad de la oferta no satisfaga el nivel exigido en las condiciones, por ejemplo si el TTF es de 5.

Plantear la comprobación de la conformidad como un CSP también es muy sencillo. Nuestra propuesta es considerar esta demostración equivalente a determinar si las soluciones de la restricción asociada a la oferta O es un subconjunto de las soluciones de la restricción asociada a las condiciones C (ver figura §5.2). Si el conjunto de soluciones de O es un subconjunto de C podemos estar seguros de que el nivel de calidad de la oferta siempre satisfará al nivel de calidad de las condiciones. Sin embargo, si existe al menos una solución de la restricción asociada a la oferta que no lo es de la restricción asociada a las condiciones, no podemos garantizar que la oferta satisfaga las condiciones en todos los casos.

5.1.2.4. Utilidad de un requisito

No todas las posibilidades que existen para satisfacer un requisito son igualmente interesantes. Por ejemplo, si un cliente está buscando el proveedor que ofrece un determinado producto con la garantía de que el tiempo entre fallos sea superior a 4 minutos, un proveedor que ofrece un TTF>10 podría ser más interesante o útil que otro que ofreciera un TTF>6.

Con objeto de automatizar el proceso de elección del proveedor que resulta más interesante definimos una operación que calcula el grado de interés o la utilidad que tiene una determinada restricción. Una primera aproximación utilizando como base la teoría de decisión con múltiples criterios [Miller 1970] sería:

$$\mathcal{U}(c) = \sum_{i=1}^{n} w_i v_i, \qquad \sum_{i=1}^{n} w_i = 1, \quad w_i > 0, \quad v_i \mapsto [0,1], \quad i = 1, \ldots, n \tag{5.1}$$

donde v_i y w_i representan respectivamente a la función de utilidad[2] y a la importancia relativa de la utilidad del atributo i-ésimo que participa en la restricción c.

Esta primera aproximación resulta adecuada para calcular la utilidad de un requisito siempre y cuando está representado por una restricción en igualdad, que como vimos en el capítulo §3 es el caso de la mayoría de las propuestas actuales. Sin embargo, en el caso de requisitos expresados con restricciones en desigualdad esta forma de calcular la utilidad necesita ser refinada pues no es posible conocer para qué valor del atributo debe evaluarse la función de utilidad: ¿el valor mínimo? ¿el valor máximo?. Por ejemplo, si quisiéramos calcular la utilidad de TTF∈[5..10], al intentar aplicar la ecuación §5.1 tendríamos que decidir para que valor de TTF tomaríamos su utilidad ¿en TTF=5?,¿en TTF=10?.

Nosotros proponemos transformar el cálculo de la utilidad en un problema de optimación, más concretamente, en el cálculo del valor mínimo de la utilidad que ofrece un determinado atributo. De este modo, cuando digamos que la restricción c_1 es más útil que c_2 tendremos la garantía de que no existe ninguna solución de c_2 que tenga una mayor utilidad que c_1. Por tanto, la utilidad de una restricción se calculará a partir de

[2]El significado de la función de utilidad de un atributo en QRL es el mismo que en el modelo de Dujmovic (sección §3.2.5).

$$\mathcal{U}(c) = \begin{array}{l} \text{mín} \\ \text{st } c \end{array} \sum_{i=1}^{n} w_i v_i$$

Se podría haber optado por un criterio más sofisticado pero como veremos más adelante, este criterio es el más adecuado para los problemas que se resuelven con QRL.

5.2. Construcción de catálogos de atributos

En QRL se considera como atributo de calidad cualquier característica que puede medirse con métodos directos o indirectos. Un método es directo cuando sirve para el obtener el valor de un atributo realizando una medición directa del producto software en el que estamos interesados, mientras que es indirecto cuando su valor se obtiene como una combinación de los valores de otros atributos. Esto da lugar a una subclasificación de los atributos en simples y derivados, respectivamente.

El tiempo de respuesta de un servicio WEB, el tamaño en bytes de un COTS y el precio de un ERP son tres atributos simples pues son medibles directamente. Por contra, el grado de flexibilidad de un diseño arquitectónico, la fiabilidad de un componente, o la facilidad de uso de un sitio WEB son atributos derivados pues su medida se obtiene como una función de las medidas de otros atributos.

Por regla general, los atributos simples son definidos y medidos por el proveedor de un producto y son válidos para todos los posibles clientes. Por su parte, los atributos derivados suelen ser definidos y medidos por el cliente de un producto, aunque es aconsejable utilizar los atributos derivados de catálogos estándares.

Tal y como se apunta en [Botella *et al.* 2001], diseñar un catálogo de atributos correcto y estable implica una revisión concienzuda del dominio del problema por lo que se convierte en una tarea que requiere un considerable esfuerzo. Con objeto de minimizar el esfuerzo necesario para construir nuevos catálogos resulta crucial poder construir catálogos extendiendo otros.

En este sentido, QRL distingue entre dos tipos de catálogos: aquéllos que contienen únicamente definiciones de atributos simples que suelen estar definidos por los fabricantes del producto o por organizaciones de

estandarización y aquéllos que contienen definiciones de atributos derivados y que suelen ser definidos por los equipos de desarrollo de cada aplicación.

Además de aumentar el grado de reutilización de los catálogos de atributos simples, la separación de catálogos forma parte de la estrategia utilizada QRL para conseguir la interoperabilidad entre documentos definidos con diferentes catálogos de atributos.

QRL también exige que por cada atributo, ya sea simple o derivado, se defina al menos un requisito abstracto. De este modo, se reduce la probabilidad de definir atributos innecesarios.

5.2.1. Catálogos de atributos simples

Para que un atributo simple quede completamente definido se requiere un identificador, una descripción en lenguaje natural de su significado, los valores que puede tomar su medida y la unidad en la que se expresa el valor del atributo. Fíjese que no se incluye en la definición la función de utilidad del atributo (algo habitual en los modelos de tercera generación), pues de ese modo aumentan las posibilidad de utilizar la misma definición del atributo en varios documentos.

La figura §5.3 y §5.4 muestran dos catálogos de atributos de calidad. El primero recoge algunos atributos habituales en el terreno de los MOWS y el segundo en los sistemas multimedia. Como puede observarse, un catálogo se identifica por un URI (*Universal Resource Identifier*) de forma similar a la utilizada en XML para los espacios de nombres [Consortium 2000].

En QRL los atributos pueden tomar valores en cinco dominios base diferentes:

- **Enteros**: Sus elementos son números enteros pertenecientes a un intervalo. Por ejemplo, un posible dominio para el tiempo entre fallos (TTF) podría ser [10..100000]. Si no se especifica ningún intervalo se considera que por defecto es $[-\infty..\infty]$.

- **Reales**: Sus elementos son números reales pertenecientes a un intervalo. Por ejemplo, un posible dominio para la disponibilidad (AVAIL), definida como la probabilidad (expresada en %) de que el sistema se

```
using org.mks.units;
catalogue com.acme.stdMOWS {
 attributes {
  TTF {
   description:"(Time To Fails). Tiempo que transcurre entre dos fallos consecutivos";
   domain: int [0..∞] min;
   statistic: min, mean, percentile80; }
  TTR {
   description: "(Time To Repair). Tiempo de recuperación ante un error";
   domain: int sec;
   statistic: max, mean, percentile90;}
  DELAY {
   description: "(Delay). Tiempo de respuesta de un servicio";
   domain: int [0, 60*60] sec; }
  AVAIL {
   description: "Availability. Probabilidad de poder usar el servicio cuando se solicita";
   domain: real [0..100] %; }
  SFAIL {
   description: "(Server Failure). Indica cómo se comporta el servicio tras un error.
   Puede tomar tres valores: halt, si se queda inactivo indefinidamente,
   initialState si vuelve a un estado inicial conocido y rollBack si vuelve a un
   punto de sincronización y deshace todos los cambios realizados antes del fallo.";
   domain: enum {halt, initialState, rolledBack}; }
  FMASK {
   description: "(Failure masking). Indica el tipo de fallo que puede presentar el
   uso de un servicio. Puede tomar un conjunto de valores entre los siguientes:
   early indica que puede dar la respuesta demasiado pronto, late indica que puede
   dar la respuesta demasiado tarde, state indica que puede tras la respuesta puede
   quedar en un estado inconsistente, value indica que puede dar una respuesta
   equivocada, omission indica que puede no dar una respuesta.";
   domain: set {early, late, state, value, omission}; }
  REBIND {
   description: "(Rebinding). Indica si es necesario realizar una nueva conexión
   (binding) con el servicio tras un fallo de éste".
   domain: bool; }
  THR {
   description: "(Throughput). Indica el número de veces por minuto que puede ser
   invocado el servicio".
   domain: int [0, 1000]; }
 }
}
```

Figura 5.3: *Definición de un catálogo de atributos simples en* QRL.

```
using org.mks.units;
catalogue com.acme.multimedia {
 attributes {
  STREAM {
   description: "(Streaming). Indica que está disponible la posibilidad de "streaming";
   domain: bool; }
  COST {
   description: "(Cost). Indica el precio de utilizar el servicio";
   domain: real [0..∞] euro; }
  MEDIA {
   description: "(Media). Indica el tipo de enlace que puede ser utilizado para
   acceder al servicio: ADSL línea ADSL, ISDN red digital de servicios
   integrados, Modem-56k red telefónica conmutada ";
   domain: set {ADSL, ISDN, Modem56k}; }
  FORMAT {
   description: "(Format). Indica el formato del archivo de la película: quicktime,
   microsoft, DVD_1, DVD_2, MPEG}";
   domain: enum {MOV, WMV, DVD_1, DVD_2, MPEG}; }
  RES {
   description: "(Resolutions). Indica la resolución para la que está preparada la
   película: LOW (320*240), MEDIUM (640*480), HIGH (1024*768)".
   domain: set {LOW, MEDIUM, HIGH}"; }
  LANG {
   description: "(Language). Idiomas en los que está disponible el contenido";
   domain: set {EN, FR, DE, SP}; }
  SUBT {
   description: "(Subtitle). Indica los subtítulos disponibles";
   domain: set {EN, FR, DE, SP}; }
 }
}
```

Figura 5.4: *Definición de un catálogo de atributos en* QRL.

encuentre fuera de servicio, podría ser [0..100]. Al igual que con los enteros el intervalo por defecto es $[-\infty..\infty]$.

- **Lógicos**: Sus elementos sólo pueden tomar dos valores **true** o **false**. Por ejemplo, el soporte de *streaming* (STREAM) por parte de un servidor de video.

- **Enumerados**. Sus elementos son valores que se denotan mediante identificadores. Por ejemplo, para describir las posibles maneras que tiene un sistema de comportarse tras un fallo (SFAIL, *Server Failure*), no se requiere un dominio tan amplio como los numéricos, basta con algunos valores que puedan ser referenciados por identificadores, por ejemplo: halt, si se queda inactivo indefinidamente, initialState si vuelve a un estado inicial conocido y rollBack si vuelve a un punto de sincronización y deshace todos los cambios realizados antes del fallo.

- **Conjuntos**. En ocasiones, necesitamos varios literales para establecer el valor de un atributo. Por ejemplo, el atributo FMASK especifica los tipos de fallos de los que se debe preocupar el cliente de un servicio WEB, los cuales, siguiendo la clasificación expuesta en [Cristian 1991], pueden ser: i) fallos de omisión (omission), indican que el servidor puede no responder a las peticiones realizadas, ii) fallos en las respuestas, indica que el servidor puede devolver valores incorrectos (value) o realizar transiciones de estado incorrectas (state) y iii) los errores de tiempo, que indican que el servicio puede responder demasiado pronto (early) o demasiado tarde (late).

  Es evidente que un servicio puede presentar más de un tipo de fallo de manera simultánea (p.e. fallos de omisión y devolver valores incorrectos), siendo necesario para describir tales situaciones, utilizar conjuntos de valores enumerados ({ omission, value}).

5.2.1.1. Modificadores estadísticos

Existen atributos de tipo numérico (**real** e **int**) para los que resulta muy útil disponer de una caracterización estadística de su medida. Por ejemplo, decir que el tiempo medio de respuesta de un servicio es de 2 segundos puede no ser suficiente en todas las situaciones, pues no es lo mismo que la desviación media del tiempo de respuesta sea de 20 segundos o que sea de 0.5 segundos.

Para estos casos, QRL propone el uso de modificadores estadísticos, esto es, un conjunto de palabras reservadas que representan las medidas de localización más habituales en distribuciones estadísticas. Los modificadores que se pueden utilizar son: el valor medio (mean), la desviación típica (variance), todos los percentiles posibles (desde percentile 1 hasta percentile 99), el valor máximo (max) y el valor mínimo (min).

En el catálogo de la figura §5.3 existen algunos atributos para los que se indican algunos modificadores estadísticos. El significado de las restricciones de calidad en las que aparecen modificadores se corresponde se ve afectado de acuerdo con el significado de dichos modificadores. Por ejemplo:

- TTF.min > 10, indica que en el peor de los casos, transcurrirán más de 10 segundos entre dos fallos consecutivos.
- TTF.percentile 80 > 5, indica que el tiempo entre fallos será mayor que 5 en al menos el 20 % (100 − 80) de los casos.

5.2.2. Catálogo de atributos derivados

El principal objetivo que se persigue con la definición de atributos derivados es facilitar la definición de requisitos en los que intervienen varios atributos. Por ejemplo, un requisito expresado como

El servicio deberá tener un tiempo entre fallos deberá ser superior a 120 minutos y un tiempo de recuperación inferior a 90 minutos. Además, deberá permitir la reconexión automática tras un error.

puede quedar reducido a una expresión tal como:

El servicio deberá ofrecer un alto grado de fiabilidad.

siempre que definamos un atributo para medir la fiabilidad a partir del tiempo entre fallos (TTF), el tiempo de recuperación (TTR) y la política de tolerancia a fallos (REBIND). La figura §5.5 muestra una posible definición de la fiabilidad (*reliability*) que cumple este propósito.

La definición de un atributo derivado también puede ser interpretada como la definición de una función que clasifica en diferentes grupos los

```
attributes {
 reliability {
  description: "Da idea de la fiabilidad de un producto. Se corresponde con la
  característica Reliability de la ISO 9126."
  relies: TTF, TTR, REBIND;
  stereotypes:
   LOW: TTF < 90 and TTR < 120;
   MEDIUM: TTF ∈ [90..120) and TTR < 90;
   HIGH: (TTF ∈ [120..240) and TTR < 90 and REBIND = true) or TTF > 240;
 }

 availability {
  description: "Atributo derivado que define diferentes estereotipos de medidas
  sobre la disponibilidad. Algunos estereotipos están basados en [Gray y Reuter 1993]"
  relies: AVAIL;
  stereotypes:
   LOW: AVAIL < 90;
   MEDIUM: AVAIL ∈ [90..98);
   HIGH: AVAIL ∈ [98.. 99.999);
   HIGHLY: AVAIL ≥ 99.999;
 }

 maturity {
  description: "Da idea de la madurez de un producto. Se corresponde con la
  subcaracterística maturity de la ISO 9126, la cual coincide esencialmente
  con el atributo TTF"
  relies: TTF;
  stereotypes:
   LOW: TTF < 90;
   MEDIUM: TTF ∈ [90..120);
   HIGH: TTF ∈ [120..240);
   VERY_HIGH: TTF ≥ 240;
  }

 recoverability {
  description: "Da idea de la capacidad de reestablecer el servicio después de un fallo"
  relies: TTR, REBIND
  stereotypes:
   LOW: TTR ≥ 90;
   MEDIUM: TTR < 90;
   HIGH: TTR < 90 and REBIND = true;
 }
}
```

Figura 5.5: *Definición de atributos derivados en* QRL.

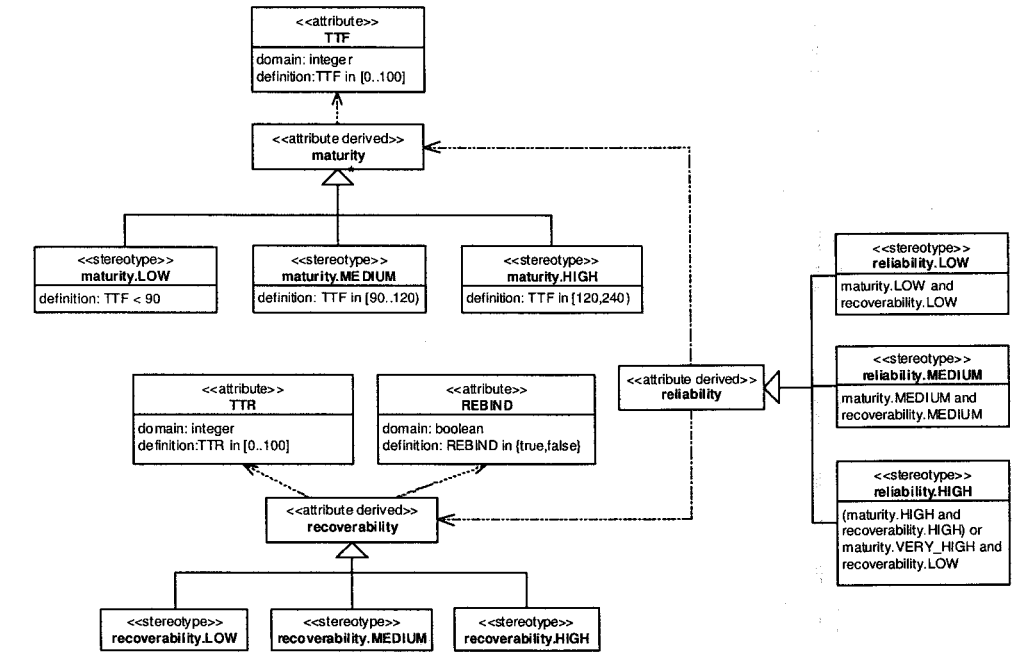

Figura 5.6: *Ejemplo de las relaciones existentes entre atributos, atributos derivados y estereotipos.*

valores de los atributos de los que depende. La función asigna a cada uno de estos grupos una etiqueta cuyo texto refleja un estereotipo del atributo. Por ejemplo, según [Gray y Reuter 1993] los sistemas de telefonía son "altamente–disponibles" cuando no están fuera de servicio más de cinco minutos al año, es decir, cuando se cumple que AVAIL $\geq$ 99.999. De este modo, con objeto de facilitar la definición de requisitos sobre la disponibilidad haciendo uso del estereotipo de Gray se podría utilizar la definición del atributo availability de la figura §5.5.

La definición de atributos derivados haciendo uso de estereotipos es un mecanismo que aumenta el grado de abstracción de las definiciones. De hecho, un atributo derivado puede verse como un elemento abstracto a partir del cual pueden definirse medidas estándares o estereotipos (ver figura §5.6).

Este aumento del grado de abstracción reduce el impacto de los cambios en las definiciones de los atributos simples o derivados de los que se

depende. Por ejemplo, si redefinimos el atributo reliability de la figura §5.5 como

```
reliability {
  description: "Da idea de la fiabilidad de un producto. Se corresponde con la
  característica Reliability de la ISO 9126."
  relies: maturity, recoverability
  stereotypes:
    LOW: maturity.LOW and recoverability.LOW;
    MEDIUM: maturity.MEDIUM and recoverability.MEDIUM;
    HIGH: maturity.HIGH and recoverability.HIGH
        or maturity.VERY_HIGH and recoverability.LOW;
}
```

y modificamos los criterios para clasificar los diferentes niveles de recoverability según los criterios de la empresa ACME a

```
recoverability {
  description: "Da idea de la capacidad de reestablecer el servicio después de
  un fallo según los criterios de la empresa ACME";
  relies: TTR, REBIND, SFAIL
  stereotypes:
    LOW: TTR ≥ 90;
    MEDIUM: TTR < 90 and SFAIL= initialState;
    HIGH: TTR < 90 and REBIND = true and SFAIL= rollBack;
}
```

la definición de la fiabilidad seguiría siendo válida.

La definición de atributos derivados y estereotipos puede verse como una generalización de los contratos de QML (ver sección §3.3.2.2). De hecho, el criterio de validación de los estereotipos de un atributo derivado es el mismo que en QML: la definición de un estereotipo debe ser conforme con la del atributo o estereotipo del que deriva. Según esto, la definición de los estereotipos maturity.MEDIUM no es válida, pues no es conforme con la definición del atributo TTF.

5.2.3. Patrones lingüísticos formales

Los catálogos de atributos son la base sobre la que se construyen documentos de calidad, por lo que deben estar diseñados para facilitar la escritura de los documentos de requisitos y en la medida de lo posible

con soporte para la dualidad. Con objeto de conseguir dicho propósito, QRL propone la definición de *patrones lingüísticos formales*, o abreviadamente FLP (*Formal Linguistic Pattern*) que son una extensión de los *patrones lingüísticos* presentados en [Durán *et al.* 1999a, Durán 2000] para describir requisitos (tanto funcionales como de calidad) mediante frases pero sin los problemas de ambigüedad inherentes al lenguaje en natural.

Un FLP es una estructura compuesta por una frase estándar o patrón lingüístico que representa a todos aquellos requisitos que se pueden expresar reemplazando los *placeholder* y una restricción matemática estándar que expresa la condición de satisfactibilidad de los requisitos que se pueden expresar con la frase estándar.

En la notación usada para describir los FLP, las expresiones entre <y> deben ser convenientemente reemplazadas, mientras que las expresiones que se encuentren entre {y} y separadas por comas representan opciones de las que se deberá escoger una.

Por ejemplo, a partir del FLP

El tiempo entre fallos deberá ser {superior a, de, inferior a} <n> minutos
= TTF {>, =, <} n

Se pueden definir requisitos tales como

El tiempo entre fallos deberá ser superior a 120 minutos
El tiempo entre fallos deberá ser inferior a 30 minutos

La posibilidad de definir PLFs proporciona tres grandes beneficios:

- *Validación de atributos*. Si no es posible definir al menos un FLP sobre un atributo, es conveniente revisar la definición de dicho atributo, pues de poco sirve un atributo sobre el que no se puede definir un requisito.
- *Facilita la dualidad*. La ligadura que se establece entre la expresión en lenguaje natural del requisito y su restricción matemática equivalente facilita la traducción automática.
- *Facilitar la escritura*. Se pueden definir requisitos instanciando FLPs lo que proporciona un ahorro importante de tiempo.

```
catalogue com.acme.stdMOWS {
 pattern {
   reliability {
     p1: "La fiabilidad del producto debe ser {baja, media, alta}"
         = reliability.{LOW, MEDIUM, HIGH};
     p2: "El producto debe ofrecer como mínimo un grado de fiabilidad {bajo,medio,alto}"
         = reliability.{LOW, MEDIUM, HIGH};
   }
   maturity {
     p3: "La probabilidad de que exista un error debe ser {pequeña,media,alta}"
         = "maturity.{LOW,MEDIUM,HIGH} ";
   }
   TTF {
     p4: "El tiempo medio entre fallos será de al menos <n> minutos"
         = "TTF.mean ≥ <n> minutos";
     p5: "El tiempo entre fallos seguirá una distribución de valor medio <valor>
          y una desviación típica máxima de <valor>"
         = " TTF.mean = <valor>  and TTF.variance ≤ <valor>";
   }
   FMASK {
     p6:  "Debe estar garantizada la respuesta del servicio";
         = "{ omission } ⊈ FMASK;
 }
}
```

Figura 5.7: *Definición de patrones lingüísticos formales en* QRL.

La figura §5.7 muestra varias definiciones de FLPs en QRL. Con objeto de simplificar al máximo la definición, QRL asume que la correspondencia entre las alternativas utilizadas por la frase y la restricción sigue el orden posicional.

5.3. Creación de un documento básico de calidad

El documento de calidad más simple que puede especificarse en QRL tiene tres secciones: la definición de los atributos, la definición del producto y la definición de los requisitos. La figura §5.8 muestra un documento de este tipo expresado en lenguaje natural y en QRL.

A primera vista, incluso sin conocer aún la sintaxis de QRL, es fácil comprobar que haciendo uso de FLPs no resulta demasiado complicado desarrollar una herramienta CASE que permita especificar un documento de requisitos en lenguaje natural y que genere automáticamente su documento equivalente en QRL [3]. No obstante, dado que implementar el proceso de traducción no forma parte de los objetivos de nuestro trabajo, no se discutirá más sobre este tema.

5.3.1. Especificación de atributos

La primera sección de un documento de calidad está dedicada a la especificación de atributos de calidad. Esta especificación puede realizarse importando catálogos de atributos y/o definiendo nuevos catálogos de atributos. En la figura §5.8 se hace uso de la instrucción **using** para importar dos catálogos: com.acme.stdMOWS y com.acme.multimedia.

5.3.2. Especificación de productos

Para especificar un producto en QRL tan sólo es necesario definir un identificador que lo represente. En el caso de estar interesados en especificar requisitos sobre alguno de sus subproductos será necesario definir un identificador para cada uno de ellos. Con estas posibilidades se consigue un alto grado de abstracción en el sentido descrito en la sección §4.3.

La figura §5.8.b la especificación de un producto que representa a cualquier servicio WEB que implemente la interfaz IVideoServer. También se especifican algunos métodos de dicha interfaz (no se obliga a que sean todos), por lo que será posible especificar requisitos que afecten únicamente a dichos métodos.

5.3.3. Requisitos básicos

Una vez especificado un producto y un catálogo de atributos ya es posible definir requisitos de calidad sobre dicho producto. Dichos requisitos pueden estar referidos sobre el producto en su conjunto (cláusula **global**),

[3] Actualmente estamos extendiendo la herramienta CASE REM desarrollada en [Durán 2000] para lograr este propósito.

Condiciones de uso para utilizar el servicio que implementa la interfaz IVideoServer

1. **Catálogos de atributos.** Los catálogos de referencia son: com.acme.stdMOWS y com.acme.multimedia.
2. **Productos.** Estamos interesados en cualquier servicio WEB que implemente la interfaz IVideoServer. En adelante nos referiremos a este tipo de servicio como P. P debe ofrecer la posibilidad de establecer condiciones sobre las siguientes operaciones de su interfaz: find, select, play, stop y rewind.
3. **Condiciones básicas.**

    3.1 Condiciones sobre el producto en su conjunto.

        3.1.1. El coste de uso debe ser como máximo de 2€ por cada sesión.

        3.1.2. El servicio debe estar disponible al menos para usuarios de modem e ISDN.

    3.2. Condiciones sobre cada una de las operaciones.

        3.2.1. El tiempo entre fallos debe ser alto.

        3.2.2. La capacidad de restablecer el servicio debe ser normal.

    3.3. Condiciones particulares.

        3.3.1. La capacidad de restablecer el servicio debe ser alta para la operación play.

        3.3.2. El retraso en la operación stop debe ser inferior a 250 milisegundos

a) Documento de condiciones en lenguaje natural

```
using com.acme.stdMOWS, com.acme.multimedia;
products { IVideoServer {find, select, play, stop, rewind}; }
conditions for IVideoServer {
 global {
  r1: COST ≤ 2;
  r2: MEDIA ⊃ {Modem, ISDN};
 }
 all {
  r3: maturity.HIGH; // equivale a TTF ∈ [120,240)
  r4: recoverability.MEDIUM; // equivale a TTR < 90
 }
 for play  { r5: recoverability.HIGH; } // equivale a TTR < 90 and REBIND = true;
 for stop { r6: DELAY < 0.250; }
}
```

b) Documento de condiciones en QRL equivalente al de arriba

Figura 5.8: *Documento de condiciones básico especificado en lenguaje natural y en* QRL.

sobre un subproducto en particular (cláusula **for** idSubproducto) o común a todos los subproductos (cláusula **all**).

La figura §5.8 muestra un ejemplo de uso de estas tres situaciones. El precio de utilización (COST) y el tipo de enlace que puede ser utilizado para acceder al servicio (MEDIA) se indican en la cláusula **global**, pues suponemos que en este caso no pueden ser aplicados independientemente a cada operación de la interfaz. Las condiciones sobre el tiempo entre fallos y el tiempo de recuperación se indican en la cláusula **all**, por entender que todas las operaciones (find, select, play, stop y rewind) deben ofrecer el mismo grado de fiabilidad. Por último, dado que sólo se especifica un requisito sobre el tiempo de retraso (atributo DELAY), concretamente para la operación stop, dicho requisito se especifica haciendo uso de la cláusula **for**.

La especificación de un requisito básico consiste de un identificador del requisito[4] y su restricción matemática equivalente. En cuanto al tipo de restricciones que puede especificarse, QRL ofrece la posibilidad de definir restricciones sobre varios atributos utilizando los operadores aritméticos y operaciones matemáticas más habituales. En la sección §?? se describe con detalle el tipo de restricciones que pueden ser especificadas en QRL, no obstante, el ejemplo de la figura §5.7 muestran varios tipos de restricciones válidas en QRL.

5.3.3.1. Comprobación de la consistencia

Comprobar la validez de la sección de requisitos básicos equivale a comprobar que todas las restricciones asociadas a los requisitos son consistentes entre sí, lo cual garantiza la ausencia de contradicciones, o lo que es lo mismo, que existe posibilidad de satisfacer a todos los requisitos.

Según esto, el documento de requisitos del ejemplo sería válido pues todas las restricciones son consistentes entre sí, incluso las restricciones de los requisitos r4 y r5, pues aunque aparentemente recoverability.MEDIUM es inconsistente con recoverability.HIGH (si es alto no puede ser medio) las restricciones asociadas de estos requisitos si lo son. Basta comprobar la satisfactibilidad de TTR $<$ 90 **and** TTR $<$ 90 **and** REBIND $=$ true.

[4]No es necesario en el caso de un documento básico, pero si para los documentos con cláusulas de negociación. Hemos decidido utilizarlo para podernos referir a un requisito por su identificador durante la explicación.

5.4. Documentos de calidad con consciencia temporal

QRL hace posible la especificación de documentos con consciencia temporal. La figura §5.9 muestra un documento de condiciones basado en el de la figura §5.8 que establece un período de vigencia para el documento en su conjunto y uno diferente para el requisito sobre el tiempo medio entre fallos.

Como puede apreciarse a simple vista, la sintaxis elegida para especificar un período de vigencia ofrece bastantes posibilidades, ya que no sólo permite indicar una fecha inicial y final, también es posible indicar fechas sueltas, los días de la semana y las franjas horarias que conforman el período de vigencia. De este modo, un período de vigencia no está restringido a un único intervalo de tiempo (fecha inicial, fecha final) sino que puede referirse a una familia de intervalos con una duración mínima de 1 minuto.

Las facilidades sintácticas que proporcionar QRL para especificar un período de vigencia son muy variadas. En el anexo §A se describen con detalle estas facilidades, no obstante, a continuación comentamos el formato básico de especificación.

El formato básico de un período es **from** patronHorario **on** patronSemanal **during** intervalosFechas, donde

- patronHorario es una secuencia de intervalos horarios con el formato $h_1{:}m_1..h_2{:}m_2,\ldots,h_{n-1}{:}m_{n-1}..h_n{:}m_n$, donde h_i representa a una hora y m_i representa los minutos. Por ejemplo, para indicar que estamos interesados únicamente en el intervalo que va de las 9:00 a las 14:00 y de las 16:00 a las 18:00, debemos especificar **from** 9:00..14:00,16:00..18:00.
- patronSemanal es una secuencia de días de la semana con el formato **on** $w_1..w_2,\ldots,w_{n-1}..w_n$, donde w_i representa a los días de la semana. Por ejemplo, para indicar que estamos interesados únicamente en los lunes, miércoles, sábados y domingos, debemos especificar **from** MON,WED, SAT, SUN o bien **from** SAT..MON,WED.
- intervalosFechas es una secuencia de fechas que sigue el formato **during** $d_1..d_2,\ldots,d_{n-1}..d_n$, donde d_i representa a una fecha. Por ejemplo, para indicar que estamos interesados en el verano del año 2002, debemos especificar **during** 21/06/2002..21/09/2002.

Condiciones de uso para utilizar el servicio que implementa la interfaz IVideoServer

1. **Catálogos de atributos.** Idem figura §5.8.
2. **Productos.** Idem figura §5.8.
3. **Período de vigencia.** El período de vigencia común para las condiciones recogidas en este documento es el comprendido entre el 15 de Julio y el 15 de Diciembre de 2002. Además, las condiciones podrán estar referidas a los siguientes subperíodos:
   - **PUNTA**: Todos los días del el 15 de Julio hasta el 15 de Septiembre (ambos inclusive) desde las 8:00 hasta las 2:00. Todos los sábados y domingos desde el 16 de Septiembre hasta el 15 de Diciembre y los días 1 de noviembre, 6 y 9 de diciembre desde las 8:00 hasta las 0:00.
4. **Condiciones básicas.** Idem a las de la figura §5.8 salvo el requisito sobre el tiempo entre fallos que ahora pasa a ser:
   - 4.2.1. El tiempo entre fallos debe ser alto en hora PUNTA y normal el resto del tiempo.

a) Documento de condiciones en lenguaje natural

```
using com.acme.stdMOWS, com.acme.multimedia;
products { P: IVideoServer {find, select, play, stop, rewind}; }
valid zone: GMT +1 {
  global { during 15/JUL/2002..15/DEC/2002;}
  PUNTA {
    from 0..2,8..24 during 15/JUL/2002..15/SEP/2002 and
    from 8..24 on SAT,SUN during 16/JUL/2002..15/DEC/2002,1/NOV/2002,
                                 6/DIC/2002, 9/DIC/2002;}
}
conditions for P {
 global { ...};
 all {
  r3: maturity.HIGH on PUNTA; // equivale a TTF ∈ [120,240);
  r3b: maturity.MEDIUM on global except PUNTA; // equivale a TTF ∈ [90,120);
  r4: recoverability.MEDIUM;
 }
 for play { ... };
 for stop { ... };
}
```

b) Documento de condiciones en QRL equivalente al de arriba

Figura 5.9: *Documento de condiciones con consciencia temporal especificado en lenguaje natural y en* QRL.

Períodos de vigencia con nombres

Por regla general, los períodos de vigencia individuales son comunes a varios requisitos. Por ejemplo, es frecuente que algunos requisitos sean válidos únicamente en horario de oficina y otros únicamente en horario de descanso. Para facilitar la escritura, es posible asignar un identificador a dichos períodos para referirse a ellos de manera más abreviada. En el ejemplo de la figura, se ha definido el identificado PUNTA para referirnos al período de tiempo en el que se prevee una mayor demanda del servicio.

Por defecto, cuando un documento tiene consciencia temporal (incluye al menos una cláusula **valid**) todos los requisitos tienen asociado el mismo período de vigencia que el documento en su conjunto, que se conoce como **global** (ver requisito r3b de la figura §5.9). No obstante, su indicación no es obligatoria.

Uso de la zona horaria

Si una documento de calidad se utiliza para especificar la calidad de un servicio WEB que esta accesible a cualquier zona del planeta, resulta necesario especificar el período de vigencia respecto a una referencia horaria común. Con tal propósito, en QRL es posible especificar la zona horaria (cláusula **zone**) sobre la que están referidos los períodos de vigencia. Si no se indica la zona horaria se considerará por defecto que está en la zona Greenwich.

5.4.1. Comprobación de la consistencia

En principio puede parecer que comprobar la validez de un documento de requisitos con consciencia temporal sólo se diferencia respecto al caso de los documentos en comprobar que el período de vigencia es válido, es decir, que los intervalos horarios semanales y de fechas corresponden a períodos válidos. Evidentemente estas comprobaciones son necesarias, pero no son las únicas, también es necesario comprobar que:

- El período de vigencia no es vacío, pues en tal caso estaríamos anulando la validez del requisito asociado.
- El período de vigencia asociado explícitamente a un requisito está incluido en el período de vigencia del documento. No tiene sentido indicar que el documento tiene validez durante el primer semestre del año y algunos requisitos individuales el año entero. Evidentemente es necesario revisar la especificación de estos períodos.

- La consistencia entre dos requisitos no puede ser entendida como hasta ahora, pues de ese modo, se consideran inconsistentes requisitos que no lo son. Por ejemplo, en la figura §5.9 y de acuerdo con el criterio de validez de una sección de requisitos que vimos en la sección anterior, los requisitos r3 y r3b no son consistentes pues la restricción TTF $\in$ [120,240) **and** TTF $\in$ [90,120) no es satisfactible. Sin embargo, si tenemos en cuenta que los requisitos asociados a dichas restricciones están referidos a períodos de vigencia disyuntos deberíamos considerarlos consistentes, pues en realidad no está provocando ninguna contradicción.

  Consistencia entre requisitos con consciencia temporal

  Por tanto, es necesario revisar la noción de consistencia entre requisitos que vimos en la sección §5.1.2.1 cuando estos poseen consciencia temporal. Diremos que dos requisitos son consistentes si sus restricciones asociadas son satisfactibles durante el período de vigencia correspondiente (esto garantiza la validez individual de un requisito) y consistentes entre sí en aquellos períodos de tiempo comunes a ambos períodos de vigencia. De este modo, comprobar la validez de un documento sin consciencia temporal no es más que un caso particular de esta comprobación más general, en la que todos los requisitos tienen el mismo período de vigencia.

  La figura §5.10 ilustra gráficamente cuando es necesario comprobar la consistencia entre dos requisitos con consciencia temporal, el primer requisito tiene un período de vigencia (P_1) compuesto por tres intervalos (I_1, I_2, I_3) y el segundo requisito un período (P_2) compuesto únicamente por dos intervalos (I'_1, I'_2), por lo que sólo es necesario cuando $P_1 \cap P_2 \neq \emptyset$, que en este caso viene dado por los intervalos $I_1 \cap I'_1$ e $I_3 \cap I'_2$.

  En el caso de nuestro documento de ejemplo, es fácil comprobar su validez, pues dado que los períodos de vigencia de r3 y r3b son disyuntos, no importa que sus restricciones no sean consistentes entre sí, basta conque sean satisfactibles individualmente.

5.5. Documentos con criterios de utilidad

Como vimos en la sección §5.1.2.4, QRL utiliza el concepto de utilidad de una restricción para poder elegir la oferta que mejor satisface una restricción. Para calcular la utilidad de una restricción cualquiera c propusimos la fórmula

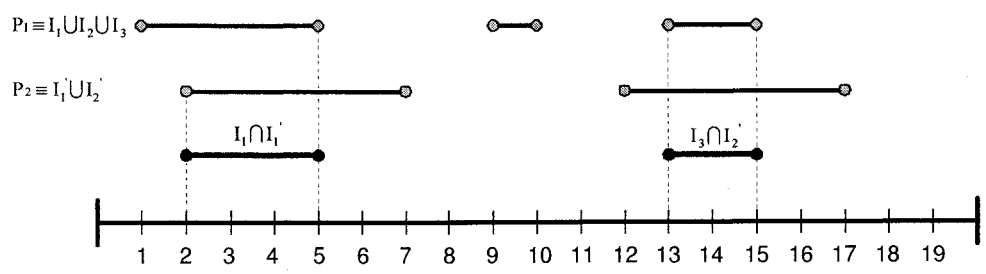

Figura 5.10: *Comprobación de la consistencia entre dos requisitos con consciencia temporal.*

$$\begin{array}{ll} \text{mín} & \sum_{i=1}^{n} w_i v_i \\ \text{st} & c \end{array}$$

donde v_i representaba a la función de utilidad del atributo i-ésimo que participa en la restricción c y w_i la importancia relativa de la utilidad del atributo i-ésimo.

Existen dos estrategias para expresar la función de utilidad y el peso de un atributo: añadir dicha información en el catálogo de atributos o añadirlo en una sección diferente del documento de requisitos. La primera estrategia presenta un grave inconveniente, y es que obliga a trabajar al menos con dos catálogos diferentes pues la utilidad de un atributo desde el punto de vista del cliente suele ser contraria (aunque no obligatoriamente) que desde el punto de vista del proveedor. Paliar este inconveniente añadiendo información en el catálogo para distinguir entre cliente y proveedor no es suficiente, pues dos proveedores pueden tener criterios de utilidad diferentes, incluso para el mismo cliente puede variar de un producto a otro.

Con objeto de aumentar las probabilidades de reutilizar el catálogo de atributos (lo que a su vez facilita la interoperabilidad), QRL utiliza la segunda estrategia. La figura §5.11 muestra un documento de requisitos que incluye una sección con la información sobre el peso y la utilidad de algunos atributos. En adelante nos referiremos a esta sección, como *sección de criterios de utilidad*.

En el ejemplo se especifican cuatro criterios de utilidad. El primero sobre el precio de uso del servicio, el segundo sobre el medio de conexión, el

Condiciones de uso para utilizar el servicio que implementa la interfaz IVideoServer

1-4. Idénticas a las de la figura §5.9.

5. **Criterios de utilidad**

5.1. La importancia del precio (4.1.1) es alta. Su utilidad será máxima para un precio de 1€, será del 65 % para un precio de 2€, será del 35 % para un precio de 3€ y dejará de tener utilidad a partir de 4€.

5.2. La importancia del medio de conexión (4.1.2) es baja. Su utilidad es mínima (1 %) si permite el acceso vía modem, es del 30 % si permite el acceso vía ISDN, es del 50 % si permite el acceso vía ISDN y modem, es del 90 % si permite el acceso vía ADSL y del 100 % si permite ADSL, modem e ISDN.

5.3. La importancia del tiempo entre fallos (4.2.1) es muy alta. Su utilidad es mínima (1 %) si el tiempo es bajo, es media (50 %) si el tiempo es medio, es alta (90 %) si el tiempo es alto y es máxima (100 %) si el tiempo es muy alto.

5.4. La importancia del retraso en la operación de parada (4.3.2) es muy alta. Su utilidad es mínima (1 %) para el valor mínimo de retraso que satisface la condición, y es máxima (100 %) para cualquier valor inferior o igual al 80 % del valor mínimo.

a) Documento de condiciones en lenguaje natural

```
// Las secciones previas del documento son idénticas a las de la figura §5.9
utility {
 global {
   COST {HIGH, line { (1,1), (2, 0.65), (3, 0.35) (4,0) }}

   MEDIA { LOW,
    discrete { ({modem}, 0.01), ({ISDN}, 0.3), ({ISDN,modem}, 0.5), ({ADSL}, 0.9),
              ({ADSL,ISDN,modem}, 1) } }
 }

 for all {
   maturity {VERY_HIGH,
    discrete { (LOW, 0.01), (MEDIUM, 0.5), (HIGH, 0.9), (VERY_HIGH, 1) } }
 }

 for stop.DELAY {VERY_HIGH,
    rampe { (min stop.DELAY, 0.01), (min stop.DELAY – 0.8 min stop.DELAY, 1)}}
 }
}
```

b) Documento de condiciones en QRL equivalente al de arriba

Figura 5.11: *Documento de condiciones con criterios de utilidad.*

tercero sobre la madurez de cada una de las operaciones del servicio y el cuarto sobre el tiempo de retraso de la operación stop.

Como puede deducirse de manera intuitiva a partir de la figura, la indicación de la importancia de cada medida se hace utilizando valores predeterminados. QRL permite siete valores estándar: MAX (1), VERY_HIGH (0.9), HIGH (0.7), MEDIUM (0.5), LOW (0.3) y VERY_LOW (0.01); aunque también es posible utilizar directamente un valor numérico entre 0 y 1.

Obsérvese, que no se han definido criterios de utilidad para todas las medidas, pues no todas tienen porque ser decisorias.

En cuanto a la especificación de las funciones de utilidad, existen más posibilidades de las que aparecen en la figura del ejemplo, por lo que dedicaremos la siguiente sección a describirlas.

5.5.1. Especificación de funciones de utilidad

En [Dujmovic y Elnicki 1982, Olsina 1999] se proponen cuatro diferentes maneras de expresar una función de utilidad: gráficamente a partir de diagramas de líneas, barras, etcétera, gráficamente a partir de una escala de preferencias, textualmente indicando las coordenadas relevantes y, por último, textualmente indicando la expresión analítica de la función.

Si se dispone de una herramienta CASE para construir documentos de requisitos en lenguaje natural y en QRL, sería deseable poder ofrecer las cuatro posibilidades. No obstante, en la versión formal del documento sólo es posible especificar una función de utilidad con las dos últimas: como una secuencia de coordenadas relevantes o con una expresión.

5.5.1.1. Funciones de utilidad definidas mediante una secuencia de coordenadas

La figura §5.12 muestra cuatro ejemplos de funciones de utilidad que pueden ser especificadas en QRL como

```
utility {
  TTF {peso, line {(120,0), (240,80), (1440,100), (MAX, 100)}};
  TTR {peso, line {(0,100), (5,10), (10,5), (20, 0)}};
  COST {peso, line {(0,0), (1,1), (2, 0.65), (3, 0.35), (4, 0)}};
  AVAIL {peso, line {(89,0), (90,1), (91,4), ... (99,99), (100,100)}};
```

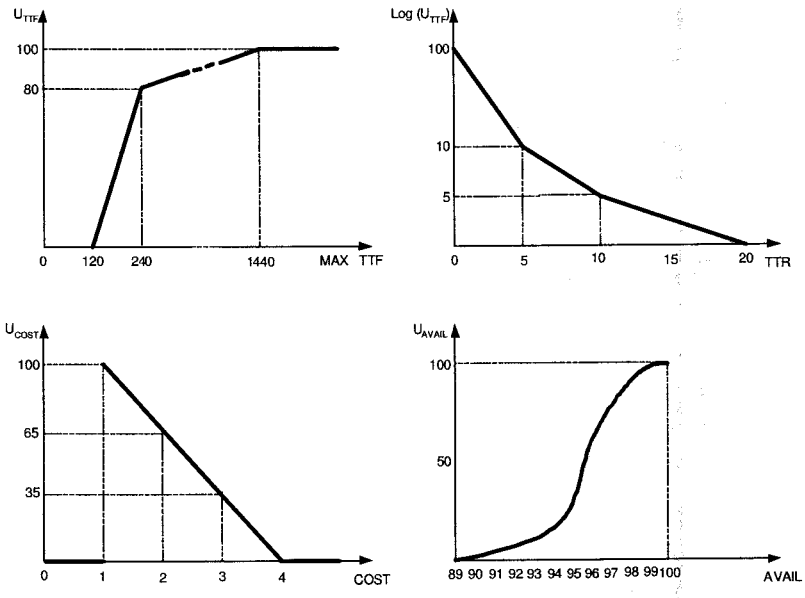

Figura 5.12: *Ejemplos de funciones de utilidad.*

}

Interpretación de una función de utilidad

La interpretación de estas funciones es muy sencilla. Por ejemplo, para el tiempo medio entre fallos (TTF), se prefieren los valores altos, y se exige que sea como mínimo de 120 minutos, a partir de este valor se incrementa la utilidad linealmente hasta los 240 minutos donde se considera que está a un 80 % del valor ideal, que es 1440 minutos (24 horas). Cualquier valor de TTF superior a las 24 horas tendrá la misma utilidad. Es decir, ofrece la misma utilidad un servicio que falla una vez al día que si falla una vez al año.

Para el TTR sin embargo, se prefieren valores pequeños, lo ideal sería un TTR de valor cero, que sólo sería posible en un sistema altamente redundante. La preferencia decrece linealmente hasta el 10 % para un TTR de 5 minutos, hasta el 5 % para un TTR de 10 minutos y hasta el 0 a partir de los 20 minutos.

La utilidad del precio (COST) también es decreciente, es decir deseamos que el precio sea tan bajo como sea posible, aunque nunca inferior a 1 ni igual o superior a 4. Por último la disponibilidad (AVAIL) es una función creciente y sólo esta definida para valores superiores o iguales al 89 %.

Como veremos en el capítulo §6 en el proceso de compilación de un documento de requisitos, estos puntos formarán el soporte para aproximar la función de utilidad mediante interpolación lineal.

5.5.1.2. Funciones de utilidad definidas mediante una expresión

En ocasiones puede interesarnos especificar analíticamente una función de utilidad tanto por reducir la verbosidad como por evitar la necesidad de interpolar. Por ejemplo, la especificación de la utilidad de la disponibilidad de la sección anterior se podría haber especificado como

```
AVAIL { peso, {1/(1 + exp(95 - AVAIL)); }}
```

que define una función sigmoidal centrada para un valor de AVAIL= 95. El lenguaje utilizado para especificar analíticamente una función de utilidad es el mismo que se puede utilizar para especificar expresiones, el cual se describe con detalle en el anexo §A.

Funciones por trozos

QRL también permite especificar la función de utilidad por trozos. Por ejemplo, la utilidad del TTF de la figura §5.11 puede definirse como:

```
TTF {VERY_HIGH,
  {
  case TTF ∈ [120..240): 0.33 TTF - 80;
  case TTF ∈ [240 .. 1440): 0.016 TTF + 76;
  case TTF ≥ 1440: 100;
  default: 0;
  }
 }
```

QRL proporciona cuatro patrones de funciones de utilidad que suelen aparecer con frecuencia, a saber:

- **rampe** { (x1, 1), (x2, 0) }. Se trata de una función lineal de tres tramos. El primer tramo desde el valor mínimo x1 (x1 debe ser menor que x2), el segundo entre x1 y x2 y el tercero entre x2 y el valor máximo del dominio.

  La función de utilidad viene dada por $\frac{x_2 - x}{x_2 - x_1}$ en el tramo central.

- **sigmoide** (center). Se trata de una función sigmoidal que tiene su centro en el valor *center*.

  La especificación anterior de la disponibilidad se reduciría a

  AVAIL { peso, **sigmoide** (95)}

- **parabola** (center). Se trata de una parábola que tiene su centro en el valor *center* y que tiene los ceros en center±1. Por ejemplo, para especificar que el coste de determinado productos que no nos interesa ni los más baratos ni los más caros.

$$1 - (x - center)^2$$

Junto a estos patrones de funciones también existen patrones lingüísticos para su especificación en lenguaje natural. Por ejemplo:

- La disponibilidad del sistema vendrá dada por una sigmoide centrada en <v>= $\frac{1}{1+e^{v-AVAIL}}$

QRL permite especificar una función de utilidad de manera relativa lo que aumenta su posibilidades de ser reutilizada. En el documento de la figura §5.11, la expresión mín stop.DELAY equivale al valor mínimo de stop.DELAY dentro del conjunto de soluciones de la restricción asociada al documento. En este caso, el criterio de utilidad se correspondería con **line** { (1,0), (2, 1), (3,0) } .

5.5.1.3. Funciones de utilidad para atributos discretos

Para los atributos de tipo booleano, enumerado y conjunto la mejor forma de especificar la utilidad es como una secuencia de coordenadas. La figura §5.11 muestra como especificar la utilidad sobre la atributo MEDIA que es de tipo conjunto. A continuación mostramos posibles funciones de utilidad para atributos de tipo enumerado y booleano.

La especificación QRL de estas funciones sería el siguiente:

```
utility {
  SFAIL {peso, {(initialState,50), (rollBack,100)}}
  STREAMING {peso, {(false ,0), (true ,100)}}
}
```

no obstante, también puede ser especificados de manera más legible con:

```
SFAIL {peso
 {
  case initialState:  50;
  case rollBack: 100;
 }
};
```

```
STREAMING {peso,
 {
  case false: 0;
  case true: 1;
 }
};
```

5.5.1.4. Funciones de utilidad sobre atributos derivados

QRL también permite que la utilidad se defina explícitamente sobre atributos derivados. En la figura §5.11 se especifica una función de utilidad para el atributo maturity. Como puede observarse, se asigna una utilidad diferente a cada uno de los estereotipos del atributo, pues si se considera que la madurez de un servicio con un TTF = 90 es la misma que si se tiene un TTF = 100 (en ambos casos se considera que es media (MEDIUM) entonces su utilidad también debe ser idéntica.

Una ventaja adicional de especificar la utilidad sobre un atributo derivado es reducir los inconvenientes de lo modelos de evaluación aditivo–lineales cuando el número de medidas a evaluar es muy elevado. Por ejemplo, si se establece la utilidad sobre el atributo recoverability se está definiendo implícitamente sobre otros dos atributos TTR y REBIND pero en la evaluación de la utilidad global sólo se considera la utilidad del atributo derivado de ambos.

5.5.2. Comprobación de validez

Para considerar que una sección de criterios de utilidad es válida debe verificarse:

1. Que sólo se hace referencia a medidas que están definidas.

2. Que sobre cada medida sólo existe un criterio de utilidad y un peso, pues de otro modo no sería posible saber qué criterio utilizar.

3. Que el dominio de la función de utilidad sobre un medida ya sea de un atributo simple o de un atributo derivado sea un subconjunto de dicho atributo.

Aquellas medidas para las que no se especifican criterios de utilidad se ignoran en el cálculo de la utilidad, es decir su utilidad y su peso es cero.

5.5.3. Comparación con otros métodos

El modelo de evaluación de la utilidad de un atributo derivado de QRL es diferente y entendemos que también más interesante que el de Dujmovic y Olsina (secciones §3.2.5 y §3.2.6). Dujmovic y Olsina permiten modelar muchas relaciones entre atributos que no se pueden modelar con el aditivo, concretamente 17 maneras diferentes desde la conjunción (la utilidad global viene dada por la utilidad mínima) hasta la disyunción (la utilidad global es la utilidad máxima de las utilidades agregadas). No obstante, con el modelo de QRL la aplicación lineal aditiva sólo se utiliza para calcular la utilidad global, pues la utilidad de cada tributo derivado que se desee utilizar en el cómputo de la utilidad de una restricción se asigna explícitamente por lo que se puede definir con entera libertad y tener en cuenta todas las relaciones de los modelos de Dujmovic y Olsina.

La utilidad del atributo derivado siempre se encuentra en el mínimo y el máximo de las utilidades de entrada, en el caso de QRL no, se puede asignar la utilidad sobre los estereotipos de un determinado atributo derivado sin limitación alguna, salvo que deben ser valores entre 0 y 1.

5.6. Documentos con cláusulas de negociación

QRL esta diseñado para dar soporte a dos modelos de negociación diferentes: el modelo de negociación por compromisos y el modelo de negociación Win–Win, ambos discutidos en el capítulo §3. La figura §5.13 muestra un documento de condiciones sobre el que discutiremos cómo se especifican cláusulas de negociación para ambos modelos.

Condiciones de uso para utilizar el servicio que implementa la interfaz IVideoServer

1-5 idénticas a las de la figura §5.11.

6. Cláusulas de negociación

6.1. Compromisos.

6.1.1. Más fiabilidad–Más dinero. Es posible pagar hasta 3€ (4.1.1) a cambio de tener un tiempo entre fallos muy alto en hora punta (4.2.1).

6.1.2. Menos fiabilidad–Menos dinero. Si sólo es posible tener un tiempo entre fallos normal (4.2.1) en hora PUNTA, el precio deberá reducirse en 0.3€ (4.1.1).

6.2. Renuncias.

6.2.1. Hora punta. Conformarme con un tiempo entre fallos normal (4.2.1) en hora PUNTA con el mismo precio que en las condiciones base.

a) Documento de condiciones en lenguaje natural

Las primera secciones del documento son idénticas a las de la figura §5.11

```
negotiation {
 tradeoffs {
  MasFiabilidad_MasDinero {
   lose { r1: COST ≤ 3;}
   win {r3: maturity.VERY_HIGH on PUNTA}
  }
  MenosFiabilidad_MenosDinero {
   lose {r3: maturity.MEDIUM on PUNTA}
   win {r1: COST ≤ 1.7;}
  }
 }
 renunciations {
  HoraPunta {r3: maturity.MEDIUM on PUNTA}
 }
}
```

b) Documento de condiciones en QRL equivalente al de arriba

Figura 5.13: *Documento de condiciones con cláusulas de negociación.*

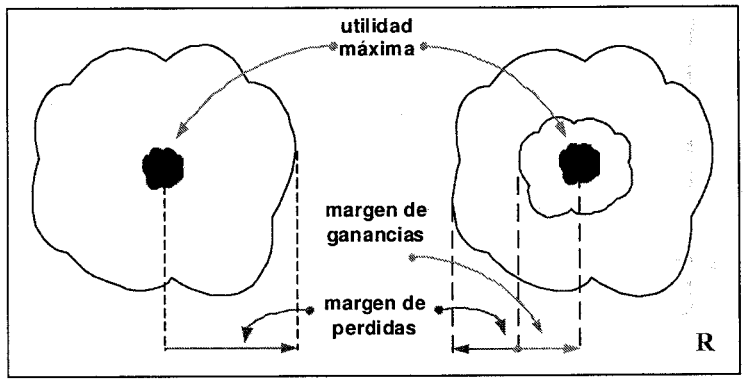

Figura 5.14: *Comparativa entre el modelo de negociación Win–Win y la negociación por compromisos.*

5.6.1. Soporte para negociación tipo Win–Win

La parte izquierda de la figura §5.14 sintetiza el efecto de la negociación en el modelo Win–Win desde el punto de vista de uno de los participantes. Partiendo de un documento de requisitos que ofrece la máxima utilidad (representado por la restricción oscura) la negociación consiste en ir relajando los requisitos, lo que matemáticamente se traduce en una perdida de la utilidad de la restricción asociada. En otras palabras, en el modelo Win–Win el margen de negociación coincide con las pérdidas que está dispuesto a aceptar el participante. De este modo, si quisiéramos utilizar agentes electrónicos para llevar a cabo una negociación siguiendo el modelo Win–Win, únicamente requeriría información sobre hasta dónde esta dispuesto a ceder.

Para expresar esta información proponemos el uso de *cláusulas de renuncias*. Una cláusula de renuncia indica hasta dónde esta dispuesto a ceder un participante para conseguir alcanzar un acuerdo. La cláusula 6.2.1 de la figura §5.13 es un ejemplo de cláusula de renuncia, pues indica que el participante está dispuesto a ceder en su intención de solicitar un mayor tiempo entre fallos en hora punta sin pedir nada a cambio.

En cuanto a comprobar la validez de una cláusula de renuncia la operativa es muy sencilla, basta con comprobar que la utilidad del requisito base es superior a la del requisito indicado en la cláusula de negociación.

En este caso, es fácil de realizar esta comprobación pues la utilidad del requisito base (r3: maturity.HIGH **on** PUNTA, ver figura §5.13) es 0.81 (pues el peso es 0.9 y la utilidad 0.9) mientras que la utilidad del requisito de la cláusula es 0.45 (pues el peso es 0.9 y la utilidad 0.5).

5.6.2. Soporte para negociación por compromisos

La parte derecha de la figura §5.14 sintetiza el efecto de la negociación por compromisos desde el punto de vista de uno de los participantes. Se parte de un documento de requisitos que ofrece un valor de utilidad está entre el máximo (el asociado a la restricción oscura) y el mínimo posible (representada por la restricción más externa). Desde esta situación intermedia la negociación consiste en ir cediendo algo de utilidad en unos requisitos e ir ganando en otros. En otras palabras, en la negociación por compromisos el margen de negociación comprende tanto a las pérdidas que está dispuesto a aceptar el participante como las ganancias que tiene que recibir a cambio.

Para expresar esta información proponemos el uso de *cláusulas de compromisos*. Una cláusula de compromisos indica en qué requisitos esta dispuesto a ceder un participante para conseguir alcanzar un acuerdo, lo que se conoce como cláusula de pérdidas (**lose**) y en que requisitos espera conseguir una mayor utilidad, lo que se conoce como cláusula de ganancias (**win**). Las cláusulas 6.1.1 y 6.1.2 de la figura §5.13 son dos ejemplos de cláusulas de compromisos que están referidos sobre los mismos requisitos. El primero indica que se está dispuesto a pagar más por el servicio si este es más fiable en hora punta, y el segundo, que si no es posible conseguir una alta fiabilidad en hora punta entonces no se está dispuesto a pagar tanto por el servicio.

En cuanto a comprobar la validez de un compromiso la operativa es muy sencilla, basta con comprobar que la utilidad del requisito base es superior a la del requisito indicado en la cláusula de pérdidas e inferior a la de la cláusula de ganancias. En ambos casos, es fácil comprobar que ambas son válidas. En el caso de la cláusula 6.1.1. la utilidad del requisito base (r1: COST $\leq$ 2) es 0.435 (pues el peso es 0.7 y la utilidad 0.65) mientras que la utilidad de la cláusula de pérdidas es 0.245 (pues el peso es 0.7 y la utilidad 0.35). Por su parte, la utilidad de la cláusula de ganancias es 0.9 (pues el peso es 0.9 y la utilidad es 1) mientras que la utilidad del requisito base es 0.81 (pues el peso es 0.9 y la utilidad es 0.9). Con un razonamiento

análogo se puede comprobar la validez de la cláusula 6.1.2 de la misma figura.

Es fácil darse cuenta de que no es posible especificar cláusulas de negociación sino se dispone de una sección de criterios de utilidad. De hecho, cualquier otro intento para validar las cláusulas de negociación a partir de la noción de conformidad y de satisfactibilidad no es suficiente para capturar su semántica (ver anexo §A).

Por último conviene resaltar que las cláusulas de negociación no son útiles únicamente a los clientes también lo son para los proveedores. Por ejemplo, un proveedor puede ofrecer un producto con la misma funcionalidad pero con tres diferentes niveles de calidad, cada uno de los cuales tendrá un precio diferente.

5.6.3. Negociando sobre el período de vigencia

Hasta ahora, la negociación entre requisitos sólo ha tenido en cuenta los atributos de calidad. Sin embargo, la negociación puede y debe tener presente el período de vigencia asociados a cada requisito, pues resulta necesaria en muchas situaciones.

Por ejemplo, según el criterio de validez de una cláusula de renuncia, la cláusula de renuncia r1: COST $\leq$ 2 **on** PUNTA no sería válida respecto al requisito base r1: COST $\leq$ 2 pues ambos requisitos poseen idéntica utilidad. Sin embargo, el período de vigencia de la cláusula de negociación es más reducido que el de la condición base, lo que puede interpretarse como una renuncia del cliente ya que está pagando lo mismo por utilizar el mismo servicio durante menos tiempo.

Primer intento para calcular la utilidad de un requisito con consciencia temporal

Para poder comprobar automáticamente la validez de una cláusula de negociación, ya sea de renuncia o de compromiso, es necesario extender la actual noción de utilidad de un requisito para que tenga en cuenta el período de vigencia. Nosotros proponemos que la utilidad de un requisito se obtenga como el promedio de la utilidad de su restricción y de su período de vigencia, esto es:

$$\mathcal{U}_r = w_c \mathcal{U}_c + w_p \mathcal{U}_p$$

donde w_c y w_p son los pesos de la restricción y del período de vigencia asociados al requisito r y U_c y U_p la utilidad de la restricción y del período de vigencia asociados al requisito r.

Como utilidad de un período de vigencia proponemos una normalización respecto al período de vigencia global de un documento. Es decir, definimos la utilidad del período de vigencia de un requisito r como el cociente P/G, donde P hace referencia a la duración de un período de vigencia, y G a la duración del período de vigencia global del documento al que pertenece r.

De este modo, ahora se puede comprobar que la cláusula de renuncia r1: COST $\leq$ 2 **on** PUNTA es válida, pues suponiendo la misma importancia al período de vigencia que a la restricción, resulta que la utilidad de r1: COST $\leq$ 2 **on** PUNTA es de 0.325 $+0{,}5P/G$ (0.5*0.65 + 0.5 * P/G) que es menor que la utilidad de r1: COST $\leq$ 2 que es de 0.825 (0.5*0.65 + 0.5 * P/G, con $P = G$).

Utilidad de un requisito con consciencia temporal

Esta noción de utilidad de un requisito no es válida cuando interesa que los requisitos tengan el período de vigencia tan reducido como sea posible. La cláusula del ejemplo utilizado anteriormente podría no ser válida si formara parte de un documento de oferta y no de uno de condiciones, pues en tal caso es más que probable que para el proveedor sale ganando al recibir el mismo dinero por ofrecer un servicio durante menos tiempo. Corregir nuestra noción de utilidad para tener en cuenta estas situaciones es muy sencillo, basta con definir la utilidad de un período de vigencia como

$$\mathcal{U}_p = \begin{cases} P/G & \text{si son más útiles los períodos mayores} \\ 1 - P/G & \text{si son más útiles los períodos pequeños} \end{cases}$$

Comprobar la validez de una cláusula de renuncia cuando se negocia sobre períodos de vigencia sigue la misma operativa que vimos en la sección anterior. No obstante, en el caso de las cláusulas de intercambio hay que ser más cuidadosos con aquellos requisitos sobre los que no existe interés en negociar con los períodos de vigencia. Por ejemplo, supongamos que un cliente solicita el requisito r1: COST $\leq$ 2 **from** 22..24 y que no tiene ningún interés por ver películas fuera de ese horario. Es evidente que el requisito r1: COST $\leq$ 2 **from** 16..24 no mejora las preferencias del cliente, pues sólo tienen interés el período que va desde las 22 hasta las 24. En tales casos, para poder comprobar la validez de una cláusula es necesario comparar la utilidad de la restricción y del período de vigencia por separado.

En QRL, esta definición de la utilidad de un período de vigencia se

incluye en la sección de criterios de utilidad, especificando separadamente aquellos requisitos para los que interesa un período de vigencia tan amplio como como sea posible (**increasing**) y aquellos para los que interesa un período de vigencia tan reducido como sea posible (**decreasing**).

```
utility {
  ....
  valid increasing { r1: peso_r1, r3: peso_r3 }
}
```

5.7. Documentos de calidad interoperables

El grado de interoperabilidad ofrecido por QRL es el más básico que puede ofrecer un QSL: la interoperabilidad por correspondencia (ver sección §4.7). Se trata de definir correspondencias entre aquellos atributos con idéntico significado o entre los que es posible definir una relación de *equivalencia semántica*. De este modo, el primer paso para comprobar la conformidad entre condiciones y ofertas que utilizan diferentes catálogos será reescribir las condiciones y la oferta de acuerdo con la equivalencia semántica definida entre sus catálogos.

Definir la relación de equivalencia entre los catálogos utilizados por un documento de condiciones y un documento de ofertas es muy sencillo y puede realizarse de manera sistemática. Para cada atributo sobre el que se define algún requisito en el documento de condiciones (id_c) se busca un atributo entre los utilizados por la oferta (id_o) cuyo significado pueda considerarse equivalente bien directamente ($id_o = id_c$) o través de una expresión que ligue a ambos atributos ($id_o = \theta(id_c)$). Por ejemplo, para las condiciones y la oferta de la figura §5.16 que hacen uso de los catálogos especificados en la figura §5.15, es fácil encontrar atributos equivalentes de TDR y TEF.

La validez de la relación $TTR = 60 \times TDR$ es fácil de ver, pues el significado de TDR es el mismo que el de TTR y lo único que les diferencia además del identificador utilizado es la unidad de medida que es el minuto en el caso del TTR y la hora en el caso del TDR.

El caso de TTF= 365 / TEF es un poco más complicado pues no existe ningún atributo en el catálogo com.acme.stdMOWS que tenga el mismo significado que TEF (número medio de días que transcurren entre dos fallos

```
catalogue tdg.stdMOWS {
 attributes {
  TEF
   description:"Tiempo entre fallos";
   domain: int día;
  TDR
   description: "Tiempo de recuperación ante un error";
   domain: int hora;
  RET
   description: "Tiempo de respuesta";
   domain: real [0..60*60] segundo;
  DIS
   description: "Disponibilidad";
   domain: real [0..100] %;
  CTE
   description: "Comportamiento tras un error";
   domain: enum {estadoInicial, deshacer};
 }
}

catalogue com.acme.stdMOWS {
 attributes {
  TTF {
   description:"Time to fail";
   domain: int;
  TTR {
   description: "Time to repair";
   domain: int min;
  DELAY {
   description: "Time to return";
   domain: real [0..60*60] sec;
  SFAIL {
   description: "Semantic failure";
   domain: enum {halt, initialState, rollBack};
 }
}
```

Figura 5.15: *Especificación de dos catálogos con una semántica cuasiequivalente.*

```
using tdg.stdMOWS
conditions for P {
  r1: TDR ≤ 10; // horas
  r2: TEF ≥ 90; // dias
  r3: RET.Percentile(90) ≤ 100; // seg
  r4: DIS ≥ 99.7; //%
  r5: CTE ≥ estadoInicial;
}

using com.acme.stdMOWS
offer for P {
  r1: TTR ≤ 480; // minutos
  r2: TTF < 5; // fallos al año
  r3: DELAY.Percentile(95) ≤ 90; //seg
  r4: SFAIL = rollBack;
}

equivalence between tdg.stdMOWS and com.acme.stdMOWS {
  TTR= 60 × TDR;
  TTF= 365 / TEF;
  DIS= 100 - TDR / (24×TEF);
  SFAIL= CTE { (initialState, estadoInicial), (rollBack, deshacer)};
  DELAY= RET;
}
```

Figura 5.16: *Ejemplo de especificación de reglas semánticas.*

consecutivos). No obstante, si es posible relacionarlo con el atributo TTF (número medio de fallos al año) si asumimos que la distribución de fallos es uniforme.

Buscar una equivalencia para la disponibilidad (DIS) es bastante más difícil que los casos anteriores, pues no existe ningún atributo en el catálogo com.acme.stdMOWS con una semántica parecida o relacionada. En general, en estas circunstacias, debe intentarse encontrar una relación con dos o más atributos. En el caso de nuestro ejemplo, es posible relacionando el tiempo entre fallos y el tiempo de recuperación ante fallos, pues el cociente TTR/TTF ofrece una idea de la probabilidad de encontrar el sistema fuera de servicio, y por tanto 100 - TTR/TTF una idea de la probabilidad de encontrar el sistema disponible. De este modo, sería posible reescribir la restricción DIS ≥ 99.7 omitiendo DIS y usando en su lugar TEF y TDR, los atributos equivalentes a TTR y TTF quedando 100 - TDR / TEF ≥ 99.7. Fíjese que en este caso no es un requisito de la oferta el que se reescribe, sino uno de las condiciones tal y como se muestra en la figura §5.17.

Fíjese que la anterior relación DIS = 100 - TDR / TEF no sería del todo válida pues dado que el TEF se expresa en días y el TDR en horas, es necesario utilizar factores de conversión entre unidades. Por ejemplo con 100

```
using tdg.stdMOWS
conditions for P {
  r1: TDR ≤ 10;
  r2: TEF ≥ 90;
  r3: RET.Percentile(90) ≤ 100;
  r4: 100 - TDR / (24×TEF) ≥ 99.7;

  r5: CTE ≥ estadoInicial;
}
```

```
using com.acme.stdMOWS
offer for P {
  r1: 60 × TDR ≤ 480 // ≡ TDR ≤ 8;
  r2: (365 / TEF) < 5 // ≡ TEF > 73;
  r3: RET.Percentile(90) ≤ 90;
  r4: CTE = deshacer;
}
```

Figura 5.17: *Resultado de aplicar las reglas semánticas.*

- TDR / (24×TEF) ≥ 99.7. Evidentemente este tipo de errores resulta muy difícil de detectar pues en definitiva estamos tratando con expresiones definidas por el usuario y por tanto con una semántica subjetiva. Es por ello, que en QRL sólo se comprueba el tipo de la expresión.

Sobre la relación de equivalencia (DIS= 100 - TDR / (24×TEF)) conviene destacar dos aspectos importantes. En primer lugar, obsérvese que relacionar una medida del documento de condiciones con una o varias medidas de la oferta permite determinar la conformidad en situaciones de *emparejamiento parcial*. En segundo lugar, obsérvese que la validez de una regla de equivalencia queda fijada por el autor de las condiciones, por lo que si el autor interpreta incorrectamente el significado de un atributo utilizado por la oferta, podría considerar que se ha violado el SLA cuando desde el punto de vista del proveedor y de su oferta original no ha existido tal violación. Para poder determinar la responsabilidad o no del autor es fundamental que en los casos en los que ha sido necesario transformar la oferta original, ésta se adjunte al SLA como parte integrante del mismo.

La relación de equivalencia entre atributos de tipo enumerado y conjunto es un poco más complicada pues aunque coincida la semántica de los atributos es posible que no tengan el mismo conjunto de valores (ver figura §5.16), por lo que en general en atributos enumerados y de tipo conjunto las relaciones serán parciales.

Por último, una característica muy interesante de QRL, es su capacidad para inferir automáticamente relaciones de equivalencia en aquellas situaciones de emparejamiento parcial en la que intervienen medidas estadísticas y donde la equivalencia entre atributos no es suficiente. Por ejemplo,

la validez de la relación DELAY=RET es fácil de ver pues el significado de ambos atributos es idéntico. No obstante, esta sustitución no garantiza que se pueda determinar la conformidad pues para el requisito r3: RET.Percentile(90) $\leq$ 100 de las condiciones no existe ningún requisito en la oferta con el que se pueda comparar. Este problema también se da cuando las condiciones y las ofertas utilizan los mismos catálogos y se conoce como el del *emparejamiento parcial* (sección §7.2.2.2).

5.8. Aplicando QRL a otros contextos

La capacidad de abstracción de QRL lo convierten en un lenguaje que puede ser utilizado en varias etapas del desarrollo de un proyecto *software*

5.8.1. Formalizando la información no funcional de un caso de uso

QRL puede ser utilizado para especificar formalmente la información no funcional asociada a un caso de uso haciendo uso de requisitos de calidad. Esta posibilidad resulta muy útil en las métodos de diseño arquitectónico guiados por casos de uso [Peña *et al.* 2000b, Ruiz-Cortés *et al.* 2000a] pues garantiza la consistencia de la información no funcional añadida a cada caso de uso. La figura §5.18 presenta como formalizar en QRL los requisitos de calidad que están asociados a un caso de uso.

La consciencia temporal puede ser muy útil para completar la especificación de requisitos de almacenamiento [Durán 2000]. Por ejemplo, se pueden tener patrones lingüísticos tales como:

- La disponibilidad de la información será del «99.9» % de «lunes» a «viernes» de «08:00» a «21:00» y del «90» en las restantes ocasiones
- El acceso estará disponible únicamente a «lista de actores»
- El acceso a «lista de campos» estará disponible únicamente a «lista de actores»

RF-10. **Consulta de una película**
Descripción. El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando el cliente del vídeo-bank solicite la consulta de una película.
Secuencia.
1. El cliente del vídeo-bank solicita al sistema comenzar el proceso de consulta.
2. El sistema solicita que se identifique la película a consultar.
3. El cliente del vídeo-bank identifica la película a consultar.
4. El sistema muestra la información publicitaria de la película.

Restricciones globales.
1. La funcionalidad debe ofrecerse durante las 24 horas de lunes a domingo con un tiempo medio entre errores superior a 48 horas y un tiempo de recuperación inferior a 5 minutos.
2. Desde 16:00 hasta las 24:00 debe tener una disponibilidad de al menos el 98 %.
3. Desde 00:00 hasta las 16:00 debe tener una disponibilidad de al menos el 90 %.
4. El sistema deberá soportar la ejecución simultánea de 1000 instancias de este caso de uso.

Restricciones individuales.
Paso 1. Debe realizarse en menos de 1 segundo.
Paso 4. Debe realizarse en menos de 3 segundo.

a) Descripción de un caso de uso.

```
products { RF-10: Consulta_de_una_pelicula= Paso[4] }
conditions for RF-10 {
 global {
  r0: TTF ≥ 48*60 and TTR ≤ 5*60;
  r1: AVAIL ≥ 98 from 16..24 and AVAIL ≥ 90 from 0..16;
  r2: CAPACITY = 1000;
 }
 for Paso[1] { DELAY < 1; }
 for Paso[4] { DELAY < 3; }
}
```

b) Descripción en QRL de las restricciones del caso de uso.

Figura 5.18: *Especificando restricciones de calidad sobre un caso de uso.*

5.8.2. Extendiendo los actuales lenguajes de descripción de arquitecturas

Una de las limitaciones más importantes de los actuales lenguajes de descripción de arquitecturas (ADLs, *Architectural Description Languages*) es la ausencia de modelos formales para expresar propiedades no funcionales [Medvidovic y Taylor 2000]. Entendemos que las características que posee QRL le hacen ser una propuesta prometedora. En [Martín *et al.* 2001b] hemos presentado una primera aproximación para evaluar automáticamente de alternativas de diseño. La idea básica es considerar que las condiciones son el documento de requisitos que debe satisfacer el diseño y que los requisitos que satisface cada alternativa son representados por una oferta. De este modo, el problema de seleccionar la mejor alternativa es muy similar puede interpretarse como un problema de minimización

Capítulo 6

Descripción rigurosa de QRL

Describir con precisión el significado de algo es condenarlo a la vulgaridad: todo el mundo acabará entendiéndolo y perderá su magia.

JOSÉ PALACIOS

Uno de los requisitos que cualquier lenguaje debería satisfacer es contar con una definición formal que recoja de forma precisa y sin ambigüedades el significado de cada una de sus características. Esto exige describir dos aspectos diferentes del mismo: por una parte la estructura de sus programas, es decir, los elementos que los componen y las relaciones que tiene que haber entre ellos para que se puedan considerar correctos; pero además hay que definir un modelo de ejecución que nos permita interpretar el significado de esos elementos cuando el programa está en ejecución. A estos dos aspectos diferentes es a lo que generalmente se suele hacer referencia como la semántica estática y dinámica, respectivamente.

El mayor problema a la hora de describir la semántica estática suele ser encontrar una técnica que sea formal a la vez que sencilla y clara. En este punto solemos

encontrarnos con dos posibilidades diferentes: utilizar algún método orientado a la obtención de un compilador del lenguaje [Lee 1989, Meyer 1990, Koskimies y Paakki 1990] o bien algún método basado en conceptos matemáticos elementales como conjuntos, tuplas, relaciones binarias, etcétera [Plotkin 1981]. El problema asociado a los primeros es que si bien proporcionan una notación clara y bien definida, en general dan lugar a descripciones que son en exceso complejas y difíciles de entender puesto que no debemos perder de vista que su objetivo es derivar un compilador a partir de la especificación. Por esta razón, la descripción formal de QRL *que presentamos en este capítulo está basada en el uso de técnicas matemáticas simples que combinan una alta capacidad expresiva y de abstracción con la rigurosidad.*

Una característica clave en la creación de los documentos de calidad es la dualidad. Tres de las cuatro actividades en las que hemos estructurado el proceso de creación de documentos de calidad, concretamente la elicitación, la validación y la verificación implican la intervención de personas con una formación no necesariamente técnica, por lo que resulta prioritario sino obligatorio disponer de una notación lo más cercana posible al lenguaje natural. Por otra parte, a fin de que el análisis de consistencia se pueda realizar automáticamente, es necesario disponer de una notación formal, la cual, por mucho que nos empeñemos siempre será más difícil de interpretar.

Una de las cuestiones que se plantean al diseñar una notación dual, son las facilidades sintácticas que debe ofrecer la notación formal. Si la notación formal sólo va a ser manejada por el entorno de desarrollo y de ejecución no es necesario añadir facilidades sintácticas. Sin embargo, si se prevé que la notación va a ser utilizada por humanos, las facilidades sintácticas son de agradecer. Dado que por ahora no disponemos de una herramienta CASE para editar documentos de calidad hemos decidido añadir un conjunto de facilidades sintácticas siendo conscientes que esta decisión complica la descripción rigurosa de la semántica.

Con objeto de simplificar la descripción formal de QRL *plantearemos su definición en dos niveles. En un primer nivel definimos la semántica axiomática de los elementos del núcleo del lenguaje (aquéllos que no pueden expresarse en función de otros) y en segundo nivel definimos la semántica de* QRL *a partir de las transformaciones necesarias para normalizar* QRL *sobre su núcleo. En adelante nos referiremos a este núcleo como* $\text{QRL}_{\heartsuit}$ *(leído como "qrlcore").*

6.1. Preliminares

6.1.1. Mapas

Definición 6.1.1 (Mapa) *Un mapa es una función σ que asocia a cada elemento de un conjunto denominado dominio otro elemento de un conjunto llamado imagen. Denotaremos el primero como* $\operatorname{dom}\sigma = \{x_1, x_2, \ldots, x_n\}$ $(n \geq 0)$*, el segundo como* $\operatorname{img}\sigma = \{y_1, y_2, \ldots, y_m\}$ $(m \geq 0)$ *y los representaremos haciendo uso de la notación* $\{x_1 \mapsto \sigma(x_1), x_2 \mapsto \sigma(x_2), \ldots, x_n \mapsto \sigma(x_n)\}$.

Ejemplo 6.1.1 $\{\}, \{a \mapsto 1, b \mapsto 2\}, \{1 \mapsto a, 2 \mapsto b\}$.

Definición 6.1.2 (Indefinición) *Dado un mapa σ y un elemento $x \notin \operatorname{dom}\sigma$, entonces consideraremos que $\sigma(x) = \perp$, en donde $\perp$ es un objeto matemático al que se suele hacer referencia como* bottom *en la bibliografía inglesa. Se utiliza generalmente para representar fallos como pueden ser indefinición de una función, insatisfactibilidad de una restricción, etcétera.*

Ejemplo 6.1.2 *Dado el mapa $\sigma = \{a \mapsto 1, b \mapsto 2\}$, entonces se verifica que $\sigma(a) = 1, \sigma(b) = 2, \sigma(c) = \perp$.*

Definición 6.1.3 (Composición) *Dados dos mapas σ_1 y σ_2 definimos su composición como* $\sigma_1 \circ \sigma_2 = \{x \mapsto \sigma_1(x) \mid x \in \operatorname{dom}\sigma_1 \wedge x \notin \operatorname{dom}\sigma_2\} \cup \{x \mapsto \sigma_2(x) \mid x \in \operatorname{dom}\sigma_2\}$.

Ejemplo 6.1.3 *Si $\sigma_1 = \{a \mapsto 1, b \mapsto 2\}$ y $\sigma_2 = \{a \mapsto 3, c \mapsto 1\}$, entonces se verifica que $\sigma_1 \circ \sigma_2 = \{a \mapsto 3, b \mapsto 2, c \mapsto 1\}$.*

6.1.2. Conjuntos

Definición 6.1.4 (Conjunto potencia) *Dado un conjunto A se define el conjunto potencia de A como la colección de todos los subconjuntos posibles de A. Lo denotaremos como como* $2^A = \{B | B \subseteq A\}$.

6.1.3. Secuencias

Definición 6.1.5 (Secuencia) *Una secuencia S es un mapa de la forma* $\{1 \mapsto x_1, 2 \mapsto x_2, \ldots, n \mapsto x_n\}$ $(n \geq 0)$ *que denotaremos como* $\langle x_1, x_2, \ldots, x_n \rangle$.

Ejemplo 6.1.4 $\langle\rangle, \langle 1, 2\rangle, \langle a, b, c\rangle$.

Definición 6.1.6 (Indización) *Dado que una secuencia es un mapa con dominio sobre una porción de los números naturales, la indización se denota como* $\sigma(i)$ $(i = 1, 2, \ldots)$.

Ejemplo 6.1.5 *Si* $S = \langle 1, 2, 3, 4\rangle$, *entonces se verifica que* $S(1) = 1, S(4) = 4, S(5) = \perp, S(a) = \perp$.

Definición 6.1.7 (Longitud) *Dada la secuencia* $S = \langle x_1, x_2, \ldots, x_n\rangle$ $(n \geq 0)$, *definimos su longitud como* $|S| = n$.

Ejemplo 6.1.6 $|\langle\rangle| = 0, \quad |\langle 1, 2\rangle| = 2, \quad |(\langle\langle 1, 2\rangle, \langle\rangle\rangle)(1)| = 2.$

Definición 6.1.8 (Concatenación) *Dadas dos secuencias* $S_1 = \langle x_1, x_2, \ldots, x_n\rangle$ $(n \geq 0)$ *y* $S_2 = \langle y_1, y_2, \ldots, y_m\rangle$ $(m \geq 0)$ *definimos su concatenación como* $S_1 \frown S_2 = \langle x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_m\rangle$.

Ejemplo 6.1.7 $\langle\rangle \frown \langle\rangle = \langle\rangle, \langle\rangle \frown \langle 1\rangle = \langle 1\rangle, \langle 1\rangle \frown \langle 2\rangle \frown \langle 3, 4\rangle = \langle 1, 2, 3, 4\rangle$.

Definición 6.1.9 (Subsecuencia) *Dada la secuencia* $S = \langle x_1, x_2, \ldots, x_n\rangle$ $(n \geq 0)$, *denotaremos la subsecuencia entre los elementos* i *y* j *con* $0 \leq i \leq j \leq n$ *como* $S(i..j)$ *que se define como* $S(i..j) = S(i) \frown S(i+1) \frown \ldots \frown S(j)$

Ejemplo 6.1.8 *Si* $S = \langle 1, 2, 3, 4, 5\rangle$, *entonces se verifica que* $S(2..4) = \langle 2, 3, 4\rangle$, $S(4..4) = \langle 4\rangle, S(4..6) = \perp$.

```
catalogue {c1:com.acme.stdMOWS; c2:com.acme.multimedia}
measures {
  c1.COST: real [0..∞];
  c1.AVAIL:  real [0..100];
  c1.play.TTR: int [0..∞];
  c1.play.TTF: int [0..∞];
  c1.play.SFAIL: enum {halt, initialState, rollBack};
  c1.play.REBIND: bool;
  c2.LANG: set { EN, FR, DE, SP};
  c2.MEDIA: set {ADSL, ISDN, Modem56k};

}
requirements {
  idr1: { (c1.COST ≥ 2, τ ≥ 0 and τ ≤ 1439),
          (c1.AVAIL ≥ 90, τ ≥ 0 and τ ≤ 1439),
          (c1.play.TTR ≤ 60, τ ≥ 0 and τ ≤ 1439)
        }
}
```

Figura 6.1: *Un documento de calidad especificado en* $\text{QRL}_{\heartsuit}$.

6.2. Semántica estática de $\text{QRL}_{\heartsuit}$

6.2.1. Estructura de un documento de condiciones

La figura §6.1 muestra un documento básico de calidad especificado en $\text{QRL}_{\heartsuit}$. Como puede apreciarse, un documento en $\text{QRL}_{\heartsuit}$ tiene cuatro secciones, a saber: sección de catálogos, sección de medidas, sección de requisitos y sección de preferencias.

6.2.2. Identificadores

Definición 6.2.1 (Identificadores) *Denotaremos el conjunto de identificadores disponibles como* $\mathcal{ID}$. *Asumiremos que estos identificadores son universales, de modo que cada uno denota un único elemento. Esta suposición no es difícil llevar a la práctica si cada uno tiene como prefijo el URI dónde reside su descripción.*

Asumiremos que $\mathcal{ID}$ *puede ser particionado en* $\mathcal{ID}_{\mathcal{T}}$, $\mathcal{ID}_{\mathcal{E}}$, $\mathcal{ID}_{\mathcal{M}}$, $\mathcal{ID}_{\mathcal{VP}}$,

$\mathcal{ID}_{\mathcal{R}}$, $\mathcal{ID}_{\mathcal{N}}$ que denotan respectivamente al conjunto de identificadores para los tipos de las medidas y de las expresiones, los valores enumerados, los períodos de vigencia, los requisitos y las cláusulas de negociación.

Esta definición de identificador es válida tanto para QRL$_{\heartsuit}$ como para QRL. No obstante, dado que QRL$_{\heartsuit}$ es un lenguaje de muy bajo nivel para conseguir la universalidad en los identificadores de las medidas y de los valores enumerados será necesario prefijarlos además de con el URI con los identificadores de productos y subproductos.

Ejemplo 6.2.1 *Algunos identificadores válidos en* QRL *son: TTF, TTR, p1, play y stop.*

Algunos identificadores válidos en QRL$_{\heartsuit}$ *son: c1.TTF, c1.TTR, c1.p1.TTF, c1.p1.play.TTF y c1.p1.stop.TTF.*

6.2.3. Sección de medidas

A partir de un catálogo de atributos y de la definición de un producto es posible obtener un conjunto de variables para representar a las medidas de los atributos de un determinado producto.

Cada una de estas variables, a las que en adelante nos referiremos como *medidas de calidad* o simplemente *medidas* pueden tomar valores en dominios numéricos, enumerados, de tipo conjunto y booleano. El conjunto de valores de una medida determina su tipo.

Definición 6.2.2 (Tipos de las medidas) *Definimos el tipo de una medida como un elemento del conjunto $\mathcal{T}$ que se define como*

$$
\begin{array}{rcll}
\mathcal{T} & \triangleq & \textbf{\textit{real}}[x_l..x_h] & \\
 & | & \textbf{\textit{int}}[n_l..n_h] & x_l, x_h \in \mathcal{L}it_{\textbf{\textit{real}}} \cup \{-\infty, \infty\} \\
 & | & \textbf{\textit{enum}}\{eid_1, \ldots, eid_n\} & n_l, n_h \in \mathcal{L}it_{\textbf{\textit{int}}} \cup \{-\infty, \infty\} \\
 & | & \textbf{\textit{set}}\{eid_1, \ldots, eid_n\} & eid_1, \ldots, eid_n \in \mathcal{ID}_{\mathcal{E}} \\
 & | & \textbf{\textit{bool}} & n \geq 1 \\
 & | & \bot &
\end{array}
$$

Dado un tipo $t \in \mathcal{T}$ diremos que t es válido si y sólo si se satisface el predicado $valid : \mathcal{T}$ definido como

$$
\begin{array}{l}
valid(\textbf{real}[x_l..x_h]) \iff x_l \leq x_h \\
valid(\textbf{int}[n_l..n_h]) \iff n_l \leq n_h \\
valid(\textbf{enum}\{eid_1, \ldots, eid_n\}) \iff \forall i, j \in \{1, 2, \ldots, n\} \cdot (i \neq j \Rightarrow eid_i \neq eid_j) \\
valid(\textbf{set}\{eid_1, \ldots, eid_n\}) \iff \forall i, j \in \{1, 2, \ldots, n\} \cdot (i \neq j \Rightarrow eid_i \neq eid_j) \\
valid(\textbf{bool}) \iff true \\
valid(\bot) \iff true
\end{array}
$$

Definición 6.2.3 (Sección de medidas) *Definimos una sección de medidas como un elemento del conjunto $\mathcal{M}$ que se define como $\mathcal{M} \triangleq \mathcal{ID}_{\mathcal{M}} \mapsto \mathcal{T}$*

Ejemplo 6.2.2 *El siguiente fragmento de especificación se corresponde con una sección de medidas*

```
measures {
  τ: int [0..∞];
  c1.COST: real [0..∞];
  c1.SFAIL: enum {c1.SFAIL.halt, c1.SFAIL.initialState, c1.SFAIL.rollBack};
  c1.play.TTR: int [0..∞];
  c1.play.REBIND: bool;
  c2.LANG: set { c2.LANG.EN, c2.LANG.FR, c2.LANG.SP};
  c2.SUBT: set { c2.LANG.EN, c2.LANG.FR, c2.LANG.SP};
}
```

La necesidad de prefijar los identificadores de medidas de tipo enumerado y conjunto lo veremos en la siguiente sección cuando discutamos el procedimiento para obtener el tipo de una expresión.

Dada una sección de medidas $M \in \mathcal{M}$, diremos que M es válida si y sólo si lo es cada uno de los tipos de las medidas que contiene y si incluye la medida que representa al tiempo. A esta medida nos referiremos en adelante como τ. Estas condiciones son recogidas por el predicado $valid : \mathcal{M}$ que se define como

$$valid(M) \iff \forall t \in \text{img}(M) \cdot valid(t) \wedge (\tau \mapsto \textbf{int}[0, \infty]) \in M$$

Como puede observarse, el tiempo se medirá en unidades enteras que representa a los diferentes "quantum" en los que se suele medir el tiempo.

6.2.4. Sección de requisitos

6.2.4.1. Restricciones

En $\text{QRL}_{\heartsuit}$ un requisito es representado por un conjunto de pares constituidos por una restricción de calidad y por una restricción temporal. Una restricción de calidad permite relacionar expresiones sobre medidas de calidad.

Definición 6.2.4 (Expresión) *Definimos una expresión como un elemento del conjunto $\mathcal{E}$ que se define como*

$$
\begin{array}{rcll}
\mathcal{E} & \triangleq & eid & eid_1, \ldots, eid_n, eid \in \mathcal{ID}_{\mathcal{E}} \\
 & | & \{eid_1, \ldots, eid_n\} & x \in \mathcal{L}it_{\textbf{\textit{real}}} \\
 & | & x & n \in \mathcal{L}it_{\textbf{\textit{int}}} \\
 & | & n & mid \in \mathcal{ID}_{\mathcal{M}} \\
 & | & mid & e_1, e_2, \ldots, e_k \in \mathcal{E} \\
 & | & \theta(e_1, e_2, \ldots, e_k) & k \geq 1 \\
 & & & \theta \in \{+, -, *, /, abs, max, pow, exp, log\}
\end{array}
$$

Ejemplo 6.2.3 *Las siguientes expresiones son válidas: 1, 1.5,* ***true*** *, n * log(n), {c2.LANG.SP, c2.LANG.ENG}, c1.play.TTF.mean, c1.play.TTR / c1.play.TTF.*

Fíjese en que aunque todos los símbolos de función se pueden aplicar en matemáticas en formato prefijo, es mucho más natural hacer uso de alguno de ellos en formato infijo. Por ejemplo, 1 + 2 puede escribirse de forma equivalente como +(1, 2) ya que es una función binaria.

Dada una expresión $e \in \mathcal{E}$ y una sección de medidas $M \in \mathcal{M}$, diremos que la expresión e es válida si y sólo si lo es su tipo. Esta condición es recogida por el predicado $valid : \mathcal{E} \times \mathcal{M}$ que se define como

$$valid(e, M) \iff type(e, M) \neq \bot$$

en donde el tipo de una expresión se obtiene a partir de la función $type : \mathcal{E} \times \mathcal{M} \mapsto \mathcal{T}$ que se define como

$$type(eid, M) = \begin{cases} \textbf{enum}\{eid_1, \ldots, eid_m\} & \text{si} \quad \begin{cases} m \in \operatorname{dom} M \wedge \\ M(m) = \Delta\{eid_1, \ldots, eid_m\} \wedge \\ \exists k \in [1..m] \cdot eid = eid_k \end{cases} \\ \bot & \text{eoc} \end{cases}$$

con $\Delta \in \{\textbf{enum}, \textbf{set}\}$, para el caso de expresiones constituidas por un valor enumerado corresponda a un dominio de tipo conjunto o enumerado. Por ejemplo, para el caso del identificador halt se tiene que su tipo es **enum** {halt, initialState, rolledBack} y para el caso del identificador ADSL su tipo es **set** {ADSL, ISDN, Modem56k}. Más adelante veremos la utilidad de definir de este modo el tipo de un valor enumerado.

$$type(\{eid_1, \ldots, eid_k\}, M) = \begin{cases} M(m) & \text{si} \quad \begin{cases} m \in \operatorname{dom} M \wedge \\ M(m) = \textbf{set}\{eid'_1, \ldots, eid'_n\} \wedge \\ \{eid_1, \ldots, eid_k\} \subseteq \{eid'_1, \ldots, eid'_n\} \end{cases} \\ \bot & \text{eoc} \end{cases}$$

para el caso de expresiones de tipo conjunto. Por ejemplo, considerando la sección de medidas de la figura §6.1 la expresión {EN, FR, IT} no es válida, pues no existe ninguna medida de tipo conjunto que verifique la condición anterior.

$type(n, M) = \textbf{int}[n..n]$ y $type(x, M) = \textbf{real}[x..x]$

para el caso de expresiones constituidas por un valor real o por un valor entero.

$$type(mid, M) = \begin{cases} M(mid) & \text{si} \quad mid \in \operatorname{dom} M \\ \bot & \text{eoc} \end{cases}$$

para el caso de expresiones constituidas por un único identificador de medida. Por ejemplo, la expresión c1.play.TTF es válida pues su tipo es **int** [0..∞].

$$type(\theta(e_1, \ldots, e_k), M) = \begin{cases} t & \text{si} \quad (type(e_1, M), \ldots, type(e_k, M), t) \in overloads(\theta) \\ \bot & \text{eoc} \end{cases}$$

para el caso de expresiones constituidas por operadores de aridad k y donde el conjunto *overloads* se define como:

$$
\begin{array}{lll}
overloads(\texttt{+}) = & \{ & (\textbf{int}[l_1,h_1],\textbf{int}[l_2,h_2],\textbf{int}[l_1+l_2,h_1+h_2]),\\
& & (\textbf{int}[l_1,h_1],\textbf{real}[l_2,h_2],\textbf{real}[l_1+l_2,h_1+h_2]),\\
& & (\textbf{real}[l_1,h_1],\textbf{int}[l_2,h_2],\textbf{real}[l_1+l_2,h_1+h_2]),\\
& & (\textbf{real}[l_1,h_1],\textbf{real}[l_2,h_2],\textbf{real}[l_1+l_2,h_1+h_2])\\
& \} & \\
overloads(\texttt{-}) = & \{ & (\textbf{int}[l_1,h_1],\textbf{int}[l_2,h_2],\textbf{int}[l_1-h_2,h_1-l_2]),\\
& & (\textbf{int}[l_1,h_1],\textbf{real}[l_2,h_2],\textbf{real}[l_1-h_2,h_1-l_2]),\\
& & (\textbf{real}[l_1,h_1],\textbf{int}[l_2,h_2],\textbf{real}[l_1-h_2,h_1-l_2]),\\
& & (\textbf{real}[l_1,h_1],\textbf{real}[l_2,h_2],\textbf{real}[l_1-h_2,h_1-l_2])\\
& \} & \\
overloads(\texttt{*}) = & \{ & (\textbf{int}[l_1,h_1],\textbf{int}[l_2,h_2],\textbf{int}[l_1*l_2,h_1*h_2]),\\
& & (\textbf{int}[l_1,h_1],\textbf{real}[l_2,h_2],\textbf{real}[l_1*l_2,h_1*h_2]),\\
& & (\textbf{real}[l_1,h_1],\textbf{int}[l_2,h_2],\textbf{real}[l_1*l_2,h_1*h_2]),\\
& & (\textbf{real}[l_1,h_1],\textbf{real}[l_2,h_2],\textbf{real}[l_1*l_2,h_1*h_2])\\
& \} & \\
overloads(\texttt{/}) = & \{ & (\textbf{int}[l_1,h_1],\textbf{int}[l_2,h_2],\textbf{int}[l_1/h_2,h_1/l_2]),\\
& & (\textbf{int}[l_1,h_1],\textbf{real}[l_2,h_2],\textbf{real}[l_1/h_2,h_1/l_2]),\\
& & (\textbf{real}[l_1,h_1],\textbf{int}[l_2,h_2],\textbf{real}[l_1/h_2,h_1/l_2]),\\
& & (\textbf{real}[l_1,h_1],\textbf{real}[l_2,h_2],\textbf{real}[l_1/h_2,h_1/l_2])\\
& \} &
\end{array}
$$

$$
\begin{array}{lll}
overloads(\textbf{pow}) = & \{ & (\textbf{int}[l_1,h_1],\textbf{int}[l_2,h_2],\textbf{int}[pow(l_1,l_2),pow(h_1,h_2)]),\\
& & (\textbf{int}[l_1,h_1],\textbf{real}[l_2,h_2],\textbf{real}[pow(l_1,l_2),pow(h_1,h_2)]),\\
& & (\textbf{real}[l_1,h_1],\textbf{int}[l_2,h_2],\textbf{real}[pow(l_1,l_2),pow(h_1,h_2)]),\\
& & (\textbf{real}[l_1,h_1],\textbf{real}[l_2,h_2],\textbf{real}[pow(l_1,l_2),pow(h_1,h_2)])\\
& \} &
\end{array}
$$

$$
\begin{array}{ll}
overloads(\textbf{max}) = & \{ \ (t[l_1,h_1],t[l_2,h_2],t[max(l_1),max(l_2)])\} \text{ donde } t \in \{\textbf{real},\textbf{int}\}\\
overloads(\textbf{abs}) = & \{ \ (t[l_1,h_1],t[abs(l_1),abs(h_1)])\} \text{ donde } t \in \{\textbf{real},\textbf{int}\}\\
overloads(\textbf{log}) = & \{ \ (t[l_1,h_1],real[log(l_1),log(h_1)])\} \text{ donde } t \in \{\textbf{real},\textbf{int}\}\\
overloads(\textbf{exp}) = & \{ \ (t[l_1,h_1],real[exp(l_1),exp(h_1)])\} \text{ donde } t \in \{\textbf{real},\textbf{int}\}
\end{array}
$$

Ejemplo 6.2.4 *La expresión c1.REBIND + c1.play.TTF no es válida pues una medida de tipo booleano no se puede sumar con una de tipo entero.*

La expresión (c1.play.TTF / c1.play.TTR) si es válida pues el resultado de la división de dos medidas de tipo entero es de tipo entero.

Definición 6.2.5 (Medidas de una expresión) *Dada una expresión* $e \in \mathcal{E}$ *definimos la función de proyección* $\mathcal{M}easures : \mathcal{E} \mapsto \mathcal{ID}_{\mathcal{M}}$ *que devuelve el conjunto de identificadores de medidas que forman parte de la expresión como:*

$$\mathcal{M}easures(eid) = \emptyset$$
$$\mathcal{M}easures(\{eid_1, \ldots, eid_k\}) = \emptyset$$
$$\mathcal{M}easures(n) = \emptyset$$
$$\mathcal{M}easures(x) = \emptyset$$
$$\mathcal{M}easures(mid) = \{mid\}$$
$$\mathcal{M}easures(\theta(e_1, e_2, \ldots, e_k\}) = \bigcup_{i=1}^{k} \mathcal{M}easures(e_i)$$

Ejemplo 6.2.5 *Los siguientes son ejemplos de proyecciones*

$$\mathcal{M}easures(\mathit{halt}) = \emptyset$$
$$\mathcal{M}easures(\mathit{p1.play.TTF}) = \{\mathit{p1.play.TTF}\}$$
$$\mathcal{M}easures(\mathit{p1.play.TTF} / \mathit{p1.play.TTR}) = \{\mathit{p1.play.TTF}, \mathit{p1.play.TTR}\}$$

Definición 6.2.6 (Evaluación de una expresión) *Dada una expresión $e \in \mathcal{E}$ y una sustitución σ que defina su contexto, denotaremos como $[\![e]\!]_\sigma$ el resultado de evaluar e tomando como valores para sus medidas los que proporciona σ.*

Ejemplo 6.2.6 *Dada la sustitución $\sigma = \{\mathit{c1.TTF} \mapsto 1440, \mathit{c1.TTR} \mapsto 10\}$, entonces se verifica que $[\![1 + 1]\!]_\sigma = 2$, $[\![100 - \mathit{c1.TTR} / \mathit{c1.TTF}]\!]_\sigma = 99.993$.*

Una definición completa y más detallada de la evaluación de una expresión no resulta relevante en nuestro contexto. Para aquellos lectores interesados en el tema le recomendamos consultar [Hennessy 1990].

Definición 6.2.7 (Restricción) *Definimos una restricción como un elemento del conjunto $\mathcal{C}$ que se define como*

$$\begin{array}{rcll} \mathcal{C} & \triangleq & C_1 \ \textbf{and} \ C_2 & \\ & | & C_1 \ \textbf{or} \ C_2 & \\ & | & \textbf{not} \ C & C, C_1, C_2 \in \mathcal{C} \\ & | & \theta(e_1, e_2) & e_1, e_2 \in \mathcal{E} \\ & | & \textbf{true} & \theta \in \{<, \leq, =, \neq, \geq, >, \in, \notin, \subset, \subseteq, \supset, \supseteq\} \\ & | & \textbf{false} & \end{array}$$

Dada una restricción $C \in \mathcal{C}$ y una sección de medidas $M \in \mathcal{M}$, diremos que C es válida si y sólo si es satisfactible y los tipos de las expresiones que se relacionan son adecuados. Estas dos condiciones son recogidas por el predicado $valid : \mathcal{C} \times \mathcal{M}$ que se define como:

$$valid(C, M) \iff sat(C, M) = true \wedge type(C, M) \neq \bot$$

en donde la función $sat : \mathcal{C} \times \mathcal{M}$ se define en la página 173 y la función $type : \mathcal{C} \times \mathcal{M} \mapsto \mathcal{T}$ se define como

$$type(C_1 \textbf{ and } C_2, M) = \begin{cases} \textbf{bool} & \text{si} \quad type(C_1, M) = type(C_2, M) = \textbf{bool} \\ \bot & \text{eoc} \end{cases}$$

$$type(C_1 \textbf{ or } C_2, M) = \begin{cases} \textbf{bool} & \text{si} \quad type(C_1, M) = type(C_2, M) = \textbf{bool} \\ \bot & \text{eoc} \end{cases}$$

$$type(\textbf{not } C, M) = \begin{cases} \textbf{bool} & \text{si} \quad type(C, M) = \textbf{bool} \\ \bot & \text{eoc} \end{cases}$$

$$type(\theta(e_1, e_2), M) = \begin{cases} \textbf{bool} & \text{si} \quad (type(e_1, M), type(e_2, M)) \in overloads(\theta) \\ \bot & \text{eoc} \end{cases}$$

$$type(\textbf{true}, M, T) = \textbf{bool}$$

$$type(\textbf{false}, M, T) = \textbf{bool}$$

en donde $overloads(\theta)$ define los tipos de medidas que se pueden relacionar el operador θ y se define como

$$\begin{aligned} overloads(=) = \{ \quad & (\textbf{int}[l_1, h_1], \textbf{int}[l_2, h_2]), \\ & (\textbf{int}[l_1, h_1], \textbf{real}[l_2, h_2]), \\ & (\textbf{real}[l_1, h_1], \textbf{int}[l_2, h_2]), \\ & (\textbf{real}[l_1, h_1], \textbf{real}[l_2, h_2]), \\ & (\textbf{enum}\{eid_1, \ldots, eid_n\}, \textbf{enum}\{eid_1, \ldots, eid_n\}), \\ & (\textbf{set}\{eid_1, \ldots, eid_n\}, \textbf{set}\{eid_1, \ldots, eid_n\}) \\ & (\textbf{bool}, \textbf{bool}) \\ \} & \end{aligned}$$

$$overloads(\neq) = overloads(=)$$

$$\begin{aligned} overloads(<) = \{ \quad & (\textbf{int}[l_1, h_1], \textbf{int}[l_2, h_2]), \\ & (\textbf{int}[l_1, h_1], \textbf{real}[l_2, h_2]), \\ & (\textbf{real}[l_1, h_1], \textbf{int}[l_2, h_2]), \\ & (\textbf{real}[l_1, h_1], \textbf{real}[l_2, h_2]), \\ & (\textbf{enum}\{eid_1, \ldots, eid_n\}, \textbf{enum}\{eid_1, \ldots, eid_n\}) \\ \} & \end{aligned}$$

$$\begin{aligned} overloads(\leq) &= overloads(<) \\ overloads(\geq) &= overloads(<) \\ overloads(>) &= overloads(<) \end{aligned}$$

$$
\begin{aligned}
&overloads(\subset) = \{(\textbf{set}\{eid_1, \ldots, eid_n\}, \textbf{set}\{eid_1, \ldots, eid_n\})\} \\
&overloads(\subseteq) = overloads(\subset) \\
&overloads(\supset) = overloads(\subset) \\
&overloads(\supseteq) = overloads(\subset)
\end{aligned}
$$

$$
\begin{aligned}
&overloads(\in) = \{\textbf{enum}\{eid_1, \ldots, eid_n\}, \textbf{set}\{eid_1, \ldots, eid_n\})\} \\
&overloads(\notin) = overloads(\in)
\end{aligned}
$$

Definición 6.2.8 (Proyección de una restricción) *Dada una restricción $C \in \mathcal{C}$, definimos la función de proyección $\mathcal{M}easures : \mathcal{C} \mapsto \mathcal{ID}_{\mathcal{M}}$ que devuelve el conjunto de identificadores de medidas que forman parte de la restricción como*

$$
\begin{aligned}
&\mathcal{M}easures(C_1 \textbf{ and } C_2) = \mathcal{M}easures(C_1) \cup \mathcal{M}easures(C_2) \\
&\mathcal{M}easures(C_1 \textbf{ or } C_2) = \mathcal{M}easures(C_1) \cup \mathcal{M}easures(C_2) \\
&\mathcal{M}easures(\textbf{not } C) = \mathcal{M}easures(C) \\
&\mathcal{M}easures(\{\theta(e_1, e_2\}) = \mathcal{M}easures(e_1) \cup \mathcal{M}easures(e_2) \\
&\mathcal{M}easures(\textbf{true}) = \emptyset \\
&\mathcal{M}easures(\textbf{false}) = \emptyset
\end{aligned}
$$

Definición 6.2.9 (Partición de restricciones) *Por conveniencia particionaremos el conjunto de restricciones $\mathcal{C}$ en el conjunto de restricciones de calidad $\mathcal{C}^{\mathcal{Q}}$ y el conjunto de restricciones temporales $\mathcal{C}^{\tau}$, verificándose que*

$$
\begin{aligned}
&\forall C \in \mathcal{C}^{\mathcal{Q}} \cdot \mathcal{M}easures(C) \cap \{\tau\} = \emptyset \\
&\forall C \in \mathcal{C}^{\tau} \cdot \mathcal{M}easures(C) = \{\tau\}
\end{aligned}
$$

Esta separación es necesaria, pues como veremos más adelante estudiar la conformidad entre restricciones definidas simultáneamente sobre medidas de calidad y sobre la variable temporal no es trivial.

Para definir con rigor el procedimiento de resolución de restricciones, es necesario definir algunos conceptos que nos permitan definir si una restricción es satisfactible en un contexto determinado.

Definición 6.2.10 (Contexto de una restricción) *Dada una restricción $C \in \mathcal{C}$, diremos que una sustitución σ define el contexto de C si y sólo si σ dispone de un valor para cada una de las medidas de C, es decir, si se verifica que $\mathcal{M}easures(C) \subseteq \mathrm{dom}\,\sigma$.*

Ejemplo 6.2.7 *La sustitución $\sigma = \{c1.TTR \mapsto 10\}$ define el contexto de la restricción $c1.TTR \leq 50$, pero no el de $100 - c1.TTR \,/\, c1.TTF \leq 99.999$.*

Definición 6.2.11 (Evaluación de una restricción) *Dada una restricción $C \in \mathcal{C}$ y una sustitución σ que defina su contexto, denotaremos como $[\![C]\!]_\sigma$ al resultado de evaluar la expresión que resulta de aplicar la sustitución σ sobre C.*

Ejemplo 6.2.8 *Dada la restricción C definida como* c1.REBIND = **true** *y la sustitución $\sigma = \{$c1.REBIND $\mapsto$ **false** $\}$ se verifica que $[\![C]\!]_\sigma = false$.*

Fíjese en que si σ es una sustitución que define el contexto de C, la aplicación de dicha sustitución sobre C da como resultado una expresión directamente evaluable. En nuestro caso se verifica que $\sigma(C)$ = **false** = **true** *por lo que $[\![C]\!]_\sigma = false$.*

Dado que en QRL$_\heartsuit$ no existen expresiones lógicas, sólo aritméticas, el operador que relaciona a las expresiones resultantes de aplicar sustituciones en una restricción es en realidad un predicado. La satisfactibilidad de este predicado se determina utilizando un resolutor.

Definición 6.2.12 (Resolutor de restricciones) *Implícitamente supondremos que* QRL$_\heartsuit$ *cuenta con un procedimiento de resolución de restricciones que a partir de una sección de medidas $M \in \mathcal{M}$ puede determinar la existencia o no de una sustitución σ que defina el contexto de cualquier restricción $c \in \mathcal{C}$. Lo representaremos como $\longrightarrow_\heartsuit$ y lo definimos como*

$$\begin{array}{lcl} (c, M) \xrightarrow{\sigma}_\heartsuit true & \iff & [\![c]\!]_\sigma = \textbf{true} \\ (c, M) \longrightarrow_\heartsuit false & \iff & \nexists \sigma \cdot [\![c]\!]_\sigma = \textbf{true} \\ (c, M) \longrightarrow_\heartsuit \bot & \iff & \forall \sigma \, [\![c]\!]_\sigma = \bot \end{array}$$

Considerar que el resultado del resolutor puede ser indeterminado es necesario pues ha de tenerse en cuenta que hasta la fecha no hay ningún resolutor completo, esto es, que resuelva cualquier tipo de restricción. Por ejemplo, una restricción tan simple como $x * y < 7$ obtiene una respuesta indeterminada si se utiliza CLP($\Re$) como resolutor, pues los algoritmos que incorpora CLP($\Re$) no son capaces de analizarla. Más concretamente no son capaces de transformarla en una restricción lineal.

Ejemplo 6.2.9 *Dada la restricción $c \triangleq$* c1.REBIND = **true and** c1.TTF $\geq$ 120, *los contextos $\sigma_1 = \{$*c1.REBIND $\mapsto$ **true**, c1.TTF $\mapsto$ 180*$\}$ y $\sigma_2 = \{$*c1.REBIND $\mapsto$ **false**, c1.TTF $\mapsto$ 135 *$\}$ verifican que $c \xrightarrow{\sigma_1}_\heartsuit y \wedge c \xrightarrow{\sigma_2}_\heartsuit false$.*

Definición 6.2.13 (Satisfactibilidad) *Dada una restricción $C \in \mathcal{C}$ y una sección de medidas $M \in \mathcal{M}$, diremos que C es satisfactible si y sólo si se verifica que sat(C, M)= true, donde la función $sat : \mathcal{C} \times \mathcal{M} \mapsto \{true, false, \perp\}$ se define formalmente como*

$$sat(C,M) = \begin{cases} true & si \quad (C,M) \xrightarrow{\sigma}_{\heartsuit} true \\ false & si \quad (C,M) \longrightarrow_{\heartsuit} false \\ \perp & si \quad (C,M) \longrightarrow_{\heartsuit} \perp \end{cases}$$

Existen muchas sustituciones que consiguen la satisfactibilidad de una restricción. El conjunto de todas las posibles sustituciones que consiguen la satisfactibilidad de una restricción determina la solución de dicha restricción.

Definición 6.2.14 (Solución de una restricción) *Dada una restricción $C \in \mathcal{C}$ y una sección de medidas $M \in \mathcal{M}$, denotamos su solución con la función $sol : \mathcal{C} \mapsto 2^{\mathcal{ID}_{\mathcal{M}} \times \mathcal{E}}$ que se define como*

$$sol(C,M) = \bigcup_{i=1}^{n} \{\sigma_i\} \cdot (C,M) \xrightarrow{\sigma_i}_{\heartsuit} true$$

Ejemplo 6.2.10 *Dada la restricción $t \triangleq \tau \geq 8$ **and** $\tau \leq 10$, su conjunto de soluciones se corresponde con $sol(t,M) = \{\{\tau \mapsto 8\}, \{\tau \mapsto 9\}, \{\tau \mapsto 10\}\}$.*

Dada la restricción $q \triangleq LANG \supseteq \{EN, SP\}$ se tiene que

$$\begin{aligned} sol(q,M) = \{ \quad & \{LANG \mapsto \{EN, SP\}, \\ & \{LANG \mapsto \{EN, FR, SP\}, \\ & \{LANG \mapsto \{EN, FR, DE\}, \\ & \{LANG \mapsto \{EN, FR, DE, SP\} \quad \} \end{aligned}$$

Definición 6.2.15 (Cardinal de una restricción) *Dada una restricción $C \in \mathcal{C}$ denotaremos como $|C|$ al cardinal del conjunto que contiene a todas las soluciones de C siendo su signatura $|\_| : \mathcal{C} \mapsto \mathbb{R} \cup \{\infty\}$.*

Ejemplo 6.2.11 *Dada la restricción $c \triangleq T \in [1..24]$ para una medida T de tipo **int**, se tiene que $|c| = 24$. En el caso de que $c \triangleq T > 24$ se tiene que $|c| = \infty$*

6.2.4.2. Requisitos

Una sección de requisitos en $\text{QRL}_{\heartsuit}$ es el resultado de la normalización de las secciones de condiciones y de negociación de un documento de requisitos en QRL.

Un requisito en QRL puede representar a una sección básica de condiciones o al resultado de aplicar una o varias cláusulas de negociación a una sección básica de condiciones. En ambos casos, un requisito hace referencia a un conjunto de restricciones de calidad, cada una de las cuales están referidas a un período de vigencia.

Con objeto de simplificar la comprobación de la conformidad entre documentos de requisitos, los períodos de vigencia y sobre todo por la obligatoriedad de expresar los períodos de vigencias mediante familias o conjuntos de restricciones, los períodos de vigencia en $\text{QRL}_{\heartsuit}$ deben estar referidos a un único intervalo, esto es, tienen que ser expresados con una restricción del tipo $\tau \geq \text{bound}_{\text{low}}$ **and** $\tau \leq \text{bound}_{\text{high}}$. Nos referiremos a las restricciones que se pueden expresar de este modo como restricciones temporales normalizadas.

Definición 6.2.16 (Restricción temporal normalizada) *Sea $t \in \mathcal{C}^{\tau}$ una restricción temporal tal que $sol(t, M) = \{\{\tau \mapsto v_1\}, \ldots, \{\tau \mapsto v_n\}\}$ donde $v_1 < v_2, \ldots, v_{n-1} < v_n$, diremos que t es una restricción temporal normalizada si se verifica que*

$$\nexists n \cdot n \in [v_l, v_n] \cdot \{\tau \mapsto n\} \notin sol(t, M)$$

*En adelante, cuando sea necesario expresar de manera extendida una restricción temporal normalizada lo denotaremos como $\tau \geq b_l$ **and** $\tau \leq b_h$, donde $b_l = v_1$ y $b_h = v_n$. Al subconjunto de restricciones de $\mathcal{C}^{\tau}$ que están normalizadas lo denotaremos como $\overline{\mathcal{C}^{\tau}}$.*

Ejemplo 6.2.12 *La restricción $\tau \geq 8$ **and** $\tau < 12$ es una restricción temporal normaliza mientras que la restricción $(\tau \geq 8$ **and** $\tau < 12)$ **or** $(\tau > 4$ **and** $\tau < 8)$ no lo es.*

Esta exigencia sobre un período de vigencia no impide poder expresar requisitos con períodos de vigencia constituidos por múltiples intervalos

como es el caso de QRL, pues como veremos en la sección §6.3.3 siempre es posible transformar un requisito de QRL a un requisito de $\text{QRL}_{\heartsuit}$ a costa de obtener especificaciones de mayor tamaño. Para distinguir entre ambas formas de expresar un requisito, nos referiremos a los requisitos de $\text{QRL}_{\heartsuit}$ como requisitos normalizados.

Definición 6.2.17 (Requisito normalizado) *Definimos un requisito normalizado como un elemento del conjunto* $\overline{\mathcal{R}}$ *que se define como* $\overline{\mathcal{R}} \triangleq 2^{C^Q \times \overline{C^\tau}}$. *Un requisito normalizado* $R \in \overline{\mathcal{R}}$ *también lo denotaremos como* $\{(q_i, t_i)\}_{i=1}^n$, *en donde* $q_i \in C^Q$ *representa a una restricción de calidad y cada* $t_i \in \overline{C^\tau}$ *a una restricción temporal normalizada.*

Ejemplo 6.2.13 *Un requisito expresado en* QRL *como*

TTF $>$ *8* **from** *8..10,12..14* **and** *TTF* $>$ *6* **from** *16..20*

se corresponde en $\text{QRL}_{\heartsuit}$ *al requisito normalizado expresado como*

$\{(TTF > 8, \tau \in [8..10]), (TTF > 8, \tau \in [12..14]), (TTF > 6, \tau \in [16..20]), \ldots\}$[1]

Dado un requisito normalizado $R \in \overline{\mathcal{R}}$ y una sección de medidas $M \in \mathcal{M}$, diremos que R es válido si y sólo si es consistente y no ambiguo. Estas condiciones son recogidas por el predicado $valid : \overline{\mathcal{R}} \times \mathcal{M}$ que se define como

$$valid(\{(q_i, t_i)\}_{i=1}^n, M) \iff \forall i, j \in [1, n] \cdot \begin{cases} sat(q_i, M) = true \wedge \\ sat(t_i, M) = true \wedge \\ (i \neq j \Rightarrow \\ sat(t_i \textbf{ and } t_j, M) = false) \end{cases}$$

Las dos primeras condiciones garantizan la consistencia al exigir la satisfactibilidad de la restricción de calidad y la restricción temporal. La última condición garantiza la ausencia de ambigüedad al exigir la ausencia de solapamiento entre los intervalos que representan las restricciones temporales, pues de este modo no es posible que para un mismo instante de tiempo existan dos restricciones de calidad diferentes.

Como veremos en la sección §6.3.4, al normalizar un requisito de QRL sobre $\text{QRL}_{\heartsuit}$ resulta necesario disponer de una operación que nos permita

[1] La normalización del período de vigencia del ejemplo genera más restricciones temporales de las que aquí se muestran (ver sección §6.3.3). No obstante, conocer todas las restricciones que se generan no es relevante en este punto, por lo que hemos preferido evitarlas a fin de ganar en claridad.

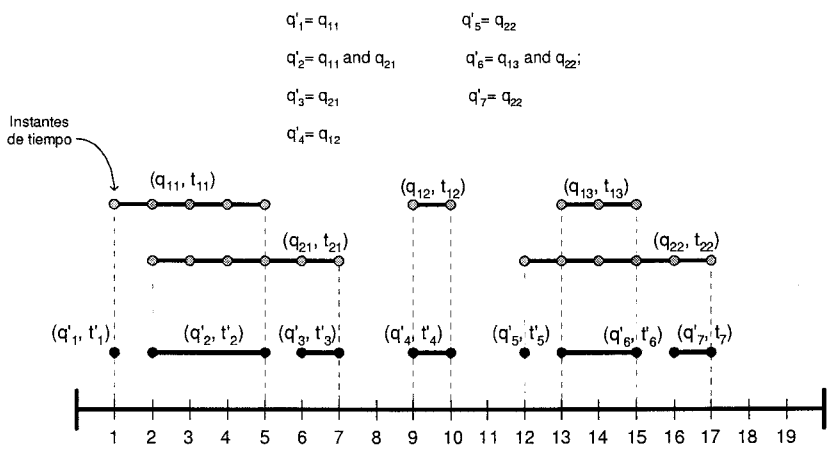

Figura 6.2: *Unión de dos requisitos normalizados.*

"unir" varios requisitos normalizados. Es fácil comprobar que si consideramos la unión de requisitos normalizados como la unión de sus elementos, en el caso general, el resultado es un conjunto que no cumple los criterios de validez de un requisito normalizado pues es más que probable la existencia de solapamiento entre los períodos de vigencia. La figura §6.2 ilustra dicha situación con la unión de dos requisitos normalizados que presentan solapamiento en dos de sus intervalos de tiempo diferentes.

La figura §6.2 también ilustra el significado que en QRL tiene la unión, que no es otro que el de conseguir un nuevo requisito normalizado que contempla todas las restricciones de calidad contempladas en los requisitos iniciales.

Definición 6.2.18 (Unión de requisitos) *Sea* $\{\{(q_{ij}, t_{ij}\}_{j=1}^{m}\}_{i=1}^{n}$ *un conjunto de requisitos normalizados, que denotaremos como* $\{(q_i, t_i)\}_{i=1}^{n}$ *o como* $\{r_i\}_{i=1}^{n}$ *de manera más simplificada, denotamos la unión de dicho conjunto de requisitos con la función* $\oplus : 2^{\overline{\mathcal{R}}} \mapsto \overline{\mathcal{R}}$ *que se define como*

$$\oplus(\{(q_i, t_i)\}_{i=1}^{n}) = \{(q'_j, t'_j)\}_{j=1}^{m}$$

siempre y cuando se verifique que

$$\forall j \in \{1,\ldots,m\}\cdot$$

$$\begin{cases} \forall s \in sol(t'_j, M)\cdot \exists i \in \{1,\ldots,n\}\cdot s \in sol(t_i, M) \wedge \\ sol(q'_j, M) = sol((\mathop{\textbf{and}}_{q_i \in H} q_i), M) \end{cases}$$

donde el conjunto H contiene a todas las restricciones de calidad activas en un intervalo determinado, esto es, las restricciones de aquellos requisitos cuyo período de vigencia se solapa con el período de vigencia asociado a un intervalo determinado.

$$H = \{q_i \in \{q_i\}_{i=1}^{n}\cdot sat(t_i \ \textbf{and} t, M) = true\}$$

Otra operación que resulta de interés para comprobar la conformidad entre documentos de requisitos es la conformidad entre requisitos normalizados.

Definición 6.2.19 (Conformidad entre requisitos) *Definimos la relación de conformidad $\rightarrow$: $\overline{\mathcal{R}} \times \overline{\mathcal{R}}$ entre requisitos normalizados como*

$$\{(q_i, t_i)\}_{i=1}^{n} \rightarrow \{(q'_j, t'_j)\}_{j=1}^{m} \iff$$

$$\iff \forall j \in \{1,\ldots,m\}\cdot \exists \mathcal{I} \subseteq \{1,\ldots,n\}\cdot \begin{cases} (\mathop{\textbf{and}}_{\iota \in \mathcal{I}} t_i) \rightarrow t'_j \wedge \\ (\mathop{\textbf{and}}_{\iota \in \mathcal{I}} q_i) \rightarrow q'_j \end{cases}$$

Para aquellos lectores familiarizados con la programación con restricciones esta relación de conformidad le debe resultar familiar, pues está basada en la *restricción de implicación* definida en [Marriot y Stuckey 1998].

Definición 6.2.20 (Sección de requisitos normalizados) *Definimos una sección de requisitos como un elemento del conjunto $\overline{\mathcal{RS}}$ que se define como: $\mathcal{RS} \triangleq \mathcal{ID}_{\mathcal{R}} \mapsto \overline{R}$*

Dada una sección de requisitos $RS \in \overline{\mathcal{RS}}$, tal que $RS = \{r_i\}_{i=1}^{n}$ diremos que RS es válida si y sólo si son válidos todos sus requisitos. Esta condición es recogida por el predicado $valid : \overline{\mathcal{RS}} \times \mathcal{M}$ que se define como

$$valid(RS, M) = \bigwedge_{i=1}^{n} valid(r_i, M)$$

Fíjese que no se imponen condiciones en cuanto a las relaciones de los requisitos entre sí, pues se considera que cada requisito normalizado se corresponde con un documento perteneciente a la clausura de una sección de condiciones (definición §6.3.25, página 201) por lo que deben ser tratados independientemente.

6.2.5. Documento de calidad

Definición 6.2.21 (Documento de calidad) *Dada una sección de sección de medidas $M \in \mathcal{M}$, una sección de requisitos normalizados $R \in \overline{\mathcal{R}}$ y una sección de preferencias $P \in \mathcal{P}$, definimos un documento de calidad como un elemento del conjunto $\mathcal{D}$ que se define como:*

$$\mathcal{D} \triangleq \textbf{\textit{measures}}\,\mathcal{M} \times \textbf{\textit{requirements}}\,\overline{\mathcal{R}} \times \textbf{\textit{utility}}\,\mathcal{P}$$

La figura §6.1 muestra un documento de calidad.

Dado un documento de calidad $D = (M, R, P) \in \mathcal{D}$, diremos que el documento D es válido si y sólo si se satisface el predicado $valid : \mathcal{D}$ que se define como

$$valid(D) \iff valid(M) \wedge valid(R, M) \wedge valid(P, M)$$

6.3. Normalización de QRL sobre QRL$_\heartsuit$

En la sección anterior hemos descrito rigurosamente la semántica estática del núcleo de QRL. En esta sección describimos la semántica de QRL indicando las transformaciones que permiten expresar los diferentes elementos de QRL sobre elementos de QRL$_\heartsuit$. La figura §6.3 muestra a grandes rasgos las correspondencias que se establecen entre los elementos de QRL y QRL$_\heartsuit$ durante la normalización.

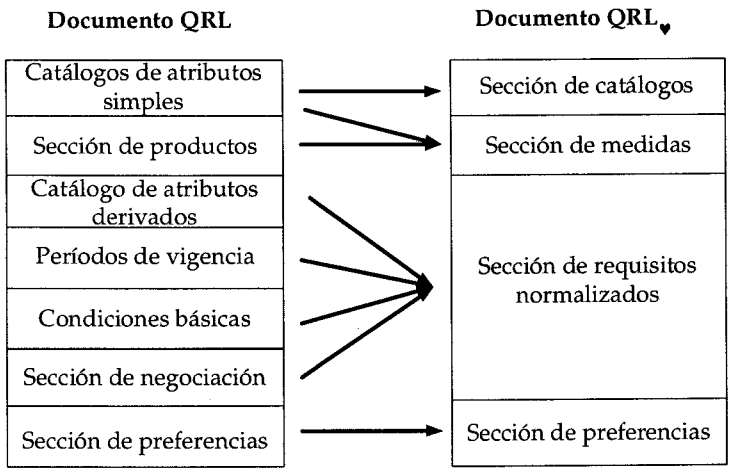

Figura 6.3: *Esquema general de la normalización de un documento* QRL *en un documento* $\text{QRL}_{\heartsuit}$.

6.3.1. Catálogo de atributos simples

La normalización de una sección de atributos da como resultado una sección de catálogos y un mapa otra de tipos. En el caso de los catálogos de atributos simples, una sección con la estructura

catalogue idc_1 {
 attributes {
  ida_1 { **description**: ...; **domain**: def_1 [unit_1]; [**statistic**: $\text{sm}_1, \ldots \text{sm}_n$];}
  ...
  ida_n { **description**: ...; **domain**: def_n [unit_n]; [**statistic**: $\text{sm}_1, \ldots \text{sm}_n$];}
 }
}
...
catalogue idc_n {
 attributes {
  ida_1 { **description**: ...; **domain**: def_1 [unit_1]; [**statistic**: $\text{sm}_1, \ldots \text{sm}_n$];}
  ...
  ida_m { **description**: ...; **domain**: def_m [unit_m]; [**statistic**: $\text{sm}_1, \ldots \text{sm}_n$];}
 }
}

se genera una sección de catálogos con la estructura

catalogue { c_1: idc_1, ..., c_n: idc_n; }

y un mapa con la estructura

{
// Normalización del primer catálogo
$idc_1.ida_1 \mapsto$ normDomain($idc_1.ida_1$, def_1),
$idc_1.ida_1.sm_1 \mapsto$ normDomain($idc_1.ida_1$, def_1),
...
$idc_1.ida_1.sm_m \mapsto$ normDomain($idc_1.ida_1$, def_1),

...

$idc_1.ida_n \mapsto$ normDomain($idc_1.ida_m$, def_m),
$idc_1.ida_n.sm_1 \mapsto$ normDomain($idc_1.ida_m$, def_m),
...
$idc_1.ida_n.sm_n \mapsto$ normDomain($idc_1.ida_m$, def_m),

... // Normalización de los catálogos intermedios

// Normalización del último catálogo
$idc_n.ida_1 \mapsto$ normDomain($idc_n.ida_1$, def_1),
$idc_n.ida_1.sm_1 \mapsto$ normDomain($idc_1.ida_1$, def_1),
...
$idc_n.ida_1.sm_m \mapsto$ normDomain($idc_1.ida_1$, def_1),

...

$idc_n.ida_n \mapsto$ normDomain($idc_1.ida_m$, def_m),
$idc_n.ida_n.sm_1 \mapsto$ normDomain($idc_1.ida_m$, def_m),
...
$idc_n.ida_n.sm_n \mapsto$ normDomain($idc_1.ida_m$, def_m),
}

donde

- $idc_j.ida_i$ es el identificador resultante de la concatenación del identificador del catálogo j-ésimo y el identificador del atributo i-ésimo.
- $idc_j.ida_i.sm_k$ es el identificador resultante de concatenar el identificador del catálogo j-ésimo, el identificador del atributo i-ésimo y el identificador del modificador estadístico k-ésimo.
- def_i es la definición del dominio del atributo i-ésimo.

- normDomain es una función que obtiene la definición del tipo equivalente en QRL$_{\heartsuit}$ a la definición del tipo que se le pasa como parámetro. normDomain se define como

normDomain (id, **int**)= **int** [$-\infty..\infty$]
normDomain (id, **int** [$l_1..l_2$])= **int** [$l_1..l_2$]
normDomain (id, **int***)= **int** [0..∞]
normDomain (id, **real**)= **real** [$-\infty..\infty$]
normDomain (id, **real** [$l_1..l_2$])= **real** [$l_1..l_2$]
normDomain (id, **real***)= **real** [0..∞]
normDomain (id, **enum** {$eid_1,\ldots,eid_n$})= **enum** {$id.eid_1,\ldots,id.eid_n$}
normDomain (id, **set** {$eid_1,\ldots,eid_n$})= **set** {$id.eid_1,\ldots,id.eid_n$}
normDomain (id, **bool**)= **bool**

La figura §6.4 muestra el resultado de normalizar dos catálogos de atributos de calidad.

El esquema de normalización empleado facilita conseguir la unicidad de los identificadores de valores enumerados tanto si están asociados a un tipo enumerado (**enum**) como a un tipo conjunto (**set**), pues éstos se obtienen derivando el identificador del catálogo de atributos que al ser un URN tienen garantizada (al menos debería) la unicidad.

En QRL se considera que el significado de la unidad de un atributo es conocido por los que utilizan los catálogos y que los identificadores empleados son correctos por definición. La función de un identificador de unidad, así como la del campo de descripción es meramente informativa y ninguno de los dos se vuelven a utilizar en ninguna otra parte de un documento de requisitos. Por tanto, no es necesario que dicha información aparezca en un documento normalizado.

No obstante, para aquellos casos en los que resulta necesario conocer esta información, como es el caso de las unidades para comprobar la conformidad entre documentos definidos sobre diferentes catálogos (sección §??), es posible recuperarla a partir de los identificadores de los catálogos de atributos normalizados, los cuales, sí forman parte del documento normalizado.

Importación de catálogos

La normalización de catálogos importados (aquéllos que están referenciados en la cláusula **using**) sigue el mismo procedimiento que el de los catálogos especificados directamente en el mismo documento.

Optimación de la normalización

Con frecuencia, los atributos utilizados para especificar requisitos son sólo una pequeña parte de todos los atributos contenidos en los catálogos

```
catalogue com.acme.stdMOWS {
 attributes {
  TTF { description:"..."; domain: int* min; statistic: min, mean, percentile80; }
  AVAIL { description: "..."; domain: real [0..100] %; }
  SFAIL { description: "..."; domain: enum {halt, initialState, rolledBack}; }
 }
catalogue com.acme.multimedia {
 attributes {
  STREAMING { description:"..."; domain: bool; }
  COST { description:"..."; domain: real* euro; }
  LANG { description:"..."; domain: set {EN, FR, GE, SP}; }
 }
}
```

a) Catálogos de atributos descritos en QRL

```
catalogue { c1: com.acme.stdMOWS; c2: com.acme.multimedia};

σ_M {
  c1.TTF↦ int [0..∞],
  c1.TTF.min↦ int [0..∞],
  c1.TTF.mean↦ int [0..∞],
  c1.TTF.percentile80↦ int [0..∞],
  c1.AVAIL↦ real [0..100],
  c1.SFAIL↦ enum {c1.SFAIL.halt, c1.SFAIL.initialState, c1.SFAIL.rollBack},
  c1.REBIND↦ bool ,
  c2.STREAMING↦ bool ,
  c2.COST↦ real [0..∞],
  c2.LANG↦ set {c2.LANG.EN, c2.LANG.FR, c2.LANG.GE, c2.LANG.SP}
 }
```

b) Sección de catálogos equivalente en $\text{QRL}_{\heartsuit}$ y mapa de medidas asociado.

Figura 6.4: *Ejemplo de normalización de dos catálogos de atributos.*

referidos en una especificación. En tales casos, la sección de tipos incluye muchas definiciones innecesarias. Es posible plantear transformaciones que consiguen una sección de tipos "mínima", es decir, que sólo tiene los tipos necesarios para poder definir las medidas utilizadas en la sección de requisitos y de negociación. Este tipo de optimaciones no presentan dificultad y entran en el ámbito de la implementación del compilador de QRL, por lo que no merece la pena entrar en detalles sobre ellas en este punto.

La secuencia natural es mostrar a continuación la normalización de los catálogos de atributos derivados, no obstante, dado que para su explicación es necesario conocer el proceso de normalización de un requisito, posponemos su explicación.

6.3.2. Productos

La normalización de una sección de productos da como resultado la definición de todas de medidas de calidad que pueden realizarse sobre el producto definido en dicha sección. De manera general, para una descripción QRL con la estructura

using $C_1, \ldots, C_n$;
catalogue $C_{n+1}\{\ldots\}, \ldots, C_{n+m}\{\ldots\}$;
products { idp = $idsp_1, \ldots, idsp_n$;}

debemos añadir una sección de medidas con la siguiente estructura:

measures {
τ: **int** [0..∞];
idp.m_1: $\sigma_{\mathcal{M}}(m_1)$;
...
idp.m_n: $\sigma_{\mathcal{M}}(m_n)$;

idp.$idsp_1$.m_1: $\sigma_{\mathcal{M}}(m_1)$;
...
idp.$idsp_1 m_n$: $\sigma_{\mathcal{M}}(m_n)$;

...

idp.$idsp_n$.m_1: $\sigma_{\mathcal{M}}(m_1)$;
...
idp.$idsp_n m_n$: $\sigma_{\mathcal{M}}(m_n)$;
}

donde $\sigma_{\mathcal{M}}$ es el mapa de medidas que se obtiene al normalizar la sección de catálogos y para el que consideramos que $\mathrm{dom}(\sigma_{\mathcal{M}}) = \{m_1, \ldots, m_n\}$

La figura §6.5 muestra un ejemplo de normalización de productos. Salta a la vista que la normalización de una sección de productos genera una sección de medidas que puede ser varios ordenes de magnitud mayor. Sin tener en cuenta los modificadores estadísticos sobre los atributos, el total de de medidas obtenidas es de $sp * n$, donde sp es el número de subproductos del producto y n es el número de atributos de todos los catálogos referenciados.

Obsérvese que en el proceso de normalización también se define la medida τ de tipo (**int** [0..∞]), sobre la cual se referirán todas las restricciones temporales resultantes de la normalización de los períodos de vigencia.

6.3.3. Períodos de vigencia

6.3.3.1. Representación mediante restricciones

Encontrar una restricción para representar a un período de vigencia cuando está compuesto por un único intervalo es sencillo. Por ejemplo, el período **from** 8..14 queda bien representado por la restricción $\tau \geq 8$ **and** $\tau \leq 14$, pues la solución de la restricción coincide con todos los puntos del período. Sin embargo, para un período de vigencia compuesto por dos intervalos es más difícil (sino imposible) encontrar una única restricción.

Por ejemplo, para el período **from** 8..14,16..18 no hemos encontrado ninguna restricción que lo represente adecuadamente. Algunas de las restricciones que hemos valorado han sido:

$(\tau \geq 8 \textbf{ and } \tau \leq 14) \textbf{ or} (\tau \geq 16 \textbf{ and } \tau \leq 18)$
$(\tau \geq 8 \textbf{ and } \tau \leq 14) \textbf{ and} (\tau \geq 16 \textbf{ and } \tau \leq 18)$
$(\tau \geq 8 \textbf{ and } \tau \leq 18) \textbf{ and} ((\tau \geq 8 \textbf{ and } \tau \leq 14) \textbf{ or} (\tau \geq 16 \textbf{ and } \tau \leq 18))$

La solución tomada a este respecto ha sido representar un período de vigencia por una familia o conjunto de restricciones. De este modo, el período **from** 8..14,16..18 será representado por el siguiente conjunto de restricciones temporales normalizadas

$$\{(\tau \geq 8 \textbf{ and } \tau \leq 14), (\tau \geq 16 \textbf{ and } \tau \leq 18)\}$$

```
catalogue com.acme.stdMOWS {
 attributes {
  TTF { description:"..."; domain: int* min; statistic: min, mean, percentile80; }
  AVAIL { description: "..."; domain: real [0..100] %; }
  SFAIL { description: "..."; domain: enum {halt, initialState, rolledBack}; }
 }
 products {org.W3C.IVideoServer= { Play, Stop; } }
```

a) Definición de productos en QRL

```
 measures {
   τ: int [0..∞];
   c1.TTF: int [0..∞];
   c1.TTF.min: int [0..∞];
   c1.TTF.mean: int [0..∞];
   c1.TTF.percentile80: : int [0..∞];
   c1.AVAIL: real [0..100];
   c1.SFAIL: enum {c1.SFAIL.halt, c1.SFAIL.initialState, c1.SFAIL.rollBack};

   Play.c1.TTF: int [0..∞];
   Play.c1.TTF.min: int [0..∞];
   ...
   Play.c1.AVAIL: c1.AVAIL;
   Play.c1.SFAIL: enum {c1.SFAIL.halt, c1.SFAIL.initialState, c1.SFAIL.rollBack};

   Stop.c1.TTF: c1.TTF;
   Stop.c1.TTF.min: c1.TTF.min;
   Stop.c1.AVAIL: real [0..100];
   Stop.c1.SFAIL: enum {c1.SFAIL.halt, c1.SFAIL.initialState, c1.SFAIL.rollBack};
   Stop.c1.REBIND: bool;
}
```

b) Sección de medidas asociada en $\text{QRL}_{\heartsuit}$, en donde $c1$ representa a com.acme.stdMOWS.

Figura 6.5: *Ejemplo de normalización de una sección de productos.*

6.3.3.2. Normalización de un período de vigencia básico

QRL permite la especificación de períodos de vigencia constituidos por múltiples intervalos. Los elementos básicos para especificar un período son los patrones horarios, semanales y de fechas, los cuales, permiten indicar las franjas horarias, días de la semana e intervalos de fechas en los que estamos interesados. También es posible realizar operación de unión y diferencia sobre los períodos de vigencia.

Definición 6.3.1 (Período de vigencia) *Definimos un período de vigencia como un elemento del conjunto $\mathcal{VP}$ que se define como:*

$$\begin{array}{rcll} \mathcal{VP} & \triangleq & VP_1 \ \textbf{\textit{and}} \ VP_2 & VP_1, VP_2 \in \mathcal{VP} \\ & | & VP_1 \ \textbf{\textit{except}} \ VP_2 & \\ & | & \textbf{\textit{from}} \ hp \ \textbf{\textit{on}} \ wp \ \textbf{\textit{during}} \ dp & hp \in \mathcal{I}_H, wp \in \mathcal{I}_W, dp \in \mathcal{I}_D \end{array}$$

donde H_P, W_P, D_P son secuencias de intervalos que representan a los patrones horarios, semanales y de fechas respectivamente.

Definición 6.3.2 (Secuencia de intervalos) *Dado un conjunto χ que dispone de una relación de orden parcial ($\leq$), definimos una secuencia de intervalos en χ como un elemento del conjunto $\mathcal{I}_\chi$ que se define como $\mathcal{I}_\chi \triangleq seq\ \chi \times \chi$.*

Dada una secuencia de intervalos $\delta \in \mathcal{I}_\chi$ diremos que δ es válida si todos sus intervalos están ordenados y son disyuntos entre sí. Estas condiciones son recogidas por el predicado $valid : \mathcal{I}_\chi$ que se define como

$$valid(\delta) \iff \forall i \geq 1 \cdot \begin{cases} \delta(i).begin \leq \delta(i).end \wedge \\ \delta(i).end < \delta(i+1).begin \end{cases}$$

donde $\delta(i).begin$ y $\delta(i).end$ hacen referencia al límite inferior y superior del intervalo i–ésimo.

Los conjuntos sobre los que se definen las secuencias de intervalos horarios, semanales y de fechas son los siguientes:

$$H = \{(h, m) \in \mathbb{N} \times \mathbb{N} \mid 0 \leq h \leq 23, 0 \leq m \leq 59\}$$
$$W = \{MON, TUE, WED, THU, FRI, SAT, SUN\}$$
$$D = \{d \in \mathcal{D}ates \mid d \geq 01/JAN/2001\}$$

La sintaxis concreta para representar cada uno de estas secuencias es la siguiente

hp ::= h_1:m_1..h_2:m_2,...,h_{m-1}:m_{m-1}..h_m:m_m
wp ::= w_1..w_2,...,w_{n-1}..w_n
dp ::= d_1..d_2,...,d_{q-1}..d_q

La normalización de un período de vigencia consiste en transformarlo en una familia de restricciones temporales normalizadas. Para describir esta transformación definiremos previamente la normalización de los patrones horarios, semanales y de fechas.

Definición 6.3.3 (Normalización de un patrón horario) *Dado un patrón horario $hp \in \mathcal{I}_H$ su normalización, queda determinada por la función $\mu_W : \mathcal{I}_W \mapsto$ seq $\mathbb{N}$ que se define como*

$$\mu_H(hp) = \langle\, (60\,h_1+m_1,\, 60\,h_2+m_2-1), \ldots, (60\,h_{m-1}+\, m_{m-1},\, 60\,h_m+m_m\, -\, 1)\,\rangle$$

De este modo

$$\mu_H(8{:}00..14{:}00,16{:}00..18{:}00) = \langle\, (480,839),(960,1079)\rangle$$

Como puede observarse, las normalización consiste en obtener los minutos equivalentes a una hora expresada en formato hh:mm. Obsérvese también que no es necesario añadir criterios de validación adicionales, pues la pertenencia a $\mathcal{I}_H$ garantiza que todos los instantes indicados en el intervalo son correctos.

Facilidades sintácticas

Reducir un minuto al limite superior de un intervalo horario facilita la especificación. Por ejemplo, supongamos un requisito definido como

TTF > 10 **from** 8:30..10:00 **and** TTF > 15 **from** 10:00..18:00

la interpretación intuitiva más probable es que a las 10:00 la condición que debe satisfacer la oferta es TTF > 15. Sin embargo, desde un punto de vista matemático no existe diferencia entre el instante temporal correspondiente a las 10:00 de r1 y el de r2. Esta ambigüedad puede ser fácilmente resuelta si se reescribe el requisito como

TTF > 10 **from** 8:30..09:59 **and** TTF > 15 **from** 10:00..18:00

salta a la vista que esta solución no es precisamente cómoda para quién tenga que especificar, por lo que hemos decidido que la lleve a cabo el compilador de QRL.

Definición 6.3.4 (Normalización de un patrón semanal) *Dada una secuencia de intervalos semanales* $wp \in \mathcal{I}_W$ *su normalización queda determinada por la función* $\mu_W : \mathcal{I}_W \mapsto$ *seq* $\mathbb{N}$ *que se define como*

$$\mu_W(wp) = \langle\, \omega(w_1)..\omega(w_2),\ldots,\omega(w_{n-1})..\omega(w_n)\,\rangle$$

donde $\omega(w_i)..\omega(w_j)$ *representa a la subsecuencia de días entre* w_i *y* w_j*, y donde*

$$\omega = \{MON \mapsto 0,\ TUE \mapsto 1,\ldots,\ SAT \mapsto 5,\ SUN \mapsto 6\}$$

de este modo

$$\begin{array}{ll} \mu_H(\ \mathsf{SAT..SUN}) = & \langle 5,6\rangle \\ \mu_H(\ \mathsf{MON..FRI, SUN..SUN}) = & \langle 0,1,2,3,4,6\rangle \\ \mu_H(\ \mathsf{SUN..SAT}) = & \bot \end{array}$$

Obsérvese que de acuerdo con la semántica asociada a una secuencia de intervalos, el patrón SUN..SAT no es válido pues el valor asociado al domingo es mayor que el del sábado.

Definición 6.3.5 (Normalización de un patrón de fechas) *Dada una secuencia de intervalos de fechas* $dp \in \mathcal{I}_D$ *su normalización queda determinada por la función* $\mu_D : \mathcal{I}_D \mapsto \mathcal{I}_{\mathbb{N}}$ *que se define como*

$$\mu_D(dp) = \langle\, (\eta(d_1), \eta(d_2)+1439),\ldots,(\eta(d_{q-1}), \eta(d_q)+1439)\,\rangle$$

donde la función $\eta : \mathcal{D}ate \mapsto \mathbb{N}$ *devuelve el número de minutos transcurridos entre las 00:00 del día 1 de enero de 2001 y las 00:00 de la fecha sobre la que se aplica.*

de este modo

μ_D(01/JAN/2001..31/JUL/2001,
01/SEP/2001..31/DEC/2001) = ⟨ (0, 305279), (349920, 525599) ⟩

Facilidades sintácticas

Obsérvese que se incrementa en 1439 minutos la fecha final de cada intervalo, pues se entiende que la fecha final de un intervalo está incluida en el intervalo. Dado que en $\text{QRL}_{\heartsuit}$ el quantum mínimo de tiempo es el minuto, el instante final de un día es el correspondiente a las 23:59.

Definición 6.3.6 (Normalización de un período de vigencia básico) *Dado un periodo de vigencia VP con una estructura del tipo* **from** *hp* **on** *wp* **during** *dp, denotaremos su proceso de normalización con la función $\mu : \mathcal{I}_H \times \mathcal{I}_W \times \mathcal{I}_D \mapsto 2^{\mathcal{C}_\tau}$ que se define como*

$$\mu(hp, wp, dp) = \bigcup_{i=1}^{n} (\tau \geq begin_i \textbf{ and } \tau \leq end_i)$$

donde

n *indica la cardinalidad del conjunto resultante de la normalización, la cual viene dada por la expresión* $|\overline{hp}| * |\overline{wp}| * \sum_{i=1}^{|\overline{dp}|} weeksIn(\overline{dp}(i), \overline{wp})$, *siendo* **weeksIn** *una función que devuelve el número de veces que el patrón semanal* **wp** *está contenido en el período de tiempo que va desde* $\overline{dp}$*(i).***begin** *hasta* $\overline{dp}$*(i).***end**.

y los valores de $begin_i$ *y* end_i *para el intervalo de fechas j–ésimo, el k–ésimo día del patrón semanal, de la semana w-ésima y el intervalo l–ésimo del patrón horario vienen dado por las expresiones*

$$begin_i = \overline{dp}(j).begin + 10080 * (w - 1) + 1440 * \overline{wp}(k) + \overline{hp}(l).begin$$
$$end_i = \overline{dp}(j).begin + 10080 * (w - 1) + 1440 * \overline{wp}(k) + \overline{hp}(l).end$$

donde 10080 *es el número de minutos que tiene una semana,* 1440 *el número de minutos que tiene un día,* w *representa al contador del número de veces que está contenido el patrón semanal en el intervalo de fechas y* $\overline{hp}, \overline{wp}, \overline{dp}$ *representan a las normalizaciones de los patrones horarios, semanales y de fechas.*

Ejemplo 6.3.1 *La normalización del período* **from** *8..14* **on** *MON..FRI* **during** *01/JAN/2001..29/JUN/2001 se corresponde con el siguiente conjunto*

μ(8:00..14:00, MON..FRI, 01/JAN/2001..12/JAN/2001) =
{
($\tau \geq 0 + 0 + 0 + 480, \tau \leq 0 + 0 + 0 + 839$), // (480, 839)
($\tau \geq 0 + 0 + 1440 + 480, \tau \leq 0 + 0 + 1440 + 839$), // (1920, 2279)
($\tau \geq 0 + 0 + 2880 + 480, \tau \leq 0 + 0 + 2880 + 839$), // (3360, 3719)
($\tau \geq 0 + 0 + 4320 + 480, \tau \leq 0 + 0 + 4320 + 839$), // (4800, 5159)
($\tau \geq 0 + 0 + 4320 + 480, \tau \leq 0 + 0 + 4320 + 839$), // (4800, 5159)
($\tau \geq 0 + 0 + 5760 + 480, \tau \leq 0 + 0 + 5760 + 839$), // (6240, 6599)

($\tau \geq 0 + 10080 + 0 + 480, \tau \leq 0 + 10080 + 0 + 839$), // (10560, 10919)
($\tau \geq 0 + 10080 + 1440 + 480, \tau \leq 0 + 10080 + 1440 + 839$), // (12000, 12359)

...
($\tau \geq 0 + 10080 + 0 + 480, \tau \leq 0 + 10080 + 0 + 839$), // (10560, 10919)
($\tau \geq 0 + 10080 + 1440 + 480, \tau \leq 0 + 10080 + 1440 + 839$), // (12000, 12359)

}

Este esquema de normalización asume que el día de la semana correspondiente al primer y último día de cada intervalo de fechas se corresponde con el primer y último día del patrón semanal, esto es, que

$$dayWeek(d_i) = w_0 \wedge dayWeek(d_{i+1}) = w_n$$

De este modo, cada intervalo de fechas debe contener un número exacto de patrones semanales. Por ejemplo, entre el 1 de Enero de 2001 y el 12 de enero de 2001 transcurren dos patrones semanales del tipo MON..FRI.

Criterios de validez
Exigir que los patrones horarios sean secuencias de intervalos ordenados y disyuntos facilita la normalización ya que garantiza los criterios de validación que en QRL se exige al período de vigencia asociados a un requisito (definición §6.2.17). Nos referimos a la consistencia y a la ambigüedad. Por ejemplo, el período especificado como

from 8..14,18..16 **on** wp **during** dp

es inconsistente pues con independencia de los patrones semanales y de fechas, el segundo intervalo del patrón horario indica la hora inicial es

posterior a la hora final. Errores de este tipo son fáciles de detectar tanto antes como después de la normalización. Antes de la normalización basta con comprobar que el intervalo es válido. Tras la normalización basta con comprobar que la restricción correspondiente es satisfactible.

Los errores de ambigüedad debidos al solapamiento entre intervalos también se evitan con la actual definición de secuencia de intervalos. Por ejemplo, el período especificado como *Situaciones de ambigüedad*

from 8..14,12..16 **on** wp **during** dp

es ambiguo debido al solapamiento existente entre ambos intervalos entre las 12:00 y las 14:00. Esta situación se detecta al comprobar la validez de la secuencia ya que el limite inferior de un intervalo no puede ser inferior al de un intervalo anterior en la secuencia.

Con idea de facilitar la lectura, en aquellas ocasiones en las que la especificación de un período de vigencia en $\text{QRL}_{\heartsuit}$ resulte excesivamente verbosa utilizaremos la notación de QRL. *Abuso de notación*

6.3.3.3. Normalización de un período de vigencia extendido

Las posibilidades para expresar períodos de vigencia de QRL van más allá del formato básico **from** hp **on** wp **during** dp, ya que QRL permite realizar sumas y diferencias entre períodos de vigencia.

La suma (operación **and**) entre dos períodos de vigencia da como resultado un nuevo período de vigencia que contiene a ambos períodos. Esta operación permite especificar de manera estructurada y flexible períodos de vigencia discontinuos en el tiempo.

Definición 6.3.7 (Normalización de la suma de períodos) *Sean VP y VP' dos períodos de vigencia con una estructura del tipo **from** hp **on** wp **during** dp, denotaremos la normalización de su suma como $\mu : (\mathcal{I}_H \times \mathcal{I}_W \times \mathcal{I}_D)^2 \mapsto 2^{\mathcal{C}^\tau}$ que se define como*

$$\mu(VP \textbf{ and } VP') = \mu(hp, wp, dp) \cup \mu(hp', wp', dp')$$

La suma de dos períodos de vigencia se considera válido siempre y cuando ambos períodos son válidos y disyuntos entre sí (esta última condición garantiza la ausencia de ambigüedad). Ambas condiciones son recogidas por el predicado $valid : (\mathcal{I}_H \times \mathcal{I}_W \times \mathcal{I}_D)^2$ que se define como

$$valid(VP \textbf{ and } VP', M) \iff valid(\mu(VP) \cup \mu(VP'), M)$$

donde el predicado $valid : 2^{\mathcal{C}^\tau} \times \mathcal{M}$ se define como

$$valid(\{t_i\}_{i=1}^n, M) \iff \forall i, j \in [1, n] \cdot \begin{cases} sat(t_i, M) = true \wedge \\ (i \neq j \Rightarrow \\ sat(t_i \textbf{ and } t_j, M) = false) \end{cases}$$

Ejemplo 6.3.2 *Sea VP = **from** 10..20 **on** MON..FRI **during** 2001 31/DEC/2001 **and from** 10..15 **on** FRI..SUN **during** 2001, es fácil comprobar que VP no es válido pues no es posible determinar cual es la franja horaria para el viernes: si de las 10:00 a las 20:00 o de las 10:00 a las 15:00.*

La diferencia (operación **except**) entre dos períodos de vigencia da como resultado un nuevo período de vigencia que representa los períodos de tiempo para los que está definido un período y no está definido otro. Esta operación permite eliminar de manera selectiva intervalos de un determinado periodo y es evidente que no es conmutativa.

Definición 6.3.8 (Normalización de la diferencia de períodos) *Sean dos períodos de vigencia $VP, VP' \in \mathcal{VP}$ denotaremos la normalización de la diferencia mediante la función $\mu_\epsilon : \mathcal{VP} \times \mathcal{VP} \mapsto 2^{\mathcal{C}^\tau}$ que se define como*

$$\mu_\epsilon(VP, VP') = \bigcup_{\substack{t_i \in \mu(VP) \\ t_j \in \mu(VP')}} \epsilon(t_i . t'_j)$$

donde la función $\epsilon : \overline{\mathcal{C}^\tau} \times \overline{\mathcal{C}^\tau} \mapsto 2^{\overline{\mathcal{C}^\tau}}$ se define como

$$\epsilon(t_i, t'_j) = \begin{cases} \{ \ \tau \geq b_l \textbf{ and } \tau \leq b'_l, \\ \quad \tau > b'_h \textbf{ and } \tau \leq b_h \ \} & si\ sol(t_i, M) \supsetneq sol(t'_j, M) \\ \\ \emptyset & eoc \end{cases}$$

Obsérvese que se ha utilizado la definición de restricción temporal normalizada presentada en la definición §6.2.16 (página 174).

Ejemplo 6.3.3 *Sea el período VP representado como* **from** *10..20 y el período VP'* **from** *15..18 se tiene que*

$$\begin{aligned}\mu_\epsilon(VP, VP') &= \epsilon(\tau \geq 10 \,\textbf{and}\, \tau \leq 20, \tau \geq 15 \,\textbf{and}\, \tau \leq 18) \\ &= \{\tau \geq 10 \,\textbf{and}\, \tau < 15, \tau > 18 \,\textbf{and}\, \tau \leq 20\}\end{aligned}$$

Sea el período VP representado como **from** *3..5 y el período VP'* **from** *3..5 se tiene que*

$$\begin{aligned}\mu_\epsilon(VP, VP') &= \epsilon(\tau \geq 3 \,\textbf{and}\, \tau \leq 5, \tau \geq 3 \,\textbf{and}\, \tau \leq 5) \\ &= \emptyset\end{aligned}$$

pues las soluciones de ambos períodos son las mismas.

La diferencia entre dos períodos de vigencia se considera válida si y sólo si todos los intervalos del segundo período de vigencia están contenidos en los intervalos del primero y el resultado de la normalización no es vacío (ambos períodos serían idénticos). Ambas condiciones son recogidas por el predicado $validExcept : \mathcal{VP} \times \mathcal{VP}$ que se define como

$$valid(VP \,\textbf{except}\, VP') \iff \begin{cases} \forall t'_j, \exists t_i \cdot t_i \rightarrow t'_j \wedge \\ \mu_\epsilon(VP, VP') \neq \emptyset \end{cases}$$

Normalización de la zona horaria

Un último aspecto que debe tenerse en cuenta es que QRL$_\heartsuit$ asume que todo instante temporal está referido a la zona horaria universal de referencia, es decir, la hora UTC o GMT+0. La operación a realizar es muy sencilla, a partir de la especificación de la zona horaria que tiene el formato **GMT** $\pm$ hh:mm, se obtiene el número de minutos de desfase. Este desfase se sumará o restará a todos los instantes temporales calculados durante la normalización.

6.3.3.4. Períodos globales, individuales y con nombre

QRL$_\heartsuit$ no tiene períodos de vigencia globales

En QRL$_\heartsuit$ no se contempla la posibilidad de asociar un período de vigencia a un documento, por lo que es necesario reflejar esta información en los períodos de vigencia de cada uno de los requisitos. El procedimiento a seguir es muy simple, si un requisito tiene asociado un período de vigencia tan sólo se comprueba que dicho período está contenido en el período

global del documento. En caso de no tener asociado ningún período se le asocia el período global.

QRL$_\heartsuit$ no tiene períodos con nombre

QRL$_\heartsuit$ tampoco permite el uso de períodos con nombre por lo que durante la normalización de dichos elementos se generará un mapa con la familia de restricciones equivalente al período referido. Dicho mapa será utilizado posteriormente durante la normalización de los requisitos.

En la figura §A.2 del anexo §A se describen las facilidades sintácticas que posee QRL para expresar períodos de vigencia.

6.3.4. Sección de condiciones

Para describir la normalización de una sección de condiciones básicas es necesario definir previamente la normalización de una sección de requisitos, de un requisito y de un requisito elemental.

Definición 6.3.9 (Requisito elemental) *Definimos un requisito elemental como un elemento del conjunto $\mathcal{R}^e$ que se define como $\mathcal{R}^e \triangleq \mathcal{C}^{\mathcal{Q}} \times \mathcal{VP}$. Un requisito elemental $r \in \mathcal{R}^e$ también lo denotaremos como el par (q, vp).*

Ejemplo 6.3.4 *El requisito AVAIL > 98 **from** 8..16 es elemental. Sin embargo, el requisito AVAIL > 98 **from** 8..16 **and** AVAIL > 90 **from** 16..20 no lo es.*

Definición 6.3.10 (Normalización de un requisito elemental) *Dado un requisito elemental $r \in \mathcal{R}^e$ en un contexto ctx, denotaremos su normalización con la función $\mu_{\mathcal{R}^e} : \mathcal{R}^e \times \mathcal{ID} \mapsto \overline{\mathcal{R}}$ que se define como*

$$\mu_{\mathcal{R}^e}((q, vp), ctx) = \mu_{\mathcal{R}^e}((q, \mu(vp)), ctx) = \{(q^{(ctx)}, t_i)\}_{i=1}^{n}$$

donde $q^{(ctx)}$ representa a la restricción resultante de sustituir los identificadores de medidas de q por identificadores prefijados por el identificador ctx.

Ejemplo 6.3.5 *El resultado de la normalización del requisito elemental*

*AVAIL > 98 **from** 8..16,18..20 con un contexto $ctx = p1$ es*

*{ (p1.AVAIL > 98, $\tau \geq$ 8 **and** $\tau \leq$ 16), (p1.AVAIL > 98, $\tau \geq$ 16 **and** $\tau \leq$ 20) }*

En la descripción informal de QRL realizada en el capítulo §5 no apareció el concepto de requisito elemental, tan sólo hablamos de requisitos. De hecho, un requisito elemental sólo es un concepto auxiliar para facilitar la definición de la semántica de un requisito. De hecho, a partir de este concepto y de la operación de unión entre requisitos normalizados, podemos definir formalmente un requisito y su normalización.

Definición 6.3.11 (Requisito) *Definimos un requisito como un elemento del conjunto $\mathcal{R}$ que se define como $\mathcal{R} \triangleq 2^{\mathcal{R}^e}$.*

Ejemplo 6.3.6 *Un ejemplo de requisito compuesto por dos requisitos elementales es* { *AVAIL* > *98* **from** *8..16, AVAIL* > *90* **from** *16..20* }

Definición 6.3.12 (Normalización de un requisito) *Dado un requisito $R \in \mathcal{R}$ en un contexto ctx, denotaremos su normalización con la función $\mu_{\mathcal{R}} : \mathcal{R} \times \mathcal{ID} \mapsto \overline{\mathcal{R}}$ que se define como*

$$\mu_{\mathcal{R}}(\{r_i\}_{i=1}^{n}, ctx) = \oplus_{i=1}^{n} \mu_{\mathcal{R}^e}(r_i, ctx)$$

Ejemplo 6.3.7 *El resultado de la normalización del requisito*

{ *AVAIL* > *98* **from** *8..16, AVAIL* > *90* **from** *16..20* } *con un contexto ctx= "p1" es*

{ *(p1.AVAIL* > *98,* $\tau \geq$ *8* **and** $\tau \leq$ *16), (p1.AVAIL* > *90,* $\tau \geq$ *16* **and** $\tau \leq$ *20)* }

Definición 6.3.13 (Sección de requisitos) *Definimos una sección de requisitos como un elemento del conjunto $\mathcal{RS}$ que se define como $\mathcal{RS} \triangleq \mathcal{ID}_{\mathcal{R}} \mapsto \mathcal{R}$*

Ejemplo 6.3.8 *El siguiente fragmento de una sección de condiciones es una sección de requisitos*

```
r0: AVAIL > 98 from 8..16;
r1: AVAIL > 90 from 16..20;
```

Definición 6.3.14 (Normalización de una sección de requisitos) *Sea una sección de requisitos $RS \in \mathcal{RS}$ en un contexto ctx, denotaremos su normalización con la función $\mu_{\mathcal{RS}} : \mathcal{RS} \times \mathcal{ID} \mapsto \overline{\mathcal{R}}$ que se define como*

$$\mu_{\mathcal{RS}}(\{idr_i \mapsto r_i\}_{i=1}^{n}, ctx) = \oplus_{i=1}^{n} \mu_{\mathcal{R}}(r_i, ctx)$$

Por tanto, en la normalización de una sección se pierde toda referencia al identificador de los requisitos cuya única función es facilitar la especificación de cláusulas de negociación.

Ejemplo 6.3.9 *El resultado de la normalización de la sección de requisitos*

r0: { AVAIL > 98 **from** 8..16, AVAIL > 90 **from** 16..20 }
r1: LANG $\supseteq$ { SPA, ENG};

con un contexto ctx= "p1" es

{ (p1.LANG $\supseteq$ { c1.SPA, c1.ENG}, $\tau \geq 0$ **and** $\tau \leq 479$),
(p1.AVAIL > 98 **and** p1.LANG $\supseteq$ { c1.SPA, c1.ENG}, $\tau \geq 480$ **and** $\tau \leq 959$),
(p1.AVAIL > 90 **and** p1.LANG $\supseteq$ { c1.SPA, c1.ENG}, $\tau \geq 960$ **and** $\tau \leq 1199$),
(p1.LANG $\supseteq$ { c1.SPA, c1.ENG}, $\tau \geq 1200$ **and** $\tau \leq 1439$),
}

A partir de estas definiciones, es posible definir una sección de condiciones así como su proceso de normalización.

Definición 6.3.15 (Sección de condiciones básicas) *Definimos una sección de condiciones básicas como un elemento del conjunto $\mathcal{BC}$ que se define como*

$$\mathcal{BC} \triangleq \mathcal{RS} \times \mathcal{RS} \times \mathcal{SP}$$

Donde $\mathcal{SP}$ hace referencia a la especificación de requisitos para los subproductos de un producto y se define como $\mathcal{SP} \triangleq \mathcal{ID}_{SP} \mapsto \mathcal{RS}$.

La figura §6.6.a muestra una sección de condiciones y su normalización a $\text{QRL}_{\heartsuit}$, que como puede observarse da lugar a una sección de requisitos normalizados.

Definición 6.3.16 (Normalización de condiciones básicas) *Dada una sección de condiciones básicas $BC \in \mathcal{BC}$ en un contexto ctx, denotaremos su normalización con la función $\mu_{\mathcal{BC}} : \mathcal{BC} \times \mathcal{ID} \mapsto \overline{\mathcal{RS}}$ que se define como*

```
using com.acme.stdMOWS;
products { p1: org.W3C.IVideoServer= Play, Stop; }
valid { }
conditions {
 global { r1: COST ≤ 1;}
 for all { r2: TTF ≥ 90; };
 for Play { r3: AVAIL ≥ 98 };
}
```

a) Sección de condiciones en QRL

```
requirements {
 dr0: { (p1.COST ≤ 1, μ(...)),
        (p1.Play.TTF ≥ 90, ),
        (p1.Stop.TTF ≥ 90, ),
        (p1.Play.AVAIL ≥ 98, ),
      }
}
```

b) Sección de requisitos equivalente en $\text{QRL}_{\heartsuit}$

Figura 6.6: *Ejemplo de normalización de una sección de condiciones.*

$$\mu_{BC}(RS_g, RS_a, \{idSp_i \mapsto RS_i\}_{i=1}^{n}, ctx) =$$
$$\{idr0 \mapsto ($$
$$\mu_{\mathcal{RS}}(RS_g, ctx) \oplus$$
$$\oplus_{i=1}^{n} \mu_{\mathcal{RS}}(RS_a, ctx.idSp_i) \oplus$$
$$\oplus_{i=1}^{n} \mu_{\mathcal{RS}}(RS_i, ctx.idSp_i))$$
$$\}$$

6.3.5. Sección de negociación

Definición 6.3.17 (Cláusulas de compromisos) *Una sección de cláusulas de negociación por compromisos se define como un elemento del conjunto $\mathcal{N}_T$ que se define como*

$$\mathcal{N}_T \triangleq \mathcal{ID}_{\mathcal{N}} \mapsto \textbf{\textit{lose}}\mathcal{RS} \times \textbf{\textit{win}}\mathcal{RS}$$

Ejemplo 6.3.10 *El siguiente fragmento de especificación se corresponde con una sección de cláusulas de negociación por compromisos*

```
MasFiabilidad_MasDinero {
 lose { r1: COST ≤ 3;}
 win {r2: maturity.VERY_HIGH on PUNTA}
}
MenosFiabilidad_MenosDinero {
 lose {r2: maturity.MEDIUM on PUNTA}
 win {r1: COST ≤ 1.7;}
}
```

Definición 6.3.18 (Aplicación de un compromiso) *Denotamos la aplicación de una cláusula de negociación por compromisos $nt \in \mathcal{N}_T$ a una sección de condiciones básicas $BC \in \mathcal{BC}$ con la operación $\otimes : \mathcal{BC} \times \mathcal{N}_T \mapsto \mathcal{RS}$, que se define como*

$$BC \otimes (idnt \mapsto (L, W)) = \pi(BC) \circ L \circ W$$

Ejemplo 6.3.11 *Para la sección de condiciones básicas representada como*

```
conditions for P {
 global {
   r1: COST ≤ 2;
   r2: maturity.HIGH;
   r3: recoverability.MEDIUM;
 }
}
```

la aplicación de la cláusula MenosFiabilidad_MenosDinero *del ejemplo anterior da como resultado la sección de requisitos*

```
r1: COST ≤ 1.7;
r2: maturity.MEDIUM;
r3: recoverability.medium;
```

Dada una sección de cláusulas de compromisos $NT \in \mathcal{N}_T$, unas condiciones básicas $BC \in \mathcal{BC}$ y una sección de preferencias $P \in \mathcal{P}$ diremos que NT es válida si y sólo si se satisface el predicado $valid : \mathcal{N}_T \times \mathcal{BC} \times \mathcal{P}$ que se define como:

$$valid(\{idnt_i \mapsto (L_i, W_i)\}_{i=1}^{n}, BC, P) \iff$$

$$\iff \forall i \in \{1, \ldots, n\} \cdot \begin{cases} (1)\, U(\pi(BC) \circ L_i, P) < U(\pi(BC), P) \wedge \\ (2)\, U(\pi(BC) \circ W_i, P) > U(\pi(BC), P) \wedge \\ (3)\, valid(\mu_{\mathcal{RS}}(\pi(BC) \otimes (L_i, W_i))) \wedge \\ (4)\ \mathrm{dom}\, L_i \cap \mathrm{dom}\, W_i = \emptyset \end{cases}$$

No estamos limitando las medidas que pueden aparecer en la cláusula de negociación ganadora, de este modo es posible añadir medidas. Tampoco estamos limitados las medidas de las cláusulas perdedoras, de ese modo se puede quitar una medida de la condición base.

Con la cuarta condición estamos exigiendo que las cláusulas modifiquen diferentes requisitos de las condiciones básicas.

Definición 6.3.19 (Cláusulas de renuncias) *Definimos una sección de cláusulas de renuncias como un elemento del conjunto $\mathcal{N}_R$ que se define como:*

$$\mathcal{N}_R \triangleq \mathcal{ID}_{\mathcal{N}} \mapsto \textbf{\textit{lose}}\mathcal{R}$$

Definición 6.3.20 (Aplicación de una renuncia) *Denotamos la operación de aplicar una cláusula de negociación por renuncias $nr \in \mathcal{N}_R$ a una sección de condiciones básicas $BC \in \mathcal{BC}$ como $BC \otimes nr$ y se define como*

$$BC \otimes L = BC \circ L$$

Dada una sección de cláusulas de renuncias $NR \in \mathcal{N}_R$, una sección de condiciones básicas $BC \in \mathcal{BC}$ y una sección de preferencias $P \in \mathcal{P}$ diremos que NR es válida si y sólo si se satisface el predicado $valid : \mathcal{N}_R \times \mathcal{BC} \times \mathcal{P}$ que se define como

$$valid(\{idnr_i \mapsto L_i\}_{i=1}^{n}, BC, P) \iff \forall i \in [1, n] \cdot \begin{cases} U(\pi(BC) \circ L_i, P) < U(\pi(BC), P) \\ valid(\mu_{\mathcal{RS}}(\pi(BC) \otimes L_i) \end{cases}$$

Definición 6.3.21 (Sección de negociación) *Definimos una sección de negociación como un elemento del conjunto $\mathcal{N}$ que se define como*

$$\mathcal{N} \triangleq \textbf{\textit{tradeoffs}}\,\mathcal{N}_T\, \textbf{\textit{renunciations}}\,\mathcal{N}_R$$

Una sección de negociación también la denotaremos como $\{nc\}_{i=1}^{n}$ donde $nc \in \mathcal{N}_T \cup \mathcal{N}_R$.

Definición 6.3.22 (Combinación de cláusulas) *Sean dos cláusulas de negociación $nc, nc' \in \mathcal{N}_T \cup \mathcal{N}_R$ denotamos su combinación como $nc \otimes nc'$ y la definimos como*

$$(L, W) \otimes (L', W') = (L \circ L', W \circ W')$$
$$(L, W) \otimes L' = (L \circ L', W)$$
$$L \otimes (L', W') = (L \circ L', W')$$
$$L \otimes L' = L \circ L'$$

La combinación de cláusulas es asociativa por la izquierda y no conmutativa, verificándose que

$$nc_1 \otimes nc_2 \otimes nc_3 = (nc_1 \otimes nc_2) \otimes nc_3$$

Definición 6.3.23 (Clausura de una sección de negociación) *Dada una sección de negociación $N \in \mathcal{N}$ denotamos su clausura como N^* y la definimos como*

$$N^* = \{P(A) \mid A \in 2^N\}$$

donde $P(A)$ devuelve el conjunto de todas las permutaciones que se pueden conseguir a partir del conjunto A, es to es,

$$P(\{a_1, \ldots, a_n\}) = \{\langle a'_1, \ldots, a'_n \rangle \mid a'_1 \neq \ldots \neq a'_n \wedge \{a'_1, \ldots, a'_n\} = \{a_1, \ldots, a_n\}\}$$

Definición 6.3.24 (Aplicación de una sección de negociación) *Sea $N \in \mathcal{N}$ una sección una sección de cláusulas de negociación tal que $N = \{nc_i\}_{i=1}^n$, definimos el resultado de aplicar N sobre una sección de condiciones básicas $BC \in \mathcal{BC}$ como*

$$BC \otimes N = BC \otimes \bigotimes_{i=1}^{n} nc_i$$

Diremos que la aplicación de una combinación es válida cuando se verifique el predicado $valid : \mathcal{BC} \times \mathcal{N}$ que se define como

$$valid(BC, \{nc_i\}_{i=1}^n) \iff (BC \otimes \bigotimes_{i=1}^{n} nc_i) \rightarrow \bigotimes_{i=1}^{n} nc_i\}$$

Definición 6.3.25 (Clausura de una sección de condiciones) *Dada una sección de condiciones básicas $BC \in \mathcal{BC}$, denotamos su clausura en relación a una sección de negociación $N \in \mathcal{N}$ como BC_N^* y la definimos como*

$$BC_N^* = \{\{idr_i \mapsto \mu_{\mathcal{RS}}(BC \otimes N_i)\} \mid valid(BC, N_i) \wedge N_i \in \mathcal{N}^*\}$$

Definición 6.3.26 (Normalización de una sección de negociación) *Sea N una sección de negociación y BC una sección de condiciones básicas, denotaremos su normalización con la función $\mu_{\mathcal{N}} : \mathcal{N} \times \mathcal{BC} \mapsto \overline{\mathcal{RS}}$ que se define como*

$$\mu_{\mathcal{N}}(N, BC) = \{idr_i \mapsto BC_N^*(idr_i)\}$$

donde $\mathrm{dom}(BC_N^*) = \{idr_1, idr_2, \ldots, idr_n\}$

Parte IV

Conclusiones y trabajo futuro

Capítulo 7

Trabajo futuro

La sabiduría consiste no sólo en ver lo que tienes ante ti, sino en prever lo que va a venir.

TERENCIO

Durante el desarrollo de esta tesis hemos tratado problemas y revisado trabajos procedentes de varias áreas de investigación, lo que nos ha proporcionado un ángulo de visión muy amplio sobre los requisitos de calidad. Prueba de ello es la diversidad de aplicaciones de los resultados obtenidos en esta tesis. Además de estas aplicaciones, en este capítulo también indicamos algunas extensiones de la propuesta presentada en esta memoria.

7.1. Aplicaciones de los resultados

Las aplicaciones de los resultados obtenidos en esta tesis están agrupadas en cuatro grandes áreas: ingeniería de requisitos, diseño arquitectónico, plataformas de ejecución y docencia.

7.1.1. Ingeniería de requisitos

La verificación y la negociación de requisitos son dos actividades en las que el uso de QRL puede resultar muy útil.

7.1.1.1. Verificación de requisitos

La posibilidad que ofrece QRL para comprobar automáticamente la consistencia de una especificación de requisitos de calidad expresada en lenguaje natural semiestructurado puede ser utilizada durante la verificación de documentos de requisitos. En este sentido, las propuestas presentadas en [Durán *et al.* 2001, Durán *et al.* 2002b, Durán *et al.* 2002a] para verificar automáticamente algunas propiedades sobre la calidad de un documento pueden ser extendidas para comprobar la ausencia de contradicciones en los requisitos de calidad.

7.1.1.2. Negociación automática de requisitos

La especificación de cláusulas de negociación puede ser empleada para resolver conflictos durante la negociación de requisitos en proyectos con múltiples participantes. En [Ruiz-Cortés *et al.* 2002b] se propone una extensión del modelo de negociación de requisitos Win–Win que tiene como principal objetivo aumentar el grado de automatización de la detección y resolución de conflictos de dicho modelo.

Esta propuesta tiene su punto más débil en la necesidad de elicitar anticipadamente los compromisos que están dispuestos a asumir los participantes, labor para la que, por regla general, no suelen estar muy receptivos. No obstante, creemos que existen elementos que hacen creible su viabilidad. Por un parte, el recelo inicial de los participantes tenderá a desaparecer si comprueba que por un esfuerzo extra durante la especificación de requisitos se evita largas sesiones de negociación. Por otra parte, el modelo no obliga a que todos los participantes den a conocer todos sus compromisos, pues para generar automáticamente una opción basta con las condiciones iniciales de los participantes y con que exista uno que indique anticipadamente un compromiso ya es posible generar una opción alternativa. Es decir, se trata de un modelo donde el grado de automatización es controlado por los participantes.

7.1.2. Diseño arquitectónico

Perhaps the reason for such slow progress in the development and evolution of software systems is that we have trained carpenters and contractors, but not architects.

[PERRY Y WOLF 1992]

Es conocido que aproximadamente un 60 % [1] del coste total de un proyecto se dedica a la fase de mantenimiento y mayoritariamente en mejoras del sistema. Gran parte de estas mejoras suelen estar referidas a requisitos no funcionales: conseguir un menor tiempo de respuesta, adaptación a un nuevo sistema operativo, ofrecer seguridad para transacciones electrónicas, etc. Llevar a cabo estas mejoras implica realizar actividades relacionadas con el diseño: diseño de un nuevo subsistema y de su integración con el sistema actual, rediseño de la arquitectura del sistema actual, etcétera..

Algunas de las preguntas que con mayor frecuencia son planteadas en este tipo de actividades son las siguientes:

- ¿Qué arquitectura le doy a un determinado subsistema?
- ¿Cómo adapto una aplicación X para que pueda ser utilizada en una Intranet? ¿Cuál es la solución más económica? ¿Y la más rápida?
- ¿Qué rendimiento está ofreciendo actualmente mi sistema?
- ¿Podrá soportar el sistema el acceso simultáneo de 1.000 clientes?, ¿podrá soportar el sistema el acceso de 2.000 clientes duplicando el *hardware*?
- ¿Cuáles son los "agujeros" de seguridad de mi sistema?

Todos sabemos que estas preguntas no tienen una fácil respuesta y que por ahora no es posible construir la herramienta que permita obtenerlas para el caso general, pues la mayoría de las actividades relacionadas con el diseño de la arquitectura de un sistema software siguen teniendo una gran componente intuitiva.

[1] Con el siguiente desglose: 12 % de adaptación, 12 % de corrección y 36 % de mejora.

Algunas de estas preguntas son abordadas en la tesis doctoral [Tekinerdogan 2000][2] en la que se discuten los diferentes enfoques que pueden seguirse durante el diseño de una arquitectura. Uno de los enfoques discutidos es el conocido como "diseño arquitectónico guiado por requisitos no funcionales", sobre el que los autores afirman que no resulta viable por la imposibilidad de formalizar este tipo de requisitos debido fundamentalmente a su su carácter subjetivo y relativo.

Por su parte, el grupo de investigación sobre requisitos de calidad de la Universidad de Toronto liderados por el profesor Mylopoulos, entiende que este enfoque si resulta viable y han centrado parte de su investigación en este problema [Mylopoulos *et al.* 1992, Chung *et al.* 2000].

En nuestra opinión, los resultados de nuestra tesis nos acercan a la tesis del grupo de Toronto y nos distancian de la de Tekinerdogan y Aksit. No obstante, creemos que algunas limitaciones del enfoque cualitativo que emplea el grupo de Toronto podrían ser evitadas utilizando algunas de las aportaciones de nuestro trabajo, sobre todo, si tenemos en cuenta que para capturar la flexibilidad de un requisito no funcional no es necesario recurrir a técnicas cualitativas.

A continuación comentamos de manera muy general cómo puede ser empleado QRL para extender UML y para extender las actuales propuestas para transformar arquitecturas.

7.1.2.1. Extendiendo UML con QRL

QRL proporciona el grado de expresividad, abstracción y formalidad necesarios para dotar de una semántica precisa a algunos de los elementos empleados en los diagramas de clases, componentes y de despliegue de UML. En el caso de los diagramas de despliegue y de componentes, el uso de etiquetas (*tags*) para añadir información que puede ser modelada como atributos de calidad puede ser dotado de una semántica precisa. Además, dicha información puede ser empleada para hacer comprobaciones de consistencia y de conformidad.

En esta línea existe una tesis doctoral en curso [Martín-Díaz 2002] que tiene entre sus principales objetivos formalizar la vista de componentes y de despliegue de UML a fin de poder aumentar el grado de automatización ciertas actividades de interés para el diseño arquitectónico: evaluar

[2]Dirigida por M. Aksit.

la utilidad que ofrece una alternativa arquitectónica, comprobar la conformidad de una alternativa respecto de las condiciones que debe cumplir, elegir la mejor alternativa arquitectónica de entre un conjunto de candidatas, etcétera. De este modo, acercamos UML a las posibilidades que deben ofrecer los lenguajes formales de descripción de arquitecturas [Medvidovic y Taylor 2000].

QRL también puede ser empleado para extender la semántica de las interfaces y de las clases de UML. En este sentido, QRL puede ser empleada como notación para extender la propuesta de contrato presentada en [Galán 2000]. En dicho trabajo, se dotó de una semántica precisa a un subconjunto de UML y se propuso un modelo de contrato con dos dimensiones: la estática y la dinámica. Actualmente empezamos a trabajar para añadir la tercera dimensión: la calidad.

En la siguiente sección justificamos con un poco más de detalle nuestro interés por extender la noción de contrato.

7.1.2.2. Extender la noción de contrato

Los beneficios de la programación por contrato están fuera de toda duda. No obstante, aún no existe un acuerdo en cuanto al alcance y estructura de un contrato software. En nuestra opinión, un contrato determina las condiciones de la colaboración entre dos o más participantes, y consideramos que un contrato es válido cuando está libre de contradicciones y su cumplimiento garantiza la interoperabilidad entre los participantes, entendiendo por interoperabilidad a la capacidad de que varias entidades se comuniquen y cooperen entre sí, aunque estén implementadas con diferentes lenguajes y para diferentes entornos de ejecución [Wegner 1996, Vallecillo *et al.* 1999].

Según [Vallecillo *et al.* 2000], es posible identificar tres niveles de interoperabilidad: el nivel *sintáctico*, el *semántico* y el de *protocolo*. Estos niveles de interoperabilidad y, por ende, aspectos de un contrato software, son ampliamente aceptados, más aún si se considera el nivel de protocolo como un caso particular del nivel semántico.

En nuestra opinión, los aspectos de calidad vuelven a ser los grandes olvidados y compartimos la opinión expresada en [Szyperski 1998, pág.46] de que tan importante es conseguir determinar si un componente satisface las condiciones sintáctico–semánticas como determinar si el nivel de cali-

dad que proporciona es el adecuado. En [Szyperski 2000] vuelve a referirse a este problema y apunta una posible razón de dicha situación:

> *While we are still struggling to write down and verify precise specifications for functionality, our arsenal of approaches and tools thins when we try to venture out on quality attributes within contracts. That is not to say they are less important; they are just much harder to handle uniformly, and a great deal of work is still required to get it right.*

El padre de la programación por contrato no está fuera de este debate y en [Meyer 2000] propone su modelo de contrato, para el cual propone cuatro niveles: sintáctico, semántico, de rendimiento y de calidad de servicio.

Independientemente de la clasificación que se lleve a cabo, lo que en cualquier caso resulta evidente es la necesidad de tener en cuenta los aspectos relacionados con la calidad. En [Ruiz-Cortés *et al.* 2000b] presentamos una propuesta para comprobar la interoperabilidad en el contexto de los MOWS y planteamos la necesidad de incluir la calidad como un nivel más de interoperabilidad. Dicha propuesta fue recogida posteriormente en [Vallecillo *et al.* 2000].

Con los resultados obtenidos en esta tesis, estamos en condiciones de proponer un modelo de especificación de contrato software que incluye cláusulas de calidad y con la posibilidad de:

- Comprobar su validez (consistencia y satisfactibilidad).
- En el caso de que existan varias implementaciones que satisfacen un contrato, determinar cuál resulta óptima de acuerdo a unas preferencias determinadas.
- Indicar cláusulas de negociación.
- Indicar períodos de vigencia complejos que pueden especificarse de manera individual sobre cada una de las operaciones de la interfaz.

7.1.2.3. Transformaciones arquitectónicas

Por regla general, la evaluación de requisitos de calidad se suele llevar a cabo una vez desarrollado un producto software. Si la evaluación arroja

unos resultados insatisfactorios, el sistema se somete a un proceso de rediseño para mejorar el grado de satisfacción de los requisitos [Bosch y Molin 1999]. Rediseñar e implementar un sistema ya construido suele resultar bastante caro por lo que en pocas ocasiones se realiza.

En [Shaw y Garlan 1996, L. Bass y Kazman 1998] se proponen métodos para guiar el diseño de la arquitectura que intentan aumentar el grado de satisfacción de los requisitos de calidad. En [Bosch y Molin 1999] se proponen varias técnicas para transformar la arquitectura de un sistema que permiten aumentar el grado de satisfacción de uno o varios requisitos de calidad sin modificar el grado de satisfacción de los requisitos funcionales.

Una de las principales limitaciones de las técnicas de transformación planteadas por Bosch (las cuales pueden verse como una extrapolación de las ideas de refactorización de código [Fowler 1999, Crespo y Marqués 2001]) es la ausencia de cuantificación del impacto de cada transformación en el grado de satisfacción de cada requisito de calidad.

Medir el impacto que supone la aplicación de un patrón es uno de los problemas que quedan por resolver en el terreno de los patrones arquitectónicos y de diseño [Buschmann *et al.* 1996]. Hasta la fecha, no conocemos propuestas que aporten resultados de interés, no obstante, creemos que el disponer de una notación formal, expresiva y fácil de utilizar para expresar el impacto facilita esta tarea.

En este sentido, creemos que QRL reúne las características necesarias para especificar el impacto de una transformación arquitectónica. El primer paso en esta dirección lo hemos dado, y en [Martín *et al.* 2001a, Martín *et al.* 2001b] presentamos una primera propuesta para describir y evaluar alternativas arquitectónicas con QRL, donde una alternativa arquitectónica es el resultado de aplicar una transformación a una arquitectura de referencia. Actualmente estamos valorando el uso de cláusulas de negociación para obtener todas las alternativas arquitectónicas que pueden obtenerse a partir de una arquitectura de referencia y de un conjunto de transformaciones.

7.1.2.4. Middlewares sensibles a la calidad y con consciencia temporal

Las posibilidades de intermediación de los actuales middlewares son muy limitadas, al menos para el desarrollo de aplicaciones distribuidas sensibles a la calidad [Frølund y Koistinen 1999]. Salvo el caso del servi-

cio de intermediacion de CORBA (sección §3.3.1), no conocemos ningún otro middleware que ofrezca un servicio para intermediar sobre atributos definidos por el programador.

De acuerdo con la norma del servicio de intermediación de CORBA [OMG 2000], es posible sustituir el lenguaje para definir las condiciones y las ofertas, por lo que un trabajo bastante interesante puede ser la implementación de un servicio de intermediación compatible CORBA utilizando QRL. Esta extensión también está siendo valorada para ser introducida en Microsoft.NET, plataforma para la que tampoco conocemos hasta la fecha ningún servicio de intermediación.

El hecho de que QRL sea un lenguaje con un alto grado de consciencia temporal también lo convierten en idóneo para plataformas de ejecución de sistemas multimedia distribuidos, pues en dichas plataformas los objetos que han de instanciarse ante la petición de un clientes varían dependiendo de la franja horaria y de la carga del sistema en la que se realiza la petición. La propuesta más avanzada que conocemos para llevar a cabo estas tareas tiene un grado de consciencia temporal muy inferior al de QRL (sección §3.3.6).

7.1.3. Docencia

Un hecho del que nos sentimos especialmente satisfechos es la posibilidad de aplicar algunos de los resultados obtenidos en esta tesis a diferentes asignaturas de las adscritas a nuestra área de conocimiento. Algunas de las previsiones son las siguientes:

- **Ingeniería del software I.** Utilizar QRL como lenguaje de especificación de requisitos de calidad. Probablemente extendiendo la metodología desarrollada en [Durán 2000] y que está soportada por la herramienta REM[3].

- **Ingeniería del software II.** Utilizar QRL para extender los diagramas componentes y de despliegue de UML.

- **Métodos Formales en Ingeniería del Software.** Utilizar QRL para especificar formalmente el nivel de calidad de los contratos software.

[3]Disponible en http://klendathu.lsi.us.es/REM/.

Hasta la fecha, algunas de estas ideas se han utilizado durante el desarrollo de algunos proyectos final de carrera y la experiencia ha sido muy positiva. Una conclusión común a todos los alumnos que han utilizado QRL, es que ha aumentado su consciencia sobre la necesidad de especificar los requisitos de calidad, sobre todo, porque les proporciona información útil para tomar decisiones durante la fase de diseño.

7.2. Extensiones de nuestra propuesta

What is important is not so much the answers that are given,
but rather the questions that are asked.

WISDOM OF THE WEST, BERTRAND RUSSELL

El trabajo presentado en esta tesis doctoral admite algunas mejoras y extensiones que no pueden ser abordadas por la necesidad de imponer un límite temporal al trabajo de investigación. Para presentar estas mejoras y extensiones, utilizaremos los tres planos identificados en [Broy 2001]: el descriptivo, el formal y el metodológico.

La figura §7.1 ilustra a grandes en cuáles de estos planos hemos desarrollado nuestro trabajo. El principal resultado de nuestro trabajo es un lenguaje de especificación de requisitos de calidad que dispone de una semántica rigurosamente definida y que le proporciona unas capacidades superiores a la de los restantes lenguajes de calidad revisados en este trabajo. Por tanto, se trata de una aportación en el plano descriptivo y formal. Esta circunstancia lo refleja el sector circular sombreado de mayor tamaño.

Este resultado principal ha sido precedido por otros resultados previos en los tres planos, situación representada por los restantes sectores sombreados. Esperamos que en el futuro, las extensiones de nuestro trabajo nos permitan seguir avanzando incrementalmente sobre estos tres planos. A continuación indicamos algunas de las extensiones que hemos identificado hasta la fecha.

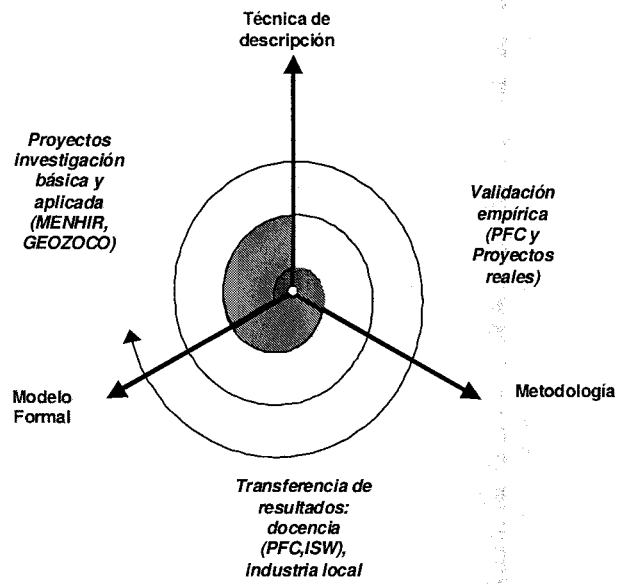

Figura 7.1: *Alcance de nuestra propuesta.*

7.2.1. Plano descriptivo

QRL cuenta con bastantes facilidades sintácticas y su grado de expresividad es bastante y alto, es por ello, que no hemos encontrado muchas extensiones en este sentido. En nuestra opinión, la más interesante esta relacionada con la definición de cláusulas de negociación, concretamente de cláusulas de compromisos. Se tratar de construir un repositorio de plantillas de cláusulas de negociación por compromisos que expresasen las condiciones de manera relativa. Estas cláusulas reflejarían, en la mayoría de las ocasiones, compromisos entre requisitos contradictorios: fiabilidad versus coste, fiabilidad versus rendimiento, etcétera. Podría verse como un caso particular de los conflictos registrados en QARCC (ver sección §3.3.4) pero expresados de manera precisa y cuantificable.

7.2.2. Plano formal

7.2.2.1. Dualidad inversa

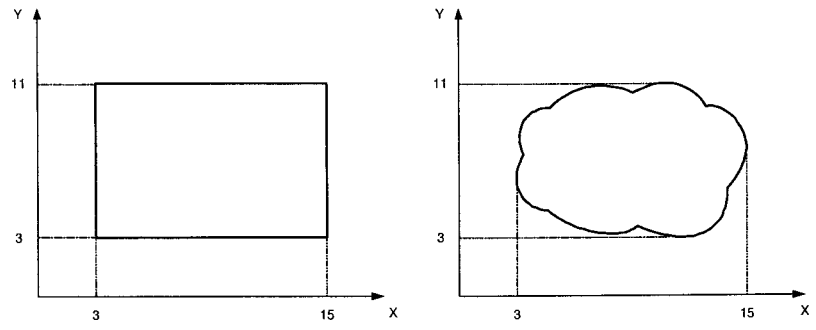

Figura 7.2: *Proyección de una restricción de doble variable.*

QRL ofrece mecanismos para traducir una especificación en lenguaje natural estructurado a un lenguaje formal, pero no para llevarla a cabo en el sentido inverso, esto es, traducir un requisito especificado formalmente a lenguaje natural. En este sentido, una primera técnica muy básica podría consistir en proyectar el requisito sobre cada uno de los atributos que lo constituyen. Una proyección sobre un atributo puede interpretarse como una frase y la proyección de un requisito completo la frase resultante de yuxtaponer la frase asociada a la proyección de cada atributo.

Por ejemplo, el requisito asociado a la restricción de la figura §7.2.a se traduciría como

"El atributo X debe estar entre 3 y 15 e el atributo Y entre 3 y 11".

Evidentemente, esta es sólo una idea inicial que necesita ser refinada, pues la proyección de un atributo no es siempre tan inmediata. Por ejemplo, la traducción del requisito representado por la figura §7.2.b se antoja mucho más compleja

7.2.2.2. Conformidad con emparejamientos parciales

En ocasiones, es posible que el proveedor no ofrezca información sobre el conjunto completo de atributos de calidad asociado a las condiciones del cliente. En tales casos decimos que existe un *emparejamiento parcial* entre las condiciones y la oferta.

	TTF.mean	TTF.variance	TTR.mean	Cost
WS1	90	40	30	15
WS2	92	35	25	16
WS3	85	–	25	17
...	...	...	...	18
WSn	100	5	10	20

Cuadro 7.1: *Repositorio de ofertas de calidad.*

Con la actual definición de conformidad, no es posible garantizar la conformidad de la oferta en situaciones de emparejamiento parcial. Sin embargo, existen situaciones en las que es posible garantizarla con el mismo grado de confianza que en el caso general y otras en las que es posible garantizarlo dentro de un margen de error conocido.

El primer grupo de situaciones se da cuando es posible determinar una cota superior o inferior de una medida a partir de medidas relacionadas. Supongamos que en las condiciones se especifica TTF.percentile90 $>$ 12 y que en la oferta no se ofrece información sobre esa medida, pero si sobre una medida estrechamente relacionada, p.e., TTF.percentile80 $>$ 14. Es fácil comprobar, que en este caso se puede garantizar la conformidad pues el percentil 90 siempre debe ser un valor mayor o igual al del percentil 80. Por tanto, el valor de una medida relacionada nos ha ayudado a determinar la conformidad sobre una medida inexistente.

El segundo grupo de situaciones se da cuando la única posibilidad de inferir el dato de una medida desconocida es mediante técnicas de inferencia "indirectas". Por ejemplo, las medidas de varias ofertas recogidas por la tabla §7.1, muestran una situación en la que no se dispone de la información relativa sobre la desviación del tiempo entre fallos (MTTF.variance) del servicio WS3. Esta información se podría inferir a partir de los atributos restantes con alguna técnica de inferencia: test estadístico, minería de datos, lógica difusa, etcétera. En caso de ser posible, estaríamos en condiciones de garantizar la conformidad, siempre, claro está, con el margen de error asociado a la técnica de inferencia utilizada.

7.2.3. Plano metodológico

En [Ciapessoni *et al.* 1999] se analiza el impacto de los métodos formales en la industria. En él se cita textualmente:

> *El problema con los métodos formales es que a pesar de que sean formales no llegan a ser métodos.* JOHN RUSHBY.

Compartimos esta opinión. Estamos convencidos de que no basta con contar con un lenguaje muy expresivo, ni con herramientas que le den soporte si sus usuarios potenciales no cuentan con una guía metodológica que les permita ser conscientes de todas las posibilidades que ofrece y los auxilie a la hora de aplicarlo a sus problemas de cada día.

En este trabajo nos hemos centrado en aspectos formales y descriptivos y hemos dejado de lado aspectos metodológicos. Por tanto, es en este terreno dónde más trabajo queda por hacer a corto plazo. Nosotros hemos identificado las siguientes líneas:

- Añadir a la metodología de especificación de requisitos propuesta en [Durán 2000] las actividades de creación de catálogos de atributos y de creación de documentos de requisitos; y desarrollar una nueva actividad para soportar la negociación de requisitos de calidad.
- Definir bancos de prueba para evaluar y comparar técnicas de especificación y negociación de requisitos. El catálogo de atributos de calidad relacionados con la WEB propuesto en [Olsina 1999] es extenso y complejo y sería un buen escenario de prueba para evaluar las posibilidades de expresar catálogos de atributos.
- Entre las tareas más tediosas y difíciles de realizar adecuadamente destacan la definición de funciones de utilidad y la obtención de pesos. En relación a los pesos, esta tarea puede ser asistida empleando técnicas empleadas en *minería de datos* para obtener la relevancia de los atributos de una base de datos [Kohavi y John 1997]. No obstante, con independencia del conjunto de técnicas que pueda emplearse para llevar a cabo estas tareas, deberían estar soportadas por una herramienta CASE.
- Estudiar las posibilidades de utilizar nuestra propuesta en el marco metodológico de la norma ISO 25000.

Parte V

Anexos

Apéndice A

Sintaxis de QRL

En este apéndice mostraremos la sintaxis del lenguaje QRL utilizando la notación EBNF que recogemos en la tabla §A.1.

```
(* Documento de requisitos *)
document ::= catalogueSec productSec validPeriodSec basicConditionsSec
             negotiationSec preferencesSec

(* Sección de catálogos de atributos *)
catalogueSec ::=  { using identList };* { catalogueDef };*
catalogueDef  ::= catalogue ident '{' basicAttribList derivedAttribList [patternSec]  '}'
basicAttribList ::= { ident '{' basicAttribDef '}' }+
basicAttribDef ::= description: "string";'
                   domain: domainDef;'
                   [statistic: statisticModifier ';']
domainDef ::= enum '{' identList '}'
            | set '{' identList '}'
            | int [range] [unit]
            | real [range] [unit]
            | int *
            | real *
range ::= Lit int ',' Lit int  |  Lit real ',' Lit real
statisticModifier ::=  max | mean | min | percentile  Lit int | variance

patternSec ::= pattern '{' { ident '{' patternDef '}' }+ '}'
patternDef ::= ident ':' l-pattern '=' simbolicConstraint
```

Elemento	Descripción
terminal, '['	Símbolos terminales en negrilla o entre apóstrofos
no_terminal	Símbolos no terminales en letra normal
::=	Definición de un símbolo no terminal
[e]	Cero o una ocurrencia de la entidad sintáctica descrita por e
$\{ e \}^{*}_{s}$	Cero, una o más ocurrencias separadas por el símbolo s de la entidad sintáctica descrita por e. Si se omite s se considera que no existe separador explícito
$\{ e \}^{+}_{s}$	Una, dos o más ocurrencias separadas por el símbolo s de la entidad sintáctica descrita por e. Si se omite s se considera que no existe separador explícito
(* *texto* *)	Comentarios

Cuadro A.1: *Notación* EBNF.

```
l-pattern ::= " { [ placeHolder ] string  [ placeHolder ] }+";
placeHolder ::= '<' string '>'
simbolicConstraint ::= l-pattern


(* Sección de productos *)
productSec ::= products '{' identProduct [= identList] ';' '}'

(* Sección de períodos de vigencia *)
validPeriodSec ::= zone: GMT [Lit int] '{' periodList '}'
periodList ::= global { validPeriod } { namedValidPeriod } * }
validPeriod ::=  validPeriod and validPeriod
               |  validPeriod except validPeriod
               |  '(' validPeriod ')'
               |  [ from hourPattern ] [ on weekPattern ] [ during datePattern ]
namedValidPeriod ::= ident '{' validPeriod '}'

hourPattern ::=  { hour '..' hour } +,
hour ::= number [ ':' number ]
weekPattern ::=  { dayName [ '..' dayName ] }*,
dayName ::= MON | TUE | WED | THU | FRI | SAT | SUN
datePattern ::=  { date [ '..' date ] }+,
date ::= number '/' monthName '/' number
       | [ monthName '/' ] number
       | monthName
monthName ::= JAN | FEB | MAR | APR | MAY | JUN | JUL | AUG | SEP | OCT | NOV | DEC
```

(* Sección de condiciones básicas *)
conditionsSec ::= **conditions for** ident '{' defConditions'}'
defConditions ::= [productConditions] [allConditions] [subProductConditions]
productConditions ::= **global** '{' requirementsList '}'
allConditions ::= **for all** '{' requirementsList '}'
subProductConditions ::= **for** ident '{' requirementsList '}'
requirementsList ::= { requirementDef }$^{+}_{;}$

(* Requisitos *)
requirementDef ::= ident':' qualityConstraint validPeriod
| ident':' ident (* para requisitos estereotipados *)
qualityConstraint ::= qualityConstraint **and** qualityConstraint
| qualityConstraint **or** qualityConstraint
| **not** qualityConstraint
| expression relOp expression
| '(' qualityConstraint ')'
| **true**
| **false**

(* Expresiones *)
expression ::= ident
| rang
| '{' identList '}'
| $\mathcal{L}it$ **int**
| $\mathcal{L}it$ **real**
| idMeasure
| '('expression')'
| expression binOp expression
| idUnaryFunction '('expression')'
| idBinaryFunction '('expression, expression')'
binOp ::= + | - | * | /
idUnaryFunction ::= log | abs
idBinaryFunction ::= pow | exp | max
relOp::= $< \mid \leq \mid = \mid \neq \mid \geq \mid > \mid \in \mid \subset \mid \subseteq \mid \supset \mid \supseteq$

(* Sección de negociación *)
negotiationSec ::= **negotiation** '{' [tradeOffSec] [renunciationsSec] '}'
tradeOffSec ::= **tradeoffs** '{' { ident: '{' loserClause winnerClause '}' }$^{+}$ '}'
renunciationsSec ::= **renunciations** '{' { ident: '{' loserClause '}' }$^{+}$ '}'
loserClause ::= **lose** '{' requirementsList '}'
winnerClause ::= **win** '{' requirementsList '}'

(* Otros elementos *)
identList ::= {ident}$^{+}_{,}$
unit ::= string

A.1. Facilidades sintácticas para los períodos de vigencia

Para los patrones horarios:

- Si se omite la cláusula **from** se considera que hp = 00:00..24:00
- Si se omiten los minutos se considera que es el instante inicial de la hora, es decir, **from** 8 se interpreta como **from** 08:00.

Para los patrones de días de la semana:

- Si se omite la cláusula **on** se considera que wp = MON..SUN
- Si se omiten los dos puntos y el límite superior se entiende que el limite superior es el mismo que el inferior, es decir, **on** MON, WED, FRI se interpreta como **on** MON..MON, WED..WED, FRI..FRI

Para los patrones de fechas:

- Si se omite la cláusula **during** se considera que dp=beg(VP)..end(VP), donde VP es el período de vigencia del documento.
- Si se omiten el día, el mes y la fecha final se considera que está referido sobre el año completo, es decir, **during** 2001 se interpreta como **during** 01/JAN/2001..31/DEC/2001.
- Si se omiten los dos puntos y el límite superior se entiende que el limite superior es el mismo que el inferior, es decir, **during** 06/DEC/2001 se interpreta como **during** 06/DEC/2001..06/DEC/2001
- Si se omiten el día y el mes se considera que el se trata del primer o último día del año según se trata de la fecha inicial o final de un período, es decir, **during** 2001..2002 se interpreta como **during** 01/JAN/2001..31/DEC/2002.

La tabla §A.2 muestra distintos ejemplos del uso de facilidades sintácticas considerando que el período de vigencia del documento abarca desde el 01/JAN/2001 hasta el 31/DEC/2002. Como puede observarse, estas facilidades reducen notablemente la longitud de las especificaciones de períodos temporales.

Si no se indica nada se asume que estamos en la zona GMT.

QRL	$QRL_{\heartsuit}$
$\emptyset$	
	from 00:00..24:00 **on** MON..SUN **during** 01/JAN/2001 ..31/DEC/2002
from 8..12	
	from 08:00..12:00 **on** MON..SAT **during** 01/JAN/2001 ..31/DEC/2002
on MON	
	from 00:00..24:00 **on** MON..MON **during** 01/JAN/2001 ..31/DEC/2002
from 10..20 **on** MON..FRI **and from** 10..15 **on** SAT	
	from 10:00..20:00 **on** MON..FRI **during** 01/JAN/2001..31/DEC/2002 **and from** 10:00..15:00 **on** SAT..SAT **during** 01/JAN/2001..31/DEC/2002
(**from** 8..14, 16..18 **on** MON, WED, FRI **during** 2001..2002 **and from** 8..14 **on** TUE, THU **during** 2001..2002) **except during** AUG	
	(**from** 8:00..14:00, 16:00..18:00 **on** MON..MON, WED..WED, FRI..FRI **during** 01/JAN/2001..31/DEC/2002 **and from** 08:00..14:00 **on** TUE..TUE, THU..THU **during** 2001..2002) **except from** 00:00..24:00 **on** MON..SUN **during** 01/AUG/2001..31/AUG/2001, 01/AUG/2002..31/AUG/2002
during 2001..2002 **except during** (AUG, 25/DEC, 14/FEB/2001)	
	from 00:00..24:00 **on** MON..SUN **during** 01/JAN/2001..31/AUG/2002 **except from** 00:00..24:00 **on** MON..SUN **during** 01/AUG/2001..31/AUG/2001, 01/AUG/2002..31/AUG/2002, 25/DEC/2001..25/DEC/2001, 14/FEB/2001..14/FEB/2002

Cuadro A.2: *Ejemplos de facilidades sintácticas para especificar períodos de vigencia.*

Bibliografía

[Azuma 2002] Motoei Azuma. SQuaRE: The next generation of the ISO/IEC 9126 and 14598 international standards series on software product quality. Informe técnico, January 2002.

[Boehm *et al.* 1978] B.W. Boehm, J.R. Brown, H. Kaspar, M. Lipow, G.J. MacLeod, y M.J. Merrit. *Characteristcs of Software Quality*. North–Holland Publishing Co, 1978.

[Boehm *et al.* 1994] B. Boehm, P. Bose, E. Horowitz, y M.-J. Lee. Software Requirements as Negotiated Win Conditions. En *Proc. of the First Intl. Conference on Requirements Engineering*, páginas 74–83. IEEE CS Press, 1994.

[Boehm *et al.* 1995] B. Boehm, P. Bose, E. Horowitz, y M.-J. Lee. Software Requirements Negotiated and Renegotiation Aids: A Theory–W Based Spiral Approach. En *Proc. of the 17th Intl. Conference on Software Engineering. ICSE' 95*. IEEE CS Press, 1995.

[Boehm y In 1996] B. Boehm y H. In. Identifying Quality–Requirements Conflicts. *IEEE Software*, 12(6):25–35, Marzo 1996.

[Boehm y Ross 1989] B.W. Boehm y R. Ross. Theory–W Software Project Management: Principles and Examples. *IEEE Transactions on Software Engineering*, 15(7):902–912, 1989.

[Booch *et al.* 1999] G. Booch, J. Rumbaugh, y I. Jacobson. *The Unified Modeling Language User Guide*. Addison–Wesley, 1^a edición, 1999.

[Bosch y Molin 1999] J. Bosch y P. Molin. Software Architecture Design: Evaluation and Transformation. En *Proc. of the IEEE Engineering of Computer Based Systems Symposium, ECBS99*, Dec 1999.

[Bosch 1998] J. Bosch. Specifying Frameworks and Design Patterns as Architectural Fragments. En *Proc. of the Technology of Object-Oriented Languages and Systems (TOOLS'98)*, páginas 1–10, Beijing, China, 1998.

[Botella *et al.* 2001] P. Botella, X. Burgués, X. Franch, M. Huerta, y G. Salazar. Modeling Non-Functional Requirements. En *Actas de las Jornadas de Ingeniería de Requisitos Aplicada (JIRA)*, Jun 2001. Disponible en `http://www.lsi.us.es/ amador/JIRA/JIRA.html`.

[Botella 2001] P. Botella. La investigación en Ingeniería de Software en nuestro país, ¿va bien? En *Adenda al Libro de Actas de las VI Jornadas de Ingeniería del Software y Bases de Datos (JISBD'01)*, Almagro, Ciudad Real, 2001.

[Bourque *et al.* 1999] P. Bourque, R. Dupuis, y A. Abran. The Guide to the Software Engineering Body of Knowledge. *IEEE Software*, 16(6), Septiembre/Diciembre 1999.

[Breitman *et al.* 1999] K. Breitman, J.C. Leite, y A. Finkelstein. The worlds a stage: a survey on requirements engineering using a real-life case study. *Journal of the Brazilian Computer Society*, 6(1), July 1999.

[Broy 2001] Manfred Broy. Toward a mathematical foundation of software enginerring methods. *IEEE Transactions on Software Engineering*, 27(1):42–57, 2001.

[Burgués *et al.* 2000] X. Burgués, X. Franch, y J.A. Pastor. Formalising ERP Selection Criteria. En *Proc. of the 10^{th} International Workshop on Software Specification and Design (IWSSD)*, San Diego, California, November 2000.

[Burgués y Franch 2000] X. Burgués y X. Franch. A Language for Stating Component Quality. En *Proc. of the 14^{th} Symposium on Software Ingineering (SBES)*, páginas 69–84, Joao Pessoa, Brasil, October 2000.

[Buschmann *et al.* 1996] F. Buschmann, R. Meunier, H. RohnertF, P. Sommerlad, y M.Stal. *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley & Sons, 1996.

[Canal *et al.* 1999] C. Canal, L. Fuentes, E. pimentel, y J.M. Troya. Coordinación de componentes distribuidos: un enfoque generativo basado en arquitectura del software. En *Actas de las IV Jornadas de Ingeniería del Software y Bases de Datos (JISBD'99)*, páginas 443–454, Cáceres, 1999.

[Cavano y McCall 1978] J. P. Cavano y J. A. McCall. A framework for the measurement of software quality. En *Proceedings of the Software & Quality Assurance Workshop*, 1978.

[Chung *et al.* 2000] L. Chung, B.Ñixon, E. Yu, y J. Mylopoulos. *Non–Functional Requirements in Software Engineering*. Kluwer Academic Publishers, 2000.

[Chung y Yu 1998] L. Chung y E. Yu. Achieving system-wide architectural qualities. En *Proc. of the Workshop on Compositional Software Architectures*, Monterey, California, 1998.

[Ciapessoni *et al.* 1999] E. Ciapessoni, A. Coen-Porisini, E. Crivelli, D. Mandrioli, P. Mirandola, y A. Morzenti. From Formal Models to Formally Based Methods: An Industrial Experience. *ACM Transactions on Software Engineering and Methodology*, 8(1):79–113, January 1999.

[Clements y Northrop 1996] P. Clements y L.Ñorthrop. Software Architecture: An Executive Overview. En A. W. Brown, editor, *Component-Based Software Engineering*, páginas 55–68. IEEE Computer Society Press, 1996.

[Coleman *et al.* 1994] D. Coleman, P. Arnold, S. Bodoff, C. Dollin, H. Gilchrist, F. Hayes, y P. Jeremaes. *Object–Oriented Development. The Fusion Method*. Prentice–Hall, 1994.

[Consortium 2000] World Wide Web Consortium. XML Schema Part 0: Primer, W3C Working Draft . Informe técnico, World Wide Web Consortium, Sep 2000. available at http://www.v3.org/TR/xmlschema-0/.

[Corchuelo *et al.* 1999] R. Corchuelo, D. Ruiz, M. Toro, y A. Ruiz-Cortés. Implementing multiparty interactions on a network computer. En *Proc. of the XXVth Euromicro Conference (Workshop on Network Computing)*, páginas 458–465, Milan, Italy, September 1999. IEEE Press.

[Corchuelo *et al.* 2001] R. Corchuelo, A. Ruiz-Cortés, J.R. Muhlbacher, y J. García-Consuegra. Object–oriented business solutions. En *ECOOP'2001 Workshop Reader*, página To be published. Springer Verlag LCNS, 2001.

[Corchuelo y Ruiz-Cortés 2002] R. Corchuelo y A. Ruiz-Cortés. La calidad en las aplicaciones de los mercados electrónicos. En F. García y Mario Piattini, editores, *Calidad en el Desarrollo y Mantenimiento del Software*, página (pendiente de publicación). Ra-Ma, 2002.

[Corchuelo 1999] R. Corchuelo. *Prototipado de Especificaciones de Sistemas Distribuidos Basadas en Retricciones. Aplicación al Lenguaje TESORO.* Tesis doctoral, Universidad de Sevilla, Octubre 1999.

[Crespo y Marqués 2001] Y. Crespo y J.M. Marqués. Definición de un marco de trabajo para el análisis de refactorización del software. En *Actas de las VI Jornadas de Ingeniería del Software y Bases de Datos (JISBD'01)*, páginas 297–310, Almagro, Ciudad Real, 2001.

[Cristian 1991] F. Cristian. Understanding Fault–Tolerant Distributed Systems. *Communications of the ACM*, 34(2), 1991.

[Cysneiros *et al.* 2001] L.M. Cysneiros, J. Leite, y J.Ñeto. A Framework for Integrating Non-Functional Requirements into Conceptual Models. *Requirements Engineering*, 6(2):97–115, 2001.

[D. D'Souza and A. Wills 1999] D. D'Souza and A. Wills. *Objects, Components and Framework with UML. The Catalysis Approach.* Addison–Wesley, 1ª edición, 1999.

[Davis 1990] A. M. Davis. The Analysis and Specification of Systems and Software Requirements. En R. H. Thayer y M. Dorfman, editores, *System and Software Requirements Engineering*, páginas 119–144. IEEE Computer Society Press, 1990.

[Davis 1993] A. M. Davis. *Software Requirements: Objects, Functions and States.* Prentice–Hall, 2ª edición, 1993.

[Díaz 1999] O. Díaz. ¿para qué la tesis doctoral? En *Actas de las IV Jornadas de Ingeniería del Software y Bases de Datos (JISBD'99)*, páginas 1–6. Springer–Verlag, 1999.

[Dong *et al.* 1999] J. Dong, P. Alencar, y D. Cowan. A Component Specification Template for COTS-based Software Development. En *Proc. of the International Workshop on Ensuring Successful COTS Development, in conjunction with 21st Intl. Conference on Software Engineering (ICSE)*, 1999.

[Dromey 1995] R. Geoff Dromey. A Model for Software Product Quality. *IEEE Transactions on Software Engineering*, 21(2):146–162, February 1995.

[Dujmovic y Elnicki 1982] J.J. Dujmovic y R. Elnicki. A DMS Cost/Benefit Decision Model: Matematical Models for Data Management System Evaluation, Comaprison and Selection. Informe Técnico NBS-GCR-82-374, NTIS Nº. PB82-170150 (155 pages), National Bureau of Standards, Washington, D.C., 1982.

[Dujmovic 1991] J.J. Dujmovic. Preferential neural networks. En *Chapter 7 in "Neural Networks: Concepts, Applications and Implementations"*, volumen II de *Prentice–Hall Advanced Reference Series*, páginas 155–206, 1991.

[Dujmovic 1996] J.J. Dujmovic. A Method for Evaluation and Selection of Complex Hardware and Software Systems. En *Proc. of the 22nd International Conference for the Resource Management and Performance Evaluation of Enterprise Computing Systems*, volumen 1, páginas 368–378, 1996.

[Durán *et al.* 1998] A. Durán, B. Bernárdez, M. Toro, y A. Ruiz-Cortés. Una propuesta metodológica para la elicitación de requisitos de un sistema software. En *Actas de las III Jornadas de Trabajo de Menhir*, Murcia, 1998. Universidad de Murcia.

[Durán *et al.* 1999a] A. Durán, B. Bernárdez, A. Ruiz-Cortés, y M. Toro. A requirements elicitation approach based in templates and patterns. En *WER'99 Proceedings*, páginas 17–29, Buenos Aires, 1999.

[Durán *et al.* 1999b] A. Durán, B. Bernárdez, M. Toro, R. Corchuelo, A. Ruiz-Cortés, y J. Pérez. Expressing customer requirements using natural language requirements templates and patterns. En *Proc. of the IIIrd Conference on Circuits, Systems, Communications and Computers CSCC'99*, Athens, (Greece), 1999. IMACS/IEEE.

[Durán *et al.* 1999c] A. Durán, B. Bernárdez, M. Toro, y A. Ruiz-Cortés. Elicitación de requisitos de usuario mediante plantillas y patrones de requisitos. En *Actas de las IV Jornadas de Ingeniería del Software y Bases de Datos (JISBD'99)*, páginas 183–194, Cáceres, 1999.

[Durán *et al.* 1999d] A. Durán, B. Bernárdez, M. Toro, y A. Ruiz-Cortés. An object–oriented model and a case tool for software requirements management and documentation. En *Fourth Workshop on Models, Environments and Tools for Requirements Engineering*, páginas 6–10, Sedano, Burgos, 1999.

[Durán *et al.* 2001] A. Durán, B. Bernárdez, A. Ruiz-Cortés, y M. Toro. An xml–based approach for the automatic verification of software requirements specifications. En *Proc. of the 4th Workshop on Requirements Engineering. WER'2001*, páginas 181–194, Buenos Aires, Argentina, 2001.

[Durán *et al.* 2002a] A. Durán, A. Ruiz-Cortés, R. Corchuelo, y M. Toro. Implementing requirements verification heuristics with XML and XSLT.

En *Proc. of the e^{rd} ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD 2002)*, 2002.

[Durán *et al.* 2002b] A. Durán, A. Ruiz-Cortés, R. Corchuelo, y M. Toro. Supporting requirements verification using XSLT. En *Proc. of the International Requirements Engineering Conference (RE'2002)*. IEEE CS Press, 2002.

[Durán 2000] A. Durán. *Un Entorno Metodológico de Ingeniería de Requisitos para Sistemas de Información*. Tesis doctoral, Universidad de Sevilla, Mayo 2000.

[Faratin *et al.* 2000] P. Faratin, C. Sierra, y N. Jennings. Using similarity criteria to make negotiation trade-offs. En *International Conference on Multiagent Systems (ICMAS-2000)*, páginas 119–126, Boston, MA., 2000.

[Faratin 2000] P. Faratin. *Automated Service Negotiation Between Autonomous Computational Agents*. Phd. thesis, University of London, Queen Mary College, Department of Electronic Engineering, 2000.

[Fenton y Pfleeger 1996] N.E. Fenton y S.L. Pfleeger. *Software Metrics: a Rigorous and Practical Approach*. International Thomson Computer Press, 1996.

[Finkelstein *et al.* 1996] A. Finkelstein, M.A. Ryan, y G. Spanoudakis. Software Package Requirements and Procurement. En *Proc. of the 8th International Workshop on Software Specification and Design IWSSD-8*, páginas 141–146. IEEE CS Press, 1996.

[Finkelstein y Dowell 1996] A. Finkelstein y J. Dowell. A Comedy of Errors: the London Ambulance Service case study. En *Proc. of the 8th International Workshop on Software Specification and Design IWSSD-8*, páginas 2–4. IEEE CS Press, 1996.

[Finkelstein 1993] A. Finkelstein. Report of the Inquiry into the London Ambulance Service. ftp://cs.ucl.ac.uk/acwf/info/lascase0.9.pdf, 1993.

[Fowler 1999] M. Fowler. *Refactoring: Improving the Design of Existing Code*. Addison–Wesley, 1999.

[Franch *et al.* 1997] X. Franch, P. Botella, X. Burgués, y J.M. Ribo. ComproLab: A Component Programming Laboratory. En *Proc. of the IX^{th} Software Engineering and Knowledge Engineering Conference (SEKE)*, páginas 397–406, Madrid, Spain, June 1997.

[Franch *et al.* 1999] X. Franch, J. Pinyol, y J. Vancells. Browsing a component library using non–functional information. En *Proc. of the International Conference on Reliabel Software Technologies. Ada Europe 99. LNCS 1622*, páginas 332–343, Santander, Spain, June 1999.

[Franch y Botella 1998] X. Franch y P. Botella. Putting non-functional requirements into software architecture. En *Proc. of the IXth Intl. Workshop on Software Specification and Design*, Ise-Shima (Isobe), Japan, April 1998.

[Franch y Pastor 2000] X. Franch y J.A. Pastor. On the formalisation of erp systems procurement. En *Proc. of the Workshop on Continuing Collaboration for Successful COTS Development, in conjunction with 22nd Intl. Conference on Software Engineering (ICSE)*, Limerick, Ireland, June 2000.

[Franch 1998] X. Franch. Systematic formulation of non–functional characteristics of software. En *Proc. of the International Conference on Requirements Engineering (ICRE'98)*, Colorado, USA, April 1998. IEEE CS Press.

[Freuder y Wallace 2000] E.C. Freuder y M. Wallace. Science and Substance: A Challenge to Software Engineers. *Constraints IEEE Inteligent Systems*, 2000.

[Frølund y Koistinen 1998a] S. Frølund y J. Koistinen. QML: A Language for Quality of Service Specification. Informe Técnico HPL-98-10, Hewlett–Packard, 1998. Disponible en `http://www.hpl.hp.com/techreports`.

[Frølund y Koistinen 1998b] S. Frølund y J. Koistinen. Quality–of–Service Specification in Distributed Object Systems. *Distributed Systems Engineering Journal*, 5(4):179–202, 1998. Disponible como informe técnico HPL–98–159 en `http://www.hpl.hp.com/techreports`.

[Frølund y Koistinen 1999] S. Frølund y J. Koistinen. Quality of Service Aware Distributed Object Systems. En *Proc. of the USENIX Conference on Object–Oriented Technologies*, 1999. Disponible como informe técnico en HPL–98–142 en `http://www.hpl.hp.com/techreports`.

[Galán 2000] F. Galán. *Formalizaciones para Sintetizar Software Orientado a Objetos*. Tesis doctoral, Universidad de Sevilla, Abril 2000.

[Gamma 1995] E. Gamma. *Design Patterns: Elements of Reusable Object-Oriented Software*. Professional Computing Series. Addison–Wesley, 1995.

[Gilb 1988] T. Gilb. *Principles of Software Engineering Management.* Addison–Wesley, 1988.

[Grady y Caswell 1987] R.B. Grady y D.L. Caswell. *Software Metrics: Establishing a Company-Wide Program.* Prentice Hall, 1987.

[Gray y Reuter 1993] J. Gray y A. Reuter. *Transaction Processing: Concepts and Techniques.* Morgan Kaufmann, 1993.

[Hafid *et al.* 1998] A. Hafid, G. Bochmann, y R. Dssouli. A quality of service negotiation approach with future reservations (NAFUR): A detailed study. *Computer Networks*, 30(8):777–794, May 1998.

[Hennessy 1990] M. Hennessy. *The semantics of programming languages.* John Wiley & Sons, 1990.

[Hentenryck y et al. 1996] P.V. Hentenryck y Saraswat et al. Strategic Directions in Constraint Programming. *ACM Computing Surveys*, 28(4), December 1996.

[IEEE 1993] IEEE. IEEE Recommended Practice for Software Requirements Specifications. IEEE/ANSI Standard 830–1993, Institute of Electrical and Electronics Engineers, 1993.

[In *et al.* 2002] H. In, D. Olson, y T. Rodgers. Multi–criteria preference analysis for systematic requirements negotiation. En *Proceedings of the IEEE International Computer Software and Applications Conference (COMPSAC 2002)*, página To be published, 2002.

[Iribarne y Vallecillo 2000] L. Iribarne y A. Vallecillo. Searching and Matching Software Components with Multiple Interfaces. En *Proc. TOOLS Europe'2000.* IEEE Press, June 2000.

[ISO 1991] ISO. ISO/IEC 9126–1991 International Standard. Quality characteristics and guidelines for their use. Informe técnico, International Standard Organization, 1991.

[ISO 1999] ISO. ISO/IEC 9126–1991 International Standard. Informe técnico, International Standard Organization, 1999.

[ITU/ISO 1995] ITU/ISO. Reference Model of Open Distributed Processing - PArt 3: Architecture. ITU-T Rec. X903, ISO/IEC 10746-3, 1995.

[Jennings *et al.* 2001] N.R. Jennings, P. Faratin, A.R. Lomuscio, S. Parsons, C. Sierra, y M. Wooldridge. Automated negotiation: Prospects, methods and challenges. *(to appear in) International Journal of Group Decision and Negotiation*, 10(2):199–215, 2001.

[Keeney y Raiffa 1976] R.L. Keeney y H. Raiffa. *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*. Wiley, 1976.

[Keller *et al.* 1990] S.E. Keller, L.G. Kahn, y R.B. Panara. Specifying Software Quality Requirements with Metrics. En R. Thayer y M. Dorfman, editores, *Systems and Software Requirements Engineering*, páginas 145–163. IEEE Computer Society, 1990.

[Kiczales *et al.* 1997] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. Loingtier, y J. Irwin. Aspect-oriented programming. En *ECOOP'97 Proc.*, páginas 220–242. Lecture Notes in Computer Science, Springer–Verlag, 1997.

[Kitchenham y Pfleeger 1996] B.A. Kitchenham y S.L. Pfleeger. Software Quality: The Elusive Target. *IEEE Software*, 12(6):12–21, January 1996.

[Kohavi y John 1997] R. Kohavi y G. H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(2):273–324, 1997.

[Koistinen y Seetharaman 1998] J. Koistinen y Seetharaman. Worth–based Multi-Category Quality–Of–Service Negotiation in Distributed Object Infrastructures. En *Proc. of the EDOC'98*, La Jolla, USA, 1998.

[Koskimies y Paakki 1990] K. Koskimies y J. Paakki. *Automating Language Implementation*. Ellis Horwood, England, 1990.

[Kramer y Wolf 1996] J. Kramer y A. Wolf. Succedings of the 8^{th} Interantional Workshop on Software Specification and Design. *ACM SIGSOFT Software Engineering Notes*, 21(5):21–35, 1996.

[Kruchten 1995] P. Kruchten. The 4+1 View Model of Architecture. *IEEE Software*, 12(6):42–50, 1995.

[L. Bass y Kazman 1998] P. Clements L. Bass y R. Kazman. *Software Architecture in Practice*. Addison–Wesley, 1998.

[Lee 1989] P. Lee. *Realistic Compiler Generation*. MIT Press, 1989.

[Lee 1996] M. J. Lee. *Foundations of the WinWin Requirements Negotiation System*. Phd. thesis, University of Southern California, August 1996.

[Liskov y Wing 1994] B. Liskov y J. Wing. A behavioral notion of subtyping. *ACM Transactions on Programming Languages and Systems*, 16(6):1811–1841, Nov 1994.

[Liu 1998] Xiaoqing (Frank) Liu. A quantitative approach for assessing the priorities of software quality requirements. *Journal of Systems and Software*, 42(8):105–113, August 1998.

[Marriot y Stuckey 1998] K. Marriot y P.J. Stuckey. *Programming with Constraints: An Introduction*. The MIT Press, 1998.

[Martín *et al.* 2001a] Octavio Martín, Antonio Ruiz-Cortés, y Miguel Toro. An approach towards automatic management of architectural alternatives. En *Actas del taller de trabajo sobre Métodos y Herramientas Desarrollo Aplicaciones Comercio Electrónico (ZOCO'01)*, páginas 29–42, Almagro (Ciudad Real, Spain), November 2001.

[Martín *et al.* 2001b] Octavio Martín, Antonio Ruiz-Cortés, y Miguel Toro. Una aproximación a la evaluación automática de alternativas de diseño. En *Actas del primer taller de trabajo sobre Apoyo a la Decisión en Ingeniería del Software (ADIS–2001)*, Almagro (Ciudad Real, Spain), November 2001.

[Martín-Díaz 2002] Octavio Martín-Díaz. Towards a quality-aware processing of architectural alternatives. En *Proc. of the 12th Workshop for PhD Students in Object-Oriented Systems (ECOOP'02)*, página To be published, Málaga, Spain, June 2002.

[McCall *et al.* 1977] James A. McCall, Paul K. Richards, y Gene F. Walters. Factors in software quality, volume III: Preliminary handbook on software quality for an acquisition manager. Informe Técnico RADC-TR-77-369, vol. III, Department of Defense of the USA, 1977.

[Medvidovic y Taylor 2000] N. Medvidovic y R.N. Taylor. A Classification and Comparison Framework for Software Architecture Description Languages. *IEEE Transactions on Software Engineering*, 26(1):70–93, 2000.

[Meyer 1990] B. Meyer. *Introduction to the Theory of Programming Languages*. Prentice Hall, Hertfordshire, England, 1990.

[Meyer 1997] B. Meyer. *Object–Oriented Software Construction*. Prentice Hall, 1997.

[Meyer 2000] B. Meyer. Contracts for components. *Software Development*, July 2000. Available at `http://www.sdmagazine.com/articles`.

[Microsoft 2001] Microsoft. Building ASP.NET Web Services. White Paper, 2001. Disponible en `http://msdn.microsoft.com/webservices/`.

[Miller 1970] J.R. Miller. *Professional Decision–Making*. Praeger Publishers, 1970.

[Mostow 1985] J. Mostow. Towards Better Models of the Design Process. *AI Magazine*, 6(1):44–57, 1985.

[Mylopoulos *et al.* 1992] J. Mylopoulos, L. Chung, y B.Ñixon. Representing and Using Non-Functional Requirements: A Process–Oriented Approach. *IEEE Transactions on Software Engineering*, 18(6):483–497, 1992.

[Nuseibeh *et al.* 1994] B.Ñuseibeh, J. Kramer, y A. Finkelstein. A framework for expressing the relationships between multiple views in requirements specifications. *IEEE Transactions on Software Engineering*, 20(10):760–773, October 1994.

[Oliver 1996] James Robert Oliver. *On Artificial Agents for Negotiation in Electronic Commerce*. Ph.d. dissertation, University of Pennsylvania, 1996.

[Olsina *et al.* 1999] L. Olsina, D. Godoy, G. Lafuente, y G. Rossi. Specifying Quality Characteristics and Attributes for Websites. En *Proc. of the Web Engineering Workshop, in conjunction with 21st Intl. Conference on Software Engineering (ICSE)*, páginas 84–93, May 1999.

[Olsina 1999] Luis Antonio Olsina. *Metodología Cuantitativa para la Evaluación y Comparación de la Calidad de Sitios WEB*. Tesis doctoral, Universidad Nacional de la Pampa, Noviembre 1999.

[OMG 2000] OMG. Trading Object Service Specification. Informe técnico, Object Management Group, 2000. Version 1.0.

[OMG 2001] OMG. Unified Modeling Language, v1.4. Informe técnico, September 2001.

[Osmond 1995] R. Osmond. Essential of Successful O–O Project Management: Designing High Performance Projects. En *Notas de un tutorial en el TOOLS USA 95*, 1995.

[Parnas 1999] D. L. Parnas. Software engineering programs are not computer science programs. *IEEE Software*, páginas 19–30, November/December 1999.

[Pastor *et al.* 2001] J.A. Pastor, X. Franch, y F. Sistach. Methodological ERP Acquisition: the ERP Experience. En *1st World Class IT Service Management Guide. 2nd edition*, TenHagenStan, 2001.

[Peña *et al.* 2000a] J. Peña, R. Corchuelo, y A. Ruiz-Cortés. Implementación Automática de Fragmentos Arquitectónicos en el Contexto de ALFA. En *Actas del (SEID'00)*, Vigo, 2000.

[Peña *et al.* 2000b] J. Peña, R. Corchuelo, A. Ruiz-Cortés, y M. Toro. Una Aproximación al Desarrollo de Software Basada en el Prototipado Arquitectónico. En *Actas de las V Jornadas de Ingeniería del Software y Bases de Datos (JISBD'00)*, páginas 191–202, Valladolid, 2000.

[Perry y Wolf 1992] D. Perry y A. Wolf. "Foundations for the Study of Software Architecture. *ACM SIGSOFT Software Engineering Notes*, 17(4), 1992.

[Plotkin 1981] G.D. Plotkin. A structural approach to operational semantics. Informe Técnico DAIMI FN–19, Computer Science Department, Aarthus University, 1981.

[Pohl 1997] K. Pohl. Requirements Engineering: An Overview. *Encyclopedia of Computer Science and Technology*, 36, 1997. Disponible en `http://sunsite.informatik.rwth-aachen.de/CREWS`.

[Pressman 1998] Pressman. *Ingeniería del Software. Un enfoque práctico.* MacGraw Hill, 1998.

[Pérez 2001] J.A. Pérez. *Un Framework Orientado a Aspectos para la Descripción del Comportamiento Coordinado en Sistemas Abiertos.* Tesis doctoral, Universidad de Sevilla, June 2001.

[Pruitt 1981] D. G. Pruitt. *Negotiation Behaviour.* Academic Press, 1981.

[Robertson y Robertson 1999] J. Robertson y Suzanne Robertson. *Mastering the Requirements Process.* Addison-Wesley, 1999.

[Roman 1985] G.C. Roman. A Taxonomy of Current Issues in Requirements Engineering. *IEEE Computer*, 18(4):14–23, April 1985.

[Ruiz-Cortés *et al.* 1999] A. Ruiz-Cortés, R. Corchuelo, J. Pérez, A. Durán, y M. Toro. An Aspect-Oriented Approach based on Multiparty Interactions to Specifying the Behaviour of a System. En *Proc. of the Intl. Conference on Principles, Logics, and Implementations of High-Level Programming Languages, PLI'99 (Workshop on Object-Oriented Specification Techniques for Distributed Systems and Behaviours)*, páginas 56–65, Paris, (France), September 1999.

[Ruiz-Cortés *et al.* 2000a] A. Ruiz-Cortés, R. Corchuelo, A. Durán, y O. Martín. Prototipado Arquitectónico de Sistemas Abiertos Distribuidos. En *Actas de las V Jornadas de Trabajo de Menhir*, Granada, (España), Marzo 2000.

[Ruiz-Cortés *et al.* 2000b] A. Ruiz-Cortés, R. Corchuelo, O. Martín, A. Durán, y M. Toro. Addressing interoperability in multi–organisational web–based systems. En *Proceedings of the European Conference on Object–Oriented Programming ECOOP'00. Workshop on Object Interoperability*, páginas 87–96. Universidad de Extremadura, June 2000.

[Ruiz-Cortés *et al.* 2001a] A. Ruiz-Cortés, R. Corchuelo, A. Durán, y M. Toro. Automated support for quality requirements in web-services-based systems. En *Proc. of the 8th IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS'2001)*, páginas 48–55, Bologna, Italy, 2001. IEEE-CS Press.

[Ruiz-Cortés *et al.* 2001b] A. Ruiz-Cortés, A. Durán, R. Corchuelo, y M. Toro. Especificación de Requisitos de Calidad en Sistema Multiorganizacionales Basados en Servicios WEB. En *Actas de las VI Jornadas de Ingeniería del Software y Bases de Datos (JISBD'01)*, páginas 615–629, Almagro, Ciudad Real, 2001.

[Ruiz-Cortés *et al.* 2002a] A. Ruiz-Cortés, R. Corchuelo, y A. Durán. An automated approach to quality-aware web applications. En *Proc. of the 4th International Conference on Enterprise Information Systems (ICEIS'2002). ISBN: 972-98050-6-7*, páginas 995–1000, Ciudad Real, Spain, April 2002.

[Ruiz-Cortés *et al.* 2002b] A. Ruiz-Cortés, R. Corchuelo, A. Durán, y M. Toro. Enhancing win–win requirements negotiation model. En A. Durán, editor, *Applied Requirements Engineering*, página (pendiente de publicación). Catedral, 2002.

[Ruiz-Cortés y Simón 1997] A. Ruiz-Cortés y F. Simón. Sistema de Alerta de Contaminación Ambiental. *Química Hoy*, páginas 36–39, Febrero 1997.

[Sawyer y Kontoya 1999] P. Sawyer y G. Kontoya. SWEBOK: Software Requirements Engineering Knowledge Area Description. Informe Técnico Versión 0.5, SWEBOK Project, 1999. Disponible en `http://www.swebok.org`.

[Shaw y Garlan 1996] M. Shaw y D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, 1996.

[Sierra *et al.* 1997] C. Sierra, P. Faratin, y N. Jennings. A service–oriented negotiation model between autonomous agents. En *8th European Workshop on Modelling Autonomous Agents in MultiAgent World, number 1237 in LNAI*, páginas 17–35. Springer-Verlag, 1997.

[Smith 1980] R. Smith. The contract net protocol: High-level communication and control in distributed problem solver. *IEEE Transactions on Computers*, 29(12):1104–1113, Dec 1980.

[Stephens y Huhns 2001] L.M. Stephens y M.N. Huhns. Consensus Ontologies: Reconciling the Semantics of Web Pages and Agents. *IEEE Internet Computing*, 5(5):92–95, 2001.

[Su *et al.* 2000] S.Y.W. Su, C. Huang, y J. Hammer. A replicable web–based negotiation server for e–commerce. En *Proc. of the 33rd Hawaii International Conference on Systems Sciences*, páginas 1–8. IEEE-CS Press, 2000.

[Sun Microsystems, Inc 2001] Sun Microsystems, Inc. Open, Smart Web Services. White Paper, 2001. Disponible en `http://www.sun.com/software/sunone`.

[Szyperski 1998] C. Szyperski. *Component Software*. Addison-Wesley, 1998.

[Szyperski 2000] C. Szyperski. Components and contracts. *Software Development*, May 2000. Available at `http://www.sdmagazine.com`.

[Tekinerdogan 2000] B. Tekinerdogan. *Synthesis Based Software Architecture Design*. PhD Thesis, University of Twente, The Netherlands, 2000.

[UDDI.ORG 2000] UDDI.ORG. Uddi technical white paper. Informe técnico, UDDI.ORG, 2000. http://www.uddi.org.

[Vallecillo *et al.* 1999] A. Vallecillo, J. Hernández, y J.M. Troya. Object Interoperability. En *Object Oriented Technology: ECOOP'99 Workshop Reader, number 1743 in LNCS*, páginas 1–21. Springer-Verlag, 1999.

[Vallecillo *et al.* 2000] A. Vallecillo, J. Hernández, y J.M. Troya. New Issues in Object Interoperability. En A. Moreira j. Malefant, S. Moisan, editor, *Object Oriented Technology: ECOOP 2000 Workshop Reader, number 1964 in LNCS*, páginas 256–269. Springer-Verlag, 2000.

[Vallecillo 1999] A. Vallecillo. *Un Modelo de Componentes para el Desarrollo de Aplicaciones Distribuidas*. PhD thesis, Universidad de Málaga, Spain, 1999.

[Voas 1998] J. Voas. Certifying Off-the-Shelf Software Components. *IEEE Computer*, páginas 53–59, 1998.

[Wegner 1996] P. Wegner. Interoperability. *ACM Comp. Surveys*, 28(1):285–287, March 1996.

[Wieringa 1996] R. J. Wieringa. *Requirements Engineering: Frameworks for Understanding*. John Wiley & Sons, 1996.

[Wurman y Wellman 1999] P. Wurman y M. Wellman. A Control Architecture for Flexible Internet Auction Servers. En *Proc. of the 1st IAC Workshop on Internet–Based Negotiation Technologies*, páginas 392–396, Yorktown Heights, New York, 1999.

[Yeh *et al.* 1984] R. Yeh, P.Zave, A. Conn, y G. Cole. Software Requirements: New Directions and Perspectives. Handbook of software engineering, 1984.

UNIVERSIDAD DE SEVILLA

Reunido el Tribunal integrado por los abajo firmantes en el día de la fecha, para juzgar la Tesis Doctoral de D. Antonio Ruiz Cortés titulada Una Aproximación Semicualitativa al Tratamiento Automático de Requisitos de Calidad

acordó otorgarle la calificación de Sobresaliente Cum Laude por Unanimidad

Sevilla, 23 de Septiembre 2002

El Vocal, El Vocal, El Vocal,

EL PRESIDENTE El Secretario, El Doctorado,