Comprobación Automática de Requisitos de Calidad en Sistemas Multiorganizacionales*

Antonio Ruiz, Amador Durán, Rafael Corchuelo, Beatriz Bernárdez y Miguel Toro

Universidad de Sevilla, Dpto. de Lenguajes y Sistemas Informáticos

aruiz,amador,corchu,beat,mtoro@lsi.us.es

Resumen El uso de servicios WEB y de servidores de aplicaciones durante el desarrollo y explotación de sistemas multiorganizacionales basados en la WEB (MOWS), ha puesto de relieve algunas limitaciones de los actuales lenguajes de especificación de requisitos de calidad para soportar el tratamiento automático que, sobre este tipo de requisitos, precisan el desarrollo y la explotación de MOWS. En este artículo se presenta un lenguaje formal de especificación de requisitos que facilita la comprobación automática de la *conformidad* y la *economía*, dos propiedades de gran interés para los MOWS.

Keywords: ingeniería de requisitos, requisitos de calidad, aplicaciones WEB

1. Introducción

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 llamados servicios WEB [1, 2], son, a grandes rasgos, las principales razones que han motivado la propuesta de un nuevo paradigma de programación conocido como WOP (*Web–Oriented Programming*) [3]. Este paradigma intenta dar respuesta a los principales problemas del desarrollo de sistemas multiorganizacionales basados en la WEB, o más abreviadamente MOWS (*Multi–Organisational Web–based Systems*), que han sido definidos como sistemas cuya funcionalidad es obtenida mayoritariamente subcontratando aplicaciones y servicios WEB proporcionados por múltiples organizaciones [3].

Como cabría esperar, la WOP debe dar una respuesta efectiva a problemas críticos en el desarrollo de un MOWS como son el de la *optimalidad* y el de la *conformidad* [3, 4]. La optimalidad se define como la capacidad de seleccionar entre un conjunto de candidatos el servicio WEB o aplicación que satisface de manera óptima un conjunto de requisitos de calidad, que incluyen desde la cuota que es preciso pagar por utilizarlos hasta su tiempo de respuesta. Por su parte, la conformidad se define como la capacidad de *garantizar* que dichos requisitos de calidad se cumplen. Conviene matizar que en este contexto la palabra garantizar no debe entenderse como sinónimo de asegurar la

* Este trabajo está parcialmente financiado por el proyecto CICYT “GEOZOCO”, TIC 2000-1106-C02-01, y por el proyecto CYTED “WEST”

satisfacción ininterrumpida de los requisitos, sino de como ofrecer a las organizaciones que intervienen en un MOWS un documento de garantía que permita identificar en caso de problemas cuál es el elemento del sistema que no ha cumplido con sus especificaciones de calidad y depurar responsabilidades.

Hasta la fecha, las limitaciones de los lenguajes de especificación de requisitos de calidad han hecho imposible obtener soluciones automáticas realmente eficaces que resuelvan este problema. En este artículo presentamos las principales características de un nuevo lenguaje de especificación de requisitos diseñado para ofrecer una respuesta adecuada y eficaz a los problemas relacionados con la comprobación de la conformidad. En adelante haremos referencia a este lenguaje como QRL (*Quality Requirements Language*).

Las principales aportaciones de QRL respecto a otras propuestas son: i) el uso de un modelo ontológico para definir atributos de calidad que hace posible la comprobación automática de propiedades y facilita la comprobación entre documentos de requisitos que utilizan diferentes catálogos de atributos ii) la interpretación de la comprobación de la conformidad como un problema de satisfacción de restricciones [5] lo que aumenta considerablemente su capacidad expresiva iii) la posibilidad de especificar períodos de validez tanto para el documento de requisitos en su conjunto como de cada requisito individualmente y iv) la capacidad de especificar reglas de negociación bilaterales de modo que los clientes puedan expresar todas las condiciones que están dispuestos a aceptar y los proveedores todas las ofertas que desean ofrecer.

El resto del artículo está organizado de la siguiente forma: en la Sección 2 se describe el problema de la conformidad en los MOWS. En la Sección 3 se presenta de manera intuitiva el modelo ontológico de los catálogos de atributos y en la Sección 4 de presenta las posibilidades que ofrece QRL para describir requisitos de calidad. En la Sección 5 se presentan las reglas de negociación y en la Sección 6 las reglas que permiten establecer los criterios de optimalidad. Seguidamente en la Sección 7, se describen algunos de los trabajos relacionados con la presente contribución. Por último, en la Sección 8, se describen las conclusiones y los trabajos futuros.

2. La Conformidad en los MOWS

Para ilustrar de manera intuitiva el problema de la conformidad en los MOWS, imaginemos que se desea construir un portal que ofrece la misma funcionalidad que un reproductor de vídeo doméstico con una videoteca potencialmente infinita y con la garantía de que el nivel de calidad se satisfará mientras dure su explotación. Durante el diseño se ha optado por utilizar tres tipos de servicios WEB: uno para la autentificación de usuarios (p.e. Microsoft Passport, http://www.passport.com), otro de almacén de películas y otro de almacén de tráilers. De este modo, el propietario del portal sólo se encargaría de la publicidad y de la coordinación entre las diferentes servicios y aplicaciones subcontratados.

Con objeto de obtener la máxima rentabilidad económica, los MOWS son diseñados como sistemas abiertos y con capacidad de razonamiento automático, de modo que, en general, los proveedores de servicios y aplicaciones que pueden formar parte del

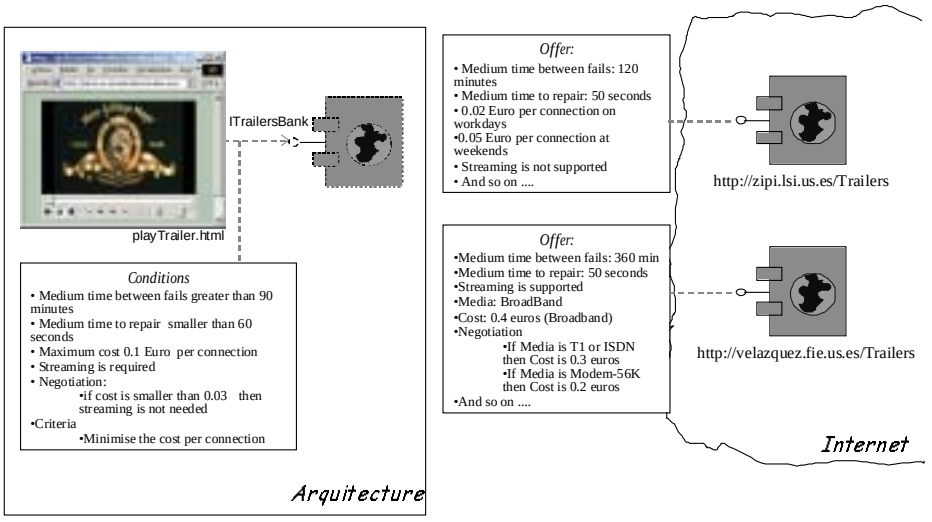

Figura1. Fragmento de la arquitectura de un portal que permite la reproducción de películas y tráilers.

sistema no son conocidos de antemano, sino que en cada conexión se elige el proveedor que resulte más beneficioso.

La Figura 1 muestra un fragmento de la vista de implementación (de acuerdo con UML [6]) de la arquitectura software del portal de video. Como puede observarse, para reproducir un tráiler (página `playTrailer.htm`) se necesita un servicio WEB que implemente la interfaz `ITrailersBank`. Por su parte, la línea discontinua del contorno del servicio WEB (componente estereotipado con una bola del mundo) significa que el servicio de almacenamiento de tráilers puede ser proporcionado por cualquier proveedor de dicho servicio que ofrezca un nivel de calidad que satisfaga las condiciones establecidas por el sistema. De ahora en adelante llamaremos *condiciones* al documento que especifica los requisitos de calidad exigidos por el sistema y *oferta* al documento que especifica los requisitos que el proveedor se compromete a garantizar.

Una de las actividades necesarias para poder garantizar la conformidad, es la comprobación de que las condiciones que debe satisfacer el proveedor del servicio pueden cumplirse con la oferta de calidad que éste proporciona. En la figura 1 se muestran dos proveedores de servicios de almacenamiento de tráilers. Es fácil comprobar que la oferta del primer proveedor (http://zipi.lsi.us.es/Trailers) no satisface las condiciones exigidas por el sistema pues no soporta `streaming`. De igual modo, es fácil comprobar que la oferta del segundo proveedor si satisface las condiciones del sistema.

Se podría pensar que comprobar la conformidad es un problema muy simple y que puede ser llevada a cabo manualmente por el administrador del portal, pero como veremos a continuación, existen varias razones que aconsejan que dicha actividad se automatice a fin de poder ser realizada por algún componente de la plataforma de ejecución:

- Cuando el número de requisitos es elevado, esta comprobación consume mucho tiempo lo que puede resultar inaceptable cuando el cliente que espera la conexión es un persona y no una aplicación. Además, existe una mayor probabilidad de que comprobaciones incorrectas den por válidas ofertas que no satisfacen los requisitos.
- El nivel de calidad de las condiciones y las ofertas pueden cambiar con mucha frecuencia, por lo que en el caso más desfavorable puede ser necesario comprobar la conformidad en cada conexión, aumentando considerablemente el número de comprobaciones necesarias.
- Si se incluyen reglas de negociación en las condiciones y/o en las ofertas, el proceso de comprobación se complica, y aumenta el tiempo necesario para realizarla.

3. Catálogos de Atributos de Calidad

Un atributo de calidad puede definirse como una propiedad cuyo grado de satisfacción por parte de un sistema software puede ser valorado por sus usuarios (atributos externos) y/o por sus diseñadores (atributos internos) [7, pág.3], en otras palabras, un atributo de calidad puede ser cualquier característica que pueda verificarse, es decir, medirse de manera objetiva o subjetiva. De este modo, dependiendo del dominio del sistema al que nos refiramos, la etapa del desarrollo en la que nos encontremos, o la experiencia de los desarrolladores, entre otros factores, para un mismo sistema podrán definirse diferentes conjuntos de atributos de calidad que lo caractericen (en [8, pág.160] se ofrece una lista con 161 atributos de calidad diferentes).

La definición de conjuntos de atributos en QRL se realiza mediante catálogos. El modelo ontológico de un catálogo de atributos consta únicamente de dos conceptos: *atributo de calidad* y *requisito abstracto*. Las propiedades de un atributo de calidad son cinco: un nombre o identificador, una descripción en lenguaje natural, una descripción formal, una definición de su dominio y su unidad de medida. Las restricciones que se establecen sobre estas propiedades son las siguientes:

- El nombre debe ser de tipo cadena y su valor será único dentro del catálogo. Su indicación es obligatoria.
- La descripción debe ser de tipo cadena. Su indicación es opcional.
- La descripción formal debe constar de al menos un patrón lingüístico [9] que describa su significado y de una restricción matemática equivalente. Su indicación es obligatoria.
- El dominio podrá ser de tipo entero, real, enumerado o de tipo conjunto. Su indicación es obligatoria.
- La unidad será de tipo cadena. Su indicación es obligatoria siempre que el atributo no sea adimensional.

Por su parte, las propiedades de un requisito abstracto son tres: su identificador, una restricción de calidad y un período de vigencia. Estas propiedades se estudiarán en detalle en la Sección 4.

La Figura 2 muestra la definición parcial de dos catálogos de atributos de calidad, un catálogo horizontal para los MOWS y otro vertical para el dominio de las aplicaciones de multimedia. El catálogo se identifica por un URI (*Universal Resource Identificator*) de

forma similar a la utilizada en XML para los espacios de nombres [10]. A continuación describimos con mayor detalle como se definen los dominios y los patrones lingüísticos.

```
catalogue com.acme.stdMOWS {
 attributes {
    TTF {
        description:"Tiempo entre fallos";
        domain: integer min;
        pattern: "El tiempo medio entre fallos será de al menos
                 <tiempo>" = "TTF.mean ≥ <tiempo>";
        pattern: "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>"; }
    TTR {
        description: "Tiempo de recuperación ante un error";
        domain: integer sec;
        pattern: "El tiempo {medio, máximo} de recuperación ante un fallo será
                 inferior a <valor>" = "TTF.{mean,max} ≤ <valor>"; }
    BCODE {
        description: "Portabilidad a nivel de código binario";
        domain: set {Win3.X, Win95, Win98, WinNT4, Win2000, Linux, Solaris};
        pattern: "Todos los componentes de la aplicación podrán ser ejecutados sin
                 necesidad de recompilar el código fuente en las siguientes
                 plataformas: <valor>" = "BCODE >= <valor>"; }
    SCODE { domain: set {Win95, Win2000, Linux, Solaris}; . . .}
    AVAIL { domain: real [0..100] %; . . .}
    FMASK { domain: set {early, late, state, value, omission}; . . .}
    SFAIL { domain: enum {halt, initialState, rollBack}; . . .}
    OPSEM { domain: enum {atLeastOnce, atMostOnce, once}; . . . }
    REBIND { domain: boolean; . . . }
    DELAY { domain: integer [0, 60*60] sec; . . . }
    THR { domain: integer [0, 1000] unit/sec; . . . }
 }
 stereotypes {
    safety.low: TTF.mean > 60 and TTR ≤ 120;
    safety.med: TTF.mean > 90 and TTR ≤ 90;
    robustness.high: REBIND = true and OPSEM = once and SFAIL ≥ initialState;
 }
catalogue com.acme.multimedia {
 attributes {
    STREAMING { domain: boolean; . . . }
    COST { domain: real euro; . . . }
    MEDIA { domain: set {BroadBand, T1, ISDN, Modem}; . . . }
    LANG { domain: set {english, french, german, spanish}; . . . }
    SUBT { domain: set {english, french, german, spanish}; . . . }
 }
}
```

Figura2. Especificación de catálogos de atributos en QRL

3.1. Dominios de Atributos de Calidad

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

- **Enteros**: Sus elementos son números enteros. Por ejemplo, la latencia (DELAY), definida como el tiempo que se tarda en responder a una determinada operación y el tiempo entre fallos (TTF).

- **Reales**: Sus elementos son números reales. Por ejemplo, la disponibilidad (`AVAIL`), definida como la probabilidad (expresada en %) de que el sistema se encuentre fuera de servicio.
- **Lógicos**: Sus elementos sólo pueden tomar dos valores **true** o **false**. Por ejemplo, el soporte de *streaming* por parte de un servidor.
- **Enumerados**. Sus elementos son literales alfanuméricos. 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 literales, 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 [11], 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`}).

3.2. Descripción Formal de Atributos

La base formal de QRL son las restricciones matemáticas y su principio básico es considerar que todo requisito de calidad puede ser expresado mediante una restricción sobre un atributo de calidad. Por ejemplo el requisito "El tiempo medio entre dos fallos consecutivos será de al menos 6 minutos", puede interpretarse como la restricción `TTF` $\geq$ `6 min`. De este modo, los problemas enunciados en la Sección 2 de manera intuitiva se pueden enunciar formalmente como problemas de satisfacción de restricciones o CSP (Constraint Satisfaction Problem) [5] y resolverse con técnicas habituales en la programación con restricciones [12].

Para facilitar la traducción de un requisito expresado en lenguaje natural a su restricción equivalente a la vez que se minimizan los problemas inherentes al uso del lenguaje natural, QRL utiliza patrones lingüísticos en el mismo sentido y con la misma forma que se utilizan en [9] (ver Figura 2). Un patrón lingüístico, o abreviadamente *patrón–L*, es una frase estándar que representa a todos aquellos requisitos que se pueden expresar respetando la forma y el fondo de la frase. En la notación usada para describir los patrones–L, las palabras o frases entre <y> deben ser convenientemente reemplazadas, mientras que palabras o frases que se encuentren entre {y} y separadas por comas representan opciones de las que se deberá escoger una. La notación para escribir restricciones es muy similar a la empleada para la formación de patrones–L.

El principio de definición dual de un requisito "frase en lenguaje natural / restricción" pese a su simplicidad tiene una gran eficacia, pues hasta la fecha no hemos encontrado ningún requisito de calidad que no haya podido ser expresado mediante una restricción. Además, resulta de gran ayuda para definir los esquemas de transformación necesarios para resolver los problemas de definiciones incompatibles que pueden aparecer cuando se utilizan diferentes catálogos de atributos de calidad.

Por ejemplo, supongamos que el "tiempo medio entre fallos" (`TTF`) se define en un primer modelo como el número de días que transcurre entre dos fallos consecutivos ($\texttt{TTF}_1$), y como el número de fallos al año en un segundo modelo ($\texttt{TTF}_2$). Es evidente que si tenemos que comparar $\texttt{TTF}_1$ con $\texttt{TTF}_2$ es necesario definir una relación que ligue ambas definiciones a fin de hacer posible la automatización. Supongamos que en nuestro esquema de transformación definimos la ecuación $\texttt{TTF}_1 \times \texttt{TTF}_2 = 365$, de este modo un problema como es el de comparar si $\texttt{TTF}_1 > \texttt{31 dias}$ satisface la restricción $\texttt{TTF}_2 > \texttt{12 errores/año}$ se convierte en el problema de comparar si $\texttt{TTF}_1 > \texttt{31 dias}$ satisface la restricción $\texttt{TTF}_1 > \texttt{30.42 dias}$.

4. Especificación de Requisitos

Con QRL se pueden especificar dos tipos de documentos: las *condiciones* y las *ofertas*. Las condiciones expresan el nivel de calidad que debe proporcionar un determinado servicio WEB para que el cliente acepte utilizarlo. Por su parte, las ofertas expresan el nivel de calidad que ofrece un determinado servicio WEB.

La Figura 3 muestra las condiciones que debe satisfacer los servicios WEB que implementen la interfaz `IVideoServer` y la interfaz `ITrailersBank`. Como puede observarse, al principio del documento se indican los catálogos de atributos que se utilizarán para especificar requisitos y la definición de las interfaces sobre las que estos se definirán (cláusula **products**). Desde el punto de vista de QRL, el lenguaje base utilizado para definir la interfaz que debe implementar un servicio WEB es completamente irrelevante. Esta decisión de diseño aumenta la aplicabilidad de QRL ya que puede ser utilizado para definir requisitos sobre cualquier "producto" sobre el que sea posible una caracterización con atributos de calidad (ver Sección 8).

QRL permite definir requisitos abstractos o estereotipos. Un requisito abstracto debe ser considerado como una facilidad sintáctica que permite disponer de definiciones de requisitos que todavía no se han asociado a ningún producto concreto. Pueden ser definidos tanto en un catálogo de atributos como en un documento de requisitos. Además, es posible definir nuevos requisitos refinando estereotipos previamente definidos.

Finalmente, la cláusula **conditions** permite definir los requisitos concretos de cada interfaz `IVideoServer`, y en general de cualquier producto definido en la Sección **products**. Dichos requisitos pueden definirse tanto para todas las operaciones de la interfaz (en general de los subelementos que definen un producto) como a operaciones concretas. La secciones de una oferta son las mismas que para las condiciones salvo que la palabra clave **conditions** se sustituye por **offer**. En las siguientes secciones comentamos con mayor detalle las posibilidad que QRL ofrece para definir requisitos de calidad.

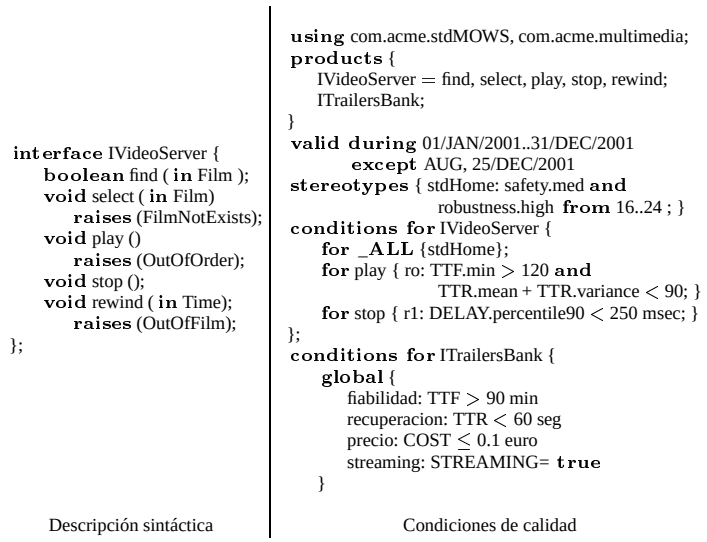

Figura3. Documento de condiciones para las interfaces IVideoServer e ITrailersBank

4.1. Restricciones Unidimensionales

El formato básico de las restricciones que se puede expresar en QRL es "`atributo` θ `valor`". Donde $\theta \in \{<, \leq, =, \neq, \geq, >, \in, \subset, \subseteq, \supset, \supseteq\}$, y valor un valor válido del dominio del atributo. Siguiendo este formato y tomando como referencia los atributos indicados en los catálogos de la Figura 2, las siguientes restricciones serían validas: c_1: `TTR` $\leq$ `10`, c_2: `SFAIL` $=$ `halt`, c_3: `FMASK` $\subseteq$ `{late, value}`. Como puede observarse se trata de restricciones en las que sólo interviene un atributo de calidad.

4.2. Caracterización Estadística

A veces, puede resultar imposible para el proveedor especificar con una medida absoluta el nivel de calidad que ofrece su servicio. Por ejemplo, el tiempo exacto que tardará un servicio WEB en responder a una petición es algo imposible de determinar, pues depende de muchas variables. No obstante, siempre es posible someter el servicio a un banco de pruebas tras el cual se disponga de la información necesaria para caracterizar estadísticamente el tiempo de respuesta: valor medio, desviación típica, etcétera.

Para especificar un atributo de manera estadística, QRL propone un conjunto de palabras reservadas que representan las medidas de localización más habituales en distribuciones estadísticas. En concreto son: el valor medio (`mean`), la desviación típica (`variance`), percentiles (desde `percentile 1` hasta `percentile 99`), el valor máximo (`max`) y el valor mínimo (`min`). Por último, también es posible especificar valores relativos a la frecuencia relativa (`frecuency`). A continuación indicamos algunos ejemplos del uso de la caracterización estadística:

`TTF.min` $>$ `10`, indica que en el peor de los casos, transcurrirán más de 10 segundos entre dos fallos consecutivos.

`TTR.frecuency [5,10]` $>$ `35%`, indica que el tiempo de recuperación tras un error está entre 5 y 10 segundos en el 35 % de los casos.

`TTF.percentile 80` $>$ `5`, indica que el tiempo entre fallos será mayor que 5 en al menos el 20 % (100 − 80) de los casos.

4.3. Restricciones Multidimensionales

Existen situaciones en las que resulta especialmente útil poder expresar restricciones sobre varios atributos de calidad simultáneamente. 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.

En este tipo de restricciones, resulta especialmente útil poder utilizar intervalos numéricos, pues reduce el tamaño de las especificaciones. Por ejemplo, `(TTR.mean` $+$ `TTR.variance)` $<$ `52` **and** `(TTR.mean` $-$ `TTR.variance)` $>$ `48` se puede reescribir como `(TTR.mean` $+$ `TTR.variance)` $\in$ `(48, 52)`.

4.4. Restricciones No Lineales

Sin duda alguna, uno de los valores añadidos de cualquier tipo de servicio es su política de precios. En el mundo de la telefonía móvil, la tarificación por segundos supuso una gran ventaja para los usuarios y la captación de un gran número de éstos para los primeros operadores de telefonía que adoptaron esta política de precios. En el caso de los servicios WEB, la adopción de este tipo de políticas puede conducir a la caracterización mediante funciones no lineales.

Pensemos en aquellos servicios WEB que tienen un consumo de recursos que depende del tamaño de la entrada. Por ejemplo, para un servicio de conversión de imágenes, está claro que convertir una imagen de 1 MB consume muchos más recursos que convertir una imagen de 1 KB, por tanto, parece lógico que el coste en el primer caso sea mayor que en el segundo. Pero, ¿cómo podemos reflejar esta realidad en el nivel de calidad proporcionado por el servicio? Con QRL existen varias posibilidades:

- **Definiendo los atributos de manera relativa**: por ejemplo, el atributo DELAY definido como **domain: real** KB/sec, permite especificar el tiempo de respuesta en función del tamaño de la imagen que se desea convertir.
  El inconveniente de definir los atributos de manera relativa es que para caracterizar propiedades de calidad bien conocidas (DELAY, COST, etcétera), necesitamos definir nuevos atributos para cada problema. Por ejemplo, supongamos un servicio WEB que encuentra el proveedor que ofrece la cesta de la compra más económica. Con toda probabilidad, su tiempo de respuesta dependerá del número de productos a comprar (n), por lo que definición del atributo DELAY del convertidor de imágenes no resulta adecuada ya que su unidad (KB/sec) no tendría sentido en este caso.
- **Definiendo restricciones no lineales**: existe una posibilidad que resuelve el problema de la definición relativa de atributos y no es otra que indicar el valor de un atributo mediante una función, generalmente no lineal, que dependa del tamaño de la entrada.
  Para el servicio que busca la cesta de la compra más económica, puede resultar más útil especificar su tiempo de respuesta con una restricción del tipo: DELAY.max $<$ log(n)+5 sec, donde n es el numero de productos a comprar, y el significado de DELAY el mismo que tenía en catálogo de atributos comunes. Las funciones que pueden ser utilizadas por las expresiones en QRL son: la suma, el producto, la división, la potencia, la exponencial, el logaritmo decimal, el valor absoluto y el valor máximo de un conjunto de valores.

4.5. Períodos de Vigencia

Continuando con el símil del sistema telefónico, nos encontramos con que la tarificación por franjas horarias puede ser llevada al mundo de los servicios WEB, de modo que el precio por usar un determinado servicio WEB u otros atributos de calidad, dependan del momento del día, o de la época del año. Por ejemplo, atributos como el tiempo de respuesta o la productividad dependen, entre otros factores, de la carga del sistema, que a su vez, pueden depender del momento del día.

QRL permite expresar para cada requisito el período temporal en el que dicho requisito debe ser satisfecho (si es un requisito de un documento de condiciones) o durante el que se garantiza su cumplimiento (si es un requisito de una oferta). En ambos casos dicho período se puede denominar simplemente como período de vigencia o validez.

QRL también permite establecer un período de vigencia para el documento de requisitos en su conjunto (cláusula **valid** de la Figura 3), que siempre deberá ser más amplio que cualquier período asociado a un requisito del documento. Las posibilidades para expresar un período de vigencia son muy variadas, a continuación indicamos algunas de estas posibilidades:

from 8..16 **on** MON..FRI **during** 2001..2003, representa al período comprendido desde las 8:00 horas hasta las 16:00, de lunes a viernes, desde el 1 de enero de 2001 hasta el 31 de diciembre de 2003.

from 8..14:30,16..18:30, representa al período que comprende a todos los días de la semana durante todo el intervalo de fechas indicado por el período de vigencia del documento.

```
using com.acme.stdMOWS;
using com.acme.multimedia;
products {p1: ITrailersBank;}
conditions for ITrailersBank {
    global {
        fiabilidad: TTF > 90 min
        recuperacion: TTR < 60 seg
        precio: COST ≤ 0.1 euro
        stream: STREAMING= true
    }
negotiation {
    servPocoFiable {
        weaken
            p1.fiabilidad: TTF > 60;
        strenght
            p1.recuperacion: TTR < 30;}
    servSinStreaming {
        weaken
            p1.stream: STREAMING= false;
        strenght
            p1.precio: COST ≤ 0.03 euro; }
    servCaro {
        weaken
            p1.precio: COST ≤ 0.3 euro;
        strenght
            p1.idioma: SUBT ⊇ {english}
            and LANG ⊇ {english, spanish}; }
preferences {
    p1.TTR.mean= 30,
    p1.TTF.mean=70, p1.CONEX=90;
 }
}
```

a) Condiciones

```
using stdMOWS.lsi.us.es;
using com.acme.multimedia;
products {p1: ITrailersBank;}
offer for ITrailersBank {
    global// provided by zipi
        c1: TTF = 120 min;
        c2: TTR = 50 seg;
        c3: COST = 0.02 euro; on MON..FRI
        c4: COST = 0.05 euro; on SAT,SUN
        c5: STREAMING= false;
}
using com.acme.stdMOWS;
using com.acme.multimedia;
products {p1: ITrailersBank;}
offer for ITrailersBank {
    global// provided by velazquez
        c1: TTF = 360 min;
        c2: TTR = 50 seg;
        c3: COST = 0.4 euros;
        c4: STREAMING= true;
        c5: MEDIA ⊇ {BroadBand};
negotiation {
    modem {
        weaken p1.c3: COST = 0.2 euro;
        strenght p1.c5: MEDIA ⊇ {BroadBand,Modem}; }
    }
    T1 {
        weaken p1.c3: COST = 0.3 euro;
        strenght p1.c5: MEDIA ⊇ {BroadBand,T1}; }
    }
}
```

b) Ofertas

Figura4. Especificación de reglas de negociación en QRL

`on MON..SAT during JAN..JUL,SEP..DIC`, representa al período que abarca todos los días de lunes a sábado de todo el año excepto el mes de agosto.

5. Reglas de Negociación

El resultado final de la comprobación de la conformidad formará parte del documento que recoge las responsabilidades del sistema y del proveedor del servicio. En el contexto de los ASPs (*Application Service Providers*) dicho documento se conoce como SLA (*Service Level Agreement*) [13]. Por desgracia, las probabilidades de alcanzar un acuerdo "a la primera" son muy bajas, pues un administrador siempre busca servicios baratos y de calidad y los proveedores siempre intentan vender muy caro los servicios de alta calidad y a un precio más barato los servicios de menor calidad.

Una posible solución para aumentar las probabilidades de encontrar ofertas que satisfagan las condiciones de un determinado sistema se encuentra en el uso de *reglas de negociación*. Por ejemplo, imaginemos que en las condiciones del ejemplo de la Figura 1 se especifica que si el coste de utilización es inferior a 0.03 euros se puede proceder a la contratación del servicio aunque éste no soporte `streaming`. En tal caso, el sistema podría contratar el servicio proporcionado por `zipi`, pues ahora su oferta sería conforme a las nuevas condiciones.

Las reglas de negociación no son aplicables únicamente a las condiciones, su uso en las ofertas hace posible que los proveedores ofrezcan la misma la funcionalidad con diferentes niveles de calidad a un precio diferente cada una. Por ejemplo, el proveedor `velazquez` tiene tres tarifas diferentes, dependiendo de la velocidad con la que se desee sea descargada la película.

Es evidente que el uso de reglas de negociación bilaterales (en las condiciones y en las ofertas) obliga a que el problema sea resuelto con mecanismos automáticos, pues en el peor de los casos es necesario realizar 2^{n+m} comprobaciones para alcanzar un acuerdo, en donde n y m corresponden al número de reglas de negociación del documento de condiciones y ofertas respectivamente.

Para que una regla sea considerada válida debe debilitar (**weaken**) uno o varios requisitos y reforzar (**strenght**) o añadir otros (**further**). Las condiciones y ofertas de la figura 4 muestra la especificación de varias reglas de negociación.

6. Reglas de Preferencia

La optimalidad en los MOWS se define como la capacidad de seleccionar de entre un conjunto de servicios WEB candidatos (implementan la misma interfaz con una oferta de calidad que satisface las condiciones del cliente), el servicio que satisface de manera óptima un conjunto de requisitos de calidad [4].

Por ejemplo, supongamos la existencia de dos servicios WEB que implementan la interfaz `ITrailersBank` con el nivel de calidad requerido por el sistema. ¿Qué servicio resultará seleccionado? Es evidente que dependerá de los criterios establecidos por el diseñador de la plataforma de ejecución (p.e. el servicio de *trading* en el caso de CORBA), los cuales no siempre tendrán porqué coincidir con los más adecuados para cada situación.

QRL permite deshacer estas situaciones de "empate" de acuerdo con las preferencias establecidas por el diseñador de la aplicación cliente y no de la plataforma de ejecución. La figura 4 muestra como QRL permite asociar pesos (cláusula **preferences**), donde el peso de un atributo refleja el grado de importancia (del 1 al 100) para el diseñador. En este caso, se le da más importancia al coste de conexión (`p1.CONEX=90`) que al tiempo medio entre fallos (`p1.TTF=70`), que a su vez es más importante que el tiempo medio de recuperación ante fallos (`p1.TTR=30`). Los atributos que tienen pesos asociados se les conoce como *decisorios*.

Con toda esta información, es posible interpretar el problema de garantizar la optimalidad como el problema de encontrar el servicio WEB que maximiza la función:

$$f(a_1, \ldots a_n) = \sum_{i=1}^{n} w_i a_i \quad que\, esta\, sujeta\, a\, \mathcal{C}$$

donde $\mathcal{A} = \{a_1, \ldots a_n\}$ es el conjunto de atributos decisorios, $\mathcal{C} = \{c_1, \ldots c_m\}$ el conjunto de restricciones que deben satisfacerse sobre $\mathcal{A}$, $\mathcal{W}$= $\{w_1, \ldots w_n\}$ el conjunto de pesos asociado a $\mathcal{A}$, y $\mathcal{S} = \{S_1, \ldots S_p\}$ el conjunto de servicios WEB candidatos.

7. Trabajo Relacionado

Existen dos grandes aproximaciones al tratamiento sistemático de requisitos de calidad: las *orientadas al producto* y las *orientadas al proceso* [8, p.3], contando cada una de ellas con soportes lingüísticos diferentes. QRL ofrece soporte para tratar automáticamente algunos problemas relacionados con las propiedades de calidad de servicios WEB, por lo que se trata de una propuesta orientada al producto. A continuación, revisamos las principales aportaciones de QRL comparándolo con varios lenguajes de especificación de requisitos de calidad orientados al producto, principalmente NOFUN [14, 15] y QML [16].

- **Modelo ontológico formal**. La definición formal de atributo de calidad tal como se propone en QRL no la hemos encontrado en ningún otro lenguaje, si bien la definición de un atributo de calidad como una restricción matemática ha sido tomada de QRL y el uso de patrones lingüísticos de [9]. Además de las ventajas ya comentadas en secciones anteriores, la doble definición que exige QRL es un buen punto de partida para abordar el tratamiento automático de requisitos de calidad expresados en lenguaje natural.
- **Interpretar la comprobación de la conformidad como un CSP**. La interpretación de la comprobación de la conformidad entre requisitos que realizan los autores de QML en [16, p.18] resulta bastante intuitiva pero su definición formal deja mucho que desear. Los autores afirman que "la comprobación de la conformidad depende de la naturaleza del atributo (dimensión en su terminología) sobre el que se define", lo cual, siendo cierto desde un punto de vista intuitivo, no lo es desde un punto de vista matemático [12], limitando innecesariamente la capacidad expresiva del lenguaje. Algunas de las limitaciones a las que nos referimos son:
  - La imposibilidad de expresar requisitos definidos sobre varios atributos, por ejemplo $\texttt{TTF.med} + \texttt{TTF.variance} < 70$.
  - No es posible definir requisitos que establezcan dos restricciones diferentes sobre el mismo atributo. Por ejemplo, no es posible especificar el requisito $\texttt{COST} \geq 10 \textbf{ and } \texttt{COSTE} \leq 20$.
  - No es posible definir restricciones no lineales, ni tampoco períodos de vigencia.

  En nuestra opinión, extender QML para solucionar las anteriores limitaciones no es una tarea en absoluto trivial, pues no se trata de aumentar el número de construcciones del lenguaje o de añadir facilidades sintácticas, sino que supone modificar el núcleo de su semántica estática tal y como está definida en [17].
- **Reglas de negociación**. En [18] se describe una propuesta para expresar de manera compacta múltiples preferencias de clientes, que pueden ser interpretada como un caso particular de las reglas de negociación de QRL. Su principal inconveniente es que no contempla la posibilidad de que el proveedor el proveedor también presente reglas de negociación.
- **Períodos de vigencia**. Hasta la fecha, no hemos encontrado ningún lenguaje que permita especificar períodos de vigencia a la especificación de requisitos, ni tampoco es posible utilizar propuestas procedentes del campo de la programación lógica con restricciones temporales, pues no son adecuadas para el tratamiento que requiere la comprobación de la conformidad cuando existen períodos de vigencia.

8. Conclusiones y trabajo futuro

En el futuro mercado de servicios WEB la gestión automática de los requisitos de calidad jugará un papel clave, pues permitirá construir sistemas con costes de explotación óptimos y con niveles de calidad garantizados. En este artículo hemos descrito uno de los principales problemas a los que se enfrenta este nuevo tipo de sistemas construidos en base a servicios WEB y hemos presentado una propuesta que ofrece una solución formal a la vez que práctica y eficaz para resolverlo.

Este trabajo presenta una aportación importante en el terreno de los lenguajes formales de especificación de requisitos de calidad. Creemos que el principio de dualidad del requisito "frase en lenguaje natural/restricción" salva algunos de los principales inconvenientes del uso de métodos formales: la dificultad para aprender a usar el método, el salto semántico entre la especificación en lenguaje natural y la notación formal y por último, la ineficiencia de la implementación generada automáticamente. En primer lugar porque el uso de restricciones es conocidos por todos los ingenieros de software, en segundo lugar porque la distancia semántica entre un patrón lingüístico y un requisito expresado como restricción es mínima y finalmente porque con los actuales motores de resolución de restricciones se puede derivar automáticamente la implementación de un CSP tan eficiente o más que muchas soluciones *ad-hoc*.

Las principales aportaciones de QRL son: i) el uso de un modelo ontológico para definir atributos de calidad que hace posible la comprobación automática de propiedades y facilita la comprobación entre documentos de requisitos que utilizan diferentes catálogos de atributos ii) la interpretación de la comprobación de la conformidad como un problema de satisfacción de restricciones lo que aumenta considerablemente su capacidad expresiva iii) la posibilidad de especificar requisitos "temporalmente" sensibles y iv) la capacidad de especificar reglas de negociación.

Actualmente estamos extendiendo este trabajo en dos direcciones. Por una parte, aplicando estas ideas en la evaluación automática 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 y puede interpretarse como un problema de minimización. Por otra parte, estamos desarrollando un entorno de ejecución y administración de MOWS que garantiza la conformidad y la optimalidad a fin de obtener los primeros resultados experimentales de nuestra propuesta.

Referencias

[1] Microsoft. Building ASP.NET Web Services. White Paper, 2001. Disponible en `http://msdn.microsoft.com/webservices/`.

[2] Sun Microsystems, Inc. Open, Smart Web Services. White Paper, 2001. Disponible en `http://www.sun.com/software/sunone`.

[3] R. Corchuelo, A. Ruiz, J. Mühlbacher, and J.D. García-Consuegra. Object-Oriented Business Solutions. In *Chapter 18 of ECOOP'2001 Workshop Reader, LNCS nº (to appear)*. Springer-Verlag, 2001.

[4] A. Ruiz, R. Corchuelo, A. Duran, and M. Toro. Automated support for quality requirements in web-services-based systems. In *Proc. of the 8th IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS'2001)*, Bologna, Italy, 2001. IEEE-CS Press.

[5] K. Marriot and P.J. Stuckey. *Programming with Constraints: An Introduction*. The MIT Press, 1998.

[6] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide*. Addison–Wesley, 1^a edition, 1999.

[7] B. Meyer. *Object–Oriented Software Construction*. Prentice Hall, 1997.

[8] L. Chung, B.Ñixon, E. Yu, and J. Mylopoulos. *Non–Functional Requirements in Software Engineering*. Kluwer Academic Publishers, 2000.

[9] A. Durán, B. Bernárdez, A. Ruiz, and M. Toro. A Requirements Elicitation Approach Based in Templates and Patterns. In *WER'99 Proceedings*, Buenos Aires, 1999.

[10] World Wide Web Consortium. XML Schema Part 0: Primer, W3C Working Draft . Technical report, World Wide Web Consortium, Sep 2000. available at http://www.v3.org/TR/xmlschema-0/.

[11] F. Cristian. Understanding Fault–Tolerant Distributed Systems. *Communications of the ACM*, 34(2), 1991.

[12] A. Ruiz, R.M. Gasca, R. Corchuelo, and M. Toro. Aplicando técnicas de satisfacción de restricciones para comprobar la conformidad de aplicaciones WEB. In *Enviado a la Conferencia de la Asociación Española para la Inteligencia Artificial (CAEPIA'01)*, 2001.

[13] ASP Industry Consortium. An Overview of The Guide to Service Level Agreements. White Paper, 2001. Disponible en `http://www.aspindustry.com`.

[14] X. Franch. Systematic formulation of non–functional cahracteristics of software. In *Proc. of the International Conference on Requirements Engineering (ICRE'98)*, Colorado, USA, April 1998.

[15] P. Botella, X. Burgués, X. Franch, M. Huerta, and G. Salazar. Modeling Non-Functional Requirements. In *Actas de las Jornadas de Ingeniería de Requisitos Aplicada (JIRA)*, Jun 2001. Disponible en `http://www.lsi.us.es/ amador/JIRA/JIRA.html`.

[16] S. Frolund and J. Koistinen. Quality–of–Service Specification in Distributed Object Systems. *Distributed Systems Engineering Journal*, 5(4), 1998. Disponible como informe técnico HPL–98–159 en `http://www.hpl.hp.com/techreports`.

[17] S. Frolund and J. Koistinen. QML: A Language for Quality of Service Specification. Technical Report HPL-98-10, Hewlett–Packard, 1998. Disponible en `http://www.hpl.hp.com/techreports`.

[18] C. Becker, K. Geihs, and J. Gramberg. Representing quality of service preferences by hierarchies of contracts. In *In Proceedings of Elektronische Dienstleistungswirtschaft und Financial Engineering (FAN'99)*, Augsburg/Germany, 1999.