WeSSQoS: Un Sistema SOA para la Selección de Servicios Web según su Calidad

Oscar Cabrera[1,2], Marc Oriol[1], Lidia López[1], Xavier Franch[1], Jordi Marco[1], Olivia Fragoso[2], René Santaolaya[2]

[1] Universitat Politècnica de Catalunya (UPC)
c/Jordi Girona, 1-3, E-08034 Barcelona, Spain
{moriol, llopez, franch, jmarco}@lsi.upc.edu
[2] Centro Nacional de Investigación y Desarrollo Tecnológico (CENIDET)
Interior Internado Palmira S/N, Col. Palmira, CP 62490, Cuernavaca, Morelos, México
{cabrera_bejar07, ofragoso, rene}@cenidet.edu.mx

Resumen. Los Servicios Web (WS) se han convertido en una tecnología altamente utilizada en el desarrollo de sistemas software. Una de sus problemáticas más importantes es la selección de los WS más apropiados para satisfacer los requisitos de dichos sistemas. Si consideramos los requisitos no funcionales (NFR), la calidad de servicio de los WS contiene la información necesaria para analizar dicha satisfacción. En este artículo se describe el sistema WeSSQoS para la ordenación de WS según su grado de satisfacción de los NFR, calculable a partir de la calidad de servicio de dichos WS, que puede declararse en el WSDL mismo o bien calcularse dinámicamente mediante monitorización. Esta información acerca de la calidad puede provenir de diversas fuentes (diferentes repositorios WSDL, diferentes monitores, etc.). La arquitectura de WeSSQoS permite la coexistencia de diversos algoritmos de ordenación de los WS, si bien en este artículo nos centramos en uno de ellos que usa la distancia euclidiana como criterio de ordenación.

Palabras clave: Arquitecturas Orientadas a Servicios (SOA), Selección de Servicios Web, Calidad de Servicio (QoS), Requisitos no Funcionales (NFR).

1 Introducción

Los Servicios Web (WS[1]) [1] son tecnologías que integran un conjunto de protocolos y estándares para intercambiar datos entre aplicaciones de software desarrolladas en distintos lenguajes de programación y ejecutadas en cualquier plataforma. Los estándares abiertos que utilizan son precisamente los que dotan a éstos de interoperabilidad, destacando: XML, SOAP, HTTP, WSDL y UDDI.

Los WS se han convertido actualmente en una tecnología de referencia en la implementación de software de todo tipo. Este éxito se ha traducido en la proliferación de WS, de manera que para una funcionalidad determinada pueden encontrase no ya docenas, sino centenares o incluso miles de WS, ora accesibles separadamente, ora residentes en catálogos. Si bien tal proliferación incrementa las

[1] En este artículo se utilizan abreviaturas de los términos ingleses por su mayor difusión.

posibilidades de encontrar WS ya existentes para satisfacer una necesidad dada, al mismo tiempo provoca diversas problemáticas y entre ellas destacamos precisamente el problema de la selección del WS más apropiado para un contexto de uso [2], que se ha convertido en un tema de investigación esencial en este ámbito. Normalmente este problema se estudia en relación con los requisitos establecidos por el cliente, es decir, se selecciona el WS que "mejor" satisface los requisitos del cliente.

Por lo que se refiere a los requisitos, consideramos la distinción clásica entre requisitos funcionales y requisitos no funcionales [3]. Por el lado de los requisitos funcionales, debe validarse que un WS cumple con la funcionalidad propiamente esperada por el cliente. Por otro lado, los requisitos no funcionales (NFR) se refieren a la calidad de servicio (QoS) que ofrece el WS, es decir, características propias del WS para poder ofrecer una cierta funcionalidad: coste de uso, tiempo de respuesta, etc. Normalmente, la expresión de los NFR en términos de QoS cristaliza en acuerdos de nivel de servicio (SLA), por lo que finalmente la comprobación de un NFR en un WS consiste en comprobar que la QoS de dicho WS satisface aquellas cláusulas del SLA que hacen referencia a los conceptos inherentes a tal NFR.

Este trabajo de investigación se centra en la ordenación de WS pertenecientes a un mismo dominio de software (que supondremos que se ha determinado previamente en función de los requisitos funcionales) según su grado de satisfacción de los NFR establecidos. Básicamente los puntos a determinar son los siguientes: cuáles son los tipos de NFR soportados; cómo se expresan dichos NFR; cuál es la medida de satisfacción de un NFR en un WS dado; cómo se combinan dichas medidas para ordenar los WS según su adecuación al conjunto de NFR; cómo se obtiene la QoS de un WS; dónde reside dicha QoS.

En concreto, en este trabajo se presenta WeSSQoS (Web Service Selection based on Quality of Service), un sistema para la selección de WS basado en QoS. WeSSQoS propone una arquitectura SOA que acoge en su núcleo uno o más algoritmos de cómputo de la adecuación de un WS respecto a los NFR. A modo de ejemplo, en este artículo utilizamos un algoritmo que mide la adecuación en términos de distancia euclidiana. Los NFR se expresan mediante expresiones formuladas sobre atributos de calidad cualesquiera; actualmente, las expresiones declaran los valores deseados para hasta 10 atributos provenientes del modelo de calidad propuesto en [4]. Los NFR se clasifican entre obligatorios y opcionales, pudiendo usarse esta información al ordenar los resultados. WeSSQoS está diseñado para trabajar sobre diversos repositorios de WS, incluso construidos con tecnologías diferentes. Para conocer el comportamiento de los WS respecto a los criterios de selección, puede usarse o bien la descripción de la QoS que eventualmente forma parte de la definición del WS, o bien puede monitorizarse el comportamiento del WS, obteniendo pues la QoS real del WS. En este sentido, compartimos la visión de [5] que propone que sólo los atributos estáticos (p.e., coste) deberían ser definidos a priori mientras que los atributos dinámicos (p.e., disponibilidad) deberían ser obtenidos mediante un sistema de monitorización.

El resto del artículo se organiza como sigue. En la Sección 2 clasificamos algunos sistemas de selección de WS basados en QoS según los criterios citados arriba. En la Sección 3 presentamos la arquitectura de WeSSQoS. En la Sección 4 describimos el algoritmo de ordenación ofrecido actualmente por WeSSQoS. En la Sección 5 presentamos los detalles de implementación y en la Sección 6 describimos los juegos de pruebas utilizados. En la Sección 7 resumimos las conclusiones y el trabajo futuro.

2 Trabajo Relacionado

Existen diversas propuestas de marcos de ordenación y/o selección de WS según su QoS. En la Tabla 1 se presentan algunas de estas propuestas, comparadas según los criterios siguientes:

- *Estilo arquitectónico*. Arquitectura de la implementación. Encontramos arquitecturas basadas en componentes (CBA), arquitecturas orientadas a servicios (SOA) y una combinación de ambas.
- *Atributos*. Atributos de calidad utilizados en los sistemas. En algunos casos se usa un conjunto predeterminado y pequeño de atributos. En otros sistemas, la arquitectura es configurable (i.e., permite tener atributos arbitrarios), si bien en los trabajos presentados se utiliza una muestra de los mismos para validar la propuesta. Destacamos que los trabajos tratan indistintamente con atributos estáticos (cuyo valor no cambia en ejecución) y atributos dinámicos.
- *Datos QoS*: describe si los valores de los atributos de calidad son declarados en la definición del servicio o, en el caso de ser dinámicos, obtenidos mediante monitorización.
- *Multialgoritmo*: Describe si el sistema es capaz de soportar múltiples algoritmos de selección mediante QoS. Según las descripciones dadas, tan sólo el sistema QCWS [6, 7] ofrece esta posibilidad, pero no permite añadir algoritmos externos (por no ser una arquitectura SOA).
- *Multirepositorio*: Describe si el sistema es capaz de obtener los datos de distintos repositorios y combinarlos para extender la cantidad de servicios y atributos de calidad a evaluar. El único sistema que presenta esta característica es el de Al-Masri et al. [5], que obtiene la lista de WS de distintos UDDIs para almacenarlos en el propio sistema. Sin embargo, no se describe un método para combinar dichos servicios.
- *Prototipo disponible*: Indica si existe un prototipo disponible para la comunidad. Aunque en la mayoría de propuestas se describe un prototipo en los artículos, e incluso algunos dispongan de página web como QCWS, sólo hemos encontrado la herramienta disponible en la propuesta de Al-Masri et al.

Tabla 1. Tabla comparativa de los diversos sistemas considerados

Propuesta	Estilo arqui-tectónico	Atributos	Datos QoS	Multial-goritmo	Multire-positorio	Prototipo disponible
[5]	CBA	4 dinámicos 2 estáticos	Est. definidos; din. monitorizados	No	Sí	Sí
[6, 7]	CBA	4 din. + 1 est.	Definidos	Sí	No	No
[2]	CBA	1 din. + 5 est.	Est. definidos; din. monitorizados	No	No	No
[8]	CBA+SOA	5 din. + 1 est.	Definidos	No	No	No
[9]	Sólo algoritmo	Configurable 1 din. + 5 est.	Definidos	No	No	No
[10]	SOA	Config.; 3 din.	Monitorizados	No	No	No
[11]	SOA	3 din. + 2 est.	Definidos	No	No	No

3. Descripción General de la Arquitectura del Sistema *WeSSQoS*

El sistema *WeSSQoS* está estructurado mediante una arquitectura SOA compuesta por los elementos siguientes (v. Fig. 1):

- *QoSSelector*. Servicio que permite obtener el orden de los WS correspondientes a un dominio de software concreto según los NFR solicitados y una lista de repositorios (*QoSRepositoryProxys*) de donde obtener la información de QoS. Este servicio utiliza los diferentes *QoSRepositoryProxys* para obtener la información de QoS de los servicios del dominio disponibles y se la proporciona, junto con los NFR, al servicio *QoSSelectionModel* para obtener la ordenación de los WS candidatos. Una vez obtenida, procesa la información sobre obligatoriedad de los NFR.
- *QoSSelectionModel*. Servicio que clasifica los WS del dominio aplicando un algoritmo de ordenación, como el presentado en la Sección 4. Aunque en la actualidad se utiliza solamente un *QoSSelectionModel* la arquitectura es lo suficientemente flexible como para permitir utilizar otros adicionales que implementen diferentes algoritmos de clasificación (uso del patrón de diseño Estrategia).
- *QoSRepositoryProxy*. Servicio que permite obtener los WS del dominio elegido que residen en un repositorio, junto con su QoS. Inicialmente se han definido dos subclases de *QoSRepositoryProxy*:
  - *DataBank*. Subclase que se caracteriza por obtener la información de QoS del proveedor. Los atributos de calidad están descritos en un WSDL extendido con QoS. En el caso de atributos de calidad dinámicos, como por ejemplo el tiempo de respuesta medio, el valor obtenido es el que el proveedor del servicio se compromete a asegurar.
  - *Monitor*. Subclase que se caracteriza por obtener la información de QoS en tiempo de ejecución mediante técnicas de monitoreo. Tal y como se comentó en la introducción, el monitor trabaja sobre un catálogo predefinido de atributos dinámicos. Esto implica que no dispondrá de información referente a atributos de calidad estáticos como por ejemplo el coste del servicio.

No se descarta la posibilidad de tener otras subclases de *QoSRepositoryProxy*, p.e. un repositorio que use tanto datos dinámicos como estáticos para obtener la QoS.

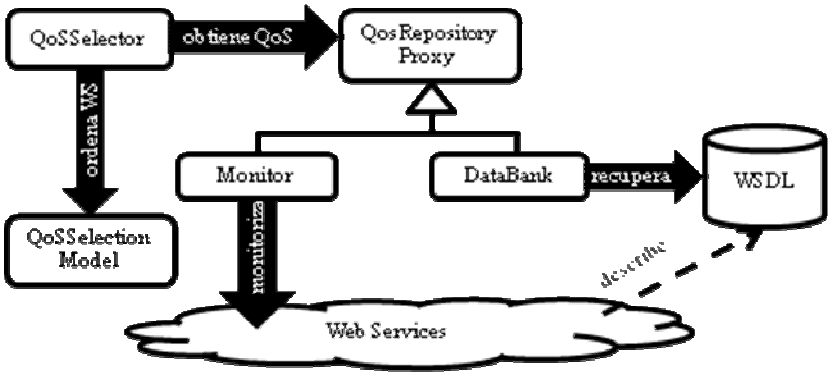

Figura 1. Arquitectura general de WeSSQoS

El caso de uso básico de WeSSQoS es el de la ordenación de WS de un dominio de software, que se traduce en la utilización de la operación *Rank4QoSRepository* del servicio *QoSSelector*. Otros casos de uso son: la utilización de la operación *GetServicesDataFromDomain* del servicio *QoSRepositoryProxy* para, con la información de QoS obtenida, posteriormente aplicar un algoritmo propio de clasificación; o la utilización del servicio *QoSSelectionMode* con datos propios. Por motivos de espacio, no presentamos estos otros casos de uso.

La Fig. 2 muestra el diagrama de secuencia del caso de uso de ordenación de WS. La operación *Rank4QoSRepository* tiene como parámetros de entrada una lista de repositorios, una lista de NFR y un dominio de software, y devuelve la ordenación, según los NFR, de todos los WS del dominio sobre los que hay información en alguno de los repositorios. Los atributos y clases involucradas se presentan en la Fig. 4.

- La lista de repositorios *lProxies* es una lista de *QoSRepositoryProxy*. Cada repositorio se representa por su nombre, el *endPoint* que permite acceder al servicio que obtiene su información (los *endpoints* corresponden a las direcciones donde se encuentran los repositorios, tanto monitores como bancos de datos) y su descripción.
- Cada requisito de la lista de NFR *lReqs* se representa por el nombre del atributo de calidad, el valor que se requiere para el atributo, un booleano que indica si el valor requerido se quiere minimizar (*false*) o maximizar (*true*) y otro booleano que indica si el valor requerido es obligatorio (*true*) o no (*false*). Un valor es obligatorio cuando no puede ser rebasado en caso de que se quiera minimizar o no puede ser inferior en el caso de que se quiera maximizar. Así, el cliente podría querer minimizar el atributo *coste* de manera obligatoria con un máximo de 100 €/mes.
- El dominio se representa por un nombre lo que permite utilizar cualquier ontología de dominios.

La operación *Rank4QoSRepository* realiza una llamada a la operación *GetServicesDataFromDomain* de cada uno de los *QoSRepositoryProxy* (*Databank* o *Monitor*), obteniendo todos los servicios del dominio que contiene cada repositorio junto con toda la información de QoS disponible. Se utiliza el *endPoint* en el momento de crear el objeto *QoSRepositoryProxy*. A medida que se van obteniendo las listas de los servicios, se prepara una lista con la información de QoS de los requisitos solicitados. En el caso de tener información de un mismo requisito en más de un repositorio, utilizamos una política de priorización simple: se utiliza la información del primer repositorio que aparece en la lista que tenga información del requisito (en otras palabras, el orden en la lista de repositorios determina la prioridad para los atributos que se encuentran en más de un repositorio). Una vez que se dispone de la lista de servicios se utiliza la operación *QoSSelectionAlgorithm* del servicio *QoSSelectionModel* para obtener la ordenación a partir de la lista de WS y la lista de requisitos. Finalmente, antes de devolver la lista como resultado, se procesa la información de los servicios sobre los atributos obligatorios.

La Fig. 3 resume las interfaces que han aparecido en el diagrama de secuencia. Los servicios *Monitor* y *DataBank* ofrecen la interfaz de *QoSRepositoryProxy* que puede ser ampliada según las necesidades, por ejemplo con operaciones para registrar un servicio en el *DataBank* o con operaciones para monitorizar un servicio en el *Monitor*. En la operación *GetServicesDataFromDomain*, se ha optado por obtener toda la información de QoS en lugar de sólo la de los requisitos solicitados para aumentar la reusabilidad del servicio *QoSRepositoryProxy*.

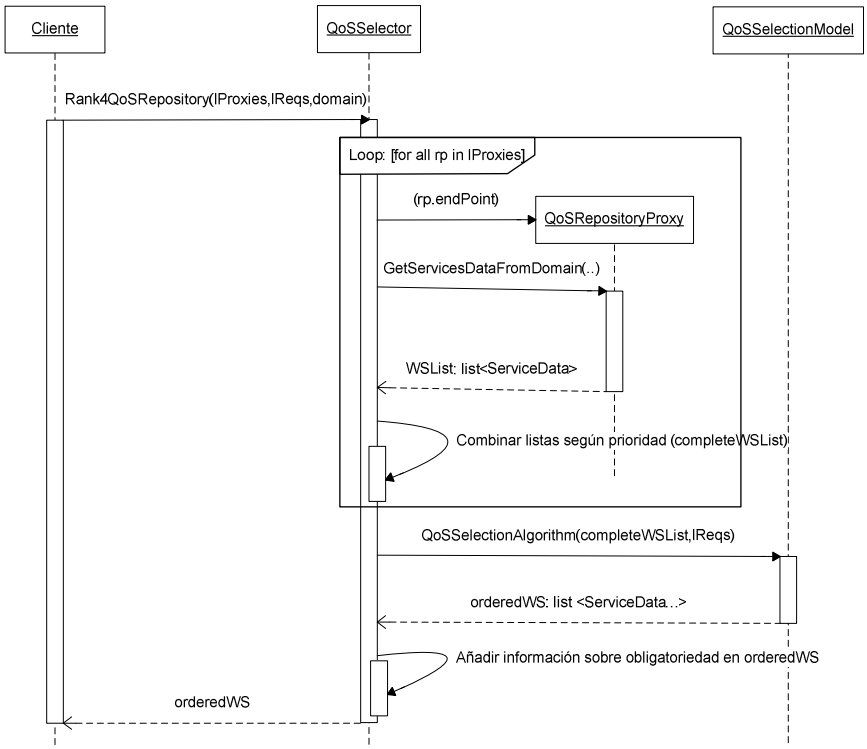

Figura 2. Diagrama de secuencia del caso de uso de ordenación de WS

QoSSelector	QoSRepositoryProxy	QoSSelectionModel
Operación: Rank4QoSRepository **Parámetros de entrada**: lProxies: list <RepositoryProxy> lReqs: list <Stakeholder Requirements> domain: String **Respuesta**: orderedWS: list <ServiceData PriorityResult>	**Operación**: GetServicesDataFromDomain **Parámetros de entrada**: domain: String **Respuesta**: WSList: list <ServiceData>	**Operación**: QoSSelectionAlgorithm **Parámetros de entrada**: CompleteWSList: list<ServiceData> lReqs: list <Stakeholder Requirements> **Respuesta**: orderedWS: list <ServiceData PriorityResult>

Figura 3. Interfaces de los servicios de WeSSQoS

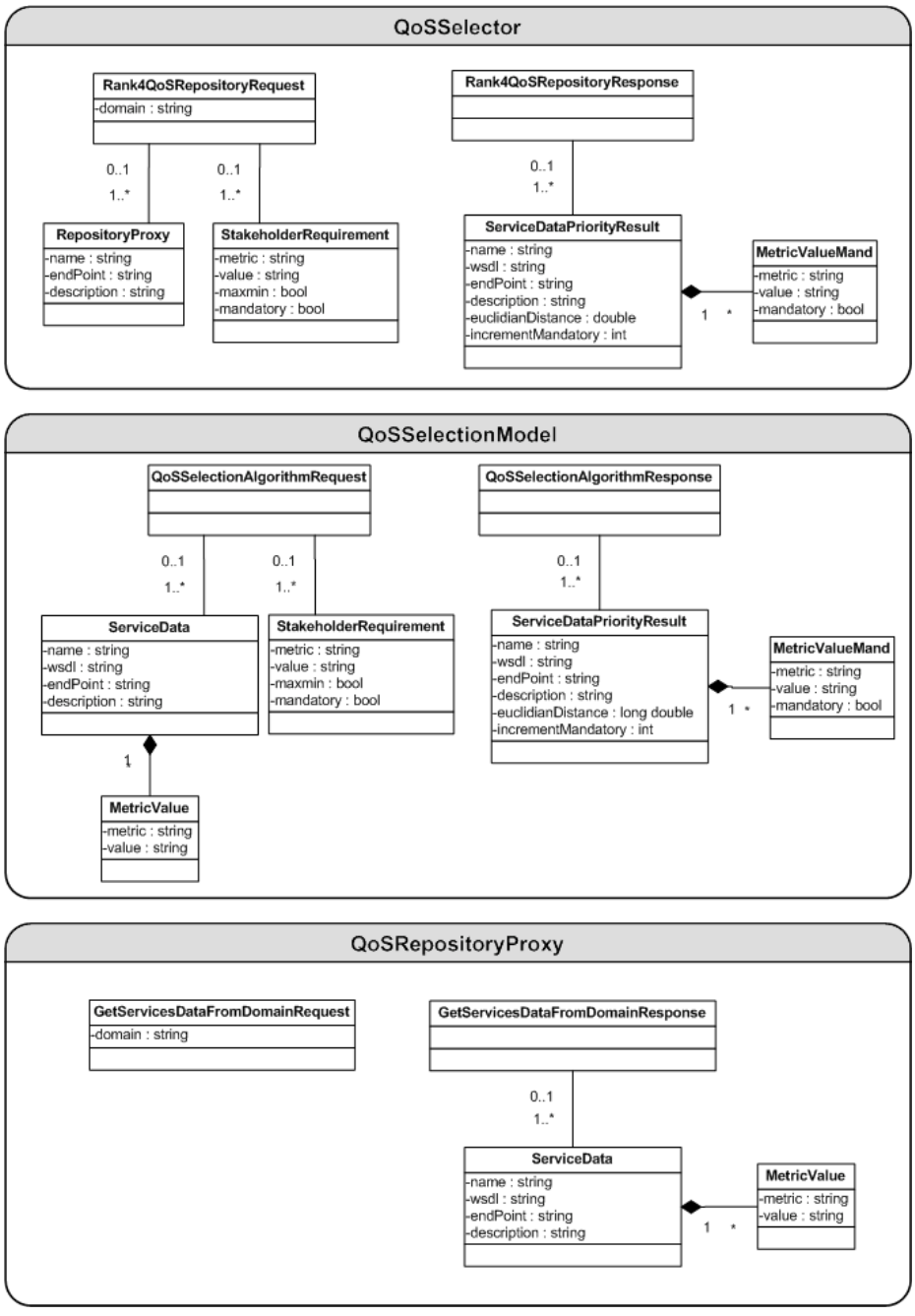

Figura 4. Modelo de datos utilizado en la arquitectura de WeSSQoS

4. Ejemplo de Algoritmo de Ordenación: la Distancia Euclidiana

La arquitectura de WeSSQoS soporta la coexistencia de varios algoritmos de ordenación. La versión actual del sistema implementa el interfaz *QoSSelectionAlgorithm* mediante la función de utilidad Distancia Euclidiana para el proceso de ordenación como se presenta en [9] y que se resume en esta sección. Dados dos vectores A y B, en donde A=(a_1, a_2, a_3,…,a_n) y B=(b_1, b_2, b_3, …,b_n), la distancia euclidiana entre los dos vectores puede calcularse de la siguiente manera:

$$d_{euclid}(A,B) = \sqrt{\sum_{i=1}^{n} |a_i - b_i|^2} \qquad (1)$$

En nuestro contexto, la distancia euclidiana busca las distancias más cortas entre la QoS de cada servicio candidato y los requisitos no funcionales que conforman la especificación del consumidor. El algoritmo de ordenación resultante se representa en el diagrama de flujo de la Figura 5. Se incluye también la consideración de la característica de obligatoriedad que según el diagrama de la Figura 2, no forma parte del algoritmo mismo (que está enmarcado en gris), pero que integramos aquí para dar una explicación más completa. A continuación describimos las fases transitadas.

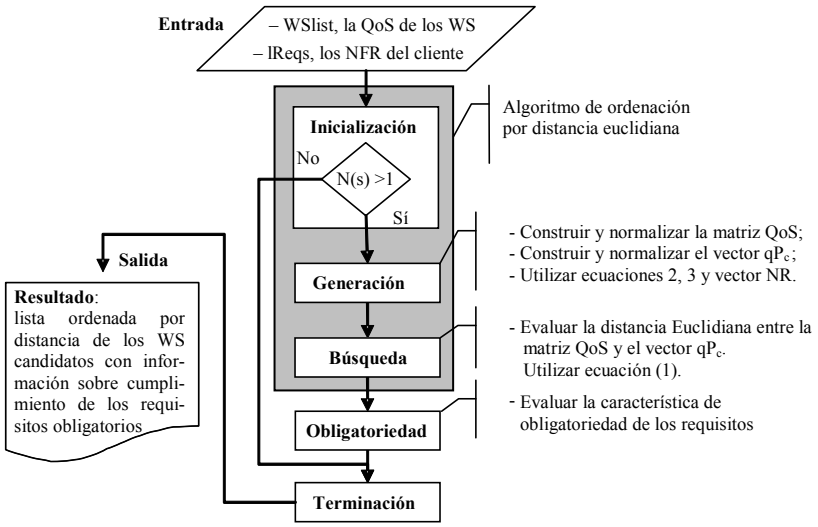

Figura 5. Algoritmo del módulo de selección

Generación. En esta fase se construyen las matrices y vectores requeridos para la aplicación de la función de distancia euclidiana. En concreto, se construye una matriz QoS de tamaño $n \times k$ a partir de la lista *lReqs*, donde n representa el número total de WS del dominio de software en consideración (los WS candidatos) y k representa el número total de atributos de calidad que aparecen en los NFR de la lista. Asimismo, la lista *lReqs* se convierte en un vector qP_c. A continuación, ambas estructuras deben normalizarse con el propósito de compensar las diferentes unidades de medida entre las diferentes propiedades de valores de QoS, y para

establecer los valores en el intervalo [0, 1]. Para ello, se necesita definir un vector NR = {nr_1, nr_2,…, nr_k}. El valor de nr_v, $1 \le v \le k$, puede ser 0 ó 1. El valor 0 indica que qP_v debe ser normalizado usando la ecuación (2) al tratarse de un atributo cuyo valor se está maximizando, mientras que el valor 1 indica que qP_v debe ser normalizado usando la ecuación (3) por el motivo contrario.

$$\mathbf{Norm(qp_{i,v})} = \frac{qp_{i,v_{max}} - qp_{i,v}}{qp_{i,v_{max}} - qp_{i,v_{min}}} \quad (2) \qquad \mathbf{Norm(qp_{i,v})} = \frac{qp_{i,v} - qp_{i,v_{min}}}{qp_{i,v_{max}} - qp_{i,v_{min}}} \quad (3)$$

La Figura 6(a), adaptada de [9], propone unos valores para el vector qP_c con la especificación del consumidor, una configuración de NR acorde con el requisito de maximizar o minimizar y una matriz QoS con información sobre cuatro WS de acuerdo con la entrada *WSlist*. En la Figura 6(b) se presentan la QoS y la especificación del consumidor, normalizadas tras aplicar las ecuaciones (2) y (3) usando el vector NR.

Especificación del consumidor: qP_c = [0.9, 20, 50, 0.9, 1, 200]	Especificación del consumidor normalizada: qP_c = [0.9, 0.0, 0.17, 0.75, 1.0, 0.75]
Vector de normalización: NR = [1, 0, 1, 1, 1, 0]	
QoS para WS candidatos: QoS = $\begin{bmatrix} WS_1 & 0.9 & 10 & 100 & 1 & 0.9 & 500 \\ WS_2 & 0 & 15 & 30 & 0.8 & 0.6 & 100 \\ WS_3 & 0.3 & 5 & 20 & 0.6 & 0.9 & 200 \\ WS_4 & 1 & 20 & 200 & 0.9 & 1 & 300 \end{bmatrix}$	QoS para WS candidatos normalizada: QoS = $\begin{bmatrix} 0.90 & 0.67 & 0.44 & 1.00 & 0.75 & 0.00 \\ 0.00 & 0.33 & 0.06 & 0.50 & 0.00 & 1.00 \\ 0.30 & 1.00 & 0.00 & 0.00 & 0.75 & 0.75 \\ 1.00 & 0.00 & 1.00 & 0.75 & 1.00 & 0.50 \end{bmatrix}$
(a)	**(b)**

Figura 6. Ejemplo del algoritmo de ordenación: fase de Generación

- *Búsqueda*. Una vez normalizadas la matriz QoS y el vector qP_c, se evalúa la distancia euclidiana entre cada servicio (filas de QoS) y los requisitos (vector qP_c) utilizando la ecuación (1). Para el ejemplo de la Figura 6, se muestra el resultado del proceso de búsqueda en la Tabla 2, segunda columna, y la ordenación resultante en la cuarta columna: WS4 tiene la menor distancia euclidiana, pues cumple con la mayoría de los NFR del cliente, por lo que parece el candidato a seleccionar, a la espera de analizar el grado de cumplimiento de los requisitos obligatorios.
- *Obligatoriedad*. Posteriormente al proceso de ordenación, se analiza la información sobre obligatoriedad. Supongamos en el ejemplo de la Figura 6 que todos los requisitos son obligatorios. En la Tabla 2 se muestra en la tercera columna que WS4 cumple tan sólo 3 de los 6 requisitos, al igual que WS1, mientras que WS2 y WS3 los cumplen todos. Si se combinan los resultados de la clasificación por QoS y Obligatorio (el criterio de obligatoriedad prima sobre el valor de QoS, ver quinta columna), el orden ha cambiado, por lo tanto, el cliente debe decidir lo que mejor se adapte a sus necesidades.

Tabla 2. Resultado de la fase de búsqueda sobre el ejemplo de la Figura 6

WS	Distancia euclidiana	Obligatorio	Orden por QoS	Orden por QoS y obligatoriedad
WS1	1.0997	3/6	2	4
WS2	1.4339	6/6	4	2
WS3	1.4191	6/6	3	1
WS4	0.8725	3/6	1	3

5. Descripción del Prototipo Actual

Se ha desarrollado una herramienta que implementa el sistema WeSSQoS, accesible en la url http://appserv.lsi.upc.es/wessqos/ y totalmente operativa.

Para el desarrollo de esta herramienta, se ha escogido el lenguaje de programación Java J2EE. Sobre éste, existe un amplio conjunto de motores de WS, entre ellos destacan Apache Axis, Apache Axis2 y Glassfish. Para la implementación de los servicios que se utilizan en este trabajo se seleccionó Apache Axis2. No obstante, para la fase de pruebas y con el objetivo de demostrar la independencia tecnológica del sistema WeSSQoS, se desarrolló un conjunto de WS a evaluar usando Glassfish como contenedor.

Los servicios colocados en *QoSRepositoryProxy* requieren, además, de un espacio para almacenar los atributos de calidad. En nuestra herramienta, tanto los repositorios estáticos como los dinámicos almacenan la información necesaria en un SGBD MySQL. No obstante, se debe tener en cuenta que cada *QoSRepositoryProxy* tiene su propia base de datos y que cada implementación puede tener el esquema conceptual y SGBD que más se ajuste a sus funcionalidades.

La interfaz del sistema es una aplicación Web que consta de tres secciones descritas a continuación usando los nombres que aparecen en dicho interfaz:

- *Repositories*: es la sección donde se introduce el dominio sobre el que se quiere realizar la búsqueda y los repositorios donde consultar la información. El sistema permite utilizar tanto dominios y repositorios definidos en nuestra herramienta como otros externos. En el caso de un dominio externo se introducirá el nombre que lo identifica en los repositorios. Por otro lado los repositorios se identifican mediante su *endpoint*. Los *endpoints* se visualizan en una tabla al final de la sección, que permite agregar/borrar. Cabe resaltar que el orden en que se añaden los repositorios se utiliza como orden de prioridad para la obtención de la información de un WS del que hay información en más de un repositorio.
- *Stakeholder Requirements*: es la sección donde el usuario de la herramienta introduce los NFR cuya satisfacción se comprueba en los WS de los repositorios identificados en la sección de *Repositories*. Estos NFR se establecen sobre atributos de calidad que pueden ser: los atributos que proporciona el propio sistema WeSSQoS o bien atributos propios del usuario. Obviamente, es responsabilidad del usuario seleccionar atributos cuya descripción o monitorización sea cubierta por los repositorios que ha elegido en la sección *Repositories*. Por cada atributo seleccionado, se debe introducir el valor requerido, especificar si se desea maximizar o minimizar y finalmente especificar si es o no obligatorio a la hora de seleccionar algún servicio (tal y como se describió en la Sección 3).
- *Results*: es la sección donde se muestra la ordenación de los WS de los repositorios tras aplicar el algoritmo elegido (actualmente, el de distancia euclidiana) según su QoS. Los WS pueden ordenarse por dos criterios: considerando sólo la distancia de su QoS respecto los NFR, o bien considerando el número de NFR obligatorios cumplidos y, en caso de empate, distancia respecto NFR. La sección permite también obtener algunas gráficas, visualizar los detalles de los WS, y refinar los NFR para depurar la búsqueda. Por ejemplo, en la Fig. 7 se puede observar el resultado de la ordenación de servicios de un repositorio con 3 requisitos obligatorios en su búsqueda.

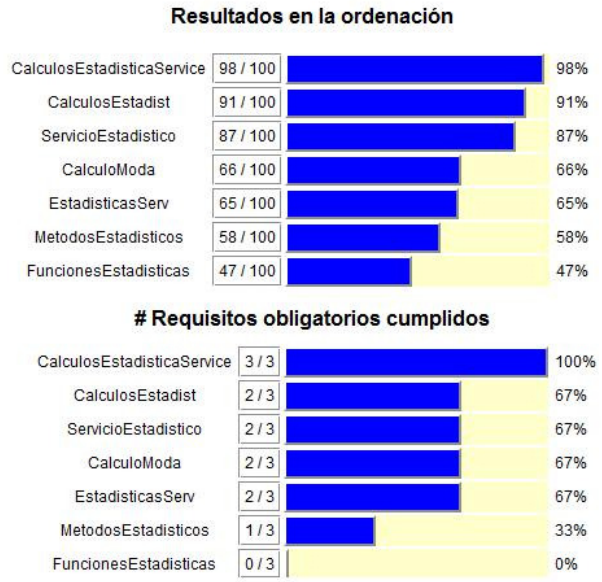

Figura 7. Ejemplo de pantalla de resultados de WeSSQoS

6 Validación

Para poder probar nuestra herramienta, hemos diseñado un escenario en el que ejecutar los casos de prueba previstos. Este escenario también está disponible en la página web de la herramienta (v. sección anterior). El objetivo del escenario es probar las siguientes características de nuestro sistema:

- Gestión de los atributos de calidad. El cliente puede pedir los atributos de calidad que le interese, y éstos pueden estar o no definidos en la información de los WS disponibles. El caso básico se da cuando el cliente pregunta por un subconjunto de los atributos definidos en el repositorio. Nuestro sistema también permite que el cliente se refiera a atributos que no están definidos para los WS, en cuyo caso serán tratados como indefinidos por el algoritmo de ordenación.
- Repositorios donde buscar la información de los WS. Nuestro sistema permite buscar en uno o más repositorios. Cada repositorio puede ser dinámico o estático. Cuando se está utilizando más de un repositorio hemos considerado la casuística siguiente:
    - Los WS de cada repositorio pueden ser diferentes. En este caso se calculará la unión de los servicios de todos los repositorios.
    - En distintos repositorios puede haber información de los mismos WS pero los atributos definidos en los diferentes repositorios pueden no ser los mismos. En este caso el algoritmo de selección combinará los atributos cogiendo los que necesite de cada repositorio.

- En distintos repositorios puede haber información de los mismos WS y de los mismos atributos. En este caso, de los atributos que están en más de un repositorio, se coge la información del repositorio más prioritario.

En la Figura 8 se muestra la implementación de la arquitectura y los datos necesarios para poder realizar todas las pruebas descritas anteriormente. Disponemos de los dos tipos de *QoSRepositoryProxy*. Los dos *Monitor* utilizan Axis, mientras que el *DataBank* (que contiene información sobre dos dominios de WS) utiliza Glassfish. Junto a los *QoSRepositoryProxy* se ha incluido los nombres de algunos de los WS de los que se tiene información. Se han seleccionado servicios cuya información se encuentra en más de un repositorio. También se muestra información sobre algunos atributos para mostrar que puede diferir según el repositorio. En el caso del *Databank*, se mantiene información de todos los atributos que proporciona la herramienta excepto el *CurrentResponseTime* (CRT) y el *CurrentAvailability* (CA). Para los servicios de los *Monitor* se muestran los atributos de los que se dispone información para cada servicio. Aparte del CRT y el CA, algunos servicios también registran información referente al *AverageResponseTime* (ART). Toda esta información permite saber cuál será la combinación de atributos que se tendrá en cuenta según el orden en que se pongan los repositorios. Por ejemplo, si el orden es *Monitor1*, *Monitor2* y *DataBank1*, en el caso del servicio *CalculosEstadist* (que aparece en los 3) los valores de ART, CRT y CA se tomarán de *Monitor1* y el resto de *DataBank1*. En cambio, si el orden fuese *Monitor2*, *Monitor1* y *DataBank1*, el valor de CRT se tomaría de *Monitor2*, los valores de ART y CA de *Monitor1*, y el resto de *DataBank1*.

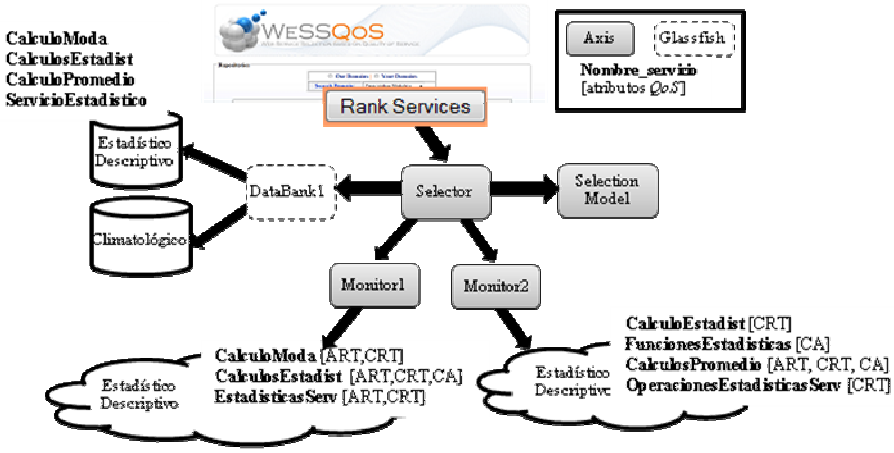

Figura 8. Escenario para las pruebas de WeSSQoS

7 Conclusiones y Trabajo Futuro

En este artículo hemos presentado el sistema WeSSQoS para la selección de servicios web (WS) en base a su calidad de servicio (QoS). Si usamos los criterios introducidos en la Sección 2 podemos analizar la propuesta:

- *Estilo arquitectónico*. WeSSQoS propone una arquitectura orientada a servicios (SOA) totalmente abierta, en el que los diferentes usuarios del sistema puedan eventualmente incorporar servicios propios respetando el interfaz de los servicios utilizados.
- *Atributos*. WeSSQoS es independiente de la ontología de atributos de calidad utilizada. La interfaz del sistema permite seleccionar atributos de un conjunto predefinido (que proponemos que, una vez completada la definición del sistema, provengan del modelo de calidad propuesto en [4]) con atributos propios arbitrarios. Como la mayoría de sistemas, WeSSQoS trabaja tanto con atributos estáticos como dinámicos, aunque es interesante destacar que tal distinción es implícita, viene dada por la forma en que se manejan los datos referentes a QoS tal y como se describe en el ítem siguiente.
- *Datos QoS*. WeSSQoS permite leer los valores de los atributos de calidad de los WS tanto de las descripciones de la definición del servicio como de sistemas de monitorización. El uso de una clase Proxy compartida permite tratar ambos casos de forma uniforme en el sistema e incluso pensar de añadir nuevas formas de obtención de valores (aprovechando la arquitectura SOA).
- *Multialgoritmo*: WeSSQoS permite trabajar con cualquier algoritmo de ordenación/selección de servicios que respete la interfaz definida en la arquitectura (descrita en detalle en la Sección 3). Eventualmente podríamos pensar en usar algoritmos arbitrariamente complejos, p. ej., agregadores de resultados mediante coreografía de otros WS que definieran algoritmos varios.
- *Multirepositorio*: WeSSQoS permite utilizar un número arbitrario de repositorios de WS con independencia de la tecnología utilizada, y además provee un mecanismo de combinación de resultados cuando un mismo WS aparece en más de un repositorio. Actualmente, el usuario es responsable de elegir repositorios "compatibles", es decir, que manejen dominios de software compatibles, que utilicen una ontología de calidad similar, etc.
- *Prototipo disponible*: Un primer prototipo de WeSSQoS está disponible en http://appserv.lsi.upc.es/wessqos/ para su libre uso. Dada su arquitectura de estilo SOA, los servicios ofrecidos pueden ser integrados en otros sistemas (p. ej., en tiempo de ejecución como soporte a la selección automática de WS en el contexto de una composición de servicios). En su versión actual, el sistema ha sido probado sobre dos dominios de software y un subconjunto del modelo de calidad de [4] que incluye 10 atributos tanto estáticos como dinámicos.

Como trabajo futuro, identificamos diversas líneas:

- Efectuar pruebas a gran escala, idealmente en entornos reales de aplicación industrial con repositorios UDDI, con su correspondiente documentación.
- Completar las posibilidades de monitorización con capacidad de medición de más atributos de calidad dinámicos.
- Ofrecer "de serie" un abanico de algoritmos de ordenación/selección.

- Permitir la expresión de requisitos más ricos. En este sentido, pretendemos adaptar la notación NoFun [12] diseñada en nuestro grupo para la modelización de componentes software al caso particular de WS.
- Pensar en mecanismos más sofisticados para combinar datos de los repositorios (y unificar las diferentes estrategias resultantes bajo un mismo interfaz, definiéndolas como servicios reemplazables).
- Considerar la problemática de interoperabilidad semántica (uso de diferentes ontologías de calidad) y lingüística (expresión de los mismos conceptos usando lenguajes diferentes).

Agradecimientos

Este trabajo ha sido subvencionado parcialmente por el proyecto de investigación TIN2007-64753. Oscar Cabrera ha completado una estancia en la UPC subvencionada por una beca CONACYT.

Referencias

[1] M.P. Papazoglou. *Web Services: Principles and Technology*. Pearson - Prentice Hall, 2007.

[2] Y. Liu, A. Ngu, L. Zeng. "QoS Computation and Policing in Dynamic Web Service Selection". *Procs. of the 13th International World Wide Web Conference* (WWW), 2004.

[3] S. Robertson, J. Robertson. *Mastering the Requirements Process (2nd Edition)*. Addison-Wesley, 2007.

[4] D. Ameller, X. Franch. "Service Level Agreement Monitor (SALMon)". *Procs. 7th IEEE Intl. Conference on Composition-Based Software Systems* (ICCBSS), 2008.

[5] E. Al-Masri, Q. Mahmoud. "QoS-based Discovery and Ranking of Web Services". *Procs. 16th Intl. Conference on Computer Communications and Networks* (ICCCN), 2007.

[6] T. Yu, K. J. Lin. "Service Selection Algorithms for Web Services with End-to-end QoS Constraints". *Procs. Information Systems and E-Business Management*, 2005.

[7] T. Yu, K. J. Lin. "A Broker-based Framework for QoS-aware Web Service Composition". *Procs. IEEE Intl.' Conference on E-Technology, e-Commerce and e-Service* (EEE), 2005.

[8] L. Taher, H. El Khatib. "A Framework and QoS Matchmaking Algorithm for Dynamic Web Services Selection". *Procs. 2nd International Conference on Innovations in Information Technology* (IIT), 2005.

[9] X. Wang, T. Vitvar, M. Kerrigan, I. Toma. "A QoS-Aware Selection Model for Semantic Web Services". *Procs. Int. Conf. on Service-Oriented Computing* (ICSOC), 2006

[10] E.M. Maximilien, M.P. Singh. "Multiagent System for Dynamic Web Services Selection". *Procs. 1st Workshop on Service-Oriented Computing and Agent-Based Engineering* (SOCABE), 2005.

[11] J. Hu, C. Guo, H. Wang, P. Zou. "Quality Driven Web Services Selection". *Procs. IEEE International Conference on e-Business Engineering* (ICEBE), 2005.

[12] X. Franch. "Systematic Formulation of Non-Functional Characteristics of Software". *Procs. 3rd IEEE International Conference on Requirements Engineering* (ICRE), 1998.