Algoritmo Híbrido de Composición Automática de Servicios con QoS

Pablo Rodriguez-Mier, Manuel Mucientes, and Manuel Lama

Centro de Investigación en Tecnologías de la Información (CITIUS)
Universidad de Santiago de Compostela, E-15782 Spain,
{pablo.rodriguez.mier,manuel.mucientes,manuel.lama}@usc.es

Resumen En este trabajo se presenta una aproximación híbrida para la composición automática de servicios mediante el emparejamiento semántico de entradas y salidas, optimizando el número de servicios y la calidad de servicio (QoS) de las composiciones. La aproximación propuesta está dividida en 4 fases: 1) generación del grafo de composición para un problema concreto de composición definido mediante entradas y salidas; 2) cálculo de la calidad de servicio óptima en el grafo; 3) optimización secuencial del grafo de composición identificando servicios equivalentes y dominados; 4) búsqueda híbrida para extraer la solución con calidad de servicio óptima y menor número de servicios. Los resultados experimentales con conjuntos de datos del Web Service Challenge 2009-2010 demuestran la efectividad de esta técnica.

1. Introducción

El creciente número de servicios disponibles en Internet demanda cada vez más la necesidad de algoritmos de composición eficientes que sean capaces de: 1) seleccionar y combinar servicios de manera automática para crear nuevas funcionalidades y 2) manejar un gran tamaño de servicios de manera efectiva y asegurando una calidad de servicio (QoS) óptima. Este reto ha motivado la exploración de estrategias diversas para la generación eficiente de composiciones automáticas considerando la calidad de servicio global de las soluciones resultantes [1,2]. Muchas de estas estrategias reducen el problema de composición a un problema combinatorio tipo "Knapsack"[2], donde el objetivo consiste en seleccionar la mejor combinación de servicios entre una lista de candidatos para un flujo de composición prefijado de antemano, optimizando la calidad de servicio final sujeta a una o varias restricciones impuestas por el usuario [3]. Este tipo de propuestas son poco flexibles ya que no contemplan el problema de la generación de los flujos de composición de manera automática, ya que trabajan con esquemas fijos que definen varias tareas o servicios abstractos. El usuario debe proporcionar dicho esquema junto con los servicios candidatos para cada tarea posibles servicios que pueden implementar dichas tareas, con un número de que deben ser proporcionados por el usuario. Otro tipo de propuestas [4,5,6] consideran un problema de composición general, donde la generación de la composición óptima implica no sólo la selección de servicios sino que además generan

la composición completa mediante el análisis de dependencias de las entradas y salidas de los servicios. Sin embargo, para una misma calidad de servicio óptima, pueden existir composiciones con distinto número de servicios. El objetivo de este trabajo es precisamente encontrar aquellas soluciones óptimas no solo en términos de QoS sino de número de servicios. Las principales contribuciones son:

- Optimización secuencial del grafo semántico de dependencias entre servicios basado en el análisis de funcionalidad similar entre servicios y considerando QoS.
- Algoritmo rápido de búsqueda local heurístico capaz de obtener composiciones con QoS óptimo y con un número de servicios cercano al óptimo.
- Búsqueda combinatoria con podado de servicios ineficientes mediante propagación de QoS capaz de mejorar los resultados obtenidos por la búsqueda local mediante una exploración exhaustiva del espacio de búsqueda.

Los resultados experimentales con los conjuntos de datos del Web Service Challenge 2009-2010 demuestran la efectividad de esta propuesta, encontrando soluciones en todos los repositorios.

El resto del artículo se organiza en las siguientes secciones: Sec. 2 introduce el problema de composición, Sec. 3 describe el algoritmo propuesto, Sec. 4 presenta los resultados obtenidos y la Sec. 5 finaliza con las conclusiones del trabajo.

2. Descripción del problema

Informalmente, el problema de composición en el que se centra este trabajo se describe como sigue: dada una descripción semántica del problema de composición $R = \{I_R, O_R\}$, donde I_R es el conjunto de entradas proporcionadas y O_R el conjunto de salidas que se espera obtener, y dando un conjunto de servicios con medidas de calidad de servicio (como por ejemplo tiempo de respuesta, ancho de banda, etc) se calcula capa a capa [7] el grafo de composición que contiene todos los servicios relevantes para el problema de composición, así como las conexiones semánticas entre entradas y salidas. Cada servicio $w = \{I_w, O_w, q_w\}$ tiene un conjunto de entradas I_w necesarias para ser ejecutado, un conjunto de salidas O_w generadas tras la ejecución, y una medida de QoS q_w. Las conexiones semánticas del grafo se calculan entre salidas y entradas siguiendo el modelo definido por Paolucci et al. [8]. El grafo de composición contiene además dos servicios abstractos $So = \{\emptyset, I_R\}$ y $Si = \{O_R, \emptyset\}$. So proporciona como salidas las entradas del problema de composición y Si acepta como entradas las salidas esperadas.

La Fig. 1 muestra un ejemplo de grafo de composición para un problema de composición $R = \{\{i1\}, \{o1, o2\}\}$, donde los servicios están asociados con un valor de tiempo de respuesta medio en milisegundos. Cada servicio está representado con un cuadrado y cada entrada y salida con un círculo. Como puede verse, el grafo contiene múltiples soluciones al problema de composición dado que cada entrada de un servicio puede estar conectada con más de una salida,

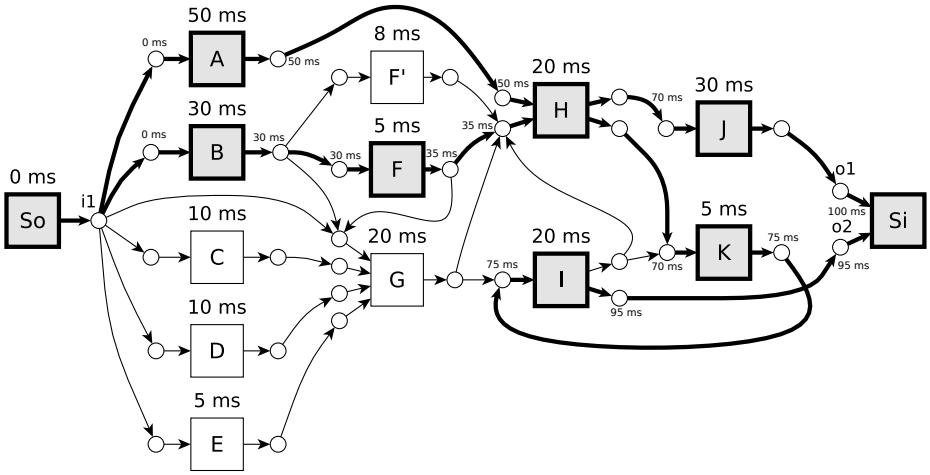

Figura 1. Ejemplo de grafo de composición con una solución con calidad de servicio óptima y mínimo número de servicios.

lo que significa que distintos servicios pueden ser usados para resolver dicha entrada. El tiempo de respuesta óptimo que se puede alcanzar en este grafo es de 100 ms, dado que el mejor coste alcanzable para $o1$ es de 100 ms y para $o2$ de 50 ms (usando los servicios C, D, E, G e I para resolver $o2$). Estos valores pueden ser calculados en tiempo polinómico mediante algoritmos de búsqueda óptimos [9] y aplicando las reglas de agregación de QoS concretas para cada tipo de calidad de servicio [10]. Sin embargo, para una calidad de servicio óptima pueden existir composiciones con distinto número de servicios. Por ejemplo, en la Fig. 1 la solución óptima con 100 ms tiene 7 servicios (excluyendo Si y So), pero es fácil ver que existe otra de 8 servicios con la misma calidad de servicio total (seleccionando los servicios A, C, D, E, G, H, I, J). El objetivo de este trabajo es no sólo determinar la calidad de servicio óptima alcanzable sino la de extraer, de manera eficiente, la composición óptima en términos de número de servicios para la calidad de servicio óptima.

3. Algoritmo de Composición con QoS

El algoritmo se divide en cuatro etapas: 1) cálculo del grafo de composición a partir de un problema de composición basado en entradas/salidas; 2) cálculo del QoS óptimo para el grafo calculado; 3) optimizaciones para reducir el tamaño del grafo; 4) búsqueda híbrida para extraer la combinación de servicios óptima.

3.1. Generación del grafo de composición

Dado un problema de composición, el grafo se calcula capa a capa, comenzando con el servicio *So* que proporciona las entradas iniciales, y seleccionando

todos los posibles servicios relevantes que pueden ser invocados con un subconjunto de las salidas producidas por servicios en capas previas [11]. Una vez se seleccionan todos los posibles servicios para formar parte del grafo, se calculan las relaciones semánticas entre ellos. El resultado de esta fase es la generación de un grafo como el mostrado en la Fig. 1.

3.2. Cálculo del QoS óptimo

El cálculo del QoS óptimo sobre el grafo se calcula en tiempo polinómico, propagando costes desde el servicio So hasta el Si hasta conocer el valor óptimo de cada entrada y salida del grafo. El algoritmo usado es una generalización del algoritmo de caminos mínimos de Dijkstra [12].

3.3. Optimizaciones del grafo

Una vez se conoce el coste óptimo de QoS para cada entrada y salida del grafo, se aplican una serie de optimizaciones secuenciales que reducen monótonamente el tamaño del grafo. Las optimizaciones que se aplican son una extensión de los explicados en [13] para soportar QoS: 1) Eliminado de servicios que empeoran el QoS óptimo detectado en el paso previo; 2) eliminación de servicios que no contribuyen directa o indirectamente a las salidas del problema de composición; 3) combinación de servicios funcionalmente equivalentes mediante el análisis de dependencias de sus entradas y salidas y del QoS; 4) combinación de servicios dominados (cuya funcionalidad sea solapada por otros servicios con mejor o igual QoS).

3.4. Algoritmo Híbrido

El objetivo de este algoritmo es el de extraer composiciones (combinaciones de servicios) del grafo de composición, optimizando el número total de servicios que forman parte de la composición y con QoS óptimo. Para ello, en cada paso se selecciona una entrada a resolver, de manera que se elige un único servicio cuyas salidas resuelven la entrada seleccionada, hasta encontrar la combinación de servicios óptima. La estrategia híbrida hace uso de dos búsquedas, una local, capaz de encontrar soluciones con un número de servicios cercano al óptimo, y la global, que mejora los resultados en el tiempo restante disponible mediante una exploración exhaustiva del espacio de búsqueda.

Búsqueda Local Fig. 2 muestra el pseudocódigo de la estrategia de búsqueda local, basado en un algoritmo heurístico recursivo con vuelta atrás. Dado un grafo de composición, el algoritmo comienza la búsqueda anotando las entradas del servicio Si como no resueltas (las salidas del problema de composición) y seleccionando Si como parte de la solución. En cada llamada al método *LS-BACKTRACK*, el algoritmo analiza los potenciales servicios candidatos para la lista de entradas sin resolver, es decir, todos aquellos servicios que tengan alguna

salida semánticamente compatible con alguna de las entradas en *unresolved*. El método *RANK-RESOLVERS* se encarga de buscar los candidatos y ordenarlos en función de dos criterios heurísticos: número de entradas que cada servicio pueden resolver en todo el grafo y número de entradas que tiene el propio servicio. El servicio más prometedor para formar parte de la composición será aquel que más entradas resuelva y menos entradas requiera para ejecutarse. Esta heurística permite una convergencia voraz hacia una solución local con un número suficientemente pequeño de servicios. Una vez seleccionado el servicio más prometedor, por cada entrada que el servicio puede resolver, se realiza un análisis *lookahead* mediante una búsqueda hacia delante (método *CYCLE*) para determinar si el servicio puede resolver la entrada o lleva a una composición irresoluble. Esta búsqueda permite descartar composiciones no válidas y acelerar la vuelta atrás en caso de fallo.

Una vez se han identificado todas las entradas que el servicio seleccionado puede resolver (almacenadas en *resolved*), el método *RESOLVE* crea una copia del grafo de composición, pero con las correspondientes entradas resueltas, es decir, sólo se conserva el enlace semántico entre el servicio seleccionado y las entradas resueltas, eliminando el enlace con el resto de candidatos del grafo. Si el servicio seleccionado no forma parte todavía del conjunto de servicios de la solución, se realiza una llamada recursiva a *LS-BACKTRACK* para repetir el proceso hasta que no quede ninguna entrada sin resolver. Si se llega a un estado irresoluble (todos los servicios candidatos para alguna entrada generan una composición con bucles) se prueba con el siguiente mejor servicio.

```
1: function LOCAL-SEARCH(graph)
2:     return LS-BACKTRACK(graph, INPUTS(Si), {Si})
3:
4: function LS-BACKTRACK(graph, unresolved, services)
5:     if unresolved = ∅ then return graph
6:     services ← RANK-RESOLVERS(unresolved)
7:     for each w ∈ services do
8:         resolved ← {}
9:         matched ← MATCH(w, unresolved)
10:        for each input ∈ matched do
11:            if ¬CYCLE(w, input) then
12:                resolved ← resolved ∪ input
13:        if resolved ≠ ∅ then
14:            unresolved ← unresolved \ resolved
15:            if ¬(w ∈ services) then
16:                unresolved ← unresolved ∪ INPUTS(w)
17:            g ← RESOLVE(graph, w, resolved)
18:            result ← LS-BACKTRACK(g, unresolved, services ∪ w)
19:            if result ≠ fail then return result
    return fail
```

Figura 2. Algoritmo de búsqueda local para extraer la composición del grafo

Búsqueda Global El objetivo de la búsqueda global es extraer la composición con menor número de servicios y calidad de servicio óptima. Esta estrategia está basada en la observación de que cada entrada del grafo de composición tiene (dependiendo de los servicios seleccionados) unos umbrales de calidad de servicio variables fuera de los cuales la calidad de servicio final de la composición se vería afectada. Por ejemplo, en la Fig. 1, el servicio H tiene una entrada que se puede resolver en 50 ms y otra que se puede resolver usando distintos servicios, cuyo mejor coste es de 35 ms. Sin embargo, dado que la ejecución más temprana posible de ese servicio son 50 ms, el margen para dicha entrada es [35 ms,50 ms]. Cualquier servicio que proporcione la entrada con coste mayor de 50 ms empeoraría el coste óptimo de las salidas del servicio H, que es de 50 ms + 20 ms = 70 ms. Esta idea se explota durante la búsqueda exhaustiva para reducir el número de potenciales combinaciones de servicios a explorar.

Para poder conocer los umbrales de coste de cada entrada, y por lo tanto para saber cuáles son los servicios candidatos válidos que pueden ser usados para resolver las entradas sin empeorar el coste óptimo de QoS, el algoritmo necesita propagar los umbrales por cada entrada $i_w \in w$. El umbral se define como $Q_{i_w} = [Q_{lmax}, Q_{gmax}]$. La cota inferior Q_{lmax} representa la calidad de servicio óptima alcanzable para dicha entrada. Q_{gmax} representa el umbral superior de calidad de servicio soportado, es decir, cualquier servicio cuya salida tenga un coste $Q > Q_{gmax}$ que sea usado para resolver dicha entrada empeoraría el coste óptimo de calidad de servicio final de la composición, con lo que puede ser descartado como candidato. Estos umbrales son usados para podar aquellos servicios sub-óptimos durante la búsqueda.

La Fig. 3 muestra el pseudocódigo de la estrategia de búsqueda global. El algoritmo comienza realizando el cálculo de los costes óptimos de cada entrada y salida del grafo mediante la función *QOS-UPDATE*. Este método devuelve una tabla *hash* Q_{opt} donde cada clave corresponde con una entrada o salida y el valor corresponde con el coste acumulado de calidad de servicio. A continuación, la búsqueda comienza seleccionando el servicio Si y anotando sus entradas como no resueltas, y se calculan los umbrales de coste de QoS (L. 6). Por ejemplo, en la Fig. 1, los umbrales para $o1$ del servicio Si son $[100, 100]$, mientras que los umbrales de $o2$ son $[50, 100]$ ms. Toda esta información se inserta en una cola de prioridad ordenada por el número de servicios seleccionados (*selected*) de modo que siempre se extraen la composiciones parciales de menor número de servicios. La cola, que contiene todas las composiciones parciales (grafos de composición), se va explorando hasta que no quedan más candidatos o hasta que se encuentra una solución (un grafo de composición sin entradas no resueltas, es decir, donde cada entrada está resuelta por un único servicio).

Cada vez que se extrae una composición parcial de la cola, se selecciona la siguiente entrada no resuelta que se va a resolver con el método *SELECT-UNRESOLVED*. Este método selecciona la siguiente entrada a resolver de manera heurística, usando una estrategia tipo *minimum-remaining-values* [14], es decir, aquella entrada con menor número de servicios candidatos es seleccionada primero. Esta heurística permite reducir el tamaño del espacio de búsqueda

```
1: function GLOBAL-SEARCH(graph)
2:     Q_opt ← QOS-UPDATE(graph)
3:     selected ← {Si}
4:     max ← MAX-QOS(INPUTS(Si), Q_opt)
5:     for i_Si ∈ INPUTS(Si) do
6:         unresolved[i_Si] ← [Q_opt[i_Si], max]
7:     queue ← INSERT(⟨graph, unresolved, Q_opt, selected⟩,queue)
8:     while queue ≠ ∅ do
9:         ⟨graph, unresolved, Q_opt, selected⟩ ← POP(queue)
10:        if unresolved = ∅ then return graph
11:        input ← SELECT-UNRESOLVED(graph)
12:        [Q_lmax, Q_gmax] ← unresolved[input]
13:        for w ∈ RESOLVERS(input, [Q_lmax, Q_gmax]) do
14:            if ¬CYCLE(w,input) then
15:                successor ← RESOLVE(graph, w, {input})
16:                unresolved ← REMOVE(i, unresolved)
17:                Q_out_w ← QOS-OUTPUT(w, Q_opt)
18:                if Q_out_w > Q_lmax then
19:                    Q_opt ← QOS-UPDATE(successor)
20:                if w ∉ selected then
21:                    [Q^w_lmax, Q^w_gmax] ← BOUNDS(w,Q_gmax)
22:                    for i_w ∈ INPUTS(w) do
23:                        unresolved[i_w] ← [Q^w_lmax, Q^w_gmax]
24:            u ← selected ∪ w
25:            queue ← INSERT(⟨successor, unresolved, Q_opt, u⟩, queue)
       return fail
```

Figura 3. Algoritmo de búsqueda global para extraer la solución con menor número de servicios

generado. Por cada servicio candidato para la entrada seleccionado, se generan tantos grafos de composición (sucesores) como candidatos hay para la entrada. Para ello, se genera un sucesor copiando el grafo actual, donde la entrada seleccionada aparecerá conectada únicamente con el servicio candidato. En caso de que el coste de la salida del servicio candidato sea mayor que Q_{lmax} de la entrada a resolver, se repropagan los costes de calidad de servicio en todo el grafo. Finalmente, si el servicio candidato no había sido seleccionado ya previamente para resolver alguna otra entrada de la solución parcial, se añaden sus entradas como nuevas entradas sin resolver con sus correspondientes umbrales de calidad de servicio calculados mediante el método *BOUNDS*, y se inserta en la cola para procesar más adelante.

4. Evaluación

Para validar la aproximación, hemos usado los conjuntos de datos del Web Service Challenge (WSC) 2009-2010 [15]. Estos conjuntos de pruebas tienen un número variable de servicios, que va desde 572 hasta 15.211 servicios con dos atributos de calidad de servicio: tiempo de respuesta (en milisegundos) y ancho de banda (en invocaciones por segundo soportadas). Los test se han ejecutado estableciendo un máximo de 5 minutos para la búsqueda de la composición.

Las Tablas 1 y 2 muestran los resultados obtenidos para cada conjunto de pruebas para el tiempo de respuesta y *throughput* respectivamente. La columna *#Servicios del grafo* muestra el número de servicios en el grafo de composición calculado inicialmente, y *#Servicios del grafo (opt)* el número de servicios del grafo después de aplicar las optimizaciones descritas en la Sec. 3. Como puede observarse, las optimizaciones reducen de media un 64 % el número de servicios del grafo. Esto indica que las técnicas de análisis de funcionalidad equivalente y dominada son efectivas a la hora de reducir el espacio de búsqueda cuando existe un número elevado de servicios con funcionalidad similar. Por otra parte, la búsqueda híbrida consiguió resolver todos los repositorios del WSC encontrando la solución óptima en un plazo de tiempo corto, excepto para el repositorio D-04, donde debido a la complejidad y tamaño la búsqueda global no encontró solución en el tiempo establecido. Sin embargo, en situaciones donde la búsqueda global no es capaz de sacar la solución óptima, la búsqueda local proporciona una alternativa buena. En el caso del repositorio D-04, la búsqueda local encontró una solución de 62 servicios en 1.6 segundos. Nótese además que la búsqueda local encuentra la solución óptima para todos los repositorios excepto en el caso del ancho de banda para los repositorios D-03 y D-04, que encuentra una subóptima (en número de servicios) en comparación con la búsqueda global. Por lo que sabemos, los resultados obtenidos son los mejores hasta la fecha en términos de número de servicios con calidad de servicio óptimo encontrados en el WSC, comparándolo tanto con las propuestas que han participado en el WSC como con otras propuestas recientes [16] (los resultados destacados en gris corresponden con los resultados que superan el estado del arte en menor número de servicios).

Cuadro 1. Validación con el WSC y tiempo de respuesta

		D-01	**D-02**	**D-03**	**D-04**	**D-05**
Tiempo de respuesta (ms)		500	1.690	760	1.470	4.070
#Servicios del grafo		81	141	154	331	238
#Servicios del grafo (opt)		21	57	15	160	126
Búsqueda local	#Servicios	5	20	10	**40**	32
	Tiempo (s)	0,613	0,998	2,608	7,767	2,920
Búsqueda global	#Servicios	5	20	10	-	32
	Tiempo (s)	0,617	1,580	2,613	-	24,971

Cuadro 2. Validación con el WSC y *throughput*

		D-01	**D-02**	**D-03**	**D-04**	**D-05**
Throughput (inv/s)		15.000	6.000	4.000	4.000	4.000
#Servicios del grafo		81	141	154	331	238
#Servicios del grafo (opt)		10	43	90	156	69
Búsqueda local	#Servicios	5	20	15	**62**	**31**
	Tiempo (s)	0,343	1,173	1,933	8,571	2,562
Búsqueda global	#Servicios	5	20	10	-	**30**
	Tiempo (s)	0,345	1,246	2,085	-	119,322

5. Conclusiones

En este artículo hemos presentado un algoritmo híbrido para la generación automática de composiciones servicios optimizando la calidad de servicio total y reduciendo el número final de servicios de la composición. Los resultados obtenidos con los repositorios del Web Service Challenge 2009-2010 muestran que la combinación de una búsqueda local y global consigue un rendimiento mejor que otras aproximaciones del estado del arte.

Agradecimientos

Este trabajo ha sido financiado con fondos del Ministerio de Economía y Competitividad (TIN2014-56633-C3-1-R). El autor Pablo Rodríguez-Mier es actualmente becario del programa FPU del Ministerio de Economía y Competitivida (AP2010-1078).

Referencias

1. J. Rao and X. Su, "A Survey of Automated Web Service Composition Methods," in *International Workshop on Semantic Web Services and Web Process Composition*, 2004.
2. A. Strunk, "QoS-Aware Service Composition: A Survey," *IEEE European Conference on Web Services*, 2010.

3. L. Zeng, B. Benatallah, A. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "Qos-Aware Middleware for Web Services Composition," *IEEE Transactions on Software Engineering*, vol. 30, no. 5, pp. 311–327, 2004.
4. W. Jiang, S. Hu, and Z. Liu, "Top K Query for QoS-Aware Automatic Service Composition," *IEEE Transactions on Services Computing*, vol. 7, no. 4, pp. 681–695, Oct. 2014.
5. Y. Yan, B. Xu, Z. Gu, and S. Luo, "A QoS-Driven Approach for Semantic Service Composition," in *IEEE International Conference on Commerce and Enterprise Computing*, 2009.
6. M. Aiello, E. E. Khoury, A. Lazovik, and P. Ratelband, "Optimal QoS-Aware Web Service Composition," in *IEEE International Conference on Commerce and Enterprise Computing*, 2009.
7. P. Rodriguez-Mier, M. Mucientes, and M. Lama, "Automatic Web Service Composition with a Heuristic-Based Search Algorithm," in *IEEE Int. Conf. on Web Services (ICWS)*, 2011, pp. 81–88.
8. M. Paolucci, T. Kawamura, T. R. Payne, and K. Sycara, "Semantic Matching of Web Services Capabilities," in *The Semantic Web—ISWC 2002*, 2002, pp. 333—-347.
9. W. Jiang, T. Wu, and S.-l. Hu, "QoS-Aware Automatic Service Composition: A Graph View," *Science And Technology*, vol. 26, no. 5, pp. 837–853, 2011.
10. J. Cardoso, A. Sheth, J. Miller, J. Arnold, and K. Kochut, "Quality of service for workflows and web service processes," *Web Semantics*, vol. 1, pp. 281–308, 2004.
11. P. Rodríguez-Mier, M. Mucientes, J. C. Vidal, and M. Lama, "An Optimal and Complete Algorithm for Automatic Web Service Composition," *International Journal of Web Service Research*, vol. 9, no. 2, pp. 1–20, 2012.
12. P. Rodriguez-Mier, M. Mucientes, and M. Lama, "A Dynamic QoS-Aware Semantic Web Service Composition Algorithm," in *International Conference on Service-Oriented Computing*, 2012.
13. P. Rodriguez Mier, C. Pedrinaci, M. Lama, and M. Mucientes, "An Integrated Semantic Web Service Discovery and Composition Framework," *IEEE Transactions on Services Computing*, 2015 (DOI 10.1109/TSC.2015.2402679).
14. R. Dechter, *Constraint processing*. Morgan Kaufmann, 2003.
15. S. Kona, A. Bansal, M. B. Blake, S. Bleul, and T. Weise, "WSC-2009: a quality of service-oriented web services challenge," in *IEEE International Conference on Commerce and Enterprise Computing*, 2009.
16. M. Chen and Y. Yan, "Redundant Service Removal in QoS-Aware Service Composition," *IEEE International Conference on Web Services (ICWS)*, 2012.