Avances Hacia un Algoritmo Optimizado para la Inferencia de Restricciones Locales de Rendimiento en Grafos

A. García Domínguez [1], I. Medina Bulo[1], M. Marcos Bárcena [2]

[1]Departamento de Lenguajes y Sistemas Informáticos, C/Chile 1, CP 11003 Cádiz. Teléfono: 956015780.
Correo electrónico: antonio.garciadominguez@uca.es.
[2] Departamento de Ingeniería Mecánica y Diseño Industrial, C/Chile 1, CP 11003 Cádiz.

Resumen

Grafos como los diagramas de actividades UML son comúnmente usados para modelar programas con diversos fines. En un trabajo anterior se presentaron algoritmos para inferir restricciones de rendimiento para cada nodo de un diagrama de actividades UML. Sin embargo, uno de los algoritmos requería recorrer todos los posibles caminos, exhibiendo costes exponenciales. En este trabajo se presenta un reemplazo más robusto en el caso promedio y generalizado a todo tipo de grafos. Tras esbozar una definición preliminar basada en reescritura de grafos, se deriva una versión más fácil de implementar basada en un simple recorrido del grafo.

1. Introducción

En muchos ámbitos, asegurar un cierto nivel de rendimiento es de gran importancia. En el ámbito de las Arquitecturas Orientadas a Servicios (AOS) [1], el software se estructura como un catálogo de servicios. Algunos servicios se construyen reutilizando servicios de otras partes de la organización o incluso de organizaciones externas: se conocen como *composiciones de servicios*.

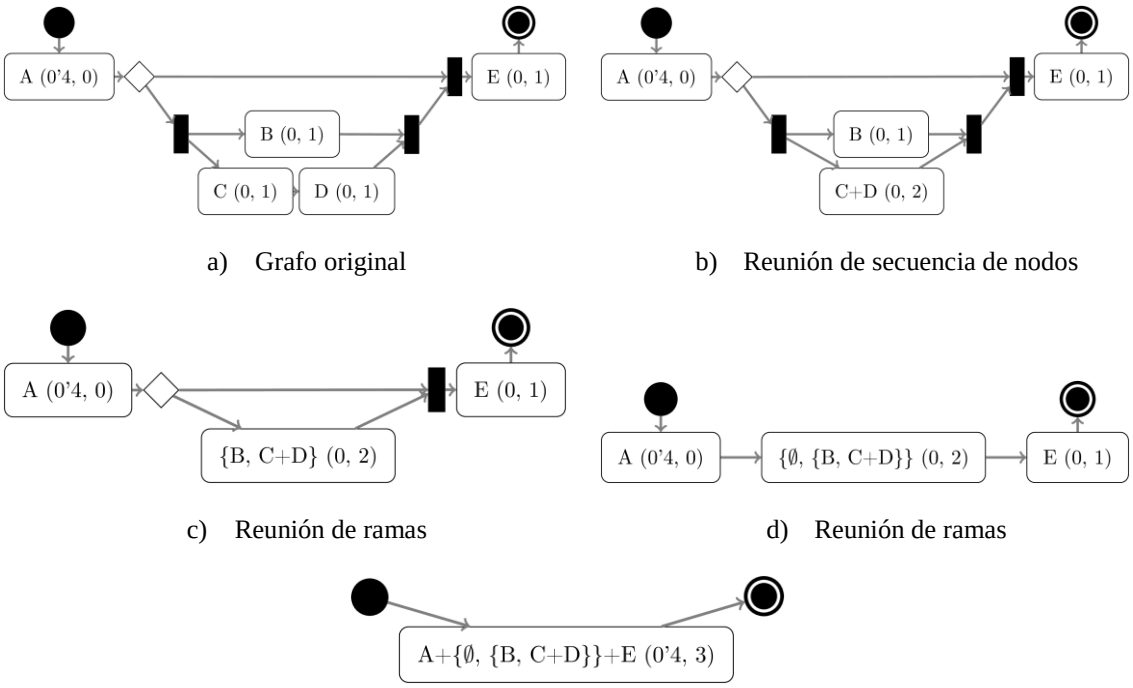
Conseguir un nivel adecuado de rendimiento de estas composiciones requiere pruebas continuas y una monitorización constante. Sin embargo, esto requiere conocer el nivel de rendimiento exigido a cada parte de la composición, a partir del rendimiento exigido al todo. En algunos casos se dispondrá de acuerdos de niveles de servicio o de información de monitorización, pero en muchos casos no dispondremos de esa información: por ejemplo, cuando el servicio externo en cuestión aún no haya sido contratado o desarrollado. Se podría asumir un valor a partir de la experiencia, pero existe el riesgo de sobreestimar o infravalorar el rendimiento necesario. En el primer caso, se incurren en costes por contratar demasiada capacidad o dedicar demasiado tiempo de desarrollo; en el segundo, se podrían violar los acuerdos de nivel de servicio con terceras partes o perder oportunidades de negocio: pedidos en empresas de fabricación, ventas en mayoristas, etc.

En un trabajo anterior, se presentaron dos algoritmos dirigidos a inferir las restricciones locales de rendimiento (tiempo límite y número de peticiones por segundo) a partir de restricciones locales y globales sobre un diagrama de actividades UML [2]. El objetivo era modelar una composición de servicios a través de un diagrama de actividades UML y obtener especificaciones para pruebas de rendimiento a partir de los valores inferidos, siguiendo un enfoque dirigido por modelos. Las herramientas dedicadas a este proceso formarán parte de la metodología AOS SODM+Testing dirigida por modelos con apoyo explícito para pruebas [3], que es el objetivo principal de la tesis asociada a este trabajo.

Tras analizar el rendimiento de estos algoritmos, se concluyó que el algoritmo de inferencia de peticiones por segundo tenía coste temporal $O(n^3)$, y el algoritmo de inferencia de tiempos límite tenía coste temporal $O(n^2 + n2^b)$, para un diagrama de actividades UML correspondiente a un grafo orientado acíclico con n vértices, de los cuales $b < n$ tenían 2 aristas salientes y el resto sólo una o ninguna. Este alto coste era debido a que la inferencia de tiempos límites de acuerdo con las restricciones impuestas necesita recorrer todos los nodos de todos los posibles caminos.

Tras hallar este problema, se observó que en la práctica, la mayoría de los caminos en los grafos correspondientes a diagramas de actividades UML no suelen ser disjuntos, sino que comparten una parte considerable de los nodos. A partir de esta idea, se ha diseñado un algoritmo mejorado más eficiente en promedio.

Este trabajo se estructura de la siguiente forma: tras presentar de forma concisa el algoritmo original de inferencia de tiempos límite, se esboza una primera formulación basada en reescritura de grafos del nuevo algoritmo. De



a) Grafo original

b) Reunión de secuencia de nodos

c) Reunión de ramas

d) Reunión de ramas

e) Reunión de secuencia de nodos y supernodos

Figura 1. *Proceso de reducción de un diagrama de actividad UML*

esta versión se deriva una definición preliminar simplificada de un algoritmo generalizado a grafos orientados acíclicos. Finalmente, se proporcionan una serie de conclusiones y líneas de trabajo futuro.

2. Algoritmo original

El algoritmo original (descrito en detalle en [2]) opera sobre diagramas de actividades UML, extendidos con una restricción global manual sobre el número de peticiones a atender por unidad de tiempo, y el tiempo límite disponible para todos los caminos. A su vez, cada actividad tiene una restricción (manual o inferida) local.

El algoritmo sigue estos pasos para anotar cada actividad libre (sin restricciones manuales) con su tiempo límite:

1. Se retiran las restricciones antes inferidas.
2. Se calculan todos los caminos posibles desde el nodo inicial hasta un nodo final.
3. Se ordenan los caminos de más a menos restrictivos. Informalmente, la métrica usada es el tiempo no usado por las restricciones manuales de sus actividades, dividido por el número de actividades libres en el camino más 1.
4. Para cada camino en la lista ordenada, se reparte el tiempo sobrante por igual entre las actividades aún sin restricciones.

El algoritmo es sencillo y consigue asignaciones de tiempos equitativas, consistentes con las restricciones manuales y tan permisivas como es posible. Sin embargo, el algoritmo ha de recorrer todos los nodos de cada uno de los caminos posibles, incurriendo en costes que crecen de forma exponencial a medida que aumenta el número de ramificaciones del grafo subyacente al diagrama de actividades UML.

3. Aproximación por reescritura de grafos

Tras los problemas identificados en el algoritmo anterior, se observó que muchos de los nodos eran compartidos entre varios caminos. Sin embargo, cada nodo sólo nos interesa en el camino más restrictivo en que aparece. Por tanto, sería útil resumir los posibles caminos de forma que las partes comunes sólo se evaluaran una vez. Para ello, puede simplificarse el grafo paso a paso, agregando sus nodos y restricciones en *supernodos* (nodos que contienen subgrafos de nodos y supernodos) hasta tener un grafo de un solo supernodo. A continuación, se puede desagregar mientras se infieren las restricciones.

En la Figura 1 puede verse un ejemplo de un proceso de reducción de un diagrama de actividades UML. Cada nodo incluye un par (m, p), donde m es la suma de los tiempos debidos a restricciones manuales en su interior, y p es el peso que tiene para el reparto del tiempo sobrante. Normalmente, un nodo con una restricción manual tendrá peso 0, y uno sin ella tendrá peso 1, pero se pueden utilizar otros pesos para conseguir asignaciones más

flexibles. Las reducciones se implementarían mediante reglas de reescritura de grafos [4].

Tras reducir el diagrama a un solo nodo, el proceso para conseguir los tiempos límite en el nodo anterior sería el siguiente:

1. Se comienza por el supernodo raíz (como el de la subfigura e) de la Figura 1) que engloba todo el grafo. Se le asigna el tiempo límite de la restricción global.
2. Se comprueba de qué tipo es el nodo actual, cuyo par es (m, p) y tiempo asignado es t:
   2.1. Si engloba una secuencia, se asigna a cada hijo h_i con par (m_i, p_i) del nodo actual el tiempo $m_i + (p_i/p)(t - m)$, y se sigue recursivamente por él.
   2.2. Si engloba una ramificación, se asigna el tiempo disponible a todas las ramas.
   2.3. Si es un nodo normal, se usa el tiempo disponible como tiempo límite.

En particular, para el ejemplo de la Figura 1 y asumiendo que todos los caminos han de acabar en 1 segundo o menos, operaría así:

1. Tiempo: 1s, nodos: (0'4, 3). Se asigna el segundo al único nodo, que representa a todo el grafo.
2. Tiempo: 1s, nodos: (0'4, 0), (0,2), (0,1). Se asignan 0'4 segundos al primer nodo, y los 0'6 restantes se reparten de acuerdo con los pesos de los demás nodos: 0'4 para el segundo y 0'2 para el tercero. El primero y tercero son nodos simples, así que no se desciende por ellos, pero sí por el segundo.
3. Tiempo: 0'4s, nodos: (0, 2). Al tratarse de una ramificación, se asignan 0'4s a ambas ramas, pero una de ellas está vacía. Se desciende por la otra.
4. Tiempo: 0'4s, nodos: (0, 1), (0, 2). De nuevo se tiene una ramificación, por lo que se asignan 0'4s a ambos nodos. Ahora se debe descender por ambas ramas.
   a. Primera rama. Tiempo: 0'4s, nodos: (0, 1). Le asignamos los 0'4s al único nodo disponible.
   b. Segunda rama. Tiempo: 0'4s, nodos: (0, 1), (0, 1). Repartimos los 0'4s por igual entre ambos.

Este enfoque consigue los mismos resultados que el original para este caso particular [2]. Sin embargo, queda pendiente demostrar si es así o no en general.

La idea de reducir un grafo a un solo nodo para obtener información acerca de su calidad de servicio no es nueva: Cardoso aplicó un enfoque similar en [5]. Sin embargo, su intención era muy distinta: agrega las restricciones manuales de cada nodo para conseguir las globales. Por el contrario, la idea de este trabajo es agregar las restricciones manuales (tanto locales como globales) para inferir las locales.

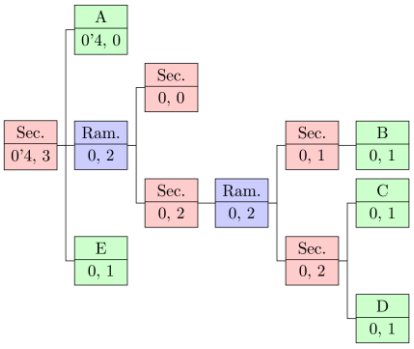


Figura 2. *Árbol de reducción del grafo de la Figura 1.*

4. Definición basada en recorrido del grafo

El anterior enfoque basado en reescritura de grafos es sencillo de comprender y puede evitar la explosión combinatoria que supone una ramificación tras otra. Sin embargo, un sistema de reescritura de grafos es, en general, difícil de implementar y puede tener problemas de rendimiento [6]. Estos problemas son resolubles en este caso, pero de todos modos es deseable buscar una alternativa más sencilla.

Tras una búsqueda bibliográfica, se observó que el algoritmo anterior producía *grafos jerárquicos* [7]. Un grafo jerárquico es un grafo que puede contener subgrafos en sus vértices y/o aristas. Busatto et ál. plantean una representación: modelar las aristas usuales con un grafo normal, y las relaciones de contención con un árbol separado [8].

En base a esta idea, puede verse que lo que interesa del anterior algoritmo no es realmente el grafo jerárquico, sino el árbol propuesto por Busatto. Por ello, se puede utilizar un algoritmo estándar para construir dicho árbol a la vez que se recorre el diagrama de actividades UML, sin tener que recurrir a toda la potencia de un sistema de reescritura de grafos. Sin embargo, como el árbol se construiría a partir de la raíz en vez de a partir de las hojas, sería necesario recorrer el árbol producido dos veces: subiendo para agregar las restricciones, y bajando para inferir. Se bajaría como antes, y subir consistiría en sumar los pares elemento a elemento para las secuencias y tomar el elemento mayor en orden lexicográfico en las ramificaciones.

Por limitaciones de espacio, se esboza el algoritmo sobre el ejemplo de la Figura 1. El árbol de reducción resultante es el de la Figura 2: los hijos están ordenados de arriba abajo.

1. Se comienza por el *nodo inicial* (nodo sin aristas entrantes). El grafo completo se modela como una secuencia, así que el árbol de reducción comienza con un único nodo de secuencia.

2. Se visita el nodo A que viene a continuación, añadiéndolo como primer hijo de la secuencia.
3. A continuación viene una ramificación, por lo que se añade un nodo de ramificación, que contiene un nodo de secuencia por cada rama. La primera rama está vacía, así que no hacemos nada. La segunda rama no está vacía: se prosigue por ella.
4. El primer nodo de la segunda rama es una ramificación: se añade un nodo de ramificación al nodo secuencia de la segunda rama. Este nuevo nodo de ramificación tendrá un nodo de secuencia a su vez por cada una de sus dos ramas: una contiene al nodo B, y la otra a C y D.
5. Hemos completado las ramificaciones: seguimos por el punto de reunión y encontramos el nodo E. Lo añadimos como hijo de la secuencia principal.
6. Hemos llegado al *nodo final* (nodo sin aristas salientes): hemos terminado.

El algoritmo actualmente definido es más complejo, ya que ha de tratar con grafos más flexibles que el de la Figura 1: puede haber varios nodos finales, y puede ocurrir que no todas las ramas que parten de un nodo se reúnan en otro. Incluso puede ocurrir que en un nodo se reúnan ramas provenientes de nodos distintos. Para ello, es necesario mantener más información, anotando las aristas visitadas y las ramas reunidas, entre otras cosas. Además, hay que considerar que implícitamente todos los nodos finales se reúnen en un "verdadero" nodo final. De todos modos, la idea general es la expuesta en el ejemplo anterior. Actualmente, el algoritmo se encuentra generalizado a todo grafo orientado acíclico y conexo.

5. Conclusiones

En este trabajo, se han esbozado dos algoritmos para mejorar un algoritmo previamente publicado en [2] para inferir restricciones de rendimiento en diagramas de actividades UML a partir de información local y global. El primer algoritmo utiliza reescritura de grafos, y el segundo es una simplificación del anterior, aprovechando el hecho de que sólo nos interesa el árbol que representa las relaciones jerárquicas del grafo jerárquico producido por el primer algoritmo. Además, el segundo algoritmo ha sido generalizado a todo tipo de grafo.

Actualmente, los algoritmos han sido probados manualmente sobre un conjunto de casos representativos. Queda pendiente implementar el segundo algoritmo y realizar estudios acerca de su corrección y rendimiento. Se considerarán diversos arquetipos de grafos, tales como grafos lineales, dipolos en secuencia, rejillas, etc.

6. Agradecimientos

Se agradecen a Francisco Palomo Lozano sus consejos acerca de la formulación del algoritmo y la organización de las pruebas a realizar.

7. Referencias

[1] T. Erl, *SOA: Principles of Service Design*, Indiana, EEUU: Prentice Hall, 2008.

[2] A. García Domínguez, I. Medina Bulo, and M. Marcos Bárcena, "Inference of performance constraints in Web Service composition models," *CEUR Workshop Proceedings of the 2nd International Workshop on Model-Driven Service Engineering*, vol. 608, Jun. 2010, pp. 55-66.

[3] A. García Domínguez, I. Medina Bulo, and M. Marcos Bárcena, "Hacia la Integración de Técnicas de Pruebas en Metodologías Dirigidas por Modelos para SOA," *Actas de las V Jornadas Científico-Técnicas en Servicios Web y SOA*, Madrid, España: 2009, pp. 167-180.

[4] R. Heckel, "Graph Transformation in a Nutshell," *Proceedings of the School on Foundations of Visual Modelling Techniques (FoVMT 2004) of the SegraVis Research Training Network*, Elsevier, 2006, pp. 187-198.

[5] J. Cardoso, A. Sheth, J. Miller, J. Arnold, and K. Kochut, "Quality of service for workflows and web service processes," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 1, Abril. 2004, pp. 281-308.

[6] D. Blostein, H. Fahmy, and A. Grbavec, "Issues in the Practical Use of Graph Rewriting," *Proceedings of the 5th Workshop on Graph Grammars and Their Application To Computer Science*, J. Cuny, H. Ehrig, G. Engels, and G. Rozenberg, Eds., Williamsburg, Virginia, EEUU: Springer-Verlag, 1995, pp. 38-55.

[7] F. Drewes, B. Hoffmann, and D. Plump, "Hierarchical graph transformation," *Foundations of Software Science and Computation Structures*, J. Tiuryn, Ed., Berlin, Alemania: Springer-Verlag, 2000.

[8] G. Busatto, G. Engels, K. Mehner, and A. Wagner, "A Framework for Adding Packages to Graph Transformation Approaches," *Proceedings of the 6th Int. Workshop on Theory and Application of Graph Transformation*, H. Ehrig, G. Engels, H. Kreowski, and G. Rozenberg, Eds., Paderborn, Alemania: Springer-Verlag, 1998, pp. 352-367.