Un Enfoque para la Generación de Pruebas para Múltiples Tecnologías desde Flujos de Trabajo Anotados con MARTE

A. García Domínguez[(1)], I. Medina Bulo[(1)] y Mariano 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

Obtener el rendimiento esperado de un flujo de trabajo sería más fácil si cada tarea incluyera sus expectativas individuales. Sin embargo, normalmente sólo se tienen requisitos globales de rendimiento, y los requisitos locales son derivados a mano. En anteriores trabajos se han presentado dos algoritmos que infieren automáticamente restricciones locales de rendimiento en diagramas de actividad UML con anotaciones del perfil MARTE. En este trabajo, se presenta un enfoque para utilizar estas anotaciones para generar pruebas de rendimiento para múltiples entornos, vinculando el modelo de rendimiento con el de implementación.

1. Introducción

El software debe cumplir requisitos tanto funcionales como no funcionales. Dentro de los no funcionales, los requisitos de rendimiento son muy comunes, y en ciertos contextos tienen tanta importancia como los requisitos funcionales. Además de los sistemas de tiempo real (duros o blandos), hay que considerar el caso de las arquitecturas orientadas a servicios (*Service Oriented Architectures* o SOA) [1]. En las SOA, es común asignar a cada servicio un acuerdo de nivel de servicio (Service Level Agreement o SLA): en caso de no cumplirse, el proveedor del servicio puede tener que compensar al consumidor de alguna forma.

Dada la importancia de los requisitos de rendimiento, existe una variedad de propuestas para su estimación y medición [2]. Medir el rendimiento de un sistema puede servir para diversos objetivos, como detectar pérdidas de rendimiento a largo plazo, identificar patrones de carga o comprobar si se están cumpliendo sus requisitos de rendimiento. Para comprobar los requisitos de rendimiento, se necesita que hayan sido definidos antes: sin embargo, normalmente no se suelen dar requisitos detallados de rendimiento. Esto exige cumplir requisitos de alto nivel sin saber realmente qué rendimiento se necesita en cada parte del sistema.

En un trabajo anterior [3], se presentaron dos algoritmos de inferencia de restricciones de rendimiento en modelos de flujos de trabajo. Estos algoritmos son capaces de completar los "vacíos" de un modelo en lo que respecta a peticiones por segundo y tiempos de respuesta exigidos, utilizando una serie de anotaciones locales y globales.

En este trabajo se describirá un plan de trabajo para explotar los modelos obtenidos tras la ejecución de los algoritmos de inferencia. Los modelos se usarán para generar planes de prueba parciales y para convertir pruebas funcionales ya existentes en pruebas de rendimiento. Para ello, será necesario vincular los modelos de rendimiento a modelos de diseño o implementación, de forma que se relacionen los requisitos de rendimiento con los artefactos software oportunos.

El resto del trabajo se estructurará de la siguiente forma: tras describir brevemente los modelos empleados, se describirá el enfoque general a utilizar para generar las pruebas. A continuación, se mostrarán dos posibles aplicaciones del enfoque, vinculando los requisitos de rendimiento a varios tipos de artefactos software, y se señalarán algunas de las tecnologías candidatas. Finalmente, se ofrecerán las conclusiones del trabajo, así como una serie de líneas futuras de trabajo.

2. Modelos de rendimiento en MARTE

En esta sección se dará un ejemplo de un modelo anotado por los algoritmos de inferencia de [3] con los tiempos límite y peticiones por segundo de cada actividad.

Los modelos utilizados son diagramas de actividad UML estándares, anotados con un subconjunto del perfil OMG MARTE [4]. Puede verse un ejemplo de un modelo anotado en la Figura 1: las salidas de los algoritmos de

inferencia se destacan en negrita. Se utilizan varios estereotipos MARTE:

- La actividad completa incluye un estereotipo «GaScenario», en el que *respT* indica que se exige que toda solicitud se atienda en menos de 1 segundo, y *throughput* indica que llegará una nueva petición por segundo en promedio.
- A su vez, la actividad completa declara una serie de parámetros de contexto en el campo *contextParam* de «GaAnalysisContext». Son variables que parametrizan el modelo, indicando cuánto tiempo por unidad de peso debe asignarse a cada acción, más allá de su tiempo mínimo. Sus valores son calculados por el algoritmo de inferencia de tiempos límite.
- Cada una de las acciones de la actividad emplea «GaStep», utilizando en *hostDemand* una expresión de la forma $m + ws$, donde m es el tiempo mínimo, w es el peso de la acción para repartir el tiempo sobrante, y s es el parámetro de contexto asociado a la acción. Los algoritmos de inferencia añaden una restricción nueva a *hostDemand* indicando el tiempo límite calculado, e indican en *throughput* el número de peticiones por segundo promedio.
- Las aristas salientes de los nodos condición también usan «GaStep», pero en este caso se limitan a indicar en *prob* la probabilidad de que se cumpla la condición que representan.

3. Generación de pruebas de rendimiento

El modelo mostrado en la sección anterior es puramente abstracto: en el nivel de detalle utilizado, no puede ejecutarse de forma automática. En lugar de eso, se utilizará como referencia para implementar el flujo de trabajo en un lenguaje de programación.

Una vez se haya implementado el flujo de trabajo, sería deseable utilizar el mismo modelo de análisis para producir las pruebas de rendimiento. Sin embargo, este modelo carece de los detalles de implementación y diseño necesarios para producir artefactos ejecutables. Una posibilidad sería extender el modelo abstracto con esta información, pero ello lo haría más difícil de entender y limitaría la generación de pruebas a una tecnología concreta. Igualmente, extender el modelo de implementación o diseño con los requisitos de rendimiento tampoco sería deseable.

En este caso, la solución consiste en insertar un modelo intermedio entre ambos modelos, que establezca y califique las relaciones oportunas entre las entidades de cada modelo. A esto se le

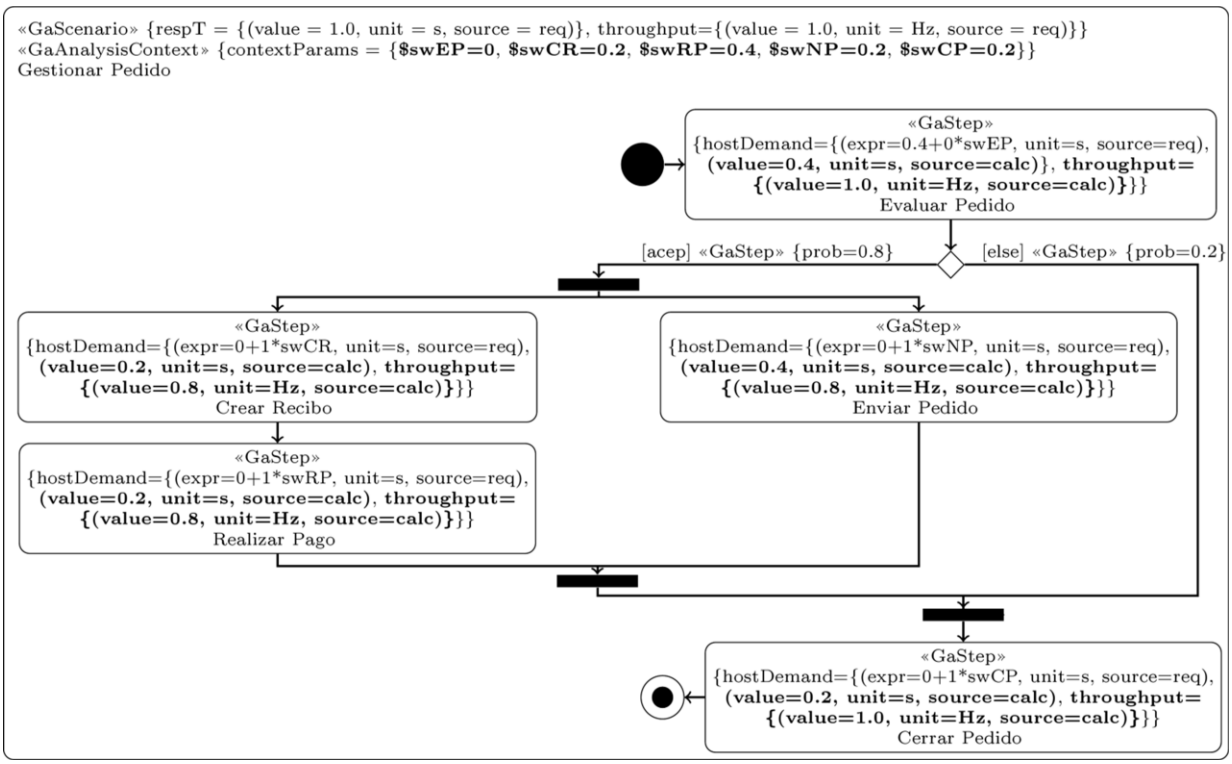

***Figura 1**. Ejemplo sencillo de modelo anotado por los algoritmos de inferencia.*

conoce como entrelazado de modelos o *model weaving*, y puede realizarse de forma genérica utilizando tecnologías como AMW [5] o Epsilon ModeLink [6].

Habiendo establecido los vínculos necesarios entre el modelo de requisitos de rendimiento y el modelo de diseño o implementación, el siguiente paso será generar las pruebas en sí. Para ello, se partirá del modelo de entrelazado y se utilizará una tecnología de transformación de modelo a texto (*model to text* o M2T) como el Epsilon Generation Language [7]. Si el modelo de entrelazado necesita algún tipo de adaptación automática, se aplicará previamente una transformación de modelo a modelo (*model to model* o M2M) en un lenguaje como el Epsilon Transformation Language [7].

El proceso que resulta de combinar estos diversos pasos se encuentra resumido en la Figura 2.

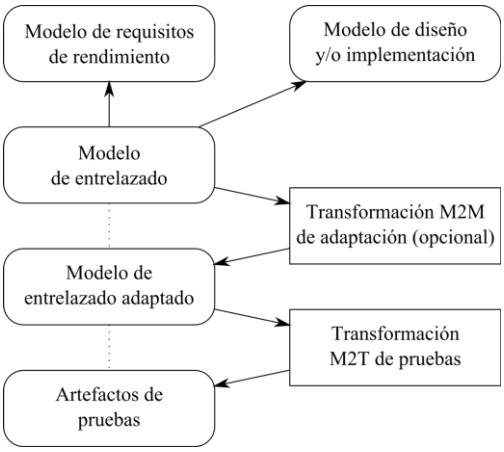

***Figura 2**. Esquema general para la generación de artefactos de pruebas de rendimiento.*

4. Conversión de pruebas JUnit en pruebas de rendimiento JUnitPerf

En esta sección se mostrará un caso concreto de cómo se podría aplicar el enfoque de la sección anterior para envolver pruebas funcionales ya existentes, escritas utilizando la biblioteca JUnit [8], empleando la biblioteca JUnitPerf [9].

La biblioteca JUnitPerf permite envolver un conjunto de pruebas basado en JUnit con un objeto decorador que verifica que se cumplen los aspectos de rendimiento. Así, si las pruebas originales se encontraran en la clase `PFuncionales` y se tuviera que atender en 1 segundo a 10 usuarios, se podría usar un fragmento de código Java como el del Listado 1.

El fragmento de código es pequeño, ya que la mayor parte de la funcionalidad está implementada en JUnitPerf. Sin embargo, generarlo de forma fiable es complejo, dado que es necesario invocar al constructor de la clase de pruebas con los argumentos apropiados y del tipo correcto. Esto exige obtener un modelo del código Java que se pueda entrelazar con el modelo de requisitos de rendimiento.

```
final int usuarios = 10;
final int tlimite_ms =  1000;
Test casoPrueba = new
  PFuncionales(…);
Test prueba1Usuario = new
  TimedTest(casoPrueba,
            tlimite_ms);
Test pruebaTodos = new
  LoadTest(prueba1Usuario,
           usuarios);
```

***Listado 1**. Parte del código a generar para envolver una prueba JUnit como una prueba de rendimiento JUnitPerf.*

Este problema ya ha sido resuelto en el proyecto Eclipse MoDisco [10], dedicado a la modernización del código ya disponible en una organización. MoDisco permite generar modelos automáticamente a partir del código Java de un proyecto Eclipse, con lo que el trabajo necesario para esta línea se reduciría a la creación del metamodelo de entrelazado y a la implementación de la transformación M2T.

5. Generación de pruebas de rendimiento para servicios Web

En el caso de tener que probar un servicio Web [11], es necesario relacionar los requisitos de rendimiento con una operación concreta de su interfaz. La relación se puede establecer a varios niveles de abstracción.

El más evidente es utilizar el documento WSDL [12] que describe su interfaz. Para describir el documento WSDL como un modelo, se podría emplear un metamodelo basado en las declaraciones XML Schema del estándar, o un perfil UML como el propuesto en [13].

De forma menos evidente, hay bibliotecas como JAX-WS [14] que permiten desarrollar servicios Web como código Java normal, al que se le han añadido una serie de anotaciones estándar de Java 6. Se podría vincular un requisito de rendimiento a uno de los métodos públicos de la clase del Listado 2, utilizando MoDisco. JAX-WS se ocupará de generar los documentos WSDL y el código adicional necesario.

A la hora de probar el servicio en sí, existen diversas opciones. En caso de sólo disponer del WSDL, se puede generar un esqueleto del plan de pruebas para herramientas como Apache JMeter [15]. Si se dispone del código JAX-WS, se podrían utilizar herramientas Java estándar, o

partir del WSDL generado. Al ser una prueba de rendimiento, se podría ofrecer la posibilidad de generar entradas aleatorias, de acuerdo con una plantilla producida automáticamente.

```
@WebService
public class HolaMundo {
  @WebMethod
  public String saludar(
     String nombre) {
    return "Hola " + nombre;
  }
}
```

***Listado 2**. Fragmento Java que define el servicio Web "HolaMundo" con la operación "saludar" utilizando JAX-WS.*

4. Conclusiones

En este trabajo se ha descrito un enfoque general para relacionar los requisitos de rendimiento inferidos por los algoritmos desarrollados en trabajos previos [3] con los artefactos de diseño e implementación.

El enfoque general se ha concretado en dos casos: la conversión de una prueba JUnit en una prueba de rendimiento con JUnitPerf, y la generación de pruebas de rendimiento para un servicio Web descrito mediante WSDL. En este último caso, se ha propuesto tanto partir del documento WSDL en sí como del código JAX-WS que implementará el servicio.

El siguiente paso es seleccionar una tecnología de *model weaving* de entre las disponibles. A continuación, se desarrollará la línea de decoración de pruebas JUnit como prueba de concepto. La línea de servicios Web es más compleja: por un lado habrá que seleccionar una herramienta concreta de pruebas, ver cómo generar los mensajes de entrada automáticamente, y evaluar las opciones de partir del WSDL o partir del código JAX-WS.

5. Agradecimientos

Este trabajo fue parcialmente financiado por la beca de investigación PU-EPIF-FPI-C 2010-065 de la Universidad de Cádiz, y por el Proyecto Puente PR2011-004 del Plan Propio de la Universidad de Cádiz.

6. Referencias

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

[2] M. Woodside, G. Franks, y D. C. Petriu, «The Future of Software Performance Engineering», in *Proceedings of FOSE 2007*, 2007, págs. 171-187.

[3] A. García-Domínguez y I. Medina-Bulo, «Inferencia Automática de Requisitos Locales de Rendimiento en Flujos de Trabajo Anotados con MARTE», in *Actas de las XVI Jornadas de Ingeniería del Software y Bases de Datos*, A Coruña, Spain, 2011, págs. 585-598.

[4] Object Management Group, «UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE) 1.0», 02-Nov-2009. [Web]. Disponible en: http://www.omg.org/spec/MARTE/1.0/. [Último acceso: 13-Nov-2011].

[5] M. D. Del Fabro, J. Bézivin, y P. Valduriez, «Weaving Models with the Eclipse AMW plugin», in *Proceedings of the 2006 Eclipse Modeling Symposium, Eclipse Summit Europe*, Esslingen, Germany, 2006.

[6] D. S. Kolovos, «Epsilon ModeLink». [Web]. Disponible en: http://eclipse.org/gmt/epsilon/doc/modelink/. [Último acceso: 13-Nov-2011].

[7] D. S. Kolovos, L. M. Rose, y R. F. Paige, «The Epsilon Book». 05-Mar-2011.

[8] K. Beck, «JUnit.org», 15-Abr-2011. [Web]. Disponible en: http://www.junit.org/. [Último acceso: 13-Nov-2011].

[9] M. Clark, «JUnitPerf», 2009. [Web]. Disponible en: http://clarkware.com/software/JUnitPerf.html. [Último acceso: 13-Nov-2011].

[10] H. Bruneliere, J. Cabot, F. Jouault, y F. Madiot, «MoDisco: a generic and extensible framework for model driven reverse engineering», in *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, Antwerp, Bélgica, 2010, págs. 173–174.

[11] W3C, «Web Services Glossary», 11-Feb-2004. [Web]. Disponible en: http://www.w3.org/TR/ws-gloss/. [Último acceso: 13-Nov-2011].

[12] «Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language», W3C, Jun. 2007.

[13] J. M. Vara, V. de Castro, y E. Marcos, «WSDL automatic generation from UML models in a MDA framework», in *Proceedings of the 2005 International Conference on Next Generation Web Services Practices*, 2005.

[14] Java.net, «JAX-WS Reference Implementation», 04-Nov-2011. [Web]. Disponible en: http://jax-ws.java.net/. [Último acceso: 13-Nov-2011].

[15] Apache Software Foundation, «JMeter - Apache JMeter». [Web]. Disponible en: http://jakarta.apache.org/jmeter/. [Último acceso: 04-Ene-2010].