Modelado y Simulación de *Middleware* Orientado a Mensajes: Un enfoque exploratorio utilizando modelado basado en componentes

Carlos Martín Flores González
Escuela de Ingenería en Computación
Tecnológico de Costa Rica
Cartago, Costa Rica
mfloresg@ieee.org

***Resumen*—This document is a model and instructions for LaTeX. This and the IEEEtran.cls file define the components of your paper [title, text, heads, etc.]. *CRITICAL: Do Not Use Symbols, Special Characters, Footnotes, or Math in Paper Title or Abstract.**

***Palabras Clave*—component, formatting, style, styling, insert**

I. Introducción

EN ..This document is a model and instructions for LaTeX. Please observe the conference page limits.

II. Trabajos Relacionados

En [1] se extiende el *Palladio Component Model* con *performance completions* para *middleware* orientado a mensajes. *Performance completions* [2] proporcionan el concepto general de incluir detalles de bajo nivel de ambientes de ejecución en modelos de rendimiento. Con la extensión del modelo, los arquitectos de software puede especificar y configurar comunicación basada en mensajes utilizando un lenguaje basado en patrones de mensajería.

La influencia en el rendimiento de *middleware* orientado a mensajes fue estudiada en [3]. Se consideró el *middleware* como un factor determinante del rendimiento en sistemas distribuidos y se hizo un mayor enfoque en su modelado y evaluación. Se propuso un enfoque basado en medición en combinación con modelos matemáticos para predecir el rendimiento de aplicaciones J2EE[1]. Las mediciones proporcionan los datos necesarios para calcular los valores de entrada de un modelo de red de cola(*queueing network model*). El cálculo refleja el comportamiento de la aplicación en cuestión. La red de cola se resuelve para derivar métricas de rendimiento tales como tiempos de respuesta y *throughput* de la aplicación.

La investigación llevada a cabo en [4] presenta un modelo abstracto de *middleware* orientado a mensajes basado en Apache Qpid junto con el uso de modelos exógenos de autorregressión (ARX por sus siglas en inglés) que describen el comportamiento del *middleware* durante condiciones de cuellos de botella. Los modelos ARX son modelos autorregresivos en donde la salida depende de la salida anterior así como de estimulos externos. Estos componentes son integrados para producir una técnica generalizada de calibración para rendimiento del *middleware* y detección de cuellos de botella en el mismo.

En [5] se construyen modelos de rendimiento para *middleware* que implementa el estándar JMS[2]. Se utiliza análisis de código y mediciones experimentales de implementaciones de JMS populares para mostrar situaciones en las que el rendimiento observado no es predecido de forma precisa por otros modelos. Se proporciona un análisis técnico detallado de los efectos observados como base para futuros trabajos de modelado. Por último, se diseña un modelo de rendimiento que captura estos efectos y se valida el modelo utilizando mediciones experimentales.

Un modelo analítico $M/M/1$ con políticas *first in - first out* y prioridad de colas fue diseñado y desarrolloado en [6] para evaluar el rendimiento de *middleware* orientado a mensajes y llamados a procedimientos remotos (RPC por sus siglas en inglés). Los modelos comparan el rendimiento de *middleware* orientado a mensajes y RPC con prioridad de colas y analizan el *throughput* de estos paradigmas de comunicación. Varios parámetros de entrada son usados para determinar la configuración óptima para lograr el máximo rendimiento. Los resultados prueban que al utilizar *middleware* orientado a mensajes con prioridad de cola, el *throughput* del sistema puede ser mejorado.

III. Ingeniería de Rendimiento de Software (*Software Performance Engineering – SPE*)

Una definición comúnmente utilizada para definir ingeniería de rendimiento de software es la que que brinda Woodside [7]: *"Ingeniería de rendimiento de software representa toda la colección de actividades de ingeniería de sofware y análisis relacionados utilizadas a través del ciclo de desarrollo de software, que están dirigidos a cumplir con los requisitos de rendimiento"*.

De acuerdo con este mismo autor, los enfoques para ingeniería de rendimiento puede ser divididos en dos categorías:

[1] *Java Enterprise Edition*

[2] *Java Messaging Service*

basadas en mediciones y basadas en modelos. La primera es la más común y utiliza pruebas, diagnóstico y ajustes una vez que existe un sistema en ejecución que se puede medir, es por esto que solamente puede ser utilizada conforme se va acercando el final del ciclo de desarrollo de software. Al contrario del enfoque basado en mediciones, el enfoque basado en modelos se centra en las etapas iniciales del desarrollo. Como el nombre lo indica, en este enfoque los modelos son clave para hacer predicciones cuantitativas de qué tan bien una arquitectura puede cumplir sus expectativas de rendimiento.

Se han propuesto otras clasificaciones de enfoques para SPE pero, con respecto a la evaluación de sistemas basados en componentes, en [8] se deja la clasificación a un lado debido a que se argumenta que la mayoría de enfoques de modelaje toman alguna medición como entrada y a la mayoría de los métodos de medición los acompaña algún modelo.

III-1. Ingeniería de rendimiento basada en mediciones: Los enfoques basados en mediciones prevalencen en la industria [9] y son típicamente utilizados para verificación(¿el sistema cumple con su requisito de rendimiento?) o para localizar y arreglar *hot-spots* (cuáles son las partes que tienen peor rendimiento en el sistema). La medición de rendimiento se remonta al inicio de la era de la computación, lo que ha generado una amplia gama de herramientas como generadores de carga y monitores para crear cargas de trabajo ficticias y llevar a cabo la medición de un sistema respectivamente.

Las pruebas de rendimiento aplican técnicas basadas en medición y usualmente esto es hecho luego de las pruebas funcionales o de carga. Las pruebas de carga verifican el funcionamiento de un sistema bajo cargas de trabajo pesadas, mientras que las pruebas de rendimiento son usadas para obtener datos cuantitativos de características de rendimiento, como tiempos de respuesta, *throughput* y utilización de hardware para una configuración de un sistema bajo una carga de trabajo definida.

III-A. Ingeniería de rendimiento a través de modelado

La importancia del modelado del rendimiento está motivada por el riesgo de problemas graves de rendimiento [**?**] y la creciente complejidad de sistemas modernos, lo que hace difícil abordar los problemas de rendimiento al nivel de código. Cambios considerables en el diseño o en las arquitecturas pueden ser requeridos para mitigar los problemas de rendimiento. Por esta razón, la comunidad de investigación de modelado de rendimiento intenta luchar contra el enfoque de "arreglar las cosas luego" durante el proceso de desarrollo. Con la aplicación de modelo del rendimiento de software se busca encontrar problemas de rendimiento y alternativas de diseño de manera temprana en el ciclo de desarrollo, evitando así el costo y la complejidad de un rediseño o cambios en los requerimientos.

Las herramientas de modelado de rendimiento ayudar a predecir la conducta del sistema antes que este sea construido o bien, evaluar el resultado de un cambio antes de su implementación. El modelado del rendimiento puede ser usado como una herramienta de alerta temprana durante todo el ciclo de desarrollo con mayor precisión y modelos cada vez más detallados a lo largo del proceso. Al inicio del desarrollo un modelo no puede ser validado contra un sistema real, por esto el modelo representa el conocimiento incierto del diseñador. Como consecuencia de esto el modelo hace suposiciones que no necesariamente se van a dar en el sistema real, pero que van a ser útiles para obtener una abstracción del comportamiento del sistema. En estas fases iniciales, la validación se obtiene mediante el uso del modelo, y existe el riesgo de conclusiones erróneas debido a su precisión limitada. Luego, el modelo puede ser validado contra mediciones en el sistema real (o parte de este) o prototipos y esto hace que la precisión del modelo se incremente.

En [10] se sugiere que los métodos actuales tienen que superar un número de retos antes que puedan ser aplicados en sistemas existentes que enfrentan cambios en su arquitectura o requerimientos. Primero, debe quedar claro cómo se obtienen los valores para los parámetros del modelo y cómo se pueden validar los supuestos. Estimaciones basadas en la experiencia para estos parámetros no son suficientes y mediciones en el sistema existente son necesarias para hacer predicciones precisas. Segundo, la caracterización de la carga del sistema en un entorno de producción es problemática debido a los recursos compartidos (bases de datos, hardware). Tercero, deben desarrollarse métodos para capturar parámetros del modelo dependientes de la carga. Por ejemplo un incremento en el tamaño de la base de datos probablemente incrementará las necesidades de procesador, memoria y disco en el servidor.

Técnicas comúnes de modelado incluyen redes de colas, extensiones de estas como redes de colas en capas y varios tipos de redes de Petri y procesos de álgebra estocástica.

III-B. Modelado de Rendimiento

En SPE, la creación y evaluación de modelos de rendimiento es un concepto clave para evaluar cuantitativamente el rendimiento del diseño de un sistema y predecir el rendimiento de otras alternativas de diseño. Un modelo de rendimiento captura el comportamiento relevante al rendimiento de un sistema para identificar el efecto de cambios en la configuración o en la carga de trabajo en el rendimiento. Permite predecir los efectos de tales cambios sin necesidad de implementación y ejecución en un ambiente de producción, que podrían ser no solamente tareas costosas sino también un desperdicio en el caso que un el hardware con el que se cuenta pruebe ser insuficiente para soportar la intensidad de la carga de trabajo. [11]

La forma del modelo de rendimiento puede comprender desde funciones matemáticas a formalismos de modelado estructural y modelos de simulación. Estos modelos varían en sus características clave, por ejemplo, las suposiciones de modelado de los formalismos, el esfuerzo de modelado requerido y el nivel de abstracción.

En cuanto a técnicas de simulación, a pesar que estas permiten un estudio más detallado de los sistemas que modelos analíticos, la construcción de un modelo de simulación requiere de conocimiento detallado tanto de desarrollo de software como de estadística [9]. Los modelos de simulación también

requieren usualmente de mayor tiempo de desarrollo que los modelos analíticos. En [7] se menciona que "la construcción de un modelo de simulación es caro, algunas veces comparable con el desarrollo de un sistema, y, los modelos de simulación detallados puede tardar casi tanto en ejecutarse como el sistema.

III-C. Modelado de Arquitecturas de Software con Palladio Component Model

El *Palladio Component Model (PCM)* es un enfoque de modelaje para arquitecturas de software basados en componentes que permite predicción de rendimiento basada en modelos. PCM contribuye al proceso de desarrollo de ingeniería basado en componentes y proporciona conceptos de modelaje para describir componentes de software, arquitectura de software, despliegue (*deployment*) de componentes y perfiles de uso de sistemas de software basados en componentes en diferentes submodelos (Figura 1).

- **Especificaciones de componentes** son descripciones abstractas y paramétricas de los componentes de software. En las especificaciones de software se proporciona una descripción del comportamiento interno del componente así como las demandas de sus recursos en RDSEFFs (*Resource Demanding Service EFFect specifications*) utilizando una sintaxis similar a los diagramas de actividad de UML.
- **Un modelo de ensamblaje** (*assembly model*) especifica qué tipo de componentes se utilizan en una instancia de aplicación modelada y si las instancias del componente se replican. Además, define cómo las instancias del componente se conectan representando la arquitectura de software.
- El entorno de ejecución y los recursos, así como el despliegue (*deployment*) de instancias de componentes para dichos contenedores de recursos se definen en un **modelo de asignación** (*allocation model*).
- El **modelo del uso** especifica la interacción de los usuarios con el sistema utilizando una sintaxis similar al diagrama de actividades de UML proporcionando una descripción abstracta de la secuencia y la frencuencia en que los usuarios activan las operaciones disponibles en un sistema.

Un modelo PCM abstrae un sistema de software a nivel de arquitectura y se anota con consumos de recursos que fueron medidos previamente u otros que son estimados. El modelo puede entonces ser usado en transformaciones de modelo-a-modelo o modelo-a-texto a un modelo de análisis en particular (redes de colas o simulación de código) que puede ser analíticamente resuelto o simulado para obtener resultados sobre el rendimiento y predicciones del sistema modelado. Los resultados del rendimiento y las predicciones pueden ser utilizadas como retroalimentación para evaluar y mejorar el diseño inicial, permitiendo así una evaluación de calidad de los sistemas de software en base a un modelo [11].

IV. *Middleware* Orientado a Mensajes

Junto con el crecimiento de Internet, los sistemas distribuidos han crecido a una escala masiva. Hoy en día, estos sistemas puede involucrar a miles de entidades distruidas globalmente. Esto ha motivado el estudio de modelos de comunicación y sistemas flexibles, que puedan reflejar la naturaleza dinámica y desacoplada de las aplicaciones.

La integración de tecnologías heterogéneas es una de las áreas principales en donde la comunicación basada en mensajes juega un papel clave. Más y más compañías se enfrentan al problema de integrar sistemas y aplicaciones heterogéneas dentro y fuera de la organización ya sea por fusiones, adquisiciones, requisitos comerciales o simplemente un cambio en la direción tecnológica. No es raro encontrar una gran cantidad de tecnologías y plataformas dentro de una solo compañía, desde productos de código libre y comerciales hasta sistemas y equipos heredados (*legacy*).

La comunicación basada en mensajes también ofrece la habilidad de procesar solicitudes de manera asincrónica, proporcionando a los arquitectos y desarrolladores soluciones para reducir o eliminar cuellos de botella en un sistema e incrementar la productividad del usuario final y del sistema en general. Dado que la comunicación basada en mensajes provee un alto grado de desacoplamiento entre componentes, los sistemas que utilizan esta tecnología también logran contar con altos grados de agilidad y flexibilidad en su arquitectura.

A los sistemas de mensajería de aplicación-a-aplicación que se utilizan sistemas de negocios se les denomina genéricamente sistemas de mensajería empresarial o *middleware* orientado a mensajes [12]. MOM permite a dos o más aplicaciones intercambiar información en forma de mensajes. Un mensaje en este caso es un paquete autocontenido de datos de negocio y encabezados de enrutamiento de red. Los datos de negocio contenidos en un mensaje puede ser cualquier cosa, van a depender de cada negocio, y usualmente contiene información acerca de algún tipo de transacción. En sistemas de mensajería empresariales, los mensajes informan a una aplicación de la ocurrencia de algún evento en otro sistema. La tasa de mensajes depende de la capacidad de la implementación de MOM o *broker*. El retraso depende de la latencia en el *broker* y en los caminos de entrada y salida [4].

Al usar MOM, los mensajes son trasmitidos desde una aplicación a otra a través de la red. Productos de *middleware* empresarial aseguran que los mensajes se distribuyan correctamente entre las aplicaciones. Además estos productos usualmente proporcionan tolerancia a fallos, balanceo de carga, escalabilidad y soporte transaccional para sistemas que necesitan que necesitan intercambiar de manera confiable grandes cantidades de mensajes.

Los fabricantes de MOM usan diferentes formatos de mensajes y protocolos de red para intercambiar mensajes pero la semántica básica es la misma. Una interfaz de programación (un API, por sus siglas en inglés) se utiliza para crear un mensaje, cargar los datos, asignar información de enrutamiento

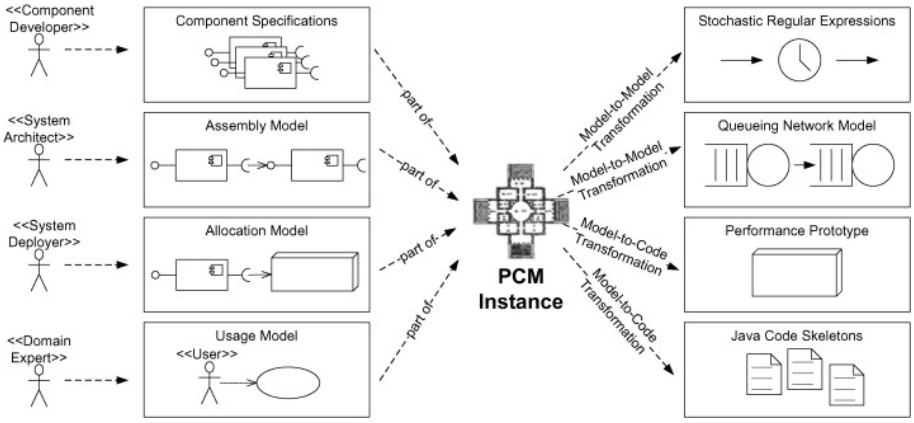

Figura 1. Instancia de un modelo PCM. Tomado de [1]

y enviar el mensaje. La misma API se utiliza para recibir los mensajes producidos por otras aplicaciones.

En todos los sistemas modernos de mensajería empresarial, las aplicaciones intercambian mensajes a través de canales virtuales llamados destinos(*destinations*). Cuando un mensaje se envía, se dirije a un destino (por ejemplo una cola o un tópico) no a una aplicación específica. Cualquier aplicación que subscriba o registre un interés en ese destino puede recibir el mensaje. De esta forma, las aplicaciones que reciben mensajes y aquellas que envían mensajes están desacopladas. Los emisores y receptores no están enlazados uno con otro de ninguna forma y pueden enviar y recibir mensajes como mejor les parezca.

Un concepto clave de MOM es que los mensajes son entregados de forma asincrónica desde un sistema a otros sobre la red. El entregar un mensaje de manera asicrónica significa que el emisor no requiere esperar a que el mensaje sea recibido o manejado por el receptor, él es libre de enviar el mensaje y continuar su procesamiento. Los mensajes asicrónicos son tratados como unidades autónomas: cada mensaje es autocontenido y lleva consigo todos los datos necesarios para ser procesado.

En comunicación de mensajes asincrónicos, las aplicaciones usan una API para construir un mensaje, luego pasarlos al MOM para su entrega a uno o varios recipientes (Figura 2). Un mensaje es un paquete de datos que es enviado desde una aplicación a otra sobre la red. El mensaje debe de ser autodescriptivo en el sentido que debe de contener todo el contexto necesario para que permit a los recipientes llevar a cabo su trabajo de forma independiente.

La arquitecturas de MOM de hoy en día varían en su implementación. Van desde las arquitecturas centralizadas que dependen de un servidor de mensajes para realizar enrutamiento a arquitecturas descentralizadas que distribuyen el procesamiento del servidor hacia los clientes. Una variedad de protocolos como `TCP/IP`, `HTTP`, `SSL` y `IP` son empleados como capa de transporte de red.

Los sistemas de mensajería están compuestos por clientes de mensajería (*messaging clients*) y algún tipo de servidor

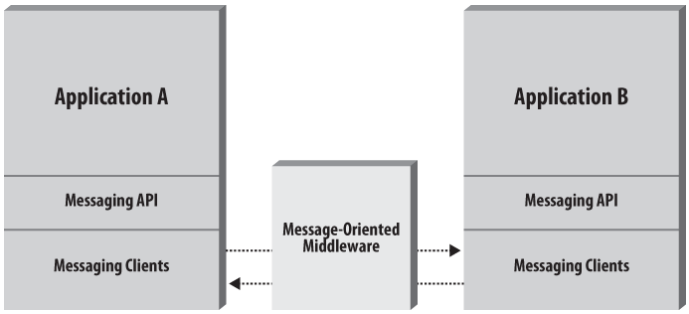

Figura 2. *Middleware* Orientado a Mensajes. Tomado de [12]

MOM. Los clientes envían mensajes al servidor de mensajería el cual distribuye estos mensajes a otros clientes. El cliente es una aplicación de negocio o componente que es usa una API de mensajería.

IV-1. Punto-a-Punto: Este modelo de mensajería permite a los clientes enviar y recibir mensajes de forma síncrona como asíncrona a través de canales virtuales conocidos como colas. En este modelo a los productores de mensajes se les llama emisores (*senders*) y a los consumidores se les conoce como receptores (*receivers*). El modelo punto-a-punto ha sido tradicionalmente un modelo basado en *polling*, en donde los mensajes son solicitados desde la cola en lugar de ser puestos en el cliente automáticamente. Los mensajes que se envían a una cola son recibidos por uno y solo un *receiver*, aunque pueden haber otros *receivers* escuchando en la cola por el mismo mensaje. Este es un modelo que promueve acoplamiento esto porque generalmente el *sender* conocer cómo el mensaje va a ser utilizado y quién lo va a recibir.

IV-2. Publish-and-Subscribe: En este modelo, los mensajes son publicados en un canal virtual llamado tópico. Los productores de mensajes son conocidos como *publishers* y a los consumidores se les llama *subscribers*. Los mensajes pueden ser recibidios por múltiples *subscribers*, a diferencia del modelo punto-a-punto. Cada *subscriber* recibe una copia de cada mensaje. Este es un modelo basado en *push* en donde los mensajes son automáticamente transmitidos a los

consumidores sin que estos tengan que solicitarlos o revisar la cola por nuevos mensajes.

Este modelo tiende a ser menos acoplado que el modelo punto-a-punto debido a que el publicador del mensaje generalmente no está conciente de cuántos subscritores hay y lo qué van a hacer estos con el mensaje.

V. Modelado de rendimiento en una Aplicación: *CloudStore*

VI. Resultados

VII. Conclusión

Referencias

[1] Jens Happe, Holger Friedrich, Steffen Becker, and Ralf H. Reussner. *A pattern-based performance completion for Message-oriented Middleware*. In Proceedings of the 7th international workshop on Software and performance (WOSP '08). ACM, New York, NY, USA, 165-176. 2008. DOI: http://ezproxy.itcr.ac.cr:2075/10.1145/1383559.1383581

[2] Murray Woodside, Dorin Petriu, and Khalid Siddiqui. *Performance-related Completions for Software Specifications*. In Proceedings of the 24th International Conference on Software Engineering, pages 22–32, New York, NY, USA, 2002.

[3] Yan Liu and Ian Gorton. *Performance prediction of J2EE applications using messaging protocols*. In Proceedings of the 8th international conference on Component-Based Software Engineering (CBSE'05), George T. Heineman, Ivica Crnkovic, Heinz W. Schmidt, Judith A. Stafford, and Clemens Szyperski (Eds.). Springer-Verlag, Berlin, Heidelberg, 1-16. 2005. DOI http://ezproxy.itcr.ac.cr:2075/10.1007/11424529_1

[4] Chew, Zen Bob. *Modelling Message-oriented-middleware Brokers Using Autoregressive Models for Bottleneck Prediction*. PhD Thesis. Queen Mary, University of London. 2013. https://qmro.qmul.ac.uk/jspui/handle/123456789/8832

[5] Tomáş Martinec, Lukáş Marek, Antonín Steinhauser, Petr Tůma, Qais Noorshams, Andreas Rentschler, and Ralf Reussner. *Constructing performance model of JMS middleware platform*. In Proceedings of the 5th ACM/SPEC international conference on Performance engineering (ICPE '14). ACM, New York, NY, USA, 123-134. 2014. DOI: https://ezproxy.itcr.ac.cr:2878/10.1145/2568088.2568096

[6] S.S. Alwakeel and H.M. Almansour. *Modeling and Performance Evaluation of Message-oriented Middleware with Priority Queuing*. Information Technology Journal, 10: 61-70. 2011. DOI http://dx.doi.org/10.3923/itj.2011.61.70

[7] Murray Woodside, Greg Franks, and Dorina C. Petriu. *The Future of Software Performance Engineering*. Future of Software Engineering (FOSE '07), pages 171-187, May 2007. DOI 10.1109/FOSE.2007.32

[8] Heiko Koziolek. 2010. *Performance evaluation of component-based software systems: A survey. Perform*. Eval. 67, 8 (August 2010), 634-658. DOI: http://ezproxy.itcr.ac.cr:2075/10.1016/j.peva.2009.07.007

[9] Thijmen de Gooijer. *Performance Modeling of ASP.Net Web Service Applications: an Industrial Case Study*. Master's thesis, Malardalen University, Vasteras, Sweden, 2011.

[10] Yan Jin, Antony Tang, Jun Han, and Yan Liu. *Performance Evaluation and Prediction for Legacy Information Systems*. 29th International Conference on Software Engineering (ICSE'07), pages 540-549, May 2007.

[11] Qais Noorshams. *Modeling and Prediction of I/O Performance in Virtualized Environments*. PhD thesis, Karlsruhe Institute of Technology (KIT), 2015. URL: http://digbib.ubka.uni-karlsruhe.de/volltexte/1000046750

[12] Mark Richards, Richard Monson-Haefel, David Chappell. *Java Message Service*. O'Reilly Media. Segunda Edición. 2009.