

**Modelado y Simulación de Colas de Mensajes Distribuidas**  
**(*Message Oriented Middleware*)**  
**Revisión de Literatura**

**Carlos Martín Flores González**

Carné: 2015183528

Profesor: César Garita  
Instituto Tecnológico de Costa Rica  
Escuela de Ingeniería en Computación  
Maestría en Computación  
MC-7205 Tema Selecto de Investigación

15 de Marzo del 2018

# Índice general

<b>Introducción</b>	<b>1</b>
<b>1. Revisión de la Literatura</b>	<b>3</b>
1.1. Estrategia . . . . .	3
1.2. Preguntas de Investigación . . . . .	3
1.3. Estrategia de Búsqueda . . . . .	3
1.4. Criterio de Selección . . . . .	4
1.5. Resultado de la Revisión . . . . .	4
<b>2. Ingeniería de rendimiento de software (<i>Software Performance Engineering</i>)</b>	<b>5</b>
2.1. Ingeniería de rendimiento basada en mediciones . . . . .	6
2.2. Ingeniería de rendimiento a través de modelado . . . . .	6
2.3. Modelado de Rendimiento . . . . .	7
2.3.1. Enfoques de ingeniería de rendimiento para software basado en componentes propuestos . . . . .	8
2.4. Modelado de Arquitecturas de Software con el <i>Palladio Component Model (PCM)</i> . . . . .	11
2.5. Retos de investigación asociados a Ingeniería de rendimiento . . . . .	12
2.5.1. Actividades de administración del rendimiento . . . . .	12
2.5.2. Modelos de rendimiento durante etapas diseño . . . . .	13
2.5.3. Retos de investigación en microservicios . . . . .	13
2.5.4. Herramientas para modelaje de rendimiento de software . . . . .	14
<b>3. Middleware Orientado a Mensajes</b>	<b>17</b>
3.1. Modelos de mensajería . . . . .	19
3.1.1. Punto-a-Punto . . . . .	19
3.1.2. <i>Publish-and-Subscribe</i> . . . . .	19

3.2. Enfoques de ingeniería de rendimiento para <i>middleware</i> orientado a mensajes . . . . .	20
<b>Bibliografía</b>	<b>21</b>

# Introducción

Los métodos de predicción de rendimiento basados en modelos permiten a los arquitectos de software evaluar el rendimiento de los sistemas de software durante las primeras etapas de desarrollo. Estos modelos de predicción se centran en los aspectos relevantes de la arquitectura y de la lógica del negocio, dejando de lado detalles de la infraestructura subyacente. Sin embargo, estos detalles son esenciales para generar predicciones de rendimiento que sean precisas.

Para los ingenieros, es una práctica común simular el modelo de un artefacto antes de construirlo. Modelos de diseños de autos, circuitos electrónicos, puentes, entre otros, son simulados para entender el impacto de decisiones de diseño en varias atributos de calidad de interés como seguridad, consumo de energía o estabilidad. La habilidad de predecir las propiedades de un artefacto en base a su diseño sin necesidad de construirlo, es una de las características centrales de una disciplina de ingeniería. A partir de esta visión de lo que se considera una disciplina de ingeniería establecida se podría decir entonces que la ingeniería de software es apenas una disciplina de ingeniería [1]. Esto porque frecuentemente los ingenieros de software carecen del entendimiento del impacto de decisiones de diseño en atributos de calidad como rendimiento o confiabilidad. Como resultado, se intenta probar la calidad del software mediante costosos ciclos de prueba y error.

El no entender el impacto en las decisiones de diseño puede ser costoso y riesgoso. Probar software significa que ya se ha hecho un esfuerzo en su implementación. Por ejemplo, si las pruebas revelan problemas de rendimiento, es muy probable que la arquitectura necesita ser modificada, lo que puede conllevar a costos adicionales. Estos costos surgen debido a que en sistemas de software empresarial un bajo rendimiento es principalmente el efecto de una arquitectura inadecuada que efecto de código.

La ingeniería de rendimiento de software (SPE por sus siglas en inglés) es una disciplina que se centra en incorporar aspecto de rendimiento dentro del proceso de desarrollo de software, con el objetivo de entregar software de confiable de acuerdo con propiedades de rendimiento particulares. Los modelos de rendimiento predictivos son una de las herramientas empleadas en SPE. Construidos en las fases tempranas del proceso de desarrollo de software, los modelos ayudan a predecir el rendimiento eventual

del software y de esta forma guiar el desarrollo, para eso los modelos de predicción de rendimiento deben capturar todos los componentes relevantes del sistema.

Para aplicaciones de software moderans, esto puede implicar modelar complejas capas tales como máquinas virtuales o *middleware* de mensajería. Componer todos estos modelos puede resultar una tarea costosa e ineficiente. En su lugar, un modelo abstracto de la aplicación se puede construir primero y luego ir agregando los modelos de los componentes del sistema.

En este trabajo se propone la construcción de un modelo de rendimiento para un sistema que utilice *middleware* orientado a mensajes con el fin de evaluar la influencia en el rendimiento de dicho sistema. Se propone evaluar esta influencia por medio de un ejemplo: tomar una aplicación de referencia con el fin de obtener sus métricas actuales de rendimiento, adaptarla para que utilice *middleware* orientado a mensajes y luego medir su rendimiento y generar un modelo a partir de esto.

# Capítulo 1

## Revisión de la Literatura

### 1.1. Estrategia

Esta sección presenta la estrategia emprendida para cubrir el cuerpo de conocimiento relacionado con ingeniería de rendimiento de software. La estrategia general se basó en un proceso iterativo de identificación y lectura de artículos, luego identificar y leer artículos relevantes a partir de referencias y citas bibliográficas.

### 1.2. Preguntas de Investigación

Las preguntas de investigación seleccionadas para conducir la revisión de la literatura fueron:

- PI1** ¿Cuáles enfoques de predicción y medición del rendimiento en sistemas de software basados en componente se han propuesto?
- PI2** ¿Cuáles enfoques de predicción y medición de rendimiento de software se han utilizado para *middleware* orientado a mensajes?
- PI3** ¿Qué retos y oportunidades existen con estos enfoques en la actualidad?
- PI4** ¿Qué herramientas hay disponibles para modelaje de rendimiento de software?

### 1.3. Estrategia de Búsqueda

Con el fin de identificar el primer conjunto de artículos relevantes, se hizo una revisión preliminar con Google Scholar<sup>1</sup> porque con este motor de búsqueda se puede abarcar un amplio número de artículos y actas académicas de diferentes fuentes. El

---

<sup>1</sup><http://scholar.google.com>

criterio de búsqueda se basó en búsquedas de palabras derivadas del tema de investigación y las preguntas de investigación. Se incluyeron palabras como “component-based software”, “software performance engineering”, “software performance modeling”, “middleware oriented messages”, “modeling middleware oriented messages”, “broker modeling”, “palladio component model”, “modeling software architecture”

La búsqueda de la literatura se realizó en Marzo del 2018 usando las siguientes bases de datos electrónicas:

- *ACM Digital Library*
- *IEEE Explore Digital Library*
- Safari Books Online<sup>2</sup>
- Google Scholar

## 1.4. Criterio de Selección

Se aplicó el siguiente criterio de inclusión de artículos para esta revisión:

- Estudios que dan a conocer enfoques de análisis de rendimiento de software basado en componentes
- Estudios sobre modelaje de rendimiento de software
- Estudios sobre modelaje de rendimiento en *middleware* orientado a mensajes
- Estudios, aplicaciones o casos de uso de *Palladio Component Model*
- Estudios que reporten sobre éxito, fracaso y retos de la ingeniería de rendimiento de software basado en componentes

Los criterios de exclusión de artículos fueron los siguiente:

- Estudios relacionados ingeniería de rendimiento de software pero que no cubren o cubren muy poco aspectos sobre modelaje y simulación de arquitecturas.
- Artículos publicados hace más de 15 años atrás (rango aceptable 2003-2018).

## 1.5. Resultado de la Revisión

A continuación se presenta...

---

<sup>2</sup><https://www.safaribooksonline.com>

## Capítulo 2

# Ingeniería de rendimiento de software (*Software Performance Engineering*)

Una definición comúnmente utilizada para definir ingeniería de rendimiento de software es la que brinda Woodside [18]: “Ingeniería de rendimiento de software (*Software Performance Engineering* - SPE) representa toda la colección de actividades de ingeniería de software 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 pueden ser divididos en dos categorías: 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 y iniciación con el enfoque de ingeniería de rendimiento propuesto por Smith [19]. 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 [14] se deja 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.



## 2.1. Ingeniería de rendimiento basada en mediciones

Los enfoques basados en mediciones prevalecen en la industria [19] 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 remota 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.

## 2.2. Ingeniería de rendimiento a través de modelado

La importancia del modelado del rendimiento está motivada por el riesgo de problemas graves de rendimiento [20] 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 ayudan 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 [21] 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 comunes 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.

### **2.3. 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 implementarlo y ponerlo en 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. [12]

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 [19]. Los modelos de simulación también requieren usualmente de mayor tiempo de desarrollo que los modelos analíticos. En [18] 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 eje-

cutarse como el sistema.

### 2.3.1. Enfoques de ingeniería de rendimiento para software basado en componentes propuestos

La encuesta llevada a cabo en [14] proporciona una clasificación de enfoques de medición y predicción de rendimiento para sistemas de software basados en componentes. Otra clasificación es la que se expone en [22] donde se presenta una revisión de métodos de predicción de rendimiento basado en modelos para sistemas en general, pero no analiza los requerimientos propios para sistemas basados en componentes.

De acuerdo con [14] durante los últimos diez años, los investigadores han propuesto muchos enfoques para evaluar el rendimiento (tiempos de respuesta, *throughput*, utilización de recursos) de sistemas de software basados en componentes. Estos enfoques tratan con predicción de rendimiento y medición del rendimiento. Los primeros analizan el rendimiento esperado de un diseño de software basado en componentes para evitar problemas de rendimiento en la implementación del sistema, lo que podría llevar a costos substanciales para rediseñar la arquitectura. Los otros analizan el rendimiento observable de sistemas de software basados en componentes implementados para entender sus propiedades, determinar su capacidad máxima, identificar componentes críticos y para remover cuellos de botella.

#### Métodos de evaluación de rendimiento

Los enfoques se agruparon en dos grandes grupos: enfoques principales que proporcionan procesos de evaluación de rendimiento completo y enfoques suplementarios que se centran en aspectos específicos como medición de componentes individuales o modelaje de las propiedades de rendimiento de los conectores de un componente.

#### Enfoques principales

- **Enfoques de predicción basados en UML:** los enfoques en este grupo se enfocan en la predicción de rendimiento en tiempo de diseño para sistemas de software basado en componentes modelados con el Lenguaje de Modelado Unificado (UML por sus siglas en inglés). UML 2.0 tiene la noción de componente de software como una clase extendida. UML permite modelar el comportamiento de un componente con diagramas de secuencia, actividad y colaboración. El alojamiento de los componentes puede ser descrito como con diagramas de despliegue(*deployment*).
- CB-SPE - *Component-Based Software Performance Engineering*

- **Enfoques de predicción basados en Meta-Modelos propietarios:** Los enfoques en este grupo apuntan a las predicciones de rendimiento de tiempo de diseño. En lugar de usar UML como lenguaje de modelado para desarrolladores y arquitectos, estos enfoques tienen meta-modelos propietarios.
  - CBML - *Component-Based Modeling Language*
  - PECT - *The Prediction Enabled Component Technology*
  - COMQUAD - *Components with Quantitative properties and Adaptivity*
  - KLAPPER
  - ROBOCOP
  - Palladio
  
- **Enfoques de predicción centrados en *middleware*:** hacen énfasis en la influencia del *middleware* en el rendimiento de un sistema basado en componentes. Por lo tanto miden y modelan el rendimiento de plataformas *middleware* como JavaEE o .Net. Se basan en la suposición que la lógica de negocio de los componentes como tal tienen poco impacto en el rendimiento general del sistema y por eso no requieren un modelado detallado.
  - NICTA
  
- **Enfoques basados en especificaciones formales:** estos enfoques siguen teorías fundamentales de especificación de rendimiento y no toman en cuenta marcos de trabajo (*frameworks*) de medición y predicción.
  - RESOLVE
  - HAMLET
  
- **Enfoques de monitoreo para sistemas implementados:** asumen que un sistema basado en componentes ha sido implementado y puede ser probado. El objetivo es encontrar problemas de rendimiento en un sistema en ejecución, identificar cuellos de botella y adaptar el sistema para que pueda lograr los requerimientos de rendimiento.
  - COMPAS
  - TESTEJB
  - AQUA
  - PAD

## Enfoques Suplementarios

- **Enfoques de monitoreo para componentes implementados:** El objetivo de los enfoques de medición para implementaciones de componentes de software individuales es derivar especificaciones de rendimiento parametrizadas a través de mediciones múltiples. El objetivo es obtener el perfil de uso, dependencias y la plataforma de implementación a partir de la especificación de rendimiento, de modo que pueda usarse en diferentes contextos.
  - RefCAM
  - COMAERA
  - ByCounter
- **Enfoques de predicción con énfasis en conectores de componentes:** Estos enfoques asumen un lenguaje de descripción de componentes existente y se centran en modelar y medir el impacto en el rendimiento de las conexiones entre componentes. Estas conexiones pueden implementarse con diferentes técnicas *middleware*.
  - Verdickt
  - Grassi
  - Becker
  - Happe

Recientemente varios enfoques de predicción basados en meta-modelos propietarios han sido propuestos para optimización de diseño de arquitecturas, modelado de calidad de servicio y escalabilidad. PerOpteryx [23] es un enfoque de optimización de diseño de arquitecturas que manipula modelos especificados en *Palladio Component Model* (PCM) [1] y utiliza el algoritmo evolucionario multi-objetivo NSGA-II. Para análisis de rendimiento utiliza redes de colas en capas. *Descartes Modeling Language* [24] es un lenguaje de modelado de arquitecturas para modelar calidad de servicio y aspectos relacionados con la gestión de recursos de los sistemas, las infraestructuras y los servicios de tecnología de información dinámicos modernos. *CloudScale*<sup>1</sup> [25] es un enfoque de diseño y evolución de aplicaciones y servicios escalables en la nube. En CloudScale se identifica y gradualmente se resuelven problemas de escalabilidad en aplicaciones existentes y también permite el modelado de alternativas de diseño y el análisis del efecto de la escalabilidad en el costo. Cabe mencionar que estos últimos enfoques han sido influenciado en gran medida por el trabajo llevado a cabo en PCM.

---

<sup>1</sup><http://www.cloudscale-project.eu/>

## 2.4. Modelado de Arquitecturas de Software con el *Palladio Component Model (PCM)*

El *Palladio Component Model* es un enfoque de modelaje para arquitecturas de software basados en componentes que permite predicción de rendimiento basada en modelos. PCM contribuye el 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 2.1).

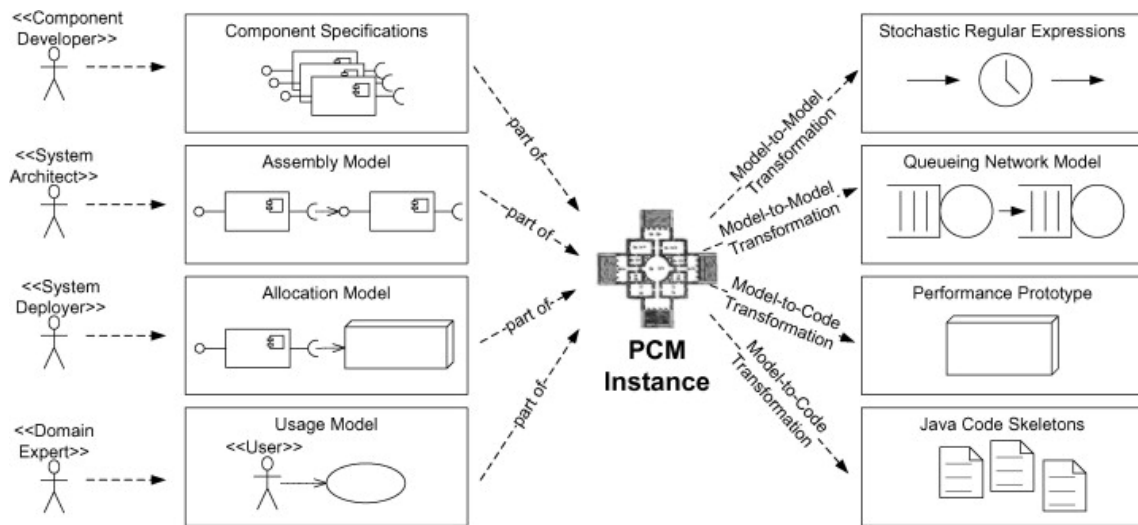


Figura 2.1: Instancia de un modelo PCM. Tomado de [13]

- **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 frecuencia 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 [12].

## **2.5. Retos de investigación asociados a Ingeniería de rendimiento**

Informes recientes como [6] y [7] señalan que los principales retos en investigación de ingeniería de rendimiento en la actualidad están asociados con generación de modelos a partir de código, integración de la evaluación de rendimiento en el ciclo de desarrollo y modelado de rendimiento de sistemas en la nube.

Con respecto a integración en ciclos de desarrollo de software y principalmente en aquellos basados en DevOps, una tendencia hacia una estrecha integración entre equipos desarrollo y operaciones, en [7] se destaca que la necesidad de tal integración está orientada por el requerimiento de adaptar aplicaciones empresariales a los cambios del ambiente del negocio continuamente. El rendimiento describe las propiedades del sistema con respecto a su ejecución y uso de recursos. Métricas comunes son tiempo de respuesta, *throughput* y la utilización de los recursos. Los objetivos de rendimiento para aplicaciones empresariales son típicamente definidos al establecer cotas superiores/inferiores para estas métricas y transacciones de negocio específicas.

### **2.5.1. Actividades de administración del rendimiento**

Evaluación del rendimiento basado en modelos:

- La representación de la memoria principal y del recolector de basura aún no están explícitamente integrados ni considerados en los modelos de rendimiento
- La selección de técnicas apropiadas de solución requiere de mucha experiencia

Extracción de modelos de rendimiento y de cargas de trabajo:

- La precisión de los modelos puede llegar a expirar si no son actualizados cuando hay cambios. Mecanismos de detección son requeridos para aprender cuando los modelos son antiguos y hay que actualizarlos.
- La extracción de capacidades de rendimiento se basa en una combinación de software y los recursos de hardware en los que se implementa. Este enfoque combinado es compatible con la precisión de la predicción, pero está menos calificado con respecto a la portabilidad de los conocimientos a otras plataformas. Una dirección de investigación futura podría ser extraer modelos separados (por ejemplo, *middleware* y modelos de aplicación separados).

### 2.5.2. Modelos de rendimiento durante etapas diseño

- Los retos de utilizar modelos de rendimiento en etapas tempranas de desarrollo es que usualmente es difícil validar la precisión de los modelos hasta que un sistema en ejecución exista. Las predicciones de rendimiento basadas en suposiciones, entrevistas y pruebas previas pueden ser también imprecisas y por tanto las decisiones que se hagan a partir de estas predicciones.

En relación con sistemas y arquitecturas en la nube como las basadas en microservicios otras tecnologías importantes para *deployment* como, virtualización basada en contenedores y soluciones de orquestación de contenedores han emergido. Estas tecnologías permiten explotar plataformas en la nube, proporcionando altos niveles de escalabilidad, disponibilidad y portabilidad para microservicios. A pesar del hecho de que es una necesidad inherente contar con escalabilidad y elasticidad, la ingeniería de rendimiento para microservicios hasta ahora ha tenido poca atención por parte de las comunidades tanto de microservicios como de comunidades de investigación de ingeniería de rendimiento [6]. Un gran cuerpo de conocimiento y buenas prácticas para ingeniería de rendimiento para desarrollo tradicional de software y arquitecturas está disponible. Sin embargo, sus aplicaciones en DevOps imponen tanto retos como oportunidades.

### 2.5.3. Retos de investigación en microservicios

#### Pruebas de rendimiento

- Reemplazar y compensar pruebas extensivas de integración y de sistema por un control detallado de los entornos de producción
- Alinear las pruebas de rendimiento y pruebas de regresión de rendimiento con prácticas de entrega continua, por ejemplo, acelerar las estas de pruebas correspondientes.



- Selección dinámica y semi automática de pruebas de rendimiento

#### Monitoreo

- Instrumentación para monitoreo distribuido arquitecturas microservicios políglotas
- Métricas adicionales para monitorear microservicios
- Técnicas de detección precisa de anomalías en arquitecturas de microservicios

Modelado del rendimiento: de momento no existen enfoques para modelado del rendimiento de microservicios

- Adoptar modelos de rendimiento para casos de uso como planeamiento de capacidad, confiabilidad y resiliencia
- Buscar abstracciones de modelado apropiadas
- Extracción automática de modelos de rendimiento
- Aprender del comportamiento de la infraestructura e integrarlo en los modelos de rendimiento

#### 2.5.4. Herramientas para modelaje de rendimiento de software

En [19] se llevó a cabo un extensa encuesta sobre el estado del arte de herramientas de modelado de rendimiento de software. Como resultado, se lograron identificar varias herramientas para construir modelos de rendimiento pero, a pesar de esto, muchas de ellas fueron consideradas como no-maduras de acuerdo a los criterios de selección aplicados.

Primero, la herramienta debería de demostrar que se puede aplicar para el modelado de arquitecturas de software, este criterio excluyó a muchas herramientas de bajo nivel. Segundo, debe de estar liberada y disponible y esto excluyó a muchas herramientas descritas en artículos de investigación. Por último, debe ser madura y medianamente estable y este fue requerimiento que no cumplieron mucha de las herramientas publicadas en artículos de investigación.

Al final, se seleccionaron seis herramientas que se consideraron suficientemente maduras y promisorias las cuales se podrían ajustar para abordar problemas de modelado de arquitecturas.

**1. Java Modeling Tools (JMT):** es un conjunto de programas en Java que apoyan el modelado y análisis de modelos de redes de colas. Dentro de las ventajas que tiene

es que puede obtener soluciones computacionalmente baratas de modelos de redes de colas. También cuenta con una extensa interfaz gráfica de usuario que ayuda a los usuarios a través del proceso de modelado y análisis, esto baja su curva de aprendizaje. Una de sus limitaciones es que se ha señalado que es desafiante modelar sistemas de software industrial usando un formalismos simples de modelos de redes de colas. La herramienta se encuentra disponible en <http://jmt.sourceforge.net>.

**2. Layered Queueing Network Solver (LQNS):** es una herramienta de redes de colas (LQN por sus siglas en inglés) en capas y tiene el conjunto de características más completo de varias herramientas de LQN en capas comunes. Las LQNs ofrecen una forma fácil de expresar el uso de los recursos y encontrar cuellos de botella en servidores. Los algoritmos de análisis para LQN pueden escalar a sistemas grandes y puede ser usados para modelar grandes cantidades de solicitudes concurrentes. Dentro de sus limitaciones se menciona el hecho que LQN son menos adecuadas para modelar restricciones de tamaño de memoria que los modelos de redes de colas. De igual forma, en su manual de usuario se listan una serie de defectos conocidos y pocos detalles sobre estos son proporcionados. La herramienta se encuentra en <http://www.sce.carleton.ca/rads/lqns/>.

**3. Palladio-Bench:** implementa el *Palladio Component Model*. El software se encuentra disponible como un *plug-in* de Eclipse IDE y puede ser obtenido en el sitio Web <http://www.palladio-simulator.com/>. Esta herramienta puede ahorrar tiempo cuando se modelan componentes que son reutilizados en otros modelos. Es posible especificar las demandas de recursos dependientes de la entrada, probabilidades condicionales y contadores de iteraciones. Por ejemplo, el tiempo de ejecución de un componente puede hacerse dependiente del tamaño del archivo que va a procesar. Las limitaciones que se han reportado de esta herramienta incluyen poco soporte para concurrencia y del modelado del ambiente de ejecución.

**4. Möbius:** es una herramienta que implementa múltiples lenguajes de modelado y inclusive permite combinarlos para explotar su expresividad o para combinar múltiples aspectos de un sistema dentro de un modelo. Dentro de los lenguajes de modelaje ofrecidos, el SAN, *Buckets and Balls* y PEPA puede ser utilizados para modelado de rendimiento. Varios de los formalismos soportados por Möbius se basan en cadenas de Markov y por lo tanto sufren del problema de explosión espacial del estado. Puede haber una reducción en el número de estados de uno a dos órdenes de magnitud utilizando técnicas de agrupamiento, pero lo que esto significa en la práctica aún no está claro. La página Web de la herramienta es <https://www.mobius.illinois.edu/>

**5. SPE-ED:** implementa el enfoque SPE documentado por Smith y Williams [26]. Es una de las herramientas más maduras que se adecua a la predicción temprana del rendimiento. Utiliza modelos simples y que son fáciles de construir y resolver. SPE-ED solamente soporta redes de tipo  $M/M/n$ , lo que significa que los tiempos de llegada y servicio solamente pueden ser modelar teniendo distribuciones exponenciales. También se menciona que SPE-ED reporta solamente tiempos medios, lo cual es una limitación importante para alguien que quiera basar sus decisiones en valores extremos. SPE-ED se puede encontrar en <http://www.spe-ed.com>.

**6. QPME:** De acuerdo con el sitio Web *Petri Nets World*<sup>2</sup> QPME es una de las pocas herramientas de redes de Petri que sportan análisis de rendimiento avanzado y es la única herramienta actual que soporta redes de colas de Petri jerárquicas. El formalismo de modelado de redes de colas de Petri ha demostrado ser muy adecuado para representar sistemas de comercio electrónico distribuidos pero, por la falta de buenas herramientas es que el desarrollo de QPME fue motivado. QPME se ocupa de reducir el problema de explosión del espacio de estado. Se señala que con QPME no se puede modelar restricciones de memoria y que de momento no hay técnicas de análisis disponible, solamente de simulación. Otra limitación es que solamente se puede modelar llamadas sincrónicas y esto no lo hace adecuado para modelar sistemas de paso de mensajes. QPME se puede encontrar en <https://se.informatik.uni-wuerzburg.de/tools/qpme/>.

---

<sup>2</sup><http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/quick.html>

## Capítulo 3

# *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 distribuidas 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 dirección tecnológica. No es raro encontrar una gran cantidad de tecnologías y plataformas dentro de una sola 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 [9]. 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 transmitidos 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 de aplicación (API, por sus siglas en inglés) se utiliza para crear un mensaje, cargar los datos, asignar información de enrutamiento 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 dirige a un destino (por ejemplo una cola o un tópico) no a una aplicación específica. Cualquier aplicación que suscriba 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 asincrónica 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 asincrónicos 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 contruir un mensaje, luego pasarlos al MOM para su entrega a uno o varios recipientes (Figura 3.1). 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.

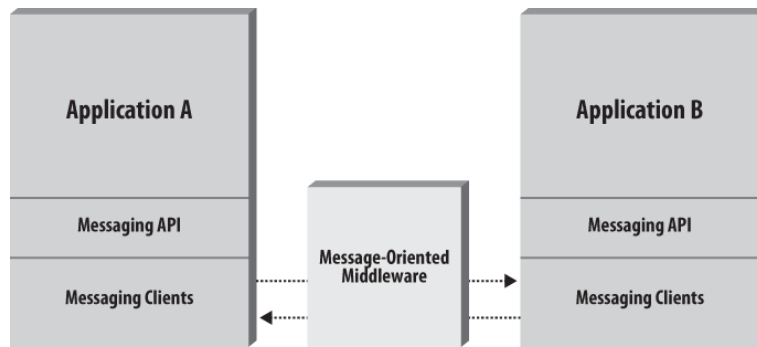


Figura 3.1: *Middleware* Orientado a Mensajes. Tomado de [9]

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

### 3.1. Modelos de mensajería

#### 3.1.1. 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* conoce cómo el mensaje va a ser utilizado y quién lo va a recibir.

#### 3.1.2. *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 recibidos 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 suscriptores hay y lo qué van a hacer estos con el mensaje.

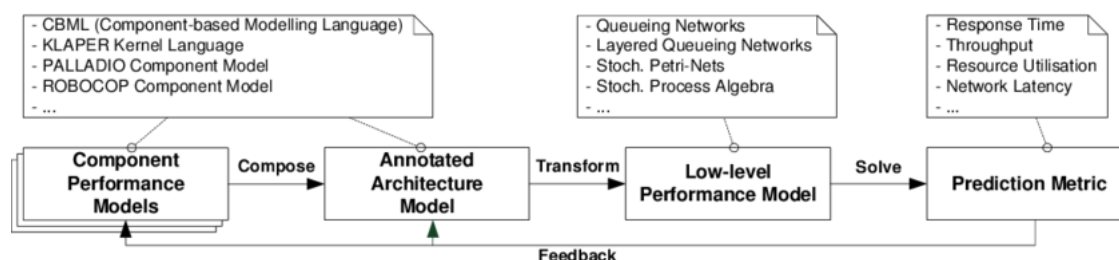


Figura 3.2: Predicción de rendimiento dirigida por modelos. Tomado de [13]

### 3.2. Enfoques de ingeniería de rendimiento para *middleware* orientado a mensajes

En [13] se extiende el *Palladio Component Model* con *performance completions* para *middleware* orientado a mensajes. *Performance completions* [27] 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 [5]. 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<sup>1</sup>. Las mediciones proporcionan los datos necesarios para calcular los valores de entrada del modelo de red de cola. El cálculo refleja el comportamiento de la aplicación en particular. 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 autorregresión (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 estímulos 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.

<sup>1</sup> Java Enterprise Edition

En [8] se construyen modelos de rendimiento para *middleware* que implementa el estándar *Java Messaging Service* (JMS). 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 esencial 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 desarrollado en [3] 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.



# Bibliografía

- [1] Ralf H. Reussner, Steffen Becker, Jens Happe, Robert Heinrich, Anne Kozirolek, Heiko Kozirolek, Max Kramer, and Klaus Krogmann. *Modeling and Simulating Software Architectures: The Palladio Approach*. The MIT Press. 2016.
- [2] Bruce Snyder, Dejan Bosnanac, Rob Davies. *ActiveMQ in Action*. Manning, 2011.
- [3] 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>
- [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] 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](http://ezproxy.itcr.ac.cr:2075/10.1007/11424529_1)
- [6] Robert Heinrich, André van Hoorn, Holger Knoche, Fei Li, Lucy Ellen Lwakatare, Claus Pahl, Stefan Schulte, and Johannes Wettinger. *Performance engineering for microservices: Research challenges and directions*. In Companion Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering, 2017, pages 223-226. DOI: <https://doi.org/10.1145/3053600.3053653>
- [7] Andreas Brunnert, André van Hoorn, Felix Willnecker, Alexandru Danciu, Wilhelm Hasselbring, Christoph Heger, Nikolas Roman Herbst, Pooyan Jamshidi, Reiner Jung, Jóakim von Kistowski, Anne Kozirolek, Johannes Kroß, Simon Spinner, Christian Vögele, Jürgen Walter, and Alexander Wert. *Performance-oriented devops: A research agenda*. Technical Report SPEC-RG-2015-01, SPEC Research Group - DevOps Performance Working Group, Standard Performance Evaluation Corporation (SPEC), 2015. <http://arxiv.org/abs/1508.04752>

- [8] Tomáš Martinec, Lukáš 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>
- [9] Mark Richards, Richard Monson-Haefel, David Chappell. *Java Message Service*. O'Reilly Media. Segunda Edición. 2009.
- [10] Nikolaus Huber, Steffen Becker, Christoph Rathfelder, Jochen Schweflinghaus, and Ralf H. Reussner. 2010. *Performance modeling in industry: a case study on storage virtualization*. In Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2 (ICSE '10), Vol. 2. ACM, New York, NY, USA, 1-10. DOI: <http://ezproxy.itcr.ac.cr:2075/10.1145/1810295.1810297>
- [11] Stephen Becker, Heiko Koziolk and Ralf Reussner. *The Palladio component model for model-driven prediction*. Journal of Systems and Software 82:3-22. Elsevier Science Inc. 2009. DOI: <http://dx.doi.org/10.1016/j.jss.2008.03.066>
- [12] 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>
- [13] 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>
- [14] Heiko Koziolk. 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>
- [15] Thijmen de Gooijer, Anton Jansen, Heiko Koziolk, and Anne Koziolk. 2012. *An industrial case study of performance and cost design space exploration*. In Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering (ICPE '12). ACM, New York, NY, USA, 205-216. DOI: <https://ezproxy.itcr.ac.cr:2878/10.1145/2188286.2188319>
- [16] Christoph Rathfelder, Steffen Becker, Klaus Krogmann and Ralf Reussner. *Workload-aware System Monitoring Using Performance Predictions Applied to a Large-scale E-Mail System*. 2012 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture, Helsinki, 2012, pp. 31-40. DOI: [10.1109/WICSA-ECSA.2012.11](https://doi.org/10.1109/WICSA-ECSA.2012.11).

- [17] Misha Strittmatter and Amine Kechaou. *The media store 3 case study system*. Technical Report 2016,1, Faculty of Informatics, Karlsruhe Institute of Technology, 2016. URL: <http://digbib.ubka.uni-karlsruhe.de/volltexte/documents/3792054>
- [18] 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
- [19] Thijmen de Gooijer. *Performance Modeling of ASP.NET Web Service Applications: an Industrial Case Study*. Master's thesis, Malardalen University, Vasteras, Sweden, 2011.
- [20] Steffen Becker. *Palladio-Bench Screenshots*. <http://www.palladio-simulator.com/tools/screenshots/> Accesado el 20 de Marzo del 2018.
- [21] 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.
- [22] Simonetta Balsamo, Antinisa DiMarco, Paola Inverardi, and Marta Simeoni. *Model-based performance prediction in software development: A survey*. IEEE Trans. Softw. Eng., 30(5):295–310, May 2004.
- [23] Anne Koziolk, Heiko Koziolk, and Ralf Reussner. *PerOpteryx: automated application of tactics in multi-objective software architecture optimization*. In Proceedings of the joint ACM SIGSOFT conference – QoSA and ACM SIGSOFT symposium – ISARCS on Quality of software architectures – QoSA and architecting critical systems – ISARCS (QoSA-ISARCS '11). ACM, New York, NY, USA, 33-42. 2011. DOI=<http://ezproxy.itcr.ac.cr:2075/10.1145/2000259.2000267>
- [24] Samuel Kounev and Fabian Brosig and Nikolaus Huber. *The Descartes Modeling Language*. Technical Report. Department of Computer Science, University of Wuerzburg, 2014.
- [25] Gunnar Brataas, Erlend Stav, Sebastian Lebrig, Steffen Becker, Goran Kopčak, and Darko Huljenic. 2013. *CloudScale: scalability management for cloud systems*. In Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering (ICPE '13), Seetharami Seelam (Ed.). ACM, New York, NY, USA, 335-338. 2013. DOI: <https://ezproxy.itcr.ac.cr:2878/10.1145/2479871.2479920>
- [26] Connie U Smith and Lloyd G Williams. *Performance solutions: a practical guide to creating responsive, scalable software*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 2002.

- [27] 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.