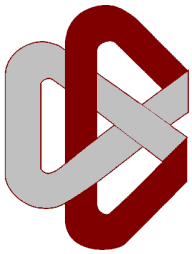

CIMAT

Centro de Investigación en Matemáticas, A.C.

Especificación no funcional de ensambles de componentes

REPORTE TÉCNICO

para obtener el grado de

Maestro en Ingeniería de Software

P r e s e n t a

Daniel Alejandro López Hernández

Director de Tesis:

Dra. Perla Inés Velasco Elizondo

Zacatecas, Zac.. Agosto de 2010

Índice

Figuras

1 Introducción

En este capítulo discutiremos el contexto, la motivación y los objetivos que relacionados a esta investigación. Adicionalmente comentaremos sobre la organización de este reporte.

1.1 Contexto y motivación

El desarrollo de software basado en componentes se ha anunciado como un paradigma que permite reducir el tiempo de desarrollo y garantizar cierta calidad de los sistemas que se construyen. Se reduce el tiempo ya que no es necesario desarrollar el componente desde el inicio, los componentes son asumidos como elementos pre-existentes de software. Se puede garantizar cierto grado de calidad en el sistema ya que los componentes que están siendo reutilizados ya fueron probados con anterioridad.

Si bien es cierto que muchos modelos y tecnologías de componentes están siendo utilizadas actualmente, por ejemplo EJB[4], CORBA[8], etc., la falta de un enfoque sistemático para la reutilización y composición de componentes en muchos de estos modelos y tecnologías ha impedido maximizar los beneficios del desarrollo basado en componentes en la práctica.

Existe un modelo de componentes descrito en [10], el cual propone una solución a estos problemas. El modelo propone un enfoque sistemático para crear ensambles de componentes y reutilizar dichos ensambles en la construcción de diferentes sistemas.

En su versión actual, el modelo de componentes considera principalmente propiedades funcionales durante la composición y especificación de componentes [10]. Sin embargo propiedades relacionadas con la funcionalidad no son los únicos elementos de información que deben considerarse al especificar componentes y los ensambles generados a partir de éstos. También es necesario considerar propiedades no funcionales que describan ciertas cualidades de los componentes de manera que podamos realizar predicciones sobre el dichas cualidades en los ensambles que realicemos con esos componentes. En un escenario ideal, por ejemplo, si conocemos una propiedad no funcional como el peor tiempo de ejecución de los elementos que constituyen un ensamble de componentes podríamos a partir esta información estimar el peor tiempo de ejecución de dicho ensamble.

De esta forma, la motivación de este trabajo es explorar la viabilidad de generar la especificación no-funcional de ensambles de componentes a partir de las propiedades no-funcionales de sus elementos constituyentes. Todo ello, dentro del contexto del modelo de componentes anteriormente mencionado y una implementación particular de este modelo en el lenguaje Java [7].

1.2 Objetivos

Al final de este trabajo se deben lograr los siguientes objetivos:

- Analizar un conjunto de propiedades no funcionales y determinar cuáles son candidatas a ser especificadas.
- Del conjunto seleccionado seleccionar una propiedad y especificar dicha propiedad para los diversos elementos arquitecturales en el contexto del modelo de componentes.
- Analizar los factores que determinan la factibilidad de generar la especificación no-funcional de ensambles de componentes a partir de las especificaciones de sus elementos constituyentes para una implementación en Java del modelo de componentes.

1.3 Organización del documento

Para el desarrollo de este trabajo primero, en el capítulo 2, se introducen algunos conceptos base que ayudarán a entender el contexto sobre el cual vamos a trabajar: software basado en componentes, propiedades no funcionales y el modelo de componentes propuesto por [10].

En el capítulo 3 se discuten distintas propiedades no funcionales con el propósito de determinar cuáles de ellas se podrían especificar en los elementos arquitecturales de un ensamble.

En el siguiente capítulo presentaremos los métodos utilizados para especificar estas propiedades en otros modelos de componentes y discutiremos que características de estos métodos.

El capítulo 5 describe un primer ejercicio, así como los resultados obtenidos, de especificación de una propiedad no funcional para un ensamble de componentes en el contexto del el modelo de componentes seleccionado.

Finalmente, en el capítulo 6 revisaremos las conclusiones de esta investigación y algunas actividades que deberán realizarse en un trabajo futuro.

2 Antecedentes

Como se mencionó en el capítulo anterior, comenzaremos con algunos conceptos base para establecer el contexto sobre el cual se desarrolla esta investigación. Este capítulo hablaremos del desarrollo basado en componentes, propiedades no funcionales y del modelo de componentes utilizado en este trabajo.

2.1 Desarrollo basado en componentes

El desarrollo basado en componentes es una rama de la ingeniería de software que ofrece como ventaja la posibilidad de crear sistemas a partir de componentes reutilizables. Desde inicios del desarrollo basado en componentes se ha creado un paradigma de repositorios de componentes *"plug & play"*. Se asume que los componentes dentro del repositorio cumplen con requerimientos de calidad, abstracción, encapsulación, bajo acoplamiento y alta cohesión.

Existen varios modelos de componentes que buscan una manera sistemática de imponer restricciones de diseño a los componentes de software. Por ejemplo CORBA[8], EJB[4], Koala[12], ProCom[19], etc.

En términos generales, un componente se define como una unidad independiente de software que implementa una abstracción lógica y proveé servicios a través de una interface bien definida. Los componentes cuentan con ciertas propiedades, funcionales y no funcionales, que deben estar documentadas para conocer el uso del componente. Cuando hablamos de propiedades funcionales en un componente nos referimos a los cálculos, manipulación de datos, procesamiento o cualquier otra funcion implementada por el componente. Cuando hablamos de propiedades funcionales, nos referimos a las características de calidad asociadas al componente, como la confiabilidad, seguridad, el performance, etc. En la siguiente sección abundaremos sobre el tema de propiedades no funcionales.

2.2 Propiedades No Funcionales

El término propiedad no funcional está asociado a muchos atributos de calidad de un sistema de software como por ejemplo su performance, portabilidad, seguridad, confiabilidad, etc. Existe en la IEEE el estándar ISO9126 [6] que da una descripción detallada de un modelo de calidad del software. Dentro de este estándar se identifican las atributos de calidad ilustrados en la Figura 2.1:

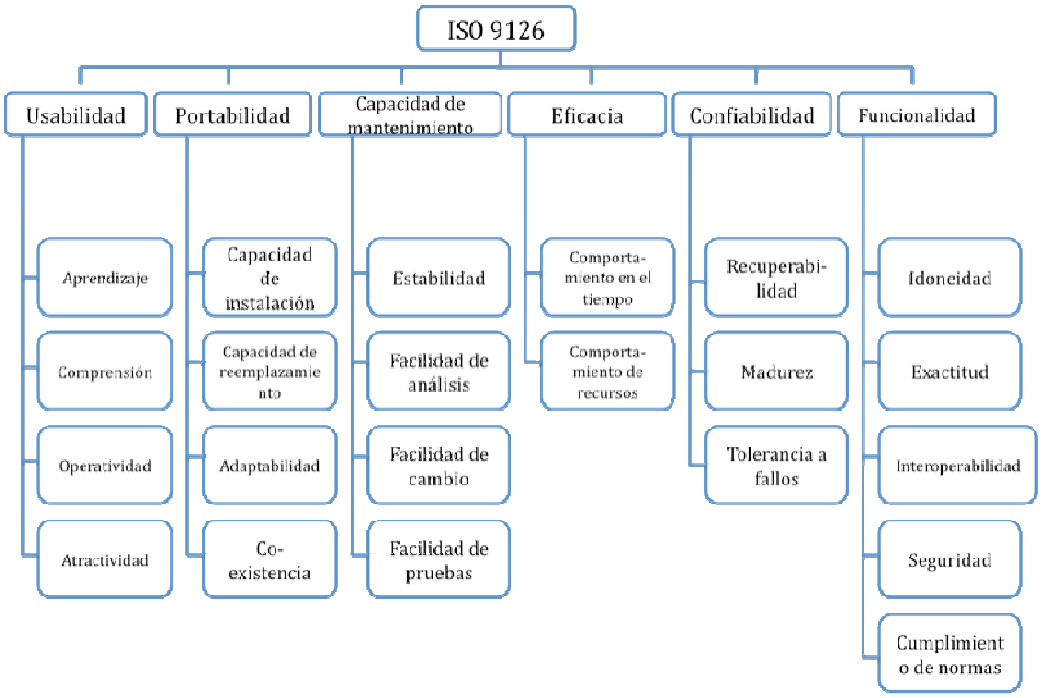

Fig. 2.1 – Atributos de calidad del estándar ISO9126

Dependiendo del dominio de aplicación de software hay propiedades no funcionales que son más importantes que otras. En los sistemas de control, las tareas tienen un riguroso tiempo límite en su ejecución. Para hacer un análisis que determine la calidad de los componentes debemos verificar que el sistema se ejecuta dentro de este tiempo límite. Para esto necesitamos conocer la especificación correspondiente con respecto a tiempo y consumo de recursos. Necesitamos conocer la eficiencia de cada componente que conforma el ensamble de manera que podamos predecir si la ejecución se hará dentro de ese tiempo límite.

En un sistema distribuido nos interesa tener propiedades como la confiabilidad, debe tener tolerancia a fallos y para esto debe conocer el tiempo que tarda el sistema en recuperarse de un fallo o la probabilidad de que ocurra un fallo.

El determinar las propiedades no funcionales de un sistema no es una tarea fácil. Sin embargo, actualmente técnicas como la prueba (testing) pueden ser utilizadas para establecer los valores de algunas propiedades funcionales. Para reducir ambigüedad, las propiedades no funcionales deben ser especificadas de forma cuantitativa. Sin embargo, en ciertas propiedades una evaluación cuantitativa es difícil y se opta por evaluaciones cualitativas.

2.3 Un modelo de componentes y su especificación de propiedades funcionales

En la motivación de este documento se mencionó el modelo de componentes que propone [10]. Una de las principales características de este modelo es que separa la computación de la composición para maximizar la reutilización. El objetivo de esta separación es tener componentes que sólo se preocupen por la computación para hacerlos más reutilizables. Así, el componente no tiene dentro de su lógica código referente a la composición del sistema, por lo que se tiene menos acoplamiento con los componentes con los que debe comunicarse.

Existen dos elementos arquitectónicos en el modelo: componentes y conectores. Como indicamos antes, en capsulan computación. Los componentes pueden ser atómicos y ensambles. Los componentes atómicos son la forma básica de un componente, mientras que los ensambles se forman mediante la composición de uno o más componentes via conectores.

Los conectores encapsulan esquemas de comunicación y coordinación útiles para crear ensambles componentes. Existen varios tipos de conectores. Todos ellos se encuentran calsificados en 2 grandes grupos: conectores de adaptación y conectores de composición.

A continuación se muestra el catálogo de conectores que se encuentran definidos hasta el momento en el modelo:

Categoría	Tipo
Conectores de adaptación	Guardia
	Ciclo controlado por condición
	Ciclo controlado por un contador
	Bloqueo
Conectores de composición	Secuenciador
	Pipe
	Selector
	Observador
	Cadena de responsabilidad
	Secuenciador con selección exclusiva
	Pipe con selección exclusiva
	Unión simple con selección
	Unión simple con pipe

Fig. 2.2 – Catálogo de conectores del modelo de componentes

Los conectores de adaptación nos dan cierto control para alterar la funcionalidad de un componente, es decir, antes que de que se ejecute el componente podemos ejecutar un esquema de control.

Un ejemplo de estos conectores es el Ciclo controlado por condición. Utilizamos este conector para repetir la ejecución de un componente de acuerdo a la evaluación de una expresión booleana. Como se puede apreciar en la figura 2.3, los parámetros de entrada (input) del componente son evaluados en una expresión booleana (expr), mientras esta sea verdadera se volverá a ejecutar

cierta computación el componente. Si tuvieramos un componente que obtiene un elemento de una lista podemos usar este conector para seguir obteniendo elementos mientras la lista no se encuentre vacía.

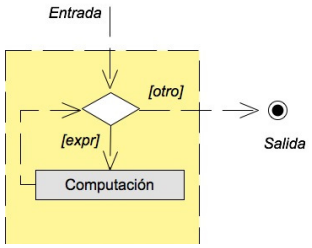

Fig. 2.3 – Conectores de adaptación: Ciclo controlado por condición

Por otra parte los conectores de composición nos permiten hacer ensambles de *n* componentes como su nombre lo sugiere. Un ejemplo de estos conectores es el Secuenciador. Este conector nos permite ejecutar un grupo de componentes de manera secuencial. Como se muestra en la figura 2.4, se reciben los parámetros de entrada y se realiza la ejecución de la computación en *n* componentes hasta producir la salida.

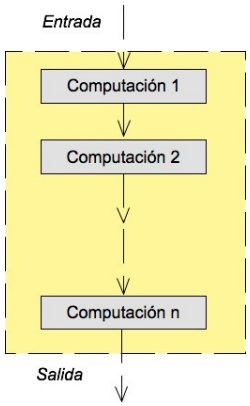

Fig. 2.4 – Conectores de composición: Secuenciador.

En este modelo de componentes los ensambles de componentes se construyen jerárquica e incrementalmente de abajo hacia arriba (bottom-up). Cada composición produce un nuevo componente que puede ser utilizado en otra composición. En la figura 2.5 se muestra un ejemplo de esto, mediante una de arquitectura de sistema. En la figura se puede apreciar cómo los conectores (K1 – K5) ensamblan a varios componentes (C1 – C5) y varios de estos han sido sujetos de posterior composición.

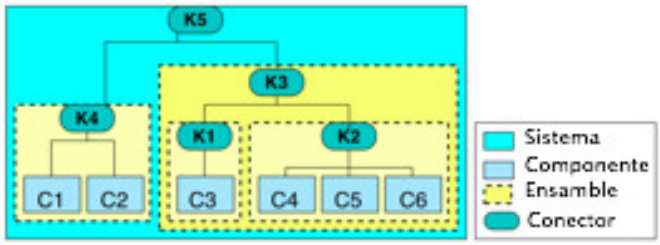

Fig.2.5 – Arquitectura de un sistema en el modelo de componentes.

En [10] se encuentra la descripción detallada de cada uno de estos conectores y su implementación.

2.3.1 Especificación funcional en el modelo de componentes

En [9] se nos presenta una propuesta para generar la especificación de propiedades funcionales en ensambles de componentes en el modelo de componentes de nuestro interés.

La especificación funcional de los componentes y ensambles del modelo debe informar acerca de los métodos que provee. En la propuesta descrita en [9], los métodos (computación) ofrecidos por un componte son especificados con base en la información de la firmas (signatures) que los implementan. Esto es, mediante un nombre, los parámetros de entrada y los parámetros de salida. En los componentes atómicos esta especificación se realiza mediante anotaciones generadas de forma manual por el desarrollador de componentes.

Cuando realizamos un ensamble de componentes los métodos ofrecidos por dicho ensamble son abstracciones que denotan secuencias de operaciones en términos de las operaciones ofrecidas por los componentes (atómicos y ensambles). Estas secuencias se obtienen a partir de la especificación funcional de los componentes y la semántica del conector que está siendo usado en la composición.

En [9] se presenta un conjunto de funciones que describen la especificación funcional de los ensambles realizados con cada conector. En la figura 2.6 se encuentra la función del conector Secuenciador. En la función Sig representa una firma (signature), FSpec es la especificación funcional formada por un conjunto de firmas y sig_concat es una función que obtiene los elementos comunes entre los parámetros de salida de un método y los parámetros de entrada de otro método.

La función nos dice que obtenemos una serie de firmas que resultan de concatenar, mediante una operación de producto cartesiano, la especificación funcional (FSpec) de los componentes del ensamble.

$$seq\_composite\_fspec : FSpec \times FSpec \times \ldots \times FSpec \rightarrow FSpec$$

$$\begin{aligned} seq\_composite\_fspec = \\ & s_1, s_2, \ldots, s_n : Sig; \\ & f_1, f_2, \ldots, f_n : FSpec; \\ & (s_1, s_2, \ldots, s_n) \in f_1 \times f_2 \times \ldots \times f_n \bullet \\ & \bigcup_{(s_1,\ldots,s_n)\in\prod_{i=1}^{n} f_i} sig\_concat(s_1, s_2, \ldots, s_n) \end{aligned}$$

Fig. 2.6 – Función de ensambles con Secuenciador

Cada vez que realizamos la composición de un ensamble obtenemos su especificación funcional, esta composición se realiza de manera iterativa en una jerarquía de abajo hacia arriba (bottom-up). Se inicia con la composición de componentes atómicos y se continua con la composición de ensambles de componentes en los siguientes niveles. Este enfoque de especificación funcional es sistemático y consistente, por lo tanto es un proceso que puede automatizarse.

Hasta ahora hemos visto cómo son especificadas las propiedades funcionales de un ensamble de componentes en el modelo de componentes de [10], en las siguientes secciones comenzaremos a explorar la posibilidad de especificar una propiedad no funcional.

2.3.2 Especificación no funcional en el modelo de componentes

Actualmente no existe un método que soporte la especificación de propiedades no funcionales en los componentes de este modelo. Como se hizo para las propiedades funcionales, se desea generar este tipo de especificaciones para los ensambles de componentes de una manera sistemática.

Se deberá incluir en la especificación de los componentes atómicos una o varias propiedades no funcionales y al momento de hacer un ensamble de componentes hacer una predicción de las propiedades no funcionales con base en dicha información.

El análisis sistemático para realizar la predicción de las propiedades no funcionales deberá ser bottom-up. Inicialmente debemos obtener la propiedad no funcional para los componentes atómicos y los conectores. Después hacer una predicción de un ensamble y medir la propiedad no funcional para todo el ensamble para comprobar la precisión de nuestra predicción.

3 Medición de Propiedades no-funcionales

En este capítulo revisaremos las distintas propiedades no funcionales para determinar cuales de ellas se podrían especificar en los elementos que conforman un ensamble.

3.1 Propiedades no funcionales y sus métricas

Como se mencionó anteriormente para poder interpretar un propiedad funcional, es necesario proveer una especificación de la misma de forma cualitativa o cuantitativa. Algunas propiedades no funcionales como usabilidad y la portabilidad son subjetivas y son descritas de una manera cualitativa.

Dependiendo del dominio de aplicación de los componentes algunas propiedades no funcionales se vuelven más importantes que otras. El modelo de componentes utilizado en este trabajo utiliza componentes muy simples que en su mayoría son sensores. Mencionamos en la sección 2.3.1 que la especificación funcional de los ensambles varía dependiendo del conector con el que hacemos la composición. Como primer ejercicio nos interesa especificar una propiedad que se pueda medir cuantitativamente y que podamos predecir cuando se realice la composición de un ensamble. Se optó por especificar el tiempo de ejecución y a continuación se discute un poco sobre las métricas de otras propiedades no funcionales que fueron consideradas.

Propiedades no funcionales como la capacidad de mantenimiento y adaptabilidad se encuentran presentes en las características del modelo de componentes. Esto debido a que nos permite aislar la computación en los componentes atómicos y que nos permite hacer ensambles de esos componentes atómicos utilizando los conectores exógenos. Nuestro análisis no se enfocará a estas dos propiedades no funcionales ya que son medidas de manera cualitativa.

La disponibilidad es una propiedad no funcional que si impacta en la ejecución del sistema. Esta propiedad no funcional nos indica la probabilidad de que un módulo esté disponible cuando sea requerido. Se puede medir con métricas como el tiempo promedio para que ocurra un fallo (Mean Time to Failure), tiempo promedio para reparar el fallo (Mean Time to Repair) y el tiempo promedio entre cada fallo (Mean Time Between Failures).

Por otra parte el performance utiliza métricas que describen el comportamiento del componente con respecto al tiempo. Algunas de ellas son: tiempo de respuesta, latencia, peor tiempo de ejecución y tiempo de procesamiento. Algunos autores hacen uso indistinto de estos términos pero existen diferencias entre ellos.

Latencia es un retraso que sufren los mensajes que viajan por la red. La latencia puede cambiar según el uso del hardware y no puede ser corregida mediante el código.

El tiempo de respuesta es el tiempo total que toma desde que el usuario hace una petición hasta que recibe respuesta del sistema. El tiempo de respuesta se ve afectado por cambios en el tiempo del procesador y por la latencia.

El tiempo de procesamiento es el tiempo que le toma al sistema procesar la petición sin tomar en cuenta el tiempo que tarda en llegar el mensaje del usuario al sistema. Es decir:

Latencia + Tiempo de procesamiento = Tiempo de respuesta [13]

Dentro de los sistemas de tiempo real el tiempo de respuesta es expresado en términos de peor caso de tiempo de ejecución (Worst Case Execution Time) que es importante conocer para garantizar que la ejecución se realizará antes de que se cumpla un deadline.

Hasta ahora hemos visto que existen propiedades no funcionales que pueden obtenerse de los componentes y otras de toda la arquitectura del sistema. También hemos visto que algunas propiedades no funcionales sólo las podemos medir mediante la ejecución del componente. Ahora veremos una clasificación de propiedades no funcionales que propone [2] y que nos dará un panorama general de que propiedades podemos medir.

3.2 Clasificación de propiedades no funcionales

Uno de los principales problemas con las propiedades no funcionales dentro del desarrollo de software es que no hay mucha precisión en su definición y son muy difíciles de medir. Como consecuencia es mucho más difícil asociar estas propiedades del sistema a un componente.

Se propone una clasificación de propiedades en **¡Error! No se encuentra el origen de la referencia.**, basándose en los principios de composición de los componentes. Se proponen los siguientes tipos de propiedades:

- Propiedades de composición directas (Directly composable properties). Estas propiedades son aquellas que pueden ser obtenidas o calculadas directamente del componente, por ejemplo la memoria que ocupa el componente cuando es ejecutado.

- Propiedades asociadas a la arquitectura (Architecture-related properties). Se refiere a aquellas propiedades que no solo dependen de las propiedades del componente, sino que dependen además de la arquitectura. Por ejemplo la confiabilidad (reliability) que depende de mecanismos de tolerancia a fallos.

- Propiedades derivadas (Derived properties) .- Una propiedad derivada es aquella que depende de varias propiedades en los componentes para ser calculada. Por ejemplo para obtener el deadline de la ejecución de un componente necesitamos conocer su peor caso de ejecución y el periodo de ejecución.

- Propiedades dependientes de uso (Usage-dependant properties) .- Estas propiedades están determinadas por el perfil de uso del sistema, su valor depende del uso que se le vaya a dar al sistema, si este uso llega a cambiar las propiedades cambiarán también. Un ejemplo de esta propiedad puede ser la confiabilidad (reliability).

- Propiedades del contexto de ambiente del sistema (System Enviroment Context Properties) .- Estas propiedades dependen no sólo del perfil de uso del sistema, también dependen del ambiente sobre el que se utiliza el sistema. Un ejemplo de esta propiedad es la seguridad, esta propiedad se refiere a una catástrofe potencial que bajo ciertas circunstancias la misma propiedad puede tener varios grados de seguridad aunque trabaje sobre el mismo perfil de uso.

En [2] se nos presenta la siguiente tabla en donde sugiere los tipos de propiedades que podemos tener para cada propiedad no-funcional.

		Clasificación				
Atributo de mas alto nivel	Atributo de Calidad	Propiedades de composición directas	Propiedades asociadas a la arquitectura	Propiedades derivadas	Propiedades dependientes del uso	Propiedades del contexto de ambiente del
Usabilidad	Accesibilidad		XX			
	Facilidad de administráción				XX	
	Facilidad de entendimiento			X		
	Generalidad			X		
	Facilidad de operación				XX	
	Simplicidad				XX	
Portabilidad	Mobilidad			XX		
	Normacidad			XX		
	Independencia de Hardware		XX			
	Independencia del Softwre		XX			
Desempeño	Presición		XX			
	Consumo de memoria		XX			
	Tiempo de respuesta				XX	
	Escalabilidad	X	XX			
	Facilidad de planeación de tareas		X	X		
	Puntualidad		X	X		
	Utilizacion del CPU			X	X	
	Latencia		X	X		
	Tasa de procesamiento de transacciones		XX		X	
	Concurrencia			XX		
	Eficiencia	XX	X			
Facilidad de mantenimiento	Flexibilidad	X	XX			
	Facilidad de evolución	X	XX			
	Facilidad de extensión	X	X			
	Facilidad de modificación	X	XX			
	Facilidad de actualización	X	X			
	Facilidad de expansión	X	X			
	Consistencia de datos	XX				
	Consistencia de versión	XX				
Facilidad de integración	Facilidad de adaptación	XX				

Fig. 3.1 – Clasificación de propiedades no funcionales

En la sección anterior se mencionó la necesidad de tener una propiedad no funcional que pudiera ser medida cuantitativamente y que se pueda predecir en los ensambles que se construyan. En la clasificación de propiedades mencionada en esta sección podemos decir que el tiempo de ejecución entra en las dos siguientes categorías: propiedad asociada a la arquitectura y propiedad de composición directa. Es decir, el tiempo de ejecución cambia dependiendo de la arquitectura y podemos medirlo directamente del componente. Estas dos categorías concuerdan con lo mencionado en la sección 2.3.2, donde se mencionó que primero se deben medir los componentes atómicos y conectores, después los ensambles de componentes.

4 Medición de propiedades no funcionales en otros modelos de componentes

Existen otros modelos de componentes en los que ya se ha definido una manera de generar la especificación de una propiedades no funcionales en ensambles de componentes. En este capítulo veremos algunos de esos modelos de componentes haciendo énfasis en las características del modelo, las propiedades no funcionales que especifican y cómo es derivada dicha especificación a nivel de ensamble.

4.1 Predicción de confiabilidad en arquitecturas basadas en componentes

En [11] se nos presenta un modelo de componentes que utiliza RADL (Rich Architecture Definition Language) para hacer la especificación de sus componentes y ensambles. Los ensambles son denominados kens y se comunican a través de gates que controlan las entradas y salidas de los kens. Los kens más básicos son componentes atómicos.

Como se mencionó anteriormente las conexiones entre kens se hace mediante gates. La conexión entre una gate provista y una gate requerida se le denomina binding, mientras que la conexión entre dos gates provistas o dos gates requeridas es denominada mapping.

Los componentes son utilizados mediante contratos. Los componentes que ofrecen un servicio tienen una pre y post condición, el componente garantiza que se cumplirá la post-condición siempre y cuando el componente cliente cumpla con la pre-condición.

La propiedad no funcional que predice este modelo de componentes es la confiabilidad (reliability). En la especificación RADL del componente se incluye la confiabilidad del componente. La confiabilidad de cada método en el componente es medida utilizando unas pruebas en las que existe una probabilidad de que el componente falle.

Para realizar la predicción se requiere también un perfil de uso. El perfil de uso comprende una serie de estadísticas que son modeladas utilizando cadenas de Markov y una máquina de estados finita que representan la probabilidad de que ocurra una transición de un estado a otro.

En este método podemos ver que es necesario hacer una especificación formal de la propiedad no funcional en los componentes y ensambles para poder realizar un análisis que nos permita hacer una predicción. Obtener la propiedad no funcional requiere de una serie de pruebas sobre los elementos que conforman el ensamble para después hacer una predicción en ensambles que utilicen esos elementos.

Nuestra especificación de tiempo de ejecución en el modelo de componentes de [10] deberá hacerse de manera formal después de realizar pruebas para obtener el tiempo de ejecución de los componentes atómicos y los conectores. La formalidad de la especificación deberá ser similar a la especificación de propiedades funcionales mencionada en la sección 2.3.1, con un enfoque de abajo hacia arriba (bottom-up).

4.2 Especificación formal y monitores de calidad de servicio

En [15] nos habla de un modelo de componentes cuyo dominio de aplicación son los sistemas de tiempo real. Este modelo, al igual que muchos otros, tiene como elementos de su arquitectura una serie de componentes que proveen servicios mediante puertos. Un puerto conecta a los componentes mediante interfaces provistas y requeridas. Existen dos clases de componentes: los componentes primitivos que son aquellos que contienen el código, y los componentes composite que representan un ensamble de componentes.

Dado que el dominio de aplicación son los sistemas de tiempo real la propiedad no funcional que se especifica es el tiempo de ejecución. A tiempo de diseño se pretende monitorear que la ejecución de las tareas se realice en determinado tiempo. Para modelar este comportamiento se define un autómata temporal, el cual describe la secuencia de mensajes que se intercambian entre componentes y las restricciones de tiempo de la ejecución de esta secuencia.

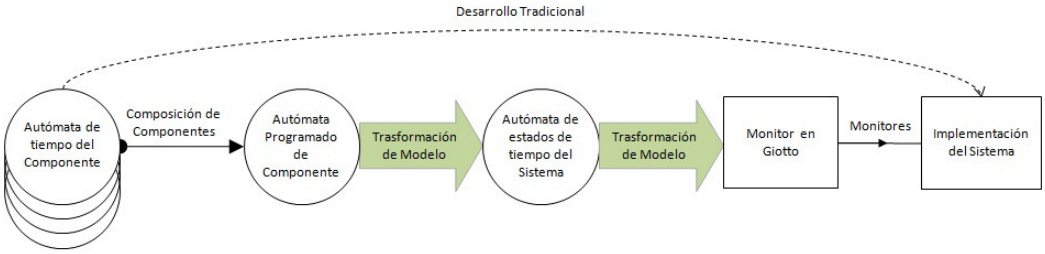

Fig. 4.1 – Descripción general del proceso que sigue este método.

El objetivo de este método es establecer en el diseño de la arquitectura las restricciones de tiempo de los componentes y después monitorear que estas restricciones se siguen cumpliendo al pasar del diseño a la implementación. Una vez que se tiene la arquitectura de componentes y sus restricciones de tiempo se realiza una transformación del modelo a una especificación en un Framework de sistemas de control embebidos (Giotto). Después se generará el código basándose en esta transformación del modelo. Un monitor de calidad de servicio debe asegurarse que al hacer la transformación del modelo se respeten las restricciones de tiempo.

Este método realiza la especificación de tiempo de ejecución a nivel de componente y después lo escala a una predicción del tiempo de ejecución de la

arquitectura del sistema, no hay un punto intermedio donde se analice sobre ensambles de componentes que después puedan ser integrados a una arquitectura. Además de que la predicción de la propiedad no funcional es realizada incluso antes de la implementación de los componentes, de hecho sirve de base para dicha implementación.

En el modelo de componentes de [10] no se considera hacer un análisis de las propiedades no funcionales antes de hacer la implementación de los componentes atómicos. En trabajo futuro esto podría ser retomado, si se establece la factibilidad de hacer una predicción del tiempo de ejecución a partir del diseño del componente ya no sería necesario medir el tiempo de ejecución del mismo.

4.3 Tecnología de componentes para predicción en ensambles

En [14] se describe el uso de una tecnología de componentes para predicción, la cual resulta de integrar una tecnología de componentes con una o más tecnologías de análisis. La tecnología análisis es la que nos permitirá hacer predicciones a nivel de ensamble de las propiedades del componente.

Actualmente existen muchas herramientas que ayudan a hacer análisis o a medir una propiedad no funcional en un sistema, pero ninguna de esas herramientas está integrada a un modelo de componentes. Por lo tanto se debe de adecuar tanto el modelo como la herramienta para especificar una propiedad no funcional específica, hacer un análisis y predicción muy precisa. La propuesta de [14] es tener además de un modelo de componentes un modelo de ensamble que cuente con un modelo de análisis para poder realizar la predicción a partir de los componentes del ensamble.

El modelo de componentes utilizado en un ejemplo de [14] es COMTEK, el cual fue desarrollado por el SEI para el departamento de calidad del agua de la Agencia de Protección Ambiental de E.U. Este modelo de componentes permite construir arquitecturas pipe and filter, realiza una planificación de tareas fija con round-robin y ejecuta el ensamble de manera secuencial. Los sistemas construidos en COMTEK corren en sistemas operativos de la familia Windows y son empaquetados en librerías Dynamic Link (DLL)

La propiedad no funcional que se especificó en el ejemplo de COMTEK fue la latencia. Para poder hacer esta predicción debemos saber cómo es la planificación de tareas del sistema operativo donde se ejecutará, si hay concurrencia en la ejecución o multi-threading. Todos estos factores ya se mencionó como son manejados en el modelo COMTEK.

Para especificar la latencia en cada componente se distinguen dos tipos de componentes. Los componentes theta son aquellos que sólo tienen dependencias internas, mientras que los componentes fi son aquellos que tienen dependencia con eventos periódicos externos, por lo tanto además de tener un tiempo de ejecución tienen especificado el periodo en el que el evento externo ocurrirá,

este evento externo puede ser la lectura de un sensor. La latencia de un componente se mide ejecutándolo varias veces y se toma el peor tiempo de ejecución.

Para hacer la predicción de la latencia en el ensamble se debe tener en cuenta el tipo de componentes que conforman el ensamble. Los ensambles de componentes theta sólo tienen un tiempo de ejecución por lo que la latencia es la suma de esos tiempos de ejecución. Un ensamble de dos componentes con tiempo de ejecución de 3 milisegundos cada uno, resulta en una latencia de 6 milisegundos.

Si el ensamble consta únicamente de componentes fi se suman los tiempos de ejecución y se compara con el periodo más grande de los componentes en el ensamble. El valor más alto de estos dos elementos será la latencia del ensamble. Por ejemplo un componente con tiempo de ejecución de 1 milisegundo y periodo de 4 milisegundos se ensambla con otro componente de 2 milisegundos de ejecución y periodo de 4 milisegundos, resulta en una latencia de 4milisegundos porque el periodo más alto es 4 milisegundos y a su vez es mayor que la suma de los tiempos de ejecución (3 milisegundos).

Para los ensambles que mezclan componentes de tipo theta con componentes fi se utiliza un razonamiento similar al de los ensambles de componentes fi, sólo tomando en cuenta que el periodo de los componentes theta siempre será cero.

Además de proponer una manera de especificar la propiedad no funcional se debe de validar que las fórmulas que describen la teoría son congruentes, es decir, tiene una validez teórica. Si las predicciones son efectivas debe haber una explicación de cómo la teoría fundamenta esa predicción.

Por otro lado se debe verificar que el modelo de análisis se integre correctamente con la tecnología de componentes. Esto se debe verificar de manera empírica utilizando análisis estadístico.

De igual manera en el modelo de componentes de [10] debe haber fórmulas que den validez a la teoría y se deben hacer mediciones de los componentes para hacer un análisis estadístico para comparar la predicción con la ejecución real.

4.4 Incluir propiedades no funcionales en arquitecturas de Software utilizando ADL

El método utilizado en [1] emplea una notación de ADL (Architecture Description Language) llamada WRIGHT. El propósito principal de un ADL es especificar la composición estructural de sistemas de Software en términos de sus componentes y conectores mediante un lenguaje de especificación formal.

WRIGHT ADL tiene 3 elementos arquitecturales: componentes, conectores y configuraciones. Un componente está formado por su computación y su interfase, esta interfase puede tener varios puertos. Cada puerto a su vez tiene

varios eventos, que son las acciones que puede realizar el componente. Los puertos son lo que une al componente con el resto del ambiente.

Por otra parte los conectores tienen dos partes: un rol y el pegamento. El rol define el comportamiento que deberá tener el componente al que se está conectando. El pegamento describe cómo los componentes del ensamble crean la composición.

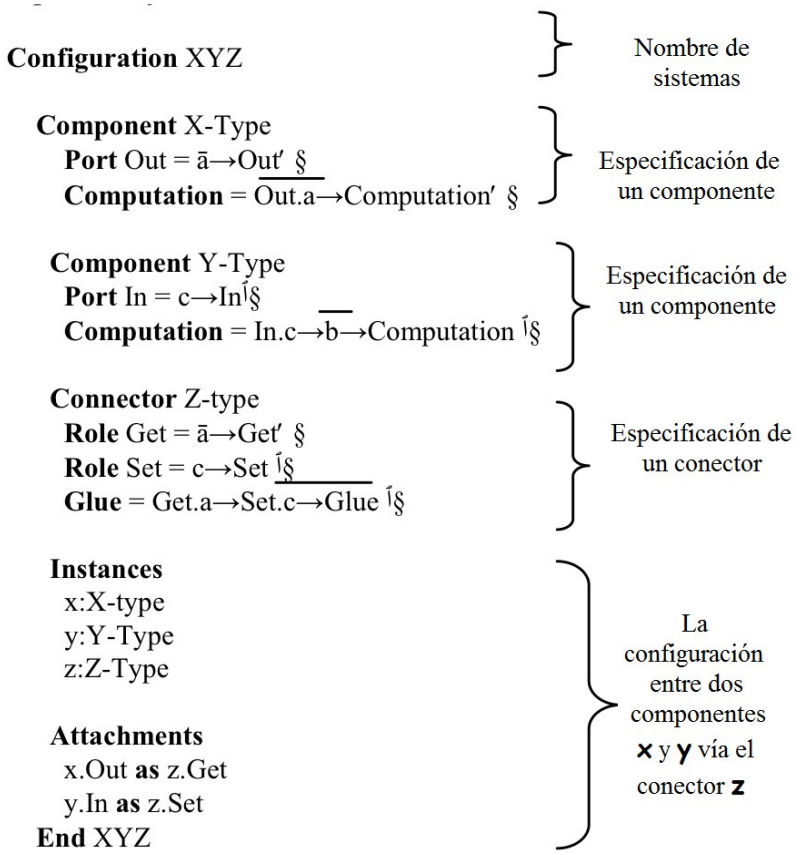

Fig. 4.2 – Ejemplo de notación WRIGHT

Se puede incluir cualquier propiedad no funcional en la especificación de WRIGHT que propone **¡Error! No se encuentra el origen de la referencia.**, sin embargo no existe formalidad en la manera que se especifica la propiedad no funcional. La especificación se realiza mediante una combinación de WRIGHT, notación CSP(Communicating Sequential Processes) e Inglés estructurado, de esta manera se vuelve flexible para permitir la especificación de cualquier propiedad no funcional.

Hay dos clases de propiedades no funcionales en la especificación: requeridas y aseguradas. Similar a las interfaces requeridas y provistas, en la especificación se indica que propiedad no funcional requiere el componente del componente con el que se conecta y que propiedades provee a otros componentes.

```
Configuration e-health
Component GP
    Port auth = authenticate?x  auth
    Port write = start  accept!x  write
    Computation auth.authenticate?x  write.start  computation
(1) Non-functional Security =
            (REQUIRES □{Remote_Procedure_Call.encryption},
            ENSURES □{auth.authentication})
```

Fig 4.3. – Ejemplo de especificación de propiedades no funcionales en WRIGHT

Este enfoque dejó de lado la formalidad para que la propuesta fuera fácil de utilizar y fácil de comprender para alguien no tan familiarizado con la notación. En el modelo de componentes [10] no es buena idea dejar de lado la formalidad porque queremos que esto sea un proceso sistemático, sin embargo algo interesante sería especificar la propiedad no funcional en función de los componentes con los que existe comunicación. Esto es importante sobre todo si se especifican propiedades no funcionales que son asociadas a la arquitectura o dependientes del uso del sistema [2].

5 Tiempo de ejecución en ensambles de componentes del modelo.

En el capítulo anterior hemos observado cómo se han especificado distintas propiedades no funcionales en otros modelos de componentes y la manera en que se realiza una predicción de esa propiedad no funcional en un ensamble de componentes o en toda la arquitectura del sistema. En este capítulo comenzaremos a explorar la factibilidad de realizar esta especificación en el modelo de [10].

Primero se establece el contexto del modelo de componentes, la propiedad no funcional que se va a medir y la metodología a seguir para medir esta propiedad no funcional.

5.1 Contexto de la implementación del modelo del componentes

El modelo de componentes fue implementado en el lenguaje Java. Existe un repositorio de conectores que contiene todo el catálogo de conectores que se mencionó en la sección 2.3. Además existe un repositorio de varios componentes atómicos que en su mayoría son componentes que implementan funciones simples de tipo sensor y actuador. En ejercicios anteriores [9] estos componentes e han utilizado para construir varios sistemas, entre ellos una máquina de café y un sistema para controlar un robot educativo de Lego Mindstorms [5].

En cuanto a la ejecución de los componentes hay dos factores importantes. El primero que en la ejecución de los componentes y sus ensambles no existe concurrencia o multi-threading. El otro factor es el uso de la API de Reflection de Java para realizar la invocación de los componentes.

La API de Reflection nos permite construir código flexible que puede ser ensamblado durante el tiempo de ejecución sin requerir ligar el código fuente entre los componentes. Eso permite establecer la conexión entre componentes y conectores sin afectar el código interno de los mismos [18].

Existen algunas características de Java que dificultan la medición del tiempo de ejecución. Estas características se discutirán más adelante.

La metodología a seguir para medir el tiempo de ejecución de los elementos que conforman la arquitectura del modelo de componentes es la siguiente:

- Establecer la forma en la que serán medidos los elementos de un ensamble en base a las características de Java.
- Determinar el tiempo de ejecución de los elementos de un ensamble (componentes y conectores).

- Determinar las características del ensamble que impactan en la predicción.

5.2 Medición de tiempo de ejecución en Java

Para hacer la especificación de la propiedad no funcional en los componentes necesitamos medir el tiempo de ejecución de los componentes atómicos. Realizando varias pruebas obtendremos datos que nos permitan obtener peor caso de ejecución, el promedio de tiempo de ejecución y obtener un rango de confianza.

Medir el tiempo de ejecución en componentes desarrollados en lenguaje Java no es fácil. La ejecución de un componente de Java es no determinista, esto debido a mecanismos en la Máquina Virtual de Java que provocan mucha variabilidad en el tiempo de ejecución.

En [3] se hacen las siguientes recomendaciones para medir el tiempo de ejecución en Java:

- Hacer la medición en distintas máquinas virtuales y distintas iteraciones.
- Obtener un intervalo de confianza de las mediciones.
- Medir en múltiples tamaños en la memoria Heap y distintas plataformas de hardware.
- Liberar la memoria con el recolector de basura antes de cada medición.

Dado que en la primera vez que se ejecuta el componente se realiza una compilación, esta es la ejecución que más tiempo toma. Algunos autores no toman en cuenta esta ejecución, ya que se sale del rango de confianza. Se recomienda configurar Java para que haga una compilación de todo el código antes de que este sea utilizado mediante el comando -Xcomp.

Existen 2 tipos de errores en las mediciones de tiempo de ejecución: los errores sistemáticos y los errores aleatorios. Los errores sistemáticos se deben a un error en el procedimiento con el que se hace la medición. Por otra parte los errores aleatorios son impredecibles y no deterministas.

En Java existe algo llamado compilación JIT (Just-In-Time) que permite pasar el código de componentes que son ejecutados frecuentemente a código nativo. Cuando se hace esta compilación se dice que la ejecución entra en un estado estable del tiempo de ejecución, es decir, se reduce en gran medida la variabilidad.

El problema es saber cuántas ejecuciones se requieren para entrar en este estado estable. Para saber en qué momento se entra al estado estable se recomienda calcular el coeficiente de variación en un rango de iteraciones. Si el valor del

coeficiente es 0.01 o 0.02 entonces se llegó a ese estado en ese número de iteraciones.

La recolección de basura afecta también al tiempo de ejecución del sistema y es otra de las causas del no determinismo en la ejecución. Al momento de hacer la recolección de basura. Cada vez que se vaya a realizar una ejecución debemos de invocar a la recolección de basura para que esta no interfiera con la ejecución y nos de un resultado incorrecto.

Al momento de determinar el tiempo de ejecución de los componentes se deberán tomar en cuenta todos estos factores para asegurarnos de que tenemos una medición confiable.

5.3 Tiempo de ejecución de componentes atómicos

Para poder realizar una predicción del tiempo de ejecución del ensamble primero necesitamos conocer el tiempo de ejecución de las partes que conforman el ensamble. Es por eso que primero debemos conocer el tiempo de ejecución de los componentes atómicos y de los conectores exógenos.

La mayoría de los componentes atómicos no son complejos y su ejecución no requiere más de 1 milisegundo. Es por esto que para realizar la medición del tiempo de ejecución se utilizó el comando System.nanoTime() de Java. Con este comando se obtiene el tiempo antes y después de hacer la ejecución, por último se obtiene la diferencia entre estos tiempos y obtenemos el tiempo de ejecución en nanosegundos.

En cada medición se ejecuta 30 veces el método del componente, sin embargo, sólo se obtiene un promedio del tiempo de ejecución de las ejecuciones que se encuentran en el estado estable. Para encontrar el estado estable se obtiene el promedio y la desviación estándar de los tiempos de ejecución. Con estos dos datos podemos calcular el coeficiente de variación, el cual nos indica la dispersión que existe entre los datos obtenidos. Lo ideal es tener un coeficiente de variación de 0.02 ó 0.01.

De todo el catálogo de componentes se decidió trabajar con sólo algunos de ellos para hacer la medición y predicción de su tiempo de ejecución en un ensamble. Se seleccionaron los componentes que conforman la CoffeeMachine, en especial el componente Dispenser. Estos componentes conforman las partes de una máquina de café, el componente Dispenser es el que administra el suministro de uno de los ingredientes del café (leche, café, agua, azúcar).

Al realizar la medición del tiempo de ejecución del método setTemperature del componente Dispenser se obtuvieron los resultados que se muestran en la fig. 5.1:

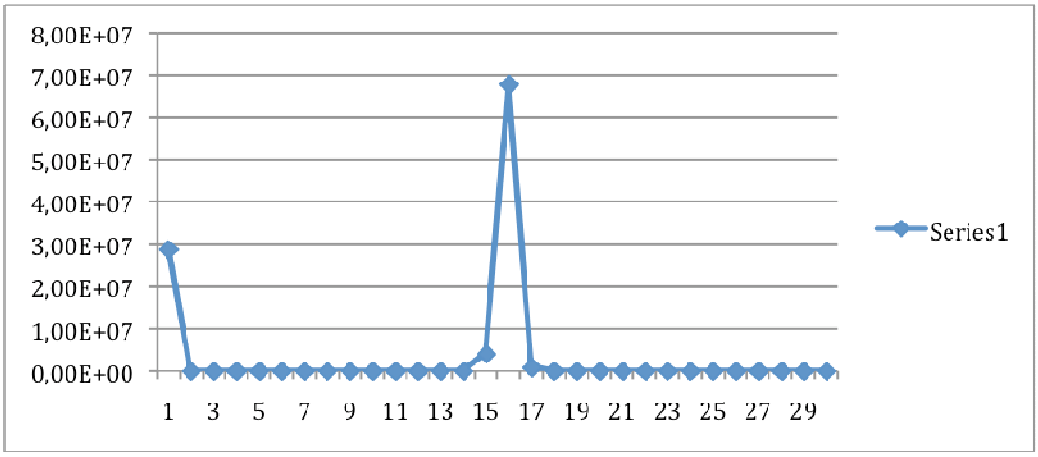

Fig. 5.1 – Tiempos de ejecución método setTemperature del componente Dispenser

En la figura anterior podemos observar que existen dos tiempos de ejecución que son muy grandes: la primera ejecución y la número 15.

Durante la primera ejecución se hace la compilación JIT (Just-In-Time). A pesar de que se utilizó el comando para hacer la compilación previo a la ejecución, la primera ejecución toma más tiempo que el resto. Sin embargo de no haber utilizado el comando de compilación este tiempo hubiera sido mayor y no le toma una sola ejecución estabilizarse.

La ejecución número 15 aumenta debido al uso de la API de Reflection de Java. Los componentes atómicos constan de dos partes: una Computation Unit(CU) y un Component. La CU es una clase con sus métodos públicos que realizan la computación. El Component es un wrapper que tiene una interfase que describe los métodos de la CU y que mediante un método execute hace la invocación a los métodos de la CU utilizando Reflection.

Después de ejecutarse 15 veces el mismo método utilizando Reflection, Java realiza una optimización de esta llamada y a partir de esa ejecución se inicia el estado estable mencionado en el capítulo 4.4. Se dice que es un estado estable porque al calcular el coeficiente de variación de las últimas 14 ejecuciones el valor obtenido oscila entre 0.05 y 0.1.

En la siguiente figura se muestra la dispersión de los tiempos de ejecución a partir del estado estable:

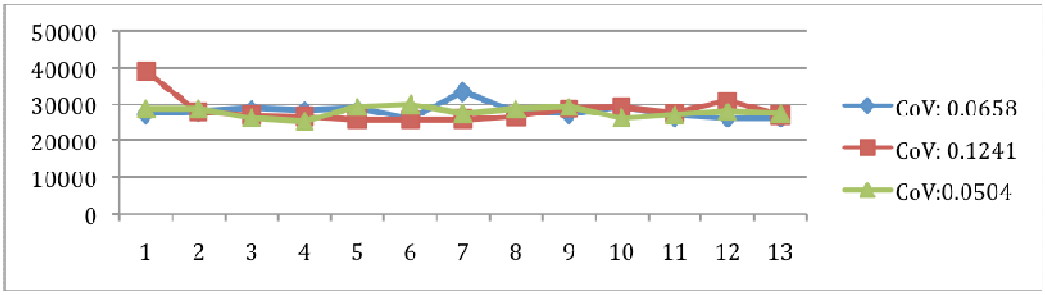

Fig. 5.2 – Tiempos de ejecución del método setTemperature del componente Dispenser durante el estado estable.

Como se puede apreciar en la figura las ejecuciones con menor dispersión son aquellas que tienen el menor coeficiente de variación. Sin embargo, se deben realizar más ejecuciones y tomar en cuenta aquellas que den un coeficiente de variación aún menor. El promedio del tiempo de ejecución de las pruebas con menor coeficiente de variación fue entre 27,768 y 28,031 nanosegundos.

5.4 Tiempo de ejecución de un conector exógeno

Para poder medir el tiempo de ejecución de un conector exógeno se tuvo que crear un componente atómico de prueba cuyos métodos no realizaran ninguna computación. Como se mencionó en el capítulo 2.3, los conectores se encargan de la interacción y comunicación entre componentes, es por eso que no es posible realizar la ejecución de un conector sin tener al menos un componente asociado al conector.

A pesar de que el componente de prueba no realiza computación dentro de sus métodos toma cierto tiempo realizar la ejecución debido a la Reflection de Java. La ejecución promedio del componente de prueba fue de 18,552 nanosegundos. Después de medir el tiempo de ejecución del conector podremos restar este tiempo para tener un tiempo más preciso.

Se midió el tiempo de ejecución del conector Secuenciador haciendo un ensamble con 3 componentes de prueba y utilizando un método que recibe como parámetro un número entero. El conector Secuenciador hace la ejecución secuencial de los 3 componentes de prueba. En la figura 5.3 se muestra el resultado de esta medición.

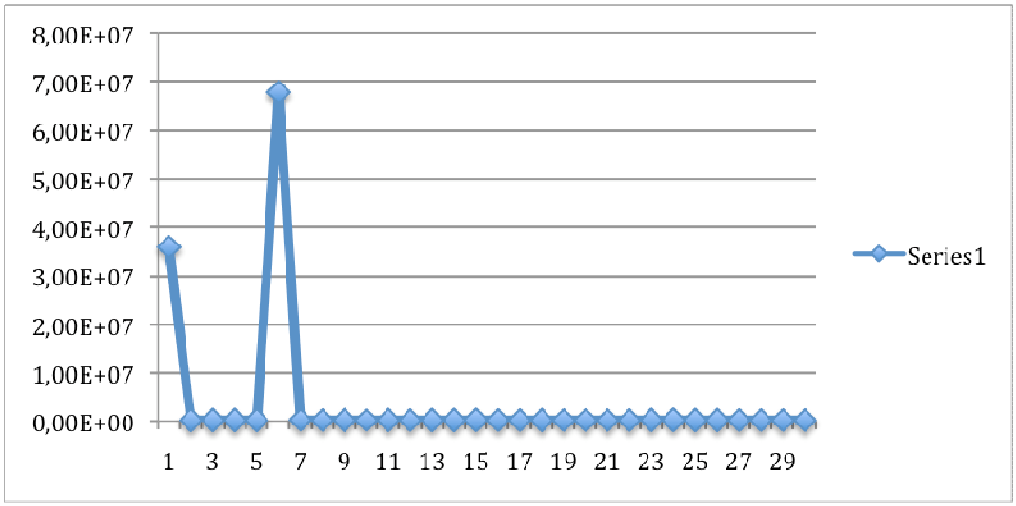

Fig. 5.3 - Tiempos de ejecución método pass del componente de prueba

En esta ocasión el mayor tiempo no ocurre en la ejecución número 15, sino en la 5, esto se debe a que en cada iteración se ejecuta el mismo método 3 veces por lo tanto para la quinta iteración ya son 15 veces que se ejecuta. Cabe señalar que no importa que se trate de una instancia distinta del componente, lo que importa es el método que es invocado.

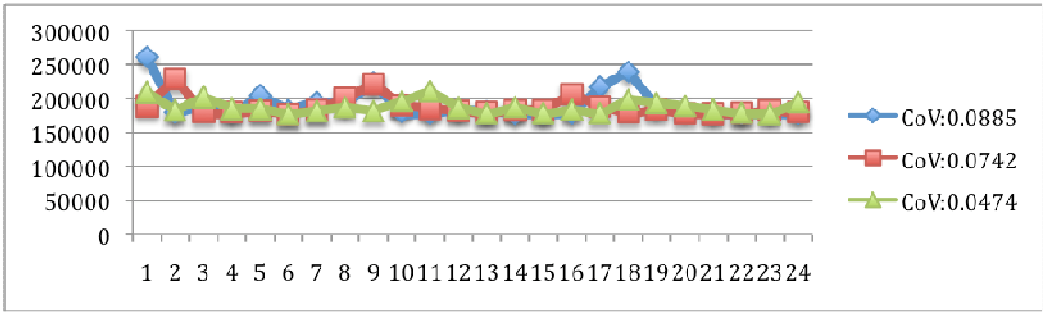

Fig. 5.4 - Tiempos de ejecución del método pass del componente de prueba durante el estado estable.

El promedio del tiempo de ejecución para este ensamble oscila entre 186,554.91 y 189,250.43 nanosegundos, a este tiempo le restamos el tiempo de ejecución del componente de prueba y obtenemos un tiempo de ejecución entre 168,003 y 170,698 nanosegundos .En las siguientes secciones utilizaremos los tiempos de ejecución obtenidos para compararlos con el tiempo de ejecución de un ensamble.

5.5 Tiempo de ejecución de un ensamble de componentes

En la sección 2.3.2 mencionamos que después de tener el tiempo de ejecución de los elementos arquitectónicos del ensamble debemos medir el tiempo del ensamble. Por ser un primer ejercicio se decidió utilizar ensambles muy simples utilizando el conector Secuenciador.

En la sección 3.2 se mencionó que la especificación en los componentes y conectores entra en la clasificación de propiedades de composición directa, y la especificación en los ensambles son propiedades asociadas a la arquitectura. En esta sección veremos que el tiempo de ejecución cambia considerablemente con respecto al tiempo de ejecución de los elementos arquitectónicos.

Para medir el tiempo de ejecución de un ensamble de componentes se pensó en medir para un ensamble de un Secuenciador con 3 componentes Dispenser utilizando el mismo método del componente Dispenser(Fig. 5.5) y después el mismo ensamble pero utilizando métodos distintos de Dispenser(Fig. 5.6). El objetivo es verificar si el tiempo de ejecución es el mismo a pesar de que el segundo caso tarda más en llegar al estado estable.

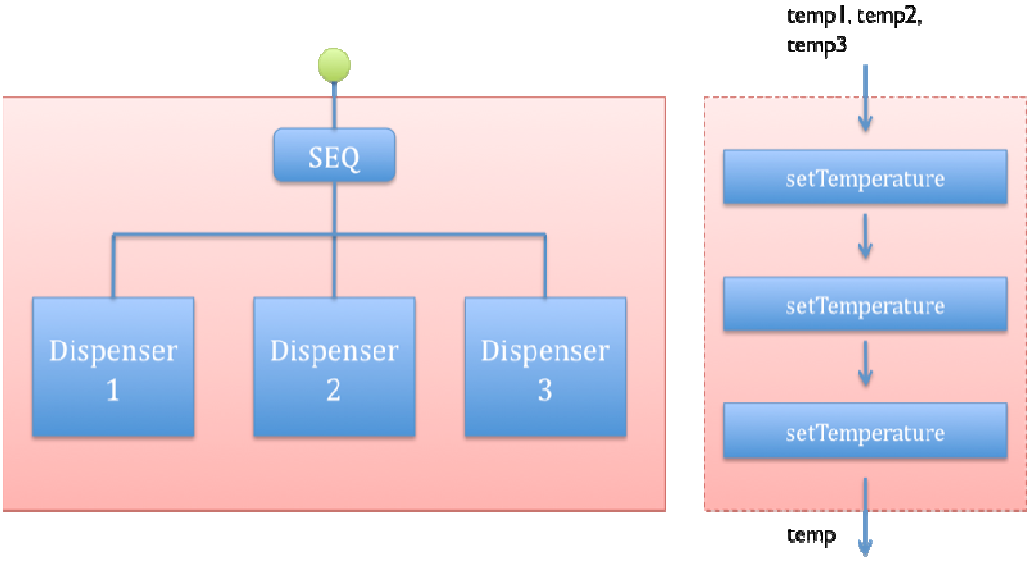

Fig 5.5 – Ensamble de un Secuenciador con 3 componentes Dispenser utilizando el mismo método.

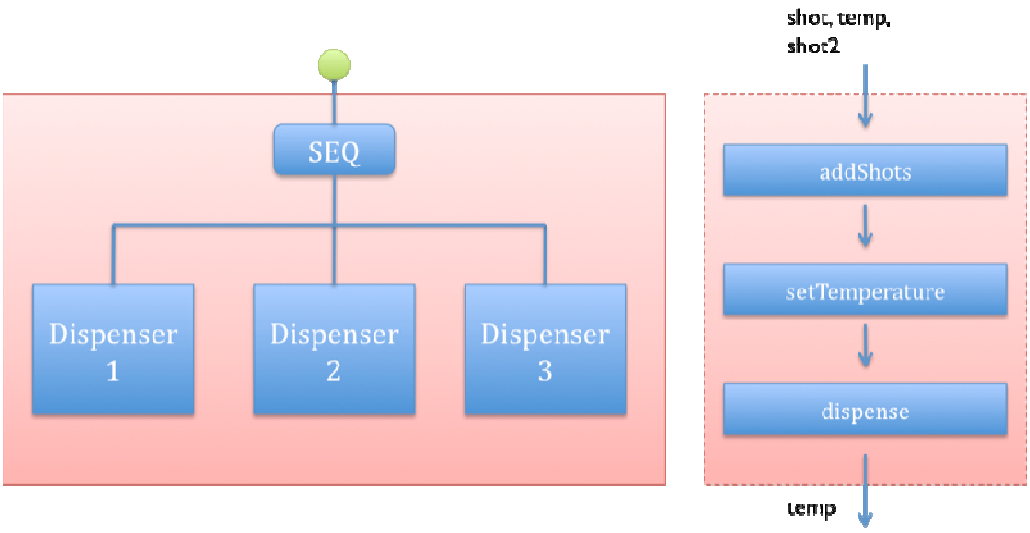

Fig. 5.6 – Ensamble de un Secuenciador con 3 componentes Dispenser utilizando métodos distintos.

En el primer caso se obtuvieron los resultados mostrados en la Figura 5.7. De esta medición el estado estable se alcanzó en la quinta ejecución y se obtuvo un tiempo promedio de ejecución entre 237,116.39 y 239,849.91 nanosegundos.

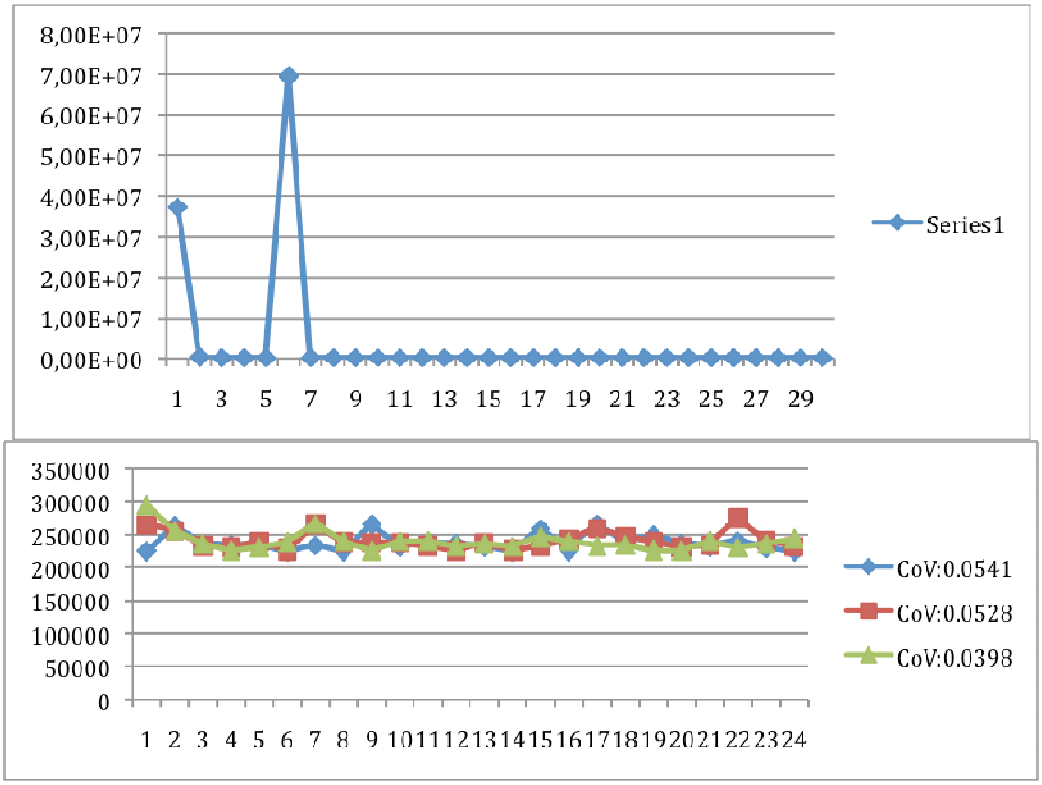

Fig. 5.7 – Tiempos de ejecución del ensamble de Secuenciador con 3 Dispenser utilizando el mismo método.

En el segundo caso el estado estable se obtuvo en la ejecución 15 y se obtuvo un tiempo promedio de ejecución entre 240,436.46 y 240,666.69 nanosegundos. Podemos concluir que no existe mucha diferencia entre el tiempo de ejecución de los dos casos.

Si comparamos el tiempo de ejecución del ensamble con el tiempo de ejecución obtenido en el componente atómico y el conector podemos ver que no existe mucha diferencia entre ellos. El tiempo de ejecución de 3 componentes Dispenser es de 84,093 nanosegundos, el tiempo del conector es de 170,698 nanosegundos, al sumarlo obtenemos 254,791 nanosegundos. Este tiempo se acerca mucho a los 240,666.69 nanosegundos obtenidos en la medición del ensamble de componentes anterior.

Algo importante que se debe señalar es que los 3 métodos distintos que se utilizaron reciben como parámetro un número entero. ¿Qué sucede cuando el método no recibe parámetros o recibe más de un parámetro?.

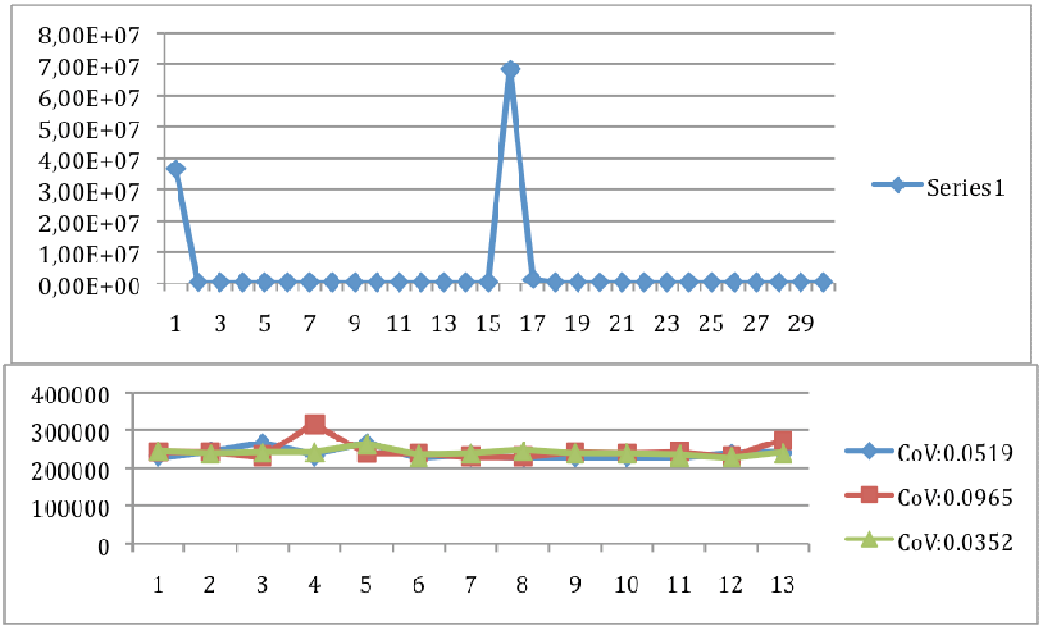

Fig. 5.8 - Tiempos de ejecución del ensamble de Secuenciador con 3 Dispenser utilizando métodos distintos.

Para comprobar que el tiempo de ejecución del conector varía dependiendo de los parámetros que recibe el método en el componente se midió el tiempo de ejecución de varios ensambles utilizando el componente de prueba. Se hicieron combinaciones utilizando el método pass que recibe como parámetro un número entero y el método run que no recibe parámetros. Los resultados de esta medición se muestran en la figura 5.9.

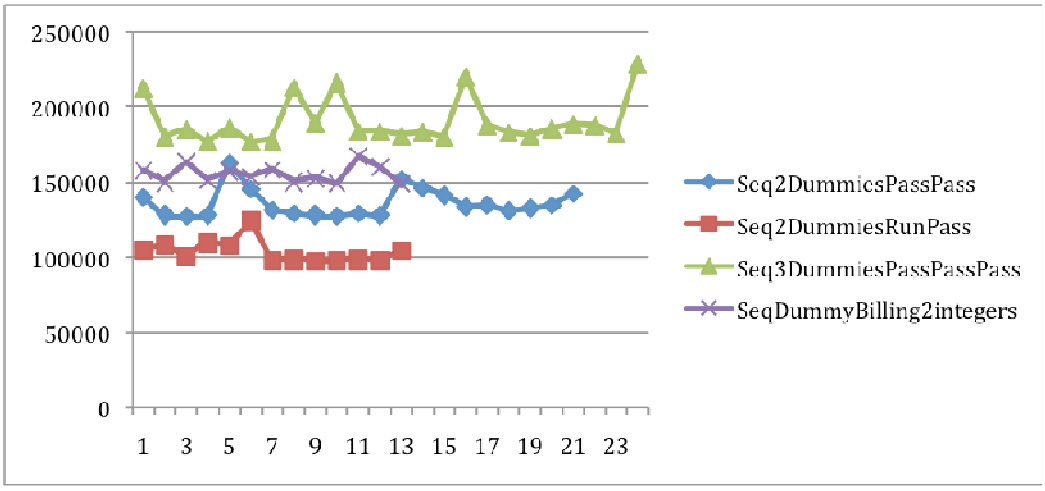

Fig.5.9 – Tiempos de ejecución de distintos según los parámetros recibidos.

En la figura 5.9 podemos ver cómo los ensambles que utilizan el método pass tienen un tiempo de ejecución mayor a los que utilizan el método run. Esto debido a que en el método pass recibe un número entero como parámetro.

Además entre más veces utilizamos el método pass es mayor el tiempo de ejecución.

Podemos concluir que el tiempo de ejecución de un ensamble está determinado por el tiempo de ejecución de los componentes que lo conforman, por el número de parámetros que reciben los métodos utilizados por el ensamble y por el tipo de dato de esos parámetros.

6 Conclusiones y trabajo futuro

Para finalizar, en este capítulo discutiremos las conclusiones de este trabajo y el trabajo futuro.

6.1 Conclusiones

En el paradigma de desarrollo basado en componentes es importante conocer las propiedades funcionales de los componentes que son reutilizados. Sin embargo, también es es de utilidad conocer al menos una propiedad no funcional que nos indique al menos una característica de calidad del sistema.

Existen varias propiedades no funcionales y la manera de medir o de utilizar cada una de ellas es distinta. Algunas de ellas se pueden medir directamente, otras dependen de la arquitectura o se derivan de otras propiedades no funcionales, todo esto se debe considerar al momento de decidir que propiedad no funcional se incluirá en los componentes.

Consideramos que para derivar la especificación de la propiedad no funcional en un ensamble de componentes es de mucha utilidad un modelo de componentes que permita hacer un ensamble de manera sistemática, composicional e incremental . De esta forma la especificacion de propiedades no funcionales del los ensamble podrían generarse a partir de las partes que lo constituyen.

El modelo de componentes utilizado en esta investigación tiene como una de sus principales características el utilizar conectores para realizar la coordinación y comunicación entre componentes, logrando de esta manera encapsular únicamente la computación en los componentes. Esta característica actualmente permite la generación de la especificación funcional en el modelo de componentes y la especificación de propiedades no funcionales podría hacerse de la misma manera.

Tomado en cuenta las carateristicas del modelo de componentes asi como una implementacion particular de este modelo en Java, después de realizar este trabajo estos son los factores identificados que determinan la factibilidad de generar la especificación no funcional de ensambles de componentes:

- Para medir la propiedad no funcional se debe considerar el contexto del modelo de componentes y del ambiente de ejecución: En el capítulo 4 pudimos observar que tanto la especificación como la medición de la propiedad no funcional varía dependiendo de las características del modelo de componentes y del ambiente en el que se ejecutan los sistemas. En la sección 5.1 revisamos el contexto del modelo de componentes de [10] y en la sección 5.2 se consideraron los factores que afectan la medición debido al ambiente de ejecución.

- A partir de la decimo quinta ejecución de un método invocado mediante Reflection se inicia el estado estable: En el capítulo 5 pudimos observar cómo al ejecutar un método por decimo quinta vez Java optimiza la manera en la que invoca el método y a partir de esa ejecución inicia el estado estable. No importa que el método se ejecute en distintas instancias de la clase, lo que importa es que es el mismo método. Además la primera vez que se realiza la optimización es la que más tiempo toma, las optimizaciones subsecuentes no varían en gran medida el tiempo de ejecución.

- Es posible medir el tiempo de ejecución de un conector en el modelo de componentes: En la sección 5.4 vimos que los conectores no se pueden ejecutar sin realizar la composición de un ensamble, sin embargo utilizando un componente de prueba que no realice ninguna computación. El tiempo de ejecución del componente de prueba es muy pequeño por lo que el tiempo de ejecución del ensamble se le puede atribuir al conector.

- Existen algunos factores que afectan a la propiedad no funcional en el ensamble: En la sección 5.5 pudimos observar que el tiempo de ejecución en los ensambles del modelo de componentes de [9] se ve afectado por la cantidad y tipo de dato de los parámetros en los métodos de los componentes que conforman dicho ensamble. Este primer ejercicio sólo contempló métodos que reciben parámetros del tipo integer, parte del trabajo futuro será averiguar de que forma afecta al tiempo de ejecución utilizar parámetros con otros tipos de dato.

6.2 Trabajo futuro

En este trabajo se hicieron mediciones con ensambles muy simples, en futuros trabajos se debe averiguar que sucede con ensambles complejos, observar que sucede cuando se hace un ensable con varios tipos de conectores.

Hasta el momento encontramos que el número de parámetros recibidos en los métodos y el tipo de dato de esos parámetros afectan al tiempo de ejecución de un ensamble, pero puede haber otros factores que afecten el tiempo de ejecución que no se hayan descubierto. Así en un trabajo futuro recomendamos realizar mediciones utilizando un método que reciba un parámetro con un tipo de dato distinto a integer y buscar la proporción en la que afecta cada tipo de dato al tiempo de ejecución.

Se recomienda además plantear una o varias fórmulas con las que se consideren todos los factores mencionados anteriormente para realizar la predicción del tiempo de ejecución al momento de construir los ensambles de componentes. Estas fórmulas se pueden incluir en la funcionalidad de la herramienta visual para realizar la predicción automáticamente al momento de construir un ensamble.

Referencias

[1] C. van Eenoo, O. Hylooz, K. M. Khan. «Addresing Non-functional Properties in Software Architecture using ADL.» School of Computing and Information Technology, University of Western Sydney, Australia.

[2] Larsson, M. «Predicting Quality Attributes in Component-based Software Systems.» *Mälardalen University*, March 2004.

[3] Georges, D. Buytaert, L Eeckhout. «Statistically Rigorous Java Performance Evaluation.» Department of Electronics and Information Systems, Ghent University, Belgium, 2007.

[4] *Enterprise Java Beans Technology.* http://java.sun.com/products/ejb/ .

[5] Group, The LEGO. *LEGO.com MINDSTORM NTX Home.* http://mindstorms.lego.com/.

[6] *ISO 9126 Software Quality Characteristics.* http://www.sqa.net/iso9126.html.

[7] *Java Technology.* http://www.sun.com/java/.

[8] *OMG's CORBA website.* http://www.corba.org/.

[9] P. Velasco Elizondo, C. Ndjatchi. «Functional Specification of Composite Components.» Center for Mathematical Research, CIMAT, Guanajuato, México, February 2010.

[10] P. Velasco Elizondo, K.-K. Lau. *A Catalogue of Component Connectors to Support Development with Reuse.* Article, Centre for Mathematical Research, CIMAT, Guanajuato: The journal of Systems and Software, 2010.

[11] R. H. Reussner, H. W. Schmidt, I. H. Poernomo. *Reliability Prediction for Component-based Software Architectures.* Distributed systems technology center, School of computer science and software engineering , Melbourne, Australia: Monash University, January 2002.

[12] R. van Ommering, F. van der Linden, J. Kramer, J. Magee. «The Koala Component Model for Consumer Electronics Software.» *IEEE Computer*, March 2000: 78-85.

[13] *Response time vs Latency.* http://javidjamae.com/2005/04/07/response-time-vs-latency/.

[14] S. Hissam, G. Moreno, J. Stafford, K. Wallnau. «Packing Predictable Assembly with Prediction-enabled Component Technology.» Carnegie Mellon Software Engineering Institute, Pittsburg, PA, November 2001.

[15] S. Saudrais, O. Barais, L. Duchien, N. Plouzeau. «From formal specifications to QoS monitors.» *Journal of Object Technology* (ETH Zurich) 6, no. 11 (December 2007).

[16] Sentilles, S. «Integration of Extra-functional Properties in Component Mdels.» Research and Technology Centre, Mälardalen University, Västeras, Sweden, 2009.

[17] Siewert, S. *Real-time Embedded Systems and Components.* Thomson Delmar Learning, 2007, 34-35, 85-86.

[18] Sosnoski, D. *Java Programming Dynamics, part 2: Introducing Reflection.* http://www.ibm.com/developerworks/library/j-dyn0603/.

[19] T. Bures, J. Carlson, I. Crnkovic, S. Sentilles and A. Vulgarakis. «ProCom — the Progress Component Model Reference Manual .» *Mälardalen University*, June 2008.