CIMAT

Centro de Investigación en Matemáticas, A.C.

Análisis y Clasificación de Técnicas Basadas en Patrones de Flujo de Trabajo para la Estimación de la Calidad de Ensambles de Componentes de Software

TESIS

Que para obtener el grado de

Maestro en Ingeniería de Software

P r e s e n t a

David Alfredo Barredo Hernández

Directoras de Tesis

Dra. Perla Velasco Elizondo

Dra. Alejandra García Hernández

Zacatecas, Zacatecas., 11 de 12 de 2013

Dedicado a mis amigos José Gayosso, Aída Vacio, Judith Navarro y Paulina García, que me hicieron sentir como en casa durante el tiempo que estuve en Zacatecas; a Texere Editores por permitirme participar en el curso Text Lab Tesis, que fue de gran ayuda para completar esta tesis; y sobre todo a mi madre y mi querida esposa que me han apoyado en todos mis proyectos.

Agradecimientos

Quiero agradecer a todo el personal docente y staff de CIMAT Unidad Zacatecas por su apoyo durante mis estudios de maestría, principalmente, a mis asesoras Dra. Perla Velasco y Dra. Alejandra García por sus consejos y motivación durante el desarrollo de esta tesis; también quiero agradecer al Consejo Zacatecano de Ciencia y Tecnología por el apoyo económico que sirvió como sustento durante mis estudios de maestría.

¡Muchas gracias a todos!

Resumen

En la actualidad cada vez más personas realizan diversas tareas apoyándose de sistemas de software. En estos últimos años se observa que, una tendencia creciente es que, muchos de estos sistemas son construidos a partir del ensamble de componentes de software. En términos generales, un componente es un elemento de software pre-existente que implementa alguna funcionalidad. Usando mecanismos de ensamblaje específicos, estos componentes pueden combinarse para desarrollar sistemas.

Un problema inherente de este enfoque de desarrollo es el de estimar la calidad de los sistemas resultantes. Esto es, si se conocen los valores de los atributos de calidad de los componentes que formarán parte del ensamble, como por ejemplo latencia, consumo de recursos, disponibilidad o seguridad, ¿cómo determinar los valores de los atributos de calidad del ensamble resultante?

Con el propósito dar respuesta a esta pregunta y evaluar la utilidad de algunas de las soluciones provistas para este propósito, en esta tesis se analizan y clasifican algunas técnicas de estimación de la calidad de ensambles de componentes de software. Específicamente, se abordarán las técnicas que utilizan patrones de flujo de trabajo como mecanismo de ensamblaje.

Índice General

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

En la actualidad el software es un producto que los individuos utilizan para realizar la mayoría de sus actividades cotidianas. La elección de qué producto de software utilizar depende de funcionalidad que se desea obtener, pero también depende, en gran medida, de que el producto cumpla con otras propiedades. Cuando se adopta un producto de software para realizar determinadas tareas, es porque se requiere que dichas tareas se realicen más rápido, con mayor precisión o con una menor cantidad de errores, por mencionar algunos ejemplos. Por esa razón, los fabricantes de productos de software deben estar conscientes de que durante su construcción, no basta con satisfacer la funcionalidad esperada por el cliente, también es necesario satisfacer todas aquellas propiedades que le dan un valor agregado al producto. Estas propiedades son conocidas como *atributos de calidad* y son un factor determinante para mantener una fuerte presencia en el mercado de la industria del software.

La mala calidad en los productos de software no sólo impacta en su venta, sino también en el desempeño de las actividades de los usuarios del producto; eso provoca que la calidad del software sea una gran preocupación, no sólo para quienes desarrollan software comercial, sino también para quienes desarrollan software en casa, es decir, para uso propio. Un proyecto de software en el que no se toman en cuenta los atributos de calidad, muy probablemente dará como resultado muchas pérdidas y frustraciones tanto para quien desarrolla el software, como para quien lo usa.

Otro aspecto importante a considerar en la industria del software es la demanda de sistemas de software cada vez más complejos y dinámicos, como grandes almacenes de datos o sistemas de inteligencia de negocios. Los clientes demandan que estos sistemas de software, además de ser de gran calidad, sean desarrollados en forma rápida. Con el fin de reducir el tiempo que toma construir software, una estrategia ampliamente adoptada es la de reutilizar elementos de software ya existentes, en lugar de desarrollar el software "desde cero". Estos elementos de software se "ensamblan" como si se tratara de piezas de una maquinaria, y así, los desarrolladores se concentran sólo en los elementos clave del nuevo producto de software, que no han sido construidos anteriormente. Es así como los equipos de desarrollo de software se van haciendo de una infraestructura de software que les permitirá construir productos cada vez más complejos y en un periodo de tiempo razonable para el cliente.

El *desarrollo basado en composición*, que es como se le conoce al enfoque de desarrollo descrito antes, es una solución para quienes desean reducir el tiempo que toma desarrollar

un sistema de software, ya que se basa en la idea de construir sistemas de software mediante el ensamble de elementos de software preexistentes [36]. A estos elementos de software se les denomina *componentes*.

A pesar de que el desarrollo basado en composición de elementos de software reduce el tiempo de desarrollo, persiste el problema de determinar la calidad de los sistemas de software resultantes, es decir, una vez decidido qué componentes formarán parte del ensamble de software, ¿cómo podemos saber que el ensamble resultante logrará satisfacer la calidad esperada previo a su construcción? En la siguientes secciones proveemos más detalles asociados a esta problemática.

1.1. Contexto General del Desarrollo Basado en Composición

En esta tesis, se hablará únicamente de sistemas de software formados por varios elementos de software denominados componentes. Los componentes de software difieren entre sí de acuerdo con el contexto de implementación. Es así como un servicio web puede ser considerado como un componente de software, al igual que un mashup en el contexto de las páginas web o el botón de un formulario en una aplicación de escritorio.

Los componentes de software están pensados para ser reutilizados en diferentes productos de software con mínimas o quizás ninguna modificación, reduciendo, de esta forma, el trabajo de los desarrolladores de software, mientras que los sistemas de software o productos de software son desarrollados para satisfacer las necesidades de uno o varios clientes. Idealmente el ciclo de vida de los componentes de software, atravesaría tres etapas [36], siendo la primera, una etapa de diseño, en donde los componentes son diseñados y desarrollados para realizar un conjunto de tareas, que pueden ser requeridas para la construcción de varios productos de sofware. En la segunda etapa, los componentes son almacenados en algún sitio para ser utilizados más tarde. En la tercera etapa, que es la de implementación, el desarrollador de software selecciona un conjunto de componentes y los une para construir un nuevo producto de software. En esta etapa se obtiene un producto terminado y por lo tanto, no será almacenado junto con los componentes de software, ya que no se espera que el software resultante sea utilizado para ensamblar un nuevo producto de software.

En algunas partes de esta tesis se hablará de aspectos que no se relacionan exclusivamente con componentes de software, sino que también aplican para productos o sistemas de software. En esos casos, se utilizará el término software, para referirse tanto a componentes de software como productos o sistemas de software.

Cuando se habla de operaciones del software, se hace referencia a las tareas que dicho software realiza. En el caso de los servicios web, las operaciones son conocidas como servicios y en otros tipos de aplicaciones, como las de java, por ejemplo, las operaciones son conocidas como funciones o métodos. En esta tesis se utilizará el término operaciones para referirse tanto a servicios, como funciones y métodos.

Los desarrolladores de software, cuentan, actualmente, con repositorios de componentes de software, que son los lugares donde se almacenan los componentes para su uso posterior,

y estos repositorios pueden ser públicos o privados. Como ejemplo de repositorios públicos podrían mencionarse librerías de PHP y Java, las APIs de google y las APIs de Facebook, Twitter y Linked-in.

Los repositorios de componentes privados constan de un conjunto de componentes para uso exclusivo de quienes han formado o formarán parte en el desarrollo de nuevos componentes o sistemas de software.

Ya sean públicos o privados, los repositorios de componentes pueden contener más de un componente que realice la misma función, pero de forma diferente. Cuando se tiene más de un componente para realizar una misma funcionalidad, cualquiera de ellos puede ser funcionalmente elegible para ensamblar un producto de software, sin embargo, cuando existen restricciones de calidad en el producto de software a desarrollar, sólo se deberán tomar en cuenta aquellos componentes que en conjunto ayuden al producto de software a satisfacer los criterios de calidad establecidos.

Cuando el desarrollador de software selecciona los componentes que formarán parte de un producto de software, éste buscará incluir únicamente aquellos componentes que ofrezcan la mayor calidad posible. A pesar de eso, la calidad del producto final suele conocerse hasta el momento en que todos los componentes han sido efectivamente ensamblados y en ocasiones hasta después de que el producto de software se encuentra en produccción, y es en esos momentos cuando resulta más costoso realizar cambios en el producto de software desarrollado.

Para evitar el riesgo de construir un producto de software que no cumpla con los criterios de calidad esperados, es conveniente tratar de predecir la calidad del producto antes de realizar su construcción. Idealmente, la calidad del producto resultante debería poder estimarse con base en la calidad de cada uno de los componentes que lo conforman.

Algunos de los ensambles de componentes generados pueden formar parte del repositorio de componentes. A partir de ese momento, el ensamble es considerado un componente más del repositorio y posteriormente formará parte de uno o varios productos de software. Este es un aspecto importante, ya que las estimaciones de calidad deben poder realizarse sobre cualquier ensamble de componentes, independientemente de si alguno de los componentes del ensamble es, a su vez, otro ensamble de componentes. En vista de lo anterior, el término ensamble de componentes de software puede hacer referencia tanto a un componente de software formado a partir de otros componentes, como a un producto de software conformado por subcomponentes de software. Es por eso, que en lo sucesivo se hablará de ensambles de componentes para generalizar. En cambio, se utilizará el término *componente compuesto* para hacer referencia a ensambles de componentes de software que dan como resultado un nuevo componente de software y *producto de software* para referirnos al software como un producto terminado.

1.2. Problemática

Cuando se habla de atributos de calidad, lo primero que viene a la mente de los usuarios son cosas como rendimiento, confiabilidad o seguridad, ya que se trata de algunos de los

atributos de calidad más comunes. Otro atributo de calidad, menos analizado por los desarrolladores de software, pero de gran importancia para los usuarios es el costo. Imagine una empresa que fabrica productos y los vende a través de un sitio web. Este sitio web es propiedad de terceros y por lo tanto, nos cobrarán por ventas realizadas; sin embargo, las ventas no son la única fuente de ingresos del sitio web en cuestión. Entre sus muchas prestaciones, ofrecen tres servicios que son de vital interés para nuestra empresa ficticia. Estos servicios son los siguientes:

- Resumen de ventas (RV): El servicio de resumen de ventas ofrece la información de ventas necesaria para generar estados financieros y realizar análisis de ventas.
- Análisis de ventas por país (AVP): Este servicio permite apreciar en qué países son más apreciados nuestros productos y planificar nuevos canales de distribución y ventas con base en este análisis.
- Análisis de interés en los productos (AIP): Si lo que queremos es vender más, no sólo debemos preocuparnos acerca de lo que se vende, sino también acerca de lo que los consumidores buscan. Este servicio provee información acerca de los productos que la gente ha consultado y no ha adquirido. Con la información provista por este servicio, es posible sacar algunas deducciones acerca de las causas de que algunas ventas no se concreten.

El hecho de que nuestro proveedor de ventas en línea nos ofrezca estos servicios, nos ahorra algunos costos operativos, pero también nos hace incurrir en costos por explotación de dichos servicios. Suponga entonces, que cada vez que se solicita un resumen de ventas, el proveedor agrega $100.00 a nuestra factura, otros $20.00 por el análisis de ventas por país y $200.00 más por el análisis de interés por parte de los consumidores. Estos tres servicios serán explotados mensualmente para producir un reporte (RP) que será evaluado por nuestra mesa directiva, por lo tanto, el generar dicho reporte tiene un costo. La pregunta es, ¿De cúanto?

Consideremos que cada servicio es un componente de software y asumamos que la generación del reporte ocurre de la siguiente manera: Primero se obtiene el resumen de ventas (RV), luego el análisis de ventas por país (AVP), posteriormente el análisis de impacto por producto (AIP) y ya que se cuenta con esta información se genera el reporte mensual (RP). Dado que el RP es un componente propio, consideraremos que su explotación no genera costos para nuestra empresa. Bajo este supuesto, el lector ya debe advertir que la solución más evidente consiste en sumar el costo de cada uno de los servicios y así obtener el costo de generar un reporte a través de estos servicios, es decir que el costo total sería de $320.00, resultado de sumar $100,00 + $20,00 + $200,00 + $0,00. Idealmente, esta sería la solución a nuestro problema de cómo conocer la calidad resultante de componer estos servicios, pero no en todos los casos suele ser tan simple ¿Qué tal si nuestro proveedor de ventas en línea nos cobrara el análisis de ventas por país, pero lo hiciera por cada producto que quisiéramos analizar. Eso significa que si nuestra empresa vende 15 productos a través de ese sitio web, habremos de explotar dicho servicio 15 veces para generar el reporte mensual, por lo tanto, el costo de dicho reporte a ascendido a $600.00, resultado de ejecutar el siguiente cálculo $100,00 + ($20,00 × 15) + $200,00.

Dejemos el costo de lado un momento y ahora concetrémonos en otro atributo de calidad, la confiabilidad. Como se mencionó anteriormente, los servicios RV, AVP y AIP son provistos por un tercero y por lo tanto no nos es posible investigar todos los detalles acerca de ellos, sin embargo, con ayuda de un contacto de la empresa proveedora de ventas en línea, hemos descubierto que la confiabilidad de los servicios es inversamente proporcional al tiempo que estos permanecen ejecutándose, es decir, que entre más tiempo dure procesando una solicitud, mayor será la probabilidad de que el servicio falle antes de emitir una respuesta. Nuestro contacto garantiza que la ejecución del servicio será exitosa para cualquier solicitud que dure 20 segundos o menos y la probabilidad de fallo incrementará un 2 % por cada segundo adicional. La probabilidad de fallo es una métrica que se utiliza para determinar si el software es confiable o no. Dado que sabemos que la probabilidad de fallo depende del tiempo de ejecución, sería imposible calcular dicha probabilidad si se desconoce la demanda de tiempo provocada por nuestras solicitudes. Para nuestra fortuna, nuestro proveedor de ventas en línea está dispuesto a proporcionarnos toda la información necesaria para que hagamos un uso informado de sus servicios y así mantener nuestra preferencia, por lo tanto, nos facilitaron la siguiente información acerca de los tiempos de ejecución promedio de nuestras solicitudes, afirmando que existe muy poca desviación con respecto a la media.

```
X(RV)=  24s
X(AVP)= 15s
X(AIP)= 38s
```

Esta información debería ser suficiente para predecir la probabilidad de fallo de cada uno de los servicios y a partir de eso, la confianza que podremos tener en que el reporte mensual se generará exitosamente, la pregunta que surge es ¿Qué razonamiento debemos seguir para que nuestro cálculo de la confiabilidad sea realista?

Ahora toca hablar acerca de la modificabilidad del software. Un detalle importante acerca de los servicios que proporciona nuestro proveedor de ventas en línea, es que todos sus servicios utilizan una Arquitectura Orientada a Servicios (SOA) [14]. Una de las principales características de esta arquitectura es la estandarizacíon del formato de mensajes entre cliente y servidor. Gracias a esta estandarización, actualmente existen herramientas de software que simplifican la integración de servicios que utilizan esta arquitectura. Una de ellas, quizás la más explotada, se menciona en [24]. Supongamos que gracias a la explotación de una herramienta visual que permite integrar varios servicios como si se tratara de uno sólo, somos capaces de realizar el ensamble de los tres servicios de nuestro proveedor, sin editar una sola línea de código y nuestro generador de reportes está diseñado para interpretar cualquier XML y darle formato de reporte, tal como se pretende ilustrar en la Figura 1.1. En consecuencia, para añadir cualquier información en formato XML a nuestro reporte, se requerirá escribir, únicamente 5 nuevas líneas en el archivo de configuración del reporte con el fin de establecer el formato. Esta es una bondad de la que no disfrutaríamos de no ser por SOA y la arquitectura diseñada, por nuestro experto en software, para el generador de reportes.

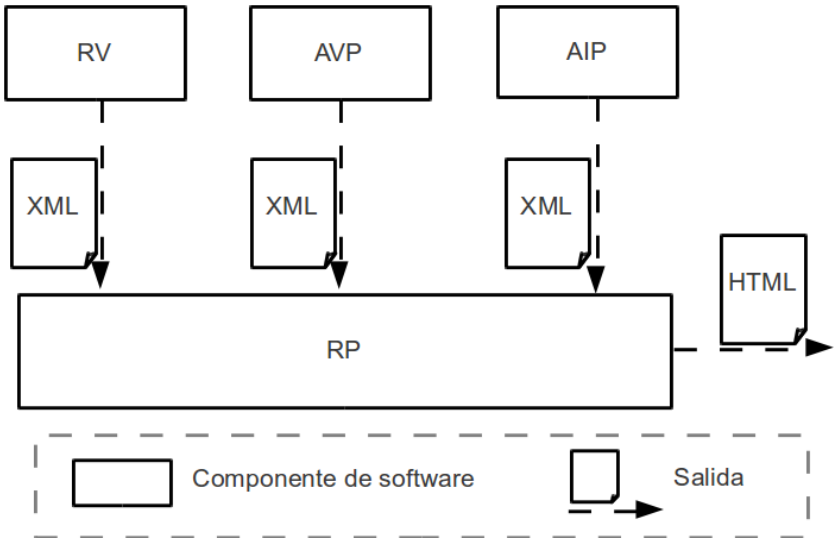

Figura 1.1: Las salidas de los servicios web RV, AVP y AIP son documentos XML que sirven como entradas para RP, que al final produce un reporte en formato HTML
Fuente: Elaboración propia

El último ejemplo, se trata de la seguridad. La seguridad se puede evaluar desde varias perspectivas, como por ejemplo, que sólo los usuarios autorizados puedan ver información clasificada o realizar cambios en el sistema, que las sesiones de usuario esten protegidas por contraseña, que la transmisión de información a través de la red no pueda ser interceptada por usuarios no autorizados o que una vez interceptada no pueda ser interpretada por el atacante. En este caso, los puntos a evaluar son tan diversos que pueden medirse de varias maneras, como si cumple o no cumple, con qué porcentaje cumple, qué tan segura es la contraseña, qué tan compleja es la llave de encriptación; pero, ¿será suficiente con tomar en cuenta estos criterios? ¿Influirá la segmentación de la red o el uso de VPNs en la seguridad de un ensamble de software?

Los servicios ofrecidos por nuestro proveedor de ventas en línea, son servicios web, por lo tanto, lo más lógico es que se transmitan a través de internet, que es una red pública, sin embargo, asumamos que nuestro proveedor ha configurado un servidor *Virtual Private Network* (VPN) que asegura una transmisión protegida de datos, pero, ¿qué tan protegida? Además, el acceso a nuestra información únicamente es posible por medio de una clave que nuestro proveedor nos ha proporcionado, pero, ¿qué tan difícil será para un atacante dar con esa clave? y lo más importante, para este estudio, ¿cómo debemos usar esta información para estimar la seguridad de nuestro ensamble de componentes de software?

Todos los ejemplos presentados asumen la existencia de componentes de software reutilizables y casos como estos se observan con gran frecuencia, ya que en la actualidad es prácticamente imposible satisfacer la demanda de software sin recurrir a componentes de software desarrollados previamente por nuestros propios desarrolladores de software o por terceros. También es un hecho que existe la preocupación por conocer la calidad resultante del ensamble de varios componentes de software, ya que de ello depende realizar la mejor selección de componentes para ensambles de software con ciertas restricciones de calidad. Lo que se pretende dar a conocer con este trabajo es qué tan factible resulta predecir la calidad de un ensamble de software en forma automática y qué tan robustas son las técnicas que

harán eso posible.

1.3. Objetivos

Con el propósito de proveer información que permita responder varias de las preguntas que se plantearon en la sección anterior, en este trabajo se realiza una revisión de las propuestas de diversos autores para estimar la calidad de componentes comupestos. Especificamente, aquellas propuestas basadas en patrones de flujo de trabajo. Con base en este próposito, se plantean los siguientes objetivos:

- Recolectar un conjunto de trabajos en donde se describan técnicas para estimar la calidad de ensambles de componentes a partir de la información de calidad provista por los componentes que constituyen dicho ensamble.
- Analizar y clasificar dichas técnicas de acuerdo a sus características principales.
- Identificar las fortalezas y debilidades de dichas técnicas.

1.4. Estructura del Documento

Este trabajo se estructura de la siguiente manera: El Capítulo 2 definirá algunos conceptos que serán mencionados a lo largo de esta tesis; el Capítulo 3 expondrá los trabajos relacionados con la estimación de la calidad de ensambles de software; en el Capítulo 4 se realizará un análisis de los hallazgos encontrados en la revisión de la literatura; y finalmente, en el Capítulo 5 se presentarán las conclusiones y el trabajo futuro.

Capítulo 2

Marco Teórico

En este capítulo se hablará acerca de los conceptos relacionados con este estudio, con el fin de que el lector se familiarice con la terminología que será utilizada a lo largo de esta tesis. Empezaremos hablando acerca de los componentes; posteriormente se hablará acerca de las interfaces y el papel que cumplen en el proceso de estimación de la calidad; luego de eso tocaremos el tema de los componentes compuestos que es el tipo de ensamble de software que resulta de mayor interés para este estudio; y finalmente se presentará un ejemplo de una técnica básica para estimar la calidad de un ensamble de componentes de software.

2.1. Componentes

En términos generales, cuando hablamos de componentes, nos referimos a las partes de un todo o los elementos de un producto. Como ejemplos de componentes, es posible mencionar el termostato de un refrigerador, el procesador de una computadora, el motor de un automóvil y los pedales de una bicicleta. En el caso del software, cuando hablamos de componentes, nos referimos a bloques de código, ya sea código máquina o scripts, que se encuentran disponibles para ser utilizados con mínimas o nulas modificaciones; o, como se define en [36], cualquier bloque de computación reutilizable que es susceptible de ser ensamblado junto con otros componentes, formando así composiciones más grandes.

Como lo mencionamos en la introducción, una estrategia ampliamente adoptada, es la de desarrollar software a partir de la composición de componentes preexistentes. Existen varios enfoques de desarrollo que se basan en esta estrategia, entre ellos se encuentran la Ingeniería de Software Basada en Componentes (CBSE) [28], la Arquitectura Orientada a Servicios (SOA) y las Líneas de Productos de Software (SPL) [12].

La importancia de los componentes de software radica en su capacidad de ser reutilizados sin realizar cambios en el código fuente de los mismos y por lo tanto, sin necesidad de recompilarlos. De esta forma, los desarrolladores de software pueden disponer de ellos sin riesgo de afectar su funcionamiento a causa de algún error de mantenimiento. Otra ventaja del uso de componentes es que el desarrollador no necesita saber cómo trabaja dicho componente para poder utilizarlo, únicamente necesita conocer la manera de invocarlo, las entradas requeridas y las salidas provistas por el componente; por esta misma razón, a los componentes de software se les considera “cajas negras”.

En la introducción se presentó un caso de ejemplo, en donde se ilustra como un conjunto de servicios web pueden ser utilizados como componentes de un ensamble de software. El resultado de ese ensamble es un producto de software que produce un reporte asociado con las ventas realizadas a través de un sitio web. El diagrama que se presenta en la introducción pretende ilustrar como las salidas de los servicios RV, AVP y AIP son utilizadas como entradas de RP para producir un reporte en formato HTML. Más adelante se presentarán otras vistas que ilustren otros aspectos acerca del ensamble.

Como se mencionó anteriormente, la principal característica de los componentes es la reutilización y su capacidad de formar parte de un ensamble junto con otros componentes. Para lograr eso, es indispensable informar al usuario de los componentes, acerca de la forma como los componentes se comunicarán entre sí y las operaciones que proveerán. La especificación de las interfaces de los componentes es el medio para comunicar esta información, respondiendo preguntas como ¿Qué servicios ofrece y qué servicios requiere el componente? ¿Cómo pueden ser invocados estos servicios? ¿De qué calidad son los servicios ofrecidos? y más.

2.1.1. Interfaces

En programación, una interfaz sirve como un contrato entre un cliente que requiere un servicio y un servidor que provee dicho servicio. Los componentes implementan los servicios especificados en sus interfaces provistas [4]. Uno de los aspectos que se abordan en esta tesis es la manera de especificar las interfaces de los componentes de tal manera que ya no sólo sirvan para conocer el conjunto de servicios provistos por un componente, sino también la manera como dichos servicios son provistos por el componente, en otras palabras, la calidad del servicio, provista por el componente. Como se menciona en la introducción, cuando se habla de calidad del servicio, nos referimos a la calidad con la que el componente realiza cada una de sus operaciones.

En el caso del ejemplo planteado en la introducción se menciona que los servicios utilizan el estándar SOA. El estándar SOA determina que los mensajes que se transmiten, tanto la llamada al servicio como la respuesta, deben ser en formato XML, sin embargo, esto no es suficiente para conocer la manera como los servicios serán invocados. Es por eso, que los proveedores de servicios web publican otro documento en formato XML llamado WSDL. El WSDL de cada servicio de nuestro proveedor de ventas en línea nos informa acerca de la información que se espera recibir al momento de la invocación y nos proporciona una descripción de la respuesta, que nos permite saber qué tipo de información esperar y cómo podemos utilizarla. Pongamos como ejemplo, el servicio de análisis de interés en los productos (AIP). El WSDL nos informa que cada vez que invoquemos a este servicio debemos proporcionar el identificador del producto que se va a analizar en formato de texto, algo parecido a lo siguiente:

```
<xs:element name="productoId" type="xs:string"/>
```

Tambiés nos informa acerca de la salida que proporcionará una vez que termine de realizar el análisis. La salida consiste en un informe de análisis, por lo que no se trata de un valor numérico o una cadena de texto, sino un tipo de dato complejo:

```
<xs:element name="reporteAnalisisInteresProducto">
    <xs:complexType>
        ...
    </xs:complexType>
</xs:element>
```

El WSDL nos ha proporcionado información acerca de los tipos de datos contenidos en los mensajes, tanto de las entradas, como de las salidas. También nos informa acerca de las operaciones que se encuentran a nuestra disposición. En este caso, nos interesa la siguiente operación:

```
<portType name="VentasEnLineaWS">
    <operation name="analisisInteresProducto">
        <input message="tns:ReporteAnalisisRequestMsg"/>
        <output message="tns:ReporteAnalisisResponseMsg"/>
    </operation>
    ...
</portType>
```

Como se aprecia en el ejemplo anterior, la especificación de la interfaz de un componente suele contener en primera instancia las propiedades funcionales del componente, esto es la lista de operaciones que el componente ofrece y la forma de invocar cada una de estas. En ocasiones, para invocar los servicios de un componente es necesario brindar ciertos datos que le indicarán al componente la manera de operar o que quizás necesitarán ser transformados por el componente para ser utilizados más tarde, a estos datos se les denomina entradas o parámetros de entrada, como es el caso del identificador del producto "productoId". Otro tipo de información provista por la especificación de la interfaz de un componente son los resultados o parámetros de salida, es decir, lo que obtendremos como respuesta al finalizar la ejecución de algún servicio del componente, tal es el caso de "reporteAnalisisInteresProducto".

En algunos casos, las empresas de desarrollo de software cuentan con repositorios de componentes [4] reutilizables que abstraen ciertas tareas y permiten elaborar sistemas rápidamente; sin embargo algunos componentes pueden contribuir, o no, a satisfacer la calidad esperada por el cliente. Para saber en qué medida, los componentes, son capaces de promover o inhibir ciertos atributos de calidad, es necesario que esta información se encuentre documentada en alguna parte, de tal forma que el desarrollador de software pueda disponer de ella para decidir qué componentes formarán parte de un sistema y la manera como deberán ser ensamblados [4], para lograr la satisfacción de sus clientes. La especificación de la interfaz de un componente sirve como el único medio para entender y utilizar un componente [35] y por lo tanto es el lugar ideal para documentar los atributos de calidad provistos por un componente específico.

La especificación de interfaces en un lenguaje de programación suele ser independiente del componente que las implementa. Esto significa que una misma interfaz puede ser implementada por más de un componente, cada uno de los cuales ofrecerá los servicios especificados en la interfaz, pero con diferencias en la calidad del servicio. Es por eso que en este punto se

plantea la cuestión de cómo especificar la interfaz de un componente de software de tal forma que ésta refleje la calidad del servicio ofrecida por un componente particular. Una posible solución a esto sería utilizar medios externos e independientes del código de programación, como puede ser un archivo xml o cualquier otro formato de archivo que pueda ser asociado a un componente específico e interpretado por humanos u ordenadores, según sea necesario, tal es el caso del WSDL que por esta y otras razones se convierte en el candidato favorito para esta tarea, sin embargo, existen otros tipos de componentes además de los servicios web, por lo que el WSDL tan solo resuelve una parte del problema. En el Capítulo 3 se abordarán con mayor detalle las alternativas para especificar la calidad de componentes de software.

2.1.2. Componentes Compuestos y Patrones de Flujo de Trabajo

En el contexto del desarrollo basado en la composición, se asume que un ensamble de componentes puede resultar en un sistema o en un componente de mayor complejidad. A los componentes que se forman del ensamble de otros componentes se les denomina *componentes compuestos*. La idea de construir componentes compuestos es reconocida como una buena práctica, ya que es un medio para maximizar la reutilización [35]. En [35] se habla de componentes compuestos como componentes reutilizables de propósito general conformados por el ensamble de dos o más componentes atómicos[1]. En [35] también se hace notar que los componentes compuestos deben ser transparentes para el usuario, por lo tanto, los componentes compuestos deben poder ser especificados y utilizados como si fueran componentes atómicos. Con base en el criterio anterior, para fines de esta tesis se considerará que un componente compuesto puede conformarse por el ensamble de dos o más componentes, sean tanto atómicos como compuestos.

En la introducción se ha presentado un ejemplo de componente compuesto. Se trata del software que construye un reporte mensual acerca de las ventas en línea, con ayuda de los servicios que nos brinda nuestro proveedor de ventas en línea. La Figura 1.1 nos muestra hacia donde se dirigen las salidas de los servicios RV, AVP y AIP, pero la Figura 2.1 es más apropiada para conocer el flujo de trabajo de los componentes de software.

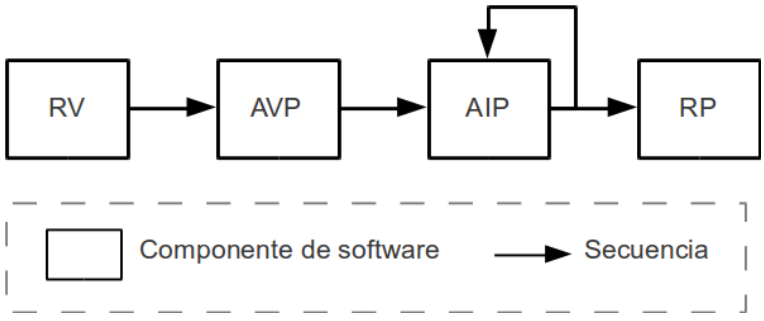

Figura 2.1: Secuencia de ejecución de los componentes del generador de reportes de ventas en línea
Fuente: Elaboración propia

[1]Se considera que un componente atómico es el tipo más básico de componentes en un modelo de componentes [35]

Como se puede apreciar en la Figura 2.1, el estilo de flecha ha cambiado y su significado también. Ya no indica hacia dónde se dirigen las salidas de los componentes, sino la secuencia con que se ejecutan. La flecha que sale del componente AIP se divide en dos, una que apunta al componente RP, el último de la secuencia, y otra que regresa al componente AIP. Esto se debe a que el componente AIP será llamado varias veces antes de pasar al componente final. A este tipo de comportamiento se le conoce como patrón LOOP, y así como este, existen varios patrones asociados al flujo de trabajo de los ensambles de software. Los *patrones de flujo de trabajo*, fueron diseñados para proveer un enfoque uniforme para la comparación de sistemas de administración de flujo de trabajo y sus lenguajes de definición de flujo [19]. En la Figura 2.1 se pueden observar dos patrones de flujo de trabajo; uno de ellos, como ya se mencionó es el patrón LOOP, que representa la ejecución iterativa del componente AIP y un patrón conocido como secuencia, que se ejemplifica con la transición entre los componentes RV y AIP, donde AIP comenzará su ejecución únicamente cuando RV haya terminado de ejecutarse.

2.1.3. Estimación de la Calidad de un Ensamble de Componentes

Al igual que con los componentes atómicos, el repositorio de componentes de un equipo de desarrollo de software puede contar con componentes compuestos. Estos componentes, a su vez, deben contar con una especificación de interfaz que provea información tanto de las propiedades funcionales, como de los atributos de calidad del componente. En el caso de los componentes atómicos, esta información debe obtenerse realizando pruebas de calidad en los componentes o recolectando datos históricos acerca de la calidad con que los componentes realizan sus funciones. Cuando se trata de ensambles de componentes, es posible conocer la calidad ofrecida, en forma similar a como se obtiene en el caso de los componentes atómicos, sin embargo, existen otras técnicas diferentes para realizar esta tarea usando la información de calidad de los componentes que conforman el ensamble.

La forma más común de estimar la calidad ofrecida por un ensamble de componentes, es la *agregación de la calidad*. La agregación de la calidad del software se realiza, comunmente, usando fórmulas de agregación. Como se describirá en el Capítulo 3, existen varias trabajos que definen fórmulas de agregación para patrones de flujos de trabajo específicos. En el ejemplo planteado en la introducción, calculamos el costo total de generar un reporte de ventas en línea explotando tres servicios, ofrecidos por nuestro proveedor de ventas en línea y uno propio. Cada servicio tiene asignado un costo que se agrega a nuestra cuenta cada vez que lo invocamos. En el primer ejemplo asumimos que los componentes del ensamble (RV, AVP, AIP, RP) se ejecutaban usando un patrón secuencial y que cada componente sería invocado una sola vez, por lo que la solución al problema consisitió en sumar el costo de cada uno de los componentes. Si quisiéramos expresar esta solución de manera formal, usaríamos una fórmula de agregación como la siguiente [10]

$$C(a) = \sum_{i=1}^{n} C(c_i)$$

En teoría podremos usar esta fórmula para calcular el costo de otros ensambles de componentes, es decir que, para un ensamble de n componentes enumerados de 1 a n ($1 \leq i \leq n$), el

costo (C) del ensamble (a) es igual a la suma ($\sum$) del costo de cada uno de los componentes (c_i).

Más adelante, se modificó el ejemplo, bajo el supuesto de que el componente AIP, debía ejecutarse una vez por cada producto que se quisiera analizar. Con el conocimiento de que los productos que se venden a través de esta página web son 15 en total, y asumiendo que se desea obtener un análisis de los 15 productos, lo que se hizo fue multiplicar el costo del componente AIP por 15 para calcular el costo de realizar el análisis para los 15 productos. Una fórmula de agregación para explicar esta solución sería la siguiente [7]:

$$C(a) = k \cdot C(c)$$

Es decir, que asumiendo que un componente c será ejecutado k veces, el costo total de la explotación de dicho componente de software ($C(a)$) se calculará multiplicando k por el costo de una ejecución individual del componente ($C(c)$). Esta fórmula será válida sólo en aquellos casos donde el costo del componente se cobre por ejecución, ya que si se cobra por periodo de tiempo, por ejemplo, el número de ejecuciones del componente será irrelevante para calcular el costo.

Esta, junto con otras técnicas de predicción de la calidad de un ensamble de componentes, serán abordadas con mayor detalle en el siguiente capítulo.

2.2. Resumen del Capítulo

En este capítulo se habló acerca de los conceptos básicos para este estudio. Las definiciones que se presentan a continuación, pretenden sintetizar lo descrito en este capítulo.

- **Componente de software:** El componente de software es un elemento de software que satisface un modelo de composición, mediante el cual pueden ser ensamblados y reutilizados sin necesidad de realizar modificaciones sobre dichos componentes [24]. Cuando hablamos de modelo de composición, nos referimos a las reglas que los componentes deben cumplir para poder comunicarse entre sí.

- **Interfaz:** Una interfaz de componente es tratada como una especificación del componente. La interfaz de un componente también es la manera programática de integrar componentes en un ensamble. Los componentes son seleccionados e integrados a través de sus interfaces [11]. Para fines de esta tesis, no se hablará de interfaces en términos de un lenguaje de programación, sino que se utilizará el concepto de especificación de interfaz para referirnos al medio que permitirá conocer a detalle las propiedades funcionales de los componentes junto con los atributos de calidad correspondientes.

- **Componente compuesto:** Un componente compuesto es un componente que se crea a partir del ensamble de otros componentes, ya sean atómicos o compuestos. Al igual que los componentes atómicos, los componentes compuestos son reutilizables y deben contar con una especificación de interfaz que proporcione un mejor entendimiento acerca de cómo utilizar el componente.

- **Patrón de Flujo de Trabajo:** Modelo que provee un enfoque para la comparación de sistemas de administración de flujo de trabajo y sus lenguajes de definición de flujo.
- **Fórmula de agregación:** Una fórmula de agregación es una función que define el cálculo de la calidad del servicio de un ensamble de componentes con base en la calidad del servicio de sus subcomponentes, junto con otras variables, tanto del componente, como factores externos a este.
- **Atributo de calidad:** Son aquellas propiedades del software que hacen referencia a la manera como debe realizar sus funciones para cumplir con las metas de negocio y así los clientes puedan percibir una mayor calidad del producto de software. Se les considera propiedades no funcionales, ya que no indican las funcionalidades que el software debe cumplir, sino la manera como deberá cumplir con sus funcionalidades.

Capítulo 3

Estado del Arte

Este capítulo contiene la revisión del estado del arte correspondiente a los trabajos que describan técnicas basadas en patrones de flujo de trabajo para estimar la calidad de ensambles de componentes a partir de la información de calidad provista por los componentes que constituyen dicho ensamble. Inicialmente discutiremos algunos aspectos importantes relacionados con la especificación de atributos de calidad de los componentes que constituyen dicho ensamble: la sintaxis y la semántica. Posteriormente se describen los trabajos revisados.

3.1. Sintaxis

Al momento de ensamblar un conjunto de componentes, disponibles, para conformar un sistema, estos serán seleccionados con base en una serie de criterios, siendo los más importantes, la funcionalidad ofrecida por los componentes, y la calidad con que los componentes realizan dicha funcionalidad. Por tal motivo, es importante que la documentación de la calidad de los componentes sea parte de la especificación de su interfaz. Los trabajos que se describen a continuación proponen alternativas para especificar las interfaces de los componentes para que estas provean información relativa a la calidad ofrecida por los mismos, de tal manera que esta información pueda ser interpretada por máquinas y personas, y así poder contar con un criterio confiable al momento de integrar dichos componentes en un ensamble de componentes de software. Cuando hablemos de sintaxis, nos referiremos a la manera como se encuentra estructurada la información dentro de una especificación de interfaz. La sintaxis de una especificación de interfaz, debe ser conocida tanto por la parte que documenta la interfaz del componente, como por la parte que consulta la información de la especificación de la interfaz. Estos agentes, tanto el que documenta como el que consulta pueden ser personas o máquinas, sin embargo, el enfoque de este trabajo será sobre aquellas sintaxis que no requieran del juicio humano, ni de algoritmos capaces de procesar lenguaje natural. En cambio se buscará trabajar con una sintaxis que pueda ser fácilmente interpretada por máquinas, lo que permitirá automatizar la mayor parte del proceso.

Algunas aportaciones referentes a la especificación de atributos de calidad, recomiendan documentar y publicar la calidad de las operaciones de los componentes en registros, que puedan ser consultados por una herramienta de software, capaz de seleccionar los componentes que cumplan con criterios de calidad específicos [21, 32]. También se ha propuesto utilizar

este sistema de registros para documentar información complementaria acerca de los atributos de calidad, es decir, sus metadatos [32]. Algunos ejemplos de metadatos de atributos de calidad son: el nombre del atributo de calidad, la unidad de medida y el tipo de dato que se utilizará para evaluar el atributo, por ejemplo: números enteros, unidades porcentuales o tipos booleanos. El objetivo de documentar los atributos de calidad por separado es construir una base de datos robusta que permita evaluar la calidad de los componentes en diferentes contextos de implementación, por ejemplo: aplicaciones web, aplicaciones de escritorio o aplicaciones para dispositivos móviles. El análisis de la calidad de los componentes, haciendo distinción del contexto en el que serán utilizados, es útil, ya que cada contexto puede tener criterios de calidad distintos, además de que un mismo componente puede variar la calidad ofrecida entre un contexto u otro. En [21] los registros que proveen información adicional acerca de los atributos de calidad son denominados registros globales ya que los metadatos de los atributos de calidad permanecen constantes independientemente del contexto en el que serán empleados.

Cuando se desea especificar los valores de la calidad del servicio de algún componente para contextos específicos, se pueden utilizar otros registros denominados registros locales [21]. Estos registros son los que contienen los valores de calidad ofrecidos por los componentes y deben tomar como base la información contenida en el registro global. Es así como un mismo componente podrá contener especificaciones de calidad en distintos registros locales, uno informando acerca de la calidad del componente en el contexto de los dispositivos móviles y otro en el contexto de las aplicaciones web, por ejemplo.

En resumen, el registro global define exclusivamente los atributos de calidad, lo cual es, a su vez, parte de la semántica, y el registro local contiene los valores de calidad que formarán parte de la especificación de la interfaz de un componente. La Figura 3.1 muestra un ejemplo de la información que podría estar contenida en el registro global. La ventaja de los registros descritos en estos trabajos es que son susceptibles de ser interpretados por máquinas, reduciendo, de esta manera, el trabajo que el desarrollador tendrá que realizar para decidir qué componentes integrar en un ensamble.

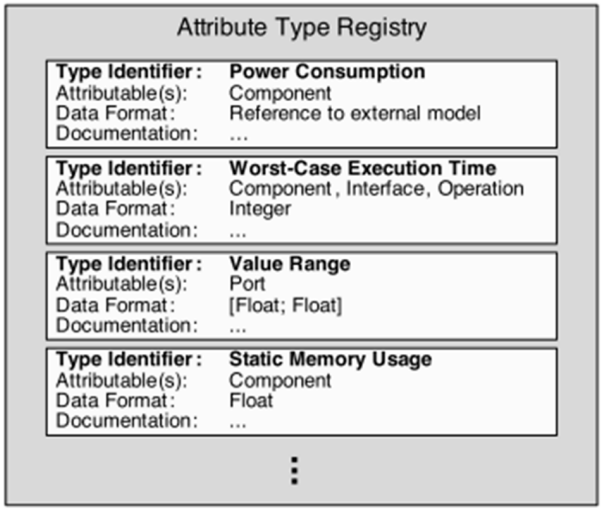

Figura 3.1: Ejemplo de un registro global, donde se almacenan los metadatos de los atributos de calidad [32]

Una forma menos común, y probablemente menos práctica, de documentar la calidad del servicio de los componentes es utilizando diagramas. La Figura 3.2 es un ejemplo de esto. Como se puede apreciar consiste en un diagrama de máquinas de estados que incluye un dato denominado demanda de recurso [4]. Este dato es introducido por el desarrollador del componente y es la base para realizar estimaciones de rendimiento del componente, y a partir de ello, estimar, a su vez, el rendimiento del sistema de software. La documentación de la demanda del recurso (RDSEFF), está pensada para poder predecir el rendimiento en diferentes contextos de implementación, y la documentación de la interfaz utilizando diagramas facilita la interpretación por parte de los seres humanos, pero resulta complicado que el análisis de la calidad, a través de medios visuales, sea realizado por máquinas en lugar de humanos.

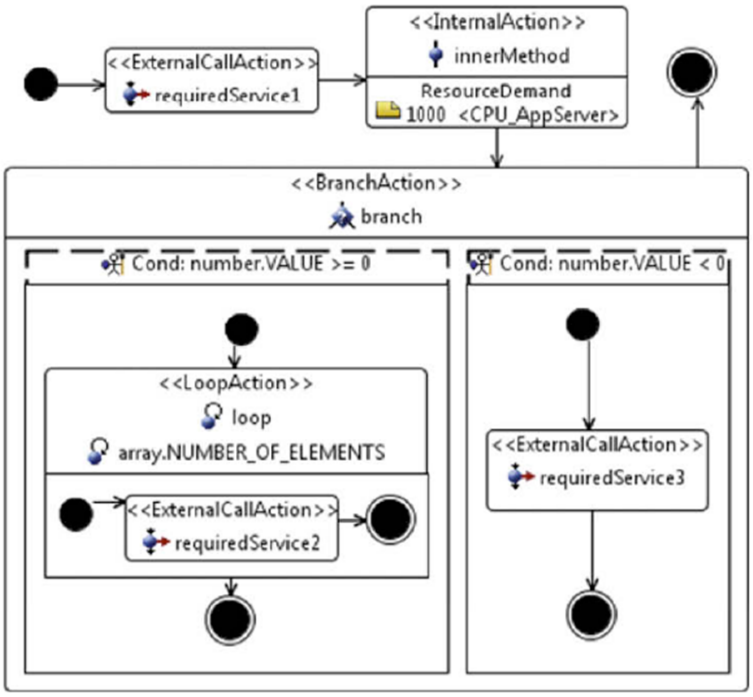

Figura 3.2: Ejemplo de Especificación del Efecto del Servicio en la Demanda de Recursos (RDSEFF) [4]

Con el fin de poder automatizar el análisis de la calidad de un ensamble, el formato de especificación de interfaz más viable es, en apariencia, XML, ya que se trata de un formato de texto estructurado y actualmente se cuenta con intérpretes de documentos XML, o parsers, para la mayorá de los lenguajes de programación actuales, y por lo tanto, pueden ser interpretados y editados, fácilmente, por máquinas además de humanos. La Figura 3.3 muestra el ejemplo de un descriptor utilizado por una herramienta que permite realizar ensambles de mashups [8]. La característica que resulta de particular interés de esta herramienta es que con base en este descriptor hace recomendaciones al usuario acerca de los componentes que ayudarán a mejorar la calidad del ensamble, por lo que se hace evidente la capacidad de la herramienta de software, para interpretar y procesar la información contenida en el descriptor.

En el Capítulo 2 se habló del WSDL que es el documento donde se especifican las interfaces para los servicios web. En los ejemplos se puede apreciar cómo el formato del WSDL, conocido

```
<component id="googlemaps" name="GoogleMaps"
    description="Geolocalization service.">

    <operation name="centerMap" description="..."
        similarity-meaning="CenterMap"
        similarity-verb="center" similarity-object="map">

        <param name="latitude" description="Latitude"
            direction="input" type="xsd:decimal"
            similarity-meaning="Geography.owl#Latitude"/>
        <param name="longitude" description="Longitude"
            direction="input" type="xsd:decimal"
            similarity-meaning="Geography.owl#Longitude"/>
    </operation>

    ...

    <event name="positionOnClick" description="..."
        similarity-meaning="ProvidePosition"
        similarity-verb="provide" similarity-object="position">

        <param name="latitude" description="Latitude"
            direction="output" type="xsd:decimal"
            similarity-meaning="Geography.owl#Latitude"/>
        <param name="longitude" description="Longitude"
            direction="output" type="xsd:decimal"
            similarity-meaning="Geography.owl#Longitude"/>
    </event>

    ...

    <qualityAttributes>
        <reputation>1</reputation>
        <languages>
            <language>javascript</language>
        </languages>
        <dataFormats>
            <dataFormat>JSON</dataFormat>
            <dataFormat>XML</dataFormat>
            <dataFormat>VML</dataFormat>
            <dataFormat>KML</dataFormat>
        </dataFormats>
        <security>no authentication</security>
        <timeliness>0.9</timeliness>
        <accuracy>0.9</accuracy>
        <completeness>0.8</completeness>
        <availability>1</availability>
        ...
    </qualityAttributes>

</component>
```

Figura 3.3: Extracto de un descriptor para la especificación de propiedades funcionales y atributos de calidad de mashups [8]

como XML consiste en un conjunto de etiquetas delimitadas por signos de < y >. El formato XML permite especificar datos en forma jerárquica y es muy sencillo de extender, en caso de que se requiera documentar información adicional. Eso significa que es posible incluir información relativa a los atributos de calidad de un servicio web en su WSDL sin necesidad de romper con el estándar establecido, ya que aquellas herramientas que se apeguen al estándar, simplemente ignorarán las extensiones realizadas en el WSDL, mientras que las herramientas diseñadas para explotar el WSDL extendido podrán hacer uso de la información adicional.

En [15], también se presenta un ejemplo de extención del WSDL, llamado WSQDL. El WSQDL contiene información útil para cada atributo de calidad, como puede ser el tipo de dato, la unidad de medida, la descripción y el método de pago, en el caso del costo; además de información complementaria, como por ejemplo, el número de procesadores del ambiente donde se realizaron las pruebas de calidad.

3.1.1. Enfoque Probabilístico para la Especificación de Interfaces

Otro aspecto relativo a la sintaxis es la información que contendrá la especificación de la interfaz de los componentes. En primera instancia, lo más lógico sería almacenar valores numéricos, como por ejemplo, el costo del componente en términos de una moneda en particular; la latencia en segundos, minutos u horas, al igual que el tiempo promedio antes de un fallo (MTTF); o la accesibilidad como una medida porcentual. En [18] se afirma que en algunos casos esta información puede no ser la adecuada para el usuario, especialmente en aquellos casos donde el usuario desea poder confiar en que la calidad que se encuentra en la especificación de la interfaz del componente es la calidad que va a obtener. Suponga que la especificación del costo de dos componentes sustitutos indica que el costo promedio es de $500.00; esto nos haría suponer que ambos componentes tienen la misma calidad, sin embargo, a la hora de ponerlos en marcha, uno de los componentes presenta costos menores o iguales a $500.00 el 80 % de las veces, mientras que el otro presenta costos menores o iguales a $500.00 el 60 % de las veces. Si el usuario desea tener la certeza de que el 80 % de las veces el costo no va a superar los $500.00, es posible que el primer componente sea la mejor opción.

La propuesta de [18] consiste en especificar no sólo una, sino varias medidas de la calidad de los componentes, indicando la probabilidad de cada una de las medidas, como se ejemplifica en la Tabla 3.1.

Tabla 3.1: Ejemplo de especificación probabilística de la calidad
Fuente: Elaboración propia

Costo	Probabilidad
$100.00	10.00 %
$200.00	10.00 %
$500.00	60.00 %
$1000.00	20.00 %

3.2. Semántica

La semántica se refiere al significado de las palabras. Una parte de la semántica fue abordada a lo largo del marco teórico, sin embargo, existe otra parte de la semántica que aún no ha sido del todo abordada y que resulta de particular interés tanto para la documentación de atributos de calidad, como para la estimación de la calidad en ensambles de componentes; y es la definición de cada uno de los atributos de calidad que serán estimados. Si los atributos de calidad no se definen correctamente, es imposible saber con seguridad si el atributo que se está midiendo es en realidad el que se desea medir. Con la finalidad de establecer una semántica coherente y efectiva para la especificación y eventual derivación de atributos de calidad, es necesario eliminar ambigüedades acerca de la interpretación de los atributos de calidad y su clasificación.

En las siguientes subsecciones se definirán los atributos de calidad estudiados en la literatura, con referencia al tema de estimación de la calidad en ensambles de componentes.

3.2.1. Atributos de Calidad

Los atributos de calidad del software son a su vez propiedades del software. Las propiedades tienen como finalidad describir un fenómeno de interés [11]. Los "stakeholders" o interesados en el desarrollo de un proyecto de software tienden a enfocar su atención en ciertos atributos de calidad, ya que no basta con saber que el software funciona, también es necesario saber cómo funciona. Los atributos de calidad suelen ser una referencia subjetiva si no se documentan adecuadamente. Un usuario puede pedir que el software sea rápido, pero su perspectiva de rápido puede diferir de la perspectiva del desarrollador de software, o lo que para unos usuarios puede ser rápido, para otros puede ser lento. Lo mismo pasa con la seguridad. No existe sistema de software impenetrable, pero existen barreras que limitan la probabilidad de sufrir un ataque, entonces la seguridad dependerá de la magnitud del ataque que se desea resistir. El usuario, seguramente querrá la mayor seguridad posible, pero como todo, la seguridad tiene un costo y por lo tanto habrá que establecer un límite con base en un análisis costo-beneficio. El problema radica en ¿cómo establecer límites si los atributos de calidad se evalúan en forma subjetiva? Para eso se utilizan las *métricas*. Si los atributos de calidad son cuantificables, entonces pueden ser medidos y evaluados. Así, si el usuario indica que el software debe proporcionar una respuesta en máximo 5 segundos, será posible evaluar si el software cumple o no con la calidad esperada.

Entre los atributos de calidad, conceptualmente hablando, existen algunos que son sinónimos y otros que abarcan varios sub atributos de calidad. Está el caso de la latencia, que puede ser considerada sinónimo de tiempo de respuesta, ya que describen el mismo fenómeno, el tiempo que transcurre entre el envío de una solicitud la recepción de la respuesta correspondiente. A su vez, la latencia puede considerarse como rendimiento, ya que el rendimiento describe la eficiencia con que se explotan los recursos del sistema, entre ellos, el tiempo. El "throughput", como se verá más adelante, no es lo mismo que la latencia, pero también es considerado como parte del rendimiento, por lo tanto, taxonómicamente hablando, tanto la latencia como el "throughput" son atributos de calidad que ayudan a describir el rendimiento del software.

En esta sección se presentará la definición de los atributos de calidad que han sido tomados en cuenta en los trabajos relacionados con el tema de estimación de la calidad en ensambles de componentes, así como algunas de las métricas que se utilizan para evaluarlos. En la Figura 3.4 se presenta una taxonomía de atributos de calidad, definida con el fin de lograr un mejor entendimiento acerca de la relación entre los mismos y así poder agruparlos en categorías, según sea conveniente.

A lo largo de la literatura, se puede observar que distintos autores pueden ofrecer una interpretación diferente para determinado atributo de calidad. Para evitar confusiones al respecto, las definiciones que se presentan a continuación pretenden informar al lector acerca de la interpretación que tendrá cada atributo de calidad en esta tesis.

- **Fiabilidad:** Es la habilidad del software para proveer operaciones confiables. También se refiere a la habilidad del software para evitar fallos que sean más severos y frecuentes de lo aceptable, desde la perspectiva del usuario [11]. Dada la generalidad de este atributo de calidad, no es posible definir una métrica única para evaluarlo, por lo tanto, es necesario recurrir a atributos de calidad con un mayor nivel de detalle. En la taxonomía

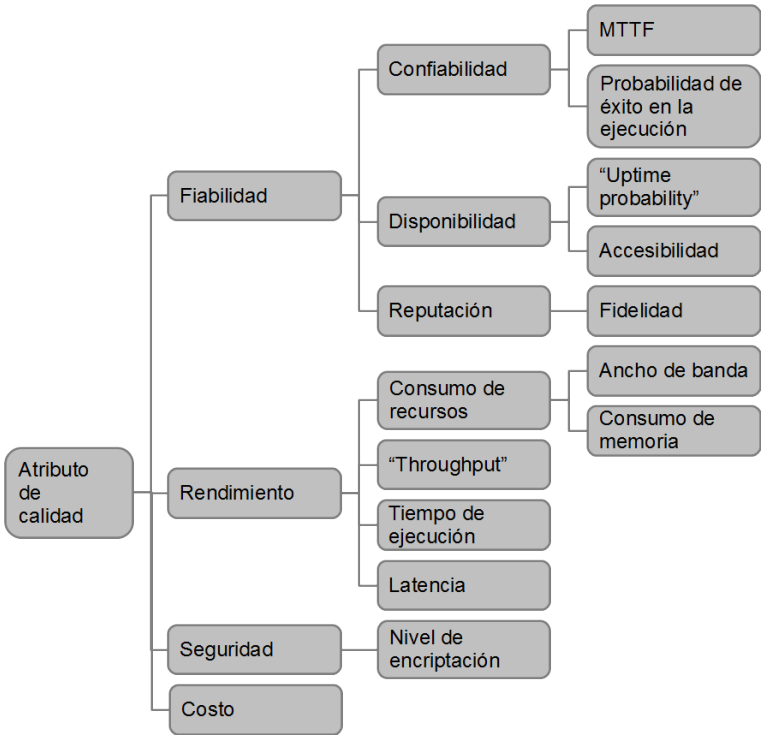

Figura 3.4: Taxonomía de atributos de calidad
Fuente: Elaboración propia

de la Figura 3.4, se muestran tres atributos de calidad que corresponden a la categoría de fiabilidad, como son la confiabilidad, disponibilidad y reputación; mismos que se definirán a continuación. Aunque las palabras fiabilidad y confiabilidad son sinónimos, en esta tesis tendrán distintas interpretaciones, de forma similar a las palabras en inglés "dependability" y "reliability". Fiabilidad se utilizará como traducción de "dependability", mientras que confiabilidad se utilizará como traducción de "reliability".

- **Confiabilidad:** La confiabilidad es un atributo de calidad dentro de la clasificación de fiabilidad, y describe la tasa de fallos del software que provocan que el software quede en un estado no funcional. También describe la habilidad del software para mantenerse operando corréctamente a través del tiempo [3]. Para medir la confiabilidad, es posible hacer referencia a atributos de calidad más específicos, de acuerdo con los criterios de confiabilidad de las personas interesadas. Una de ellas es el tiempo promedio antes de un fallo (MTTF) y otra puede ser la probabilidad de que la ejecución de un componente o servicio sea exitosa (Probabilidad de éxito en la ejecución).

- **MTTF:** El tiempo promedio antes del fallo o mid time to failure (MTTF), es considerado como un atributo de calidad, dentro de la literatura, aunque más que un atributo de calidad se trata de una métrica que permite evaluar la confiabilidad [3] y consiste en el tiempo promedio que el software permanece funcionando sin que se presenten fallos. Para su medida se utilizan unidades de tiempo, como segundos, minutos y horas, entre otras. Por ejemplo, se podría decir que el software tiene un MTTF de 48 horas, si el promedio de horas que permanece funcionando sin que ocurra un fallo es de 48 horas.

- **Probabilidad de éxito en la ejecución:** Este atributo de calidad es el criterio más

común para evaluar la confiabilidad, y consiste en la probabilidad de que el software complete sus operaciones exitosamente [18], es decir, sin la presencia de fallos. La probabilidad de éxito en la ejecución puede calcularse dividiendo el número de operaciones ejecutadas exitosamente entre el total de operaciones ejecutadas en el sistema, y representarse como una medida porcentual. Por ejemplo, si 8 de cada 10 veces que se ejecuta alguna operación, dicha operación es exitosa, entonces la probabilidad de éxito en la ejecución para dicha operación, es de 0.8 u 80 % según como resulte conveniente su representación. La mayoría de los autores no harán mención de este atributo de calidad, en cambio, se referirán a la probabilidad de éxito en la ejecución como confiabilidad.

- **Disponibilidad:** La disponibilidad es un atributo de calidad dentro de la clasificación de fiabilidad y describe de qué tanto el software está listo (o disponible) para ser utilizado [3, 26]. Como se verá más adelante, la disponibilidad comprende a otro atributo de calidad llamado accesibilidad, pero también puede ser evaluado a través de una métrica llamada "uptime probability".

- **"Uptime probability":** Este atributo de calidad es el criterio más común para evaluar la disponibilidad y consiste en la probabilidad de que la ejecución del software se lleve a cabo satisfactoriamente, cuando sea requerido [19]. Dicha probabilidad puede calcularse dividiendo el tiempo durante el cual el software se encuentra en funcionamiento, entre el tiempo transcurrido desde el momento en que se puso en marcha por primera vez. La probabilidad de funcionamiento o "uptime probability" puede representarse como una unidad porcentual. Por ejemplo, suponiendo que un elemento de software lleva un año desde que se puso en marcha y durante ese año, dicho elemento ha estado fuera de servicio durante 35 días, entonces se podría decir que tiene un "uptime probability" de 0.9041 o de 90.41 % según resulte conveniente su representación. La mayoría de los autores no harán mención de este atributo de calidad, en cambio, se referirán al "uptime probability" como disponibilidad.

- **Accesibilidad:** Es un atributo de calidad dentro de la clasificación de disponibilidad y consiste en la capacidad del software para atender a una solicitud, en otras palabras, que no esté ocupado al momento de recibir la solicitud [16]. La accesibilidad puede medirse como el porcentaje de tiempo que el software permanece desocupado y, por tanto, es capaz de atender solicitudes sin enviarlas a una cola de espera. Por ejemplo, suponiendo que el software se encuentra ocupado el 60 % del tiempo, entonces tiene una accesibilidad del 40 % o de 0.4, según resulte conveniente su representación.

- **Reputación:** Es otro atributo de calidad dentro de la clasificación de fiabilidad y describe la opinión que los usuarios tienen del software [15]. La reputación se puede medir con base en la fidelidad.

- **Fidelidad:** Es un atributo de calidad dentro de la clasificación de reputación y describe la manera como los usuarios califican la reputación [18]. No existe una unidad de medida estándar para medir la fidelidad, sin embargo es un atributo de calidad muy utilizado en la industria de software, por ejemplo, en los mercados de aplicaciones para móviles, se solicita a los usuarios calificar las aplicaciones que han utilizado, desde una hasta cinco estrellas; donde una estrella significa que el software es muy malo y cinco estrellas significa que el software cumplió a la perfección con las espectativas del usuario. En

este caso, el usuario califica su fidelidad asignando un número entre 1 y 5 y la fidelidad del software se calcula promediando la calificación de todos los usuarios. Google Play es un ejemplo concreto de una interfaz que ofrece a los usuarios una medida de fidelidad similar a lo que se acaba de explicar.

- **Rendimiento:** Atributo de calidad del software, que describe el grado en el que el software cumple las funciones designadas para este, dadas ciertas restricciones, como son velocidad, precisión o uso de memoria [33]. El rendimiento se mide en función del consumo de recursos de hardware, "throughput", tiempo de ejecución y latencia; y por lo tanto no existe una unidad de medida única para evaluarlo.

- **Consumo de Recursos:** El consumo de recursos es un atributo de calidad dentro de la clasificación de rendimiento y se refiere a la cantidad de recursos de hardware consumidos por el software para realizar una tarea específica. El consumo de recursos se calcula con base en el hardware que resulta de particular interés, desde la percepción del cliente o para el cumplimiento de los objetivos de negocio, como puede ser el consumo de memoria o el ancho de banda.

- **Ancho de Banda:** Es un atributo de calidad asociado con el rendimiento, en términos de consumo de recursos y consiste en la velocidad de transmisión de los datos en una red. El ancho de banda se mide como la cantidad de datos que serán transmitidos en unidad de tiempo, por ejemplo 60Mb/s.

- **Consumo de memoria:** También es conocido como "footprint" [22] y se trata de un atributo de calidad asociado al rendimiento, dentro de la categoría de consumo de recursos. Es la cantidad de memoria volátil, también conocida como memoria RAM, que el software consumirá durante su ejecución. El consumo de memoria o footprint se mide en términos de unidades de almacenamiento de información digital, como pueden ser bytes, kilobytes, megabytes, gigabytes o terabytes.

- **"Throughput":** El "throughput" es un atributo de calidad dentro de la clasificación de rendimiento, que consiste en el número de eventos o solicitudes que pueden ser atendidos durante un lapso de tiempo determinado [3, 22, 31], por ejemplo, la cantidad de datos que pueden ser procesados cada segundo o la cantidad de veces que una operación puede ser ejecutada en un segundo. Cuando se habla de cantidad de datos que pueden ser procesados, es importante delimitar qué es un dato en el contexto de aplicación, por ejemplo, se podría considerar dato a un registro de una tabla de base de datos.

- **Tiempo de Ejecución:** El tiempo de ejecución se encuentra dentro de la clasificación de rendimiento y es el tiempo transcurrido o el tiempo de procesador utilizado al ejecutar el software [33]. El tiempo de ejecución se calcula a partir del momento en que una solicitud es recibida por el software y hasta que la solicitud ha sido completamente procesada. El tiempo de ejecución puede medirse, como su nombre lo indica, en unidades de tiempo, como segundos, minutos, horas, etc. Por ejemplo, una operación particular del software con un tiempo de ejecución de 300 milésimas de segundo.

- **Latencia:** La latencia es otro atributo de calidad dentro de la clasificación de rendimiento, también conocido como tiempo de respuesta, es el tiempo toma responder a un

evento específico [3]. La latencia se calcula desde el momento en que un cliente envia una solicitud hasta el momento en que la respuesta es enviada al solicitante. Por ejemplo, suponiendo que un componente de software envía una solicitud a otro componente de software a través de una red, la latencia de esa transacción sería el triempo transcurrido desde que el primer componente hace la llamada al segundo componente, hasta que el segundo componente realiza la transmisión de la respuesta o hasta que la respuesta es recibida por el solicitante. En el segundo de los casos, la latencia se verá severamente afectada por los retrasos ocasionados por la transmisión de datos a través de la red, además de los tiempos de ejecución. La latencia, al igual que el tiempo de ejecución puede medirse en unidades de tiempo.

- **Seguridad:** Habilidad del software para protegerse a sí mismo contra intrusiones tanto accidentales, como deliberadas [22]. Protección de los datos del software contra consulta, modificación o destrucción no autorizada y protección del software para sí mismo [3]. Una forma de medir la seguridad de un componente o servicio, es a través del nivel de encriptación.

- **Nivel de encriptación:** Este atributo de calidad es el criterio más común para evaluar la seguridad y se refiere a qué tan ocultos se encuentran los datos. Cuando se habla de encriptación de datos, se hace referencia a la dificultad con que un individuo no autorizado podrá interpretar la informacion que se transmite en una red o se guarda en algún medio de almacenamiento. Una referencia común para medir el nivel de encriptación es el tamaño de la llave utilizada para encriptar los datos. Por ejemplo, cuando un conjunto de datos que se transmite a través de una red es encriptado con una llave numérica de 5 caracteres, el atacante podría probar con un caracter entre 0 y 9, lo que implica realizar 10 pruebas, posteriormente con dos caracteres entre 0 y 9 lo que implicaría realizar otras 100 pruebas y así hasta probar con 5 caracteres entre 0 y 9, realizando un máximo de 111,110 pruebas, lo cual no resulta un trabajo demasiado pesado para un ordenador. En cambio, podría usuarse una clave también de 5 caracteres utilizando una llave alfanumérica, donde cada caracter puede contener valores entre 0 y 9 y entre la letra a y la letra z, haciendo distinción entre letras mayúsculas y letras minúsculas; es decir, que cada caracter tomará uno de 62 posibles valores. Una llave como esta requeriría como máximo 931'151,402 pruebas, cosa que representa un esfuerzo considerable incluso para un ordenador y la dificultad aumentaría a 57'731'386,986 con tan solo incrementar un caracter a la llave. Por lo tanto, el tamaño de la llave puede asociarse con el número de pruebas requeridas para desencriptar un mensaje; otra opción es utilizar el número de caracteres como una potencia de la gama posibles valores en cada caracter, por ejemplo, se podría decir que una llave de 6 caracteres alfanuméricos tiene un nivel de encriptación de $62^6 = 56800235584$; que aunque no corresponde al número máximo de pruebas requeridas para desencriptar el mensaje, es una referencia igual de confiable acerca de la dificultad que le implicaría a un atacante desencriptar los datos.

- **Costo:** Es el importe, en términos monetarios, que, quien solicita la ejecución de un software, tiene que pagar a quien la provee para hacer uso de sus operaciones. Este atributo de calidad resulta de particular interés en el contexto de los servicios web ya que algunos de estos servicios se cobran de acuerdo con el número de solicitudes realizadas.

3.3. Técnicas de Estimación

Como lo mencionamos en el Capítulo 2, muchos de los trabajos que han propuesto técnicas para estimar la calidad de ensambles de componentes, con base en patrones de flujo de trabajo, coinciden en emplear una técnica denominada agregación de la calidad; dicha técnica consiste en el uso de fórmulas matemáticas, llamadas fórmulas de agregación. Una fórmula de agregación se define como cualquier función cuya entrada consiste en uno o varios conjuntos de valores y ofrece un sólo valor como resultado [5]. En el Capítulo 1 se presentó un ejemplo de cómo se podría calcular el costo de un ensamble de componentes de software, y en el Capítulo 2 se definieron fórmulas que plantean una solución general para dos de los casos que se presentaron en el Capítulo 1.

Para poder llevar a cabo la estimación de la calidad de ensambles de componentes de software, usando fórmulas de agregación, es necesario entender la naturaleza de los atributos de calidad que se desean estimar. De esta forma, las fórmulas de agregación pueden analizarse considerando la clasificación de atributos de calidad propuesta en [11]:

- **Atributos de calidad directamente agregables.** Estos atributos de calidad son aquellos que pueden estimarse con base en un antributo de calidad, del mismo tipo, en cada uno de sus subcomponentes, por ejemplo, la cantidad de memoria que consume un ensamble de componentes puede obtenerse sumando la cantidad de memoria consumida por sus subcomponentes. Esta solución es similar a la que se planteó en el Capítulo 1 para calcular el costo de un ensamble de componentes de software y como se muestra a continuación, la fórmula de agregación que se presenta como ejemplo en [11] es muy similar a una de las fórmulas que se presentan en el Capítulo 2.

$$\sum_{i=1}^{n} M(c_i)$$

  Lo que se debe notar tanto en la fórmula de costo como en la fórmula de consumo de memoria, es que el operador de agregación, en este caso, $\sum$ se aplica únicamente sobre variables que son del mismo tipo que el valor agregado, es decir, que si lo que queremos es estimar el consumo de memoria del ensamble, el conjunto de sumandos será conformado por el consumo de memoria de cada uno de los componentes. Lo mismo ocurre con el costo, por lo que ambos atributos de calidad pueden ser considerados como diréctamente agregables. En el momento en que una fórmula de agregación dependa de otros atributos de calidad diferentes al que se desea agregar u otros factores tanto del ensamble como factores externos, el atributo de calidad dejará de considerarse diréctamente agregable y pasará a formar parte de alguna de las otras clasificaciones.

- **Atributos de calidad dependientes de la arquitectura.** Estos atributos de calidad pueden o no derivarse de atributos de calidad del mismo tipo en sus subcomponentes, pero además deben tomar en cuenta la arquitectura de software. La arquitectura de software es usada con frecuencia como un medio para promover atributos de calidad particulares, estableciendo reglas que dictan la manera de ensamblar un conjunto de componentes [22]. Por ejemplo, en una arquitectura cliente-servidor, el desempeño del

sistema puede variar dependiendo del número de clientes conectados a la red, e intercambiando información con el servidor. Es de suponerse, por ejemplo, que si un gran número de clientes envían solicitudes al servidor en forma simultanea, algunas de estas solicitudes serán puestas en una cola de espera y por lo tanto, habrá un retraso en el procesamiento de las solicitudes que repercutirá en el tiempo de respuesta.

La fórmula de agregación que se presenta a continuación fue tomada de [11], para ejemplificar la agregación de la calidad en atributos dependientes de la arquitectura.

$$ax + b\frac{x}{y} + cy$$

La variable independiente x hace referencia al número de clientes, y al número de componentes del ensamble y a, b, y c, son factores proporcionales para una implementación particular, es decir, que para diferentes implementaciones de ensambles, que respeten una arquitectura cliente-servidor; a, b, y c, pueden tomar múltiples valores, e incluso significados. Un detalle a observar es que la variable x (número de clientes) surge a partir de la arquitectura cliente-servidor, por lo tanto, al tratarse de otra arquitectura que no contemple la noción de cliente, la variable x perderá sentido y probablemente surgirá una nueva variable asociada con la nueva arquitectura.

Para que un atributo de calidad se considere dependiente de la arquitectura, la fórmula de agregación debe considerar aspectos de la arquitectura de software. En el caso de la fórmula de ejemplo, se incluyen las variables x y y variables que son tomadas de la arquitectura.

En el Capítulo 1 se presentó un ejemplo de atributo que también depende de la arquitectura. Se trata de la modificabilidad, es decir, la capacidad del software de ser modificado con cierta facilidad. La modificabilidad puede medirse de acuerdo con el tiempo requerido para hacer cambios de cierta magnitud, o de acuerdo con el trabajo requerido para realizar cierta modificación en el software, por ejemplo, en el caso expuesto en el Capítulo 1, se indica que agregar nueva información al reporte, a partir de un documento XML requerirá escribir 5 nuevas líneas dentro de un archivo de configuración, es decir, que el esfuerzo requerido para modificar el ensamble será de 5 líneas por cada servicio web que se desee agregar al ensamble. No todas las aplicaciones de software brindan este grado de modificabilidad y tal como se explica en el ejemplo, este factor de 5 líneas por cada nuevo componente, es consecuencia de la arquitectura de los componentes internos (RP) junto con la arquitectura, SOA, de los componentes externos (RV, AVP, AIP).

- **Atributos de calidad derivados.** Estos atributos de calidad son aquellos que para su estimación toman en cuenta diferentes atributos de los componentes de un ensamble, por ejemplo, el tiempo de respuesta o latencia de un ensamble de componentes, no depende únicamente del tiempo que tardan los subcomponentes en ejecutar sus tareas, sino de la velocidad de transmisión de datos en la red, la cantidad de datos a transmitir y la prioridad de la tarea en cuestión sobre las demás tareas que se estén ejecutando en forma concurrente. A pesar de que esta explicación cobra mucho sentido, no es común ver fórmulas de agregación que consideren que unos atributos de calidad se derivan de otros. La fórmula que se muestra a continuación es un ejemplo tomado de [11] para agregar el tiempo máximo de ejecución, o Worst Case Execution Time (WCET).

$$c_i.wcet + B(c_i) + \sum_{\forall c_j \in hp(c_i)} \left\lceil \frac{L^n(c_i)}{c_j.T} \right\rceil c_j.wcet$$

En la fórmula se puede notar la presencia de otras variables además del WCET de cada componente ($c_i.wcet$), como el tiempo de bloqueo (B), componentes con tareas prioritarias ($hp(c_i)$), latencia (L) y el periodo (T), que a pesar de ser parte de la fórmula no se explica a detalle en el ejemplo. La presencia de todas estas variables convierte al WCET en un atributo de calidad derivado.

En el Capítulo 1 se presentó un caso donde la confiabilidad de los servicios RV, AVP y AIP, depende del tiempo de ejecución. El objetivo de este ejemplo es poner en evidencia la necesidad de derivar algunos atributos, en lugar de agregarlos directamente, por ejemplo, supongamos que ahora que conocemos la confiabilidad de cada componente definimos una fórmula de agregación que no tome en cuenta los tiempos de ejecución de los componentes y tiempo más tarde, nuestro proveedor hace cambios en la infraestructura que provocan que algunos tiempos de ejecución aumenten y otros disminuyan. Eso nos obligará a volver a calcular la confiabilidad de los componentes, manualmente, para poder conocer la nueva confiabilidad del ensamble, cuando lo que se desea es automatizar la estimación de la calidad tanto como sea posible.

- **Atributos de calidad dependientes del uso.** Para estimar estos atributos de calidad en un ensamble es necesario conocer los usos que los usuarios puedan dar a los componentes de software. Un ejemplo de esto es la confiabilidad, ya que para predecir la probabilidad de que la ejecución de un componente de software sea exitosa es necesario conocer y probar los diferentes usos que se le puedan dar. A pesar de definir esta clasificación, en [11] no se presentan ejemplos de fórmulas de este tipo.

  Aunque en [11] no se habla puntualmente acerca de variables relacionadas con el uso, más adelante se hablará de algunos trabajos que definen fórmulas de agregación donde se puede observar la presencia de variables como la probabilidad de ejecución o el número de iteraciones, que no dependen de los componentes del ensamble, ni de la arquitectura, sino que son consecuencia del uso que se le da al software.

  De acuerdo con lo explicado en el párrafo anterior, existe un ejemplo entre los que se presentan en la introducción que toma en cuenta el factor de uso para calcular el costo de uno de los componentes. En la introducción se menciona que uno de los componentes, identificado como AIP deberá ejecutarse una vez por cada producto que deba ser analizado y cada ejecución de este componente genera un costo, por lo tanto, el costo de la explotación del componente AIP depende de la cantidad de productos que se vayan a analizar. En este ejemplo, el factor de uso es la cantidad de productos que se venden a través de la página de nuestro proveedor (k).

- **Atributos de calidad dependientes del contexto.** Los atributos de calidad dependientes del contexto son aquellos que, ya sea que pertenezcan, o no, a cualquiera de las clasificaciones menciondas anteriormente, también dependen del contexto o del estado del sistema, por ejemplo, la salvaguarda de un servicio. Esto depende de varios factores, como el estado actual del sistema, el suministro de energía y el control de acceso físico al servidor. El contexto, también puede referirse al tipo de aplicación, según si se trata de una aplicación para computadoras personales, dispositivos móviles o aplicaciones

web. Hasta ahora no se han encontrado ejemplos de fórmulas que incluyan variables relacionadas con el contexto, ni en [11], ni en otras fuentes.

3.4. Fórmulas de Agregación Basadas en Patrones de Flujo de Trabajo

Como se mencionó en la sección anterior, la arquitectura puede ser un factor determinante en la calidad de un ensamble de componentes de software. En esta sección se presentará una perspectiva que toma como base los patrones de composición usados para integrar varios componentes dentro de un ensamble.

Un *patrón de composición* es un patrón de comportamiento que define la manera como varios elementos habrán de interactuar dentro de un ensamble [23]. Los patrones de composición describen interacciones entre un conjunto de roles, que son realizados por componentes de software [36]. En esta subsección, la atención se centrará sobre un tipo de patrones de composición, denominado *patrones de flujo de trabajo*. Como ya lo describimos antes, los patrones de flujo de trabajo, fueron diseñados para proveer un enfoque uniforme para la comparación de sistemas de administración de flujo de trabajo y sus lenguajes de definición de flujo [19]. Los elementos de un flujo de trabajo suelen ser actividades o tareas, por lo que en este contexto se dejará de hablar de componentes, para referirnos únicamente a las operaciones de los componentes.

A continuación un ejemplo burdo de la importancia de los patrones de flujo de trabajo, en la definición de fórmulas de agregación: si un grupo de operaciones es ejecutado en secuencia, la base para el cáclulo del tiempo de ejecución del ensamble será la suma del tiempo de ejecución de sus operaciones; cosa que no ocurre cuando los componentes se ejecutan en paralelo, donde la base para calcular el tiempo de ejecución será la operación con el mayor tiempo de ejecución del ensamble. Esto se debe a que cuando un grupo de operaciones se ejecuta en forma secuencial, cada operación podrá ejecutarse sólo cuando la operación anterior haya concluido su trabajo y por lo tanto el tiempo de ejecución del ensamble se incrementa por cada componente que sea ejecutado; en cambio, cuando se habla composición en paralelo, se trata de varias operaciones que inician su ejecución simultáneamente y por lo tanto aquellos componentes con menor duración, se vuelven irrelevantes. Es decir que el cuello de botella será la operación con mayor tiempo de ejecución del ensamble, y por lo tanto, será la que determine el tiempo de ejecución del ensamble.

En los trabajos relacionados, se ha encontrado que algunas de las fórmulas basadas en patrones de flujo de trabajo, además de ser dependientes de la arquitectura, contienen algunas características de las que se describen en otras clasificaciones de [11], que fueron explicadas en la subsección anterior. También se ha observado que para un mismo atributo de calidad, pueden existir fórmulas cuyas características coincidan con diferentes categorías. Por ese motivo, en esta subsección se tratará de asociar cada fórmula de agregación con una o varias de las clasificaciones de [11], a fin de encontrar puntos comunes en ambas perspectivas.

A continuación se describen los patrones de flujo de trabajo más utilizados en la literatura y se presentan ejemplos de fórmulas de agregación aplicables a estos. Se ha incluido un glosario de expresiones en la Tabla 3.2 para una mejor comprensión de las fórmulas de agregación,

Tabla 3.2: Glosario de expresiones para las fórmulas de agregación basadas en patrones de flujo de trabajo
Fuente: Elaboración propia

Expresión	Significado
o	Operación
n	Número de operaciones en el ensamble
i	Valor que hace referencia a una operación específica del ensamble (o_i)
L	Número máximo de iteraciones esperadas en un ciclo
l	Número de iteraciones
J	Número de operaciones alternativas o de respaldo para o_i
m	Número de operaciones que deben completarse en un ensamble con tolerancia a fallos
j	Cuando varias operaciones realizan la misma tarea, en ensambles con tolerancia a fallos, la expresión o_{ij} hace referencia a la operación j de la tarea i donde $1 \leq j \leq J$
a	Hace referencia a un ensamble de operaciones
Lat	Latencia
Rep	Reputación
$Cost$	Costo
$Cryp$	Nivel de encriptación
Rel	Confiabilidad
Fid	Fidelidad
Av	Disponibilidad
Ac	Accesibilidad
Tp	"Throughput"
Xt	Tiempo de ejecución
Up	"Uptime probability"
Bw	Ancho de banda
λ	Tasa de fallos
p_i	Probabilidad de que la operación i sea ejecutada
w_i	Factor de ponderación de una operación, generalmente por criticidad
z_i	Factor booleano que indica si una operación es o no crítica
h_i	Factor de ponderación de una operación al momento de la sincronización
f_i	Indica que un ciclo finalizará ejecutando la operación o_i
t_i	Probabilidad de que, entre varias operaciones ejecutándose en paralelo, la operación o_i sea la primera en responder
s_{ij}	En un ensamble con tolerancia a fallos, s_{ij} denota que la operación j que realiza la tarea i ha completado exitosamente su ejecución.
$Lavg$	Cantidad promedio de veces que se ejecuta una operación
$Trans$	Duración de la transmisión de datos a través una red
CPA	Agregación que consiste en sumar exclusivamente de aquellas operaciones que forman parte de la ruta crítica del ensamble.
$g(expr)$	Donde $expr$ es una expresión boleana, $Dom(g) = \{0, 1\}$
e	Número de Euler o constante de Napier
LB	Límite inferior
UB	Límite superior

aunque cabe aclarar, que no es el objetivo de este estudio explicar la interpretación de cada una de las fórmulas presentadas por otros autores, sin embargo, se han modificado las fórmulas de manera que puedan ser interpretadas con ayuda de la Tabla 3.2.

3.4.1. Ejecución en Paralelo

También conocido como patrón AND. Este patrón consiste en la ejecución simultánea de varias operaciones. Esto puede realizarse de varias maneras; en los sistemas multiprocesador, cada operación puede ser responsabilidad exclusiva de un procesador. Otra manera es a través de clusters de computadoras, en este caso cada operación será asignada a una computadora para que todas las operaciones se ejecuten al mismo tiempo. La Figura 3.5 muestra la representación gráfica de un flujo de trabajo que utiliza el patrón AND.

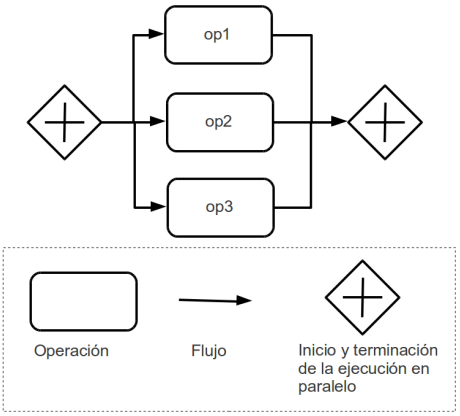

Figura 3.5: Representación gráfica del patrón AND
Fuente: Elaboración propia

A lo largo de este estudio se han encontrado, en la literatura, fórmulas de agregación que se basan en el patrón AND. En la Tabla 3.3 se presentan algunos ejemplos de estas fórmulas. La primera columna muestra el atributo de calidad estimado; la segunda columna muestra el identificador de la fórmula, para poder hacer referencia a cada una de ellas más adelante; en la tercera columna se muestra la fórmula; la cuarta columna contiene las categorías a las que pertenece cada fórmula de acuerdo con sus características; y la última columna contiene las referencias a los documentos que contienen fórmulas similares, o que han presentado ejemplos de agregación que siguen las reglas descritas por cada fórmula.

De acuerdo con las fórmulas que se presentan en la Tabla 3.3, los atributos de calidad que miden las líneas de tiempo, como latencia y tiempo de ejecución se estiman utilizando como base la operación con mayor latencia o tiempo de ejecución, según sea el caso, ya que se considera que este es el cuello de botella del ensamble. Para estimar el "throughput" del ensamble se tomará como referencia la operación con menor "throughput", ya que en este caso, el cuello de botella será la operación que responde a menos eventos durante un intervalo de tiempo. El costo del ensamble será la suma de los costos de las operaciones, ya que todas serán requeridas. El nivel de encriptación del ensamble será equivalente al de la operación con menor nivel de encriptación. Esta regla es análoga a la de que una cadena es tan fuerte como su esabón más débil. La confiabilidad, disponibilidad y accesibilidad, cuando son evaluados con base en métricas porcentuales, se estiman utilizando el operador producto. A pesar de las diferencias antes mencionadas, todas estas fórmulas tienen algo en común, y es el hecho de realizar una agregación directa, de acuerdo con la definición de [11], sin embargo, han sido definidas de esta forma con base en el patrón de flujo de trabajo, por lo que se consideran dependientes de la arquitectura. El MTTF, en cambio, se calcula con base

Tabla 3.3: Fórmulas de agregación para el patrón AND
Fuente: Elaboración propia

Atributo de calidad	ID	Fórmula	Categoría	Fuentes
Confiabilidad	1	$\prod_{i=1}^{n} Rel(o_i)$	Dependiente de la arquitectura	[7, 10, 13, 18, 26]
MTTF	2	$\frac{1}{\sum_{i=1}^{n} \lambda(o_i)}$	Dependiente de la arquitectura, derivada	[16]
Disponibilidad	3	$\prod_{i=1}^{n} Av(o_i)$	Dependiente de la arquitectura	[6, 7, 17]
"Uptime probability"	4	$\prod_{i=1}^{n} Up(o_i)$	Dependiente de la arquitectura	[19]
Accesibilidad	5	$\prod_{i=1}^{n} Ac(o_i)$	Dependiente de la arquitectura	[6]
Reputación	6	$min_{i=1}^{n}\{Rep_(o_i)\}$	Dependiente de la arquitectura	[20]
	7	$\frac{1}{n+1}\sum_{i=1}^{n} Rep(o_i)$	Dependiente de la arquitectura	[6]
Fidelidad	8	$\sum_{i=1}^{n} w_i \cdot Fid(o_i)$	Dependiente de la arquitectura, dependiente del uso	[18]
"Throughput"	9	$min_{i=1}^{n}\{Tp(o_i)\}$	Dependiente de la arquitectura	[6, 19]
Tiempo de ejecución	10	$max_{i=1}^{n}\{Xt(o_i)\}$	Dependiente de la arquitectura	[6, 13, 19]
Latencia	11	$max_{i=1}^{n}\{Lat(o_i)\}$	Dependiente de la arquitectura	[6, 7, 9, 10, 18, 29]
Nivel de encriptación	12	$min_{i=1}^{n}\{Cryp(o_i)\}$	Dependiente de la arquitectura	[19]
Costo	13	$\sum_{i=1}^{n} Cost(o_i)$	Dependiente de la arquitectura	[7, 9, 10, 13, 18, 19, 20]

en la tasa de fallos de los subcomponentes, representada por la expresión λ, por lo que se considera que también cae en la categoría de atributos de calidad derivados. La fórmula de fidelidad utiliza un factor que pondera la fidelidad de los subcomponentes (w). Este factor podría darse por varias razones, entre ellas el uso, en cuyo caso, también pertenecería a la categoría de dependientes del uso.

3.4.2. Secuencia

En un ensamble que utiliza el patrón de flujo de trabajo secuencia, una operación comienza su ejecución sólo cuando la operación anterior a esta se ha completado [17]. La ejecución secuencial es la manera más común de componer operaciones, en especial, cuando las salidas de cada elemento serán utilizadas como entradas del elemento siguiente, como si se tratara de una banda transportadora dentro de una línea de ensamble. En la Figura 3.6 se puede ver una representación gráfica del patrón de flujo de trabajo secuencial; y en la Tabla 3.4 se presentan algunos ejemplos de fórmulas de agregación para este patrón.

A diferencia del patrón AND, en el patrón secuencia, no se trata de identificar cuellos de botella, para estimar atributos de calidad como latencia y tiempo de ejecución, por lo

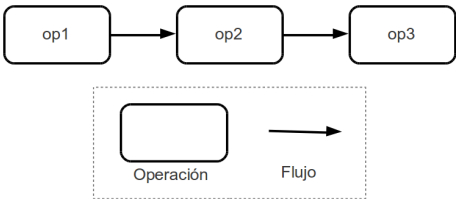

Figura 3.6: Representación gráfica del patrón secuencia
Fuente: Elaboración propia

Tabla 3.4: Fórmulas de agregación para el patrón secuencia
Fuente: Elaboración propia

Atributo de calidad	ID	Fórmula	Categoría	Fuentes
Confiabilidad	1	$\prod_{i=1}^{n} Rel(o_i)$	Dependiente de la arquitectura	[2, 7, 10, 13, 15, 18, 26]
	2	$\prod_{i=1}^{n} (e^{Rel(o_i)*z_i})$	Dependiente de la arquitectura, dependiente del uso	[39]
MTTF	3	$\frac{1}{\sum_{i=1}^{n} \lambda(o_i)}$	Dependiente de la arquitectura, derivada	[16]
Disponibilidad	4	$\prod_{i=1}^{n} Av(o_i)$	Dependiente de la arquitectura	[2, 6, 7, 15, 17]
	5	$\prod_{i=1}^{n} (e^{Av(o_i)*z_i})$	Dependiente de la arquitectura, dependiente del uso	[39]
"Uptime probability"	6	$\prod_{i=1}^{n} Up(o_i)$	Dependiente de la arquitectura	[19]
Accesibilidad	7	$\prod_{i=1}^{n} Ac(o_i)$	Dependiente de la arquitectura	[6]
Reputación	8	$min_{i=1}^{n}\{Rep(o_i)\}$	Dependiente de la arquitectura	[20]
	9	$\frac{1}{n+1} \sum_{i=1}^{n} Rep(o_i)$	Dependiente de la arquitectura	[6]
	10	$\frac{1}{n} \sum_{i=1}^{n} Rep(o_i)$	Dependiente de la arquitectura	[2, 39]
Fidelidad	11	$\sum_{i=1}^{n} w_i \cdot Fid(o_i)$	Dependiente de la arquitectura, dependiente del uso	[18]
Ancho de banda	12	$min_{i=1}^{n}\{Bw(o_i)\}$	Dependiente de la arquitectura	[15]
"Throughput"	13	$min_{i=1}^{n}\{Tp(o_i)\}$	Dependiente de la arquitectura	[2, 6, 19]
Tiempo de ejecución	14	$max_{i=1}^{n}\{Xt(o_i)\}$	Dependiente de la arquitectura	[6, 13, 19, 20, 34, 37]
Latencia	15	$\sum_{i=1}^{n} Lat(o_i)$	Dependiente de la arquitectura	[2, 6, 7, 10, 18]
	16	$\sum_{i=1}^{n} Lat(o_i) + Trans(i)$	Dependiente de la arquitectura, derivada	[15]
Nivel de encriptación	17	$min_{i=1}^{n}\{Cryp(o_i)\}$	Dependiente de la arquitectura	[19]
Costo	18	$\sum_{i=1}^{n} Cost(o_i)$	Dependiente de la arquitectura	[2, 7, 10, 13, 15, 18, 19, 20, 39]

que en lugar de buscar el valor máximo, se suma la latencia o el tiempo de ejecución de las operaciones que componen el flujo. De las fórmulas que se presentan en la Tabla 3.4, las fórmulas 1, 4, 6, 7, 8, 9, 10, 12, 13, 14, 15, 17, 18 únicamente son dependientes de la

arquitectura; las fórmulas 2, 5 y 11 se suman a la categoría de dependientes del uso; la 3 y la 16 se suman a la categoría de derivadas.

3.4.3. Selector

También conocido como patrón XOR. Este patrón provee un esquema de composición, donde sólo una de las operaciones del ensamble es ejecutada, con base en la evaluación de una expresión booleana [36]. Con base en esta descripción, se vuelve evidente que aquellos flujos donde se presente el patrón XOR tendrán comportamientos muy variables y difíciles de predecir, ya que la calidad del ensamble dependerá de la calidad de la operación que sea seleccionada. La representación gráfica del patrón XOR se muestra en la Figura 3.7. En la Tabla 3.5 se muestran los ejemplos de fórmulas de agregación para el patrón XOR.

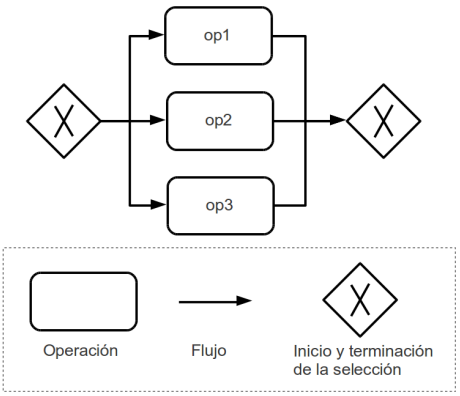

Figura 3.7: Representación gráfica del patrón XOR
Fuente: Elaboración propia

Como se puede ver en los ejemplos de fórmulas para agregar el tiempo de ejecución de la Tabla 3.5, la forma de agregar atributos en flujos XOR es muy variada y depende principalmente de lo que se quiere saber acerca del ensamble. La fórmula 10 presenta la suma ponderada de los tiempos de ejecución, donde el factor de ponderación es la probabilidad de que cada operación del ensamble sea ejecutada, con el fin de que el tiempo de ejecución agregado reciba mayor influencia de las operaciones que se ejecutan con mayor frecuencia. La fórmula 11, es adecuada para quienes desean conocer el peor tiempo de ejecución del ensamble (WCET) sin importar la probabilidad de que se presente ese caso. La fórmula 12, ofrece un promedio ponderado, donde la ponderación pudiera asociarse a la importancia o a la frecuencia de uso, en cuyo caso el tiempo de ejecución agregado sería aproximado al de la fórmula 10. La fórmula 13, ofrece dos resultados, uno optimista y otro pesimista, con el fin de conocer el rango de valores que puede tomar el tiempo de ejecución del ensamble, esto, sin importar la probabilidad de que cada valor se dé.

Analizando los ejemplos de la Tabla 3.5 podemos concluir, que las fórmulas 2, 4, 5, 6, 9, 11, 13, 15 y 16 únicamente son dependientes de la arquitectura. Esto debido a que ya sea que tomen en cuenta valores optimistas, pesimistas o promedio, no se toma en cuenta el factor de uso dentro de la fórmula; en el caso de las fórmulas 14, 7, 17, 1, 8, 3, 10 y 12, la calidad agregada sí se ve afectada por un factor de uso p o w.

Tabla 3.5: Fórmulas de agregación para el patrón XOR
Fuente: Elaboración propia

Atributo de calidad	ID	Fórmula	Categoría	Fuentes
Confiabilidad	1	$\sum_{i=1}^{n} p_i \cdot Rel(o_i)$	Dependiente de la arquitectura, dependiente del uso	[7, 10, 13, 18, 26]
Disponibilidad	2	$min_{i=1}^{n}\{Av(o_i)\}$	Dependiente de la arquitectura	[6]
	3	$\sum_{i=1}^{n} p_i \cdot Av(o_i)$	Dependiente de la arquitectura, dependiente del uso	[7]
"Uptime probability"	4	$min_{i=1}^{n}\{Up(o_i)\}$	Dependiente de la arquitectura	[19]
Accesibilidad	5	$min_{i=1}^{n}\{Ac(o_i)\}$	Dependiente de la arquitectura	[6]
Reputación	6	$\frac{1}{n+1}\sum_{i=1}^{n} Rep(o_i)$	Dependiente de la arquitectura	[6]
	7	$\frac{1}{n}\sum_{i=1}^{n} w_i \cdot Rep(o_i)$	Dependiente de la arquitectura, dependiente del uso	[20]
Fidelidad	8	$\sum_{i=1}^{n} p_i \cdot Fid(o_i)$	Dependiente de la arquitectura, dependiente del uso	[18]
"Throughput"	9	$min_{i=1}^{n}\{Tp(o_i)\}$	Dependiente de la arquitectura	[6, 19]
Tiempo de ejecución	10	$\sum_{i=1}^{n} p_i \cdot Xt(o_i)$	Dependiente de la arquitectura, dependiente del uso	[13]
	11	$max_{i=1}^{n}\{Xt(o_i)\}$	Dependiente de la arquitectura	[6]
	12	$\frac{1}{n}\sum_{i=1}^{n} w_i \cdot Xt(o_i)$	Dependiente de la arquitectura, dependiente del uso	[20]
	13	LB: $min_{i=1}^{n}\{Xt(o_i)\}$ UB: $max_{i=1}^{n}\{Xt(o_i)\}$	Dependiente de la arquitectura	[19]
Latencia	14	$\sum_{i=1}^{n} p_i \cdot Lat(o_i)$	Dependiente de la arquitectura, dependiente del uso	[7, 10, 18]
	15	$max_{i=1}^{n}\{Lat(o_i)\}$	Dependiente de la arquitectura	[6]
Nivel de encriptación	16	$min_{i=1}^{n}\{Cryp(o_i)\}$	Dependiente de la arquitectura	[19]
Costo	17	$\sum_{i=1}^{n} p_i \cdot Cost(o_i)$	Dependiente de la arquitectura, dependiente del uso	[7, 13, 18, 19, 20]

3.4.4. Iteración

También conocido como patrón LOOP. Es un conjunto de sentencias de un programa computacional, que se ejecuta repetidas veces, hasta que se cumple una condición o mientras cierta condición sea verdadera [33]. En este contexto, el concepto sentencias toma un significado más específico, donde una sentencia puede referirse tanto a una operación, como a un ensamble de operaciones, como se muestra en la Figura 3.8 b). También se debe considerar, que el número de iteraciones puede haber sido definido en tiempo de diseño o depender de que se cumpla determinada condición durante la ejecucuión del componente. La calidad de los ensambles de software que utilizan este patrón pueden ser variables o no, dependiendo de si se conoce la cantidad de veces que se repetirá el ciclo. Por ejemplo, si se conoce el número

de veces que se ejecutará una operación dentro del ciclo, el tiempo de ejecución del ensamble podrá calcularse multiplicando el tiempo de ejecución de la operación por el número de iteraciones del ciclo. Cuando no es posible determinar el número de veces que se repetirá el ciclo, se realiza una estimación de la calidad del ensamble con base en la probabilidad de que el ciclo permanezca ejecutándose. Las fórmulas de agregación para el patrón LOOP se muestran en la Tabla 3.6.

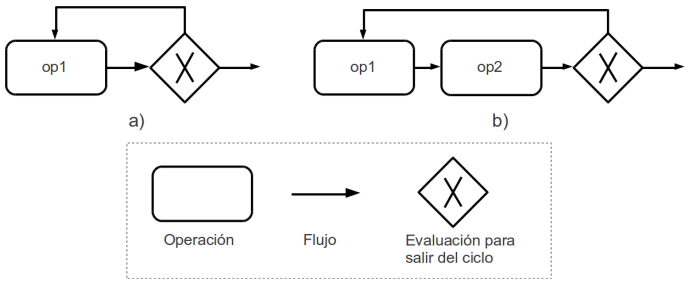

Figura 3.8: Representación gráfica del patrón LOOP
Fuente: Elaboración propia

Como se puede apreciar en la Tabla 3.6, los criterios para agregar la calidad de ensambles que siguen el patrón LOOP son incluso más variados que con el patrón XOR. Las fórmulas 10, 11, 12 y 20 se consideran dependientes de la arquitectura, las fórmulas 1, 2, 3, 4, 5, 6, 7, 8, 9, 13, 14, 15, 16, 17, 18, 19, 21, 22, 23, 24, 25 y 26 son, además, dependientes del uso, ya que consideran factores de uso, como el número de iteraciones (l) o la probabilidad de seguir iterando (p). La fórmula 7, además de ser dependiente de la arquitectura y el uso, es derivada.

Algunas de las fórmulas presentadas en la Tabla 3.6, aunque expresadas de forma diferente, son equivalentes, por ejemplo:

$$Rel(o) \cdot (1-p)^{-1} = \frac{Rel(o)}{1-p}$$

3.4.5. Multiselector

También conocido como patrón OR. Describe una estructura donde un conjunto de operaciones pueden ser ejecutadas, no necesariamente una, como en el caso del patrón XOR, ni tampoco todas como en el caso del patrón AND, sino un subconjunto de las posibles opciones. El no ejecutar ninguna operación del conjunto, también es una posibilidad [31], por lo que las posibilidades pueden expresarse como $\mathcal{P}\{c_i | 1 \leq i \leq n\}$. El conjunto de operaciones seleccionadas será ejecutado en paralelo. La representación gráfica del patrón OR se muestra en la Figura 3.9.

En la Tabla 3.7 se puede observar cómo las fórmulas de agregación para el patrón OR combinan características tanto de las fórmulas para el patrón AND, como de las fórmulas para el patrón OR. Por ejemplo, las fórmulas del patrón OR usan la probabilidad de que un conjunto de componentes sea ejecutado, como factor de ponderación de la calidad agregada, al igual que con el patrón XOR; y la calidad de cada conjunto de componentes que se ejecutarán

Tabla 3.6: Fórmulas de agregación para el patrón LOOP
Fuente: Elaboración propia

Atributo de calidad	ID	Fórmula	Categoría	Fuentes
Confiabilidad	1	$l \cdot Rel(o)$	Dependiente de la arquitectura, dependiente del uso	[7]
	2	$\frac{(1-p) \cdot Rel(o)}{1 - p \cdot Rel(o)}$	Dependiente de la arquitectura, dependiente del uso	[10]
	3	$\frac{(1-p_2) \cdot Rel(o_1)}{1 - p_2 \cdot Rel(o_1) \cdot Rel(o_2)}$	Dependiente de la arquitectura, dependiente del uso	[10, 26]
	4	$\sum_{l=1}^{L} l \cdot Rel(o) \cdot p_l$	Dependiente de la arquitectura, dependiente del uso	[18]
	5	$Rel(o) \cdot (1-p)^{-1}$	Dependiente de la arquitectura, dependiente del uso	[13]
	6	$\prod_{i=1}^{n+1} Rel(o_i)^{Lavg(o_i)} \cdot \sum_{i=1}^{n} p(f_i) \cdot Rel(f_i)$	Dependiente de la arquitectura, dependiente del uso	[13]
MTTF	7	$\frac{1}{\lambda \cdot l}$	Dependiente de la arquitectura, dependiente del uso, derivada	[16]
Disponibilidad	8	$Av(o)^l$	Dependiente de la arquitectura, dependiente del uso	[7, 17]
"Uptime probability"	9	$Up(o)^l$	Dependiente de la arquitectura, dependiente del uso	[19]
Reputación	10	$Rep(o)$	Dependiente de la arquitectura	[20]
Fidelidad	11	$Fid(o)$	Dependiente de la arquitectura	[18]
"Throughput"	12	$Tp(o)$	Dependiente de la arquitectura	[19]
Tiempo de ejecución	13	$l \cdot Xt(o)$	Dependiente de la arquitectura, dependiente del uso	[19, 20]
	14	$Xt(o) \cdot (1-p)^{-1}$	Dependiente de la arquitectura, dependiente del uso	[13]
	15	$\sum_{i=1}^{n+1} Lavg(o_i) \cdot Xt(o_i)$	Dependiente de la arquitectura, dependiente del uso	[13]
Latencia	16	$l * Lat(o)$	Dependiente de la arquitectura, dependiente del uso	[7]
	17	$\sum_{l=1}^{L} l \cdot Lat(o) \cdot p_l$	Dependiente de la arquitectura, dependiente del uso	[18]
	18	$\frac{Lat(o)}{1-p}$	Dependiente de la arquitectura, dependiente del uso	[10]
	19	$\frac{Lat(o_1) + Lat(o_2) - (1-p_2) \cdot Lat(o_2)}{1-p_2}$	Dependiente de la arquitectura, dependiente del uso	[10]
Nivel de encriptación	20	$Cryp(o)$	Dependiente de la arquitectura	[19]
Costo	21	$l \cdot Cost(o)$	Dependiente de la arquitectura, dependiente del uso	[7, 19, 20]
	22	$Cost(o) \cdot (1-p)^{-1}$	Dependiente de la arquitectura, dependiente del uso	[13]
	23	$\sum_{l=1}^{L} l \cdot Cost(o) \cdot p_l$	Dependiente de la arquitectura, dependiente del uso	[18]
	24	$\sum_{i=1}^{n} Lavg(o_i) \cdot Cost(o_i)$	Dependiente de la arquitectura, dependiente del uso	[13]
	25	$\frac{Cost(o)}{1-p}$	Dependiente de la arquitectura, dependiente del uso	[10]
	26	$\frac{Cost(o_1) + Cost(o_2) - (1-p_2) \cdot Cost(o_2)}{1-p_2}$	Dependiente de la arquitectura, dependiente del uso	[10]

en paralelo se calcula sumando, multiplicando u obteniendo valores máximos y mínimos, de forma similar a las fórmulas para el patrón AND. De las fórmulas de la Tabla 3.7, las fórmulas

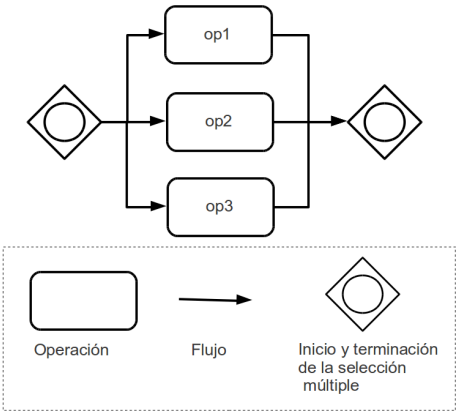

Figura 3.9: Representación gráfica del patrón OR
Fuente: Elaboración propia

3, 4, 5, 6, 7 y 9 son dependientes de la arquitectura; y las fórmulas 1, 2 y 8 son, además, dependientes del uso. Además de eso, la fórmula 1 es derivada.

3.4.6. Discriminador

También conocido como patrón de tolerancia a fallos [10]. Se da cuando dos o más operaciones que realizan la misma funcionalidad son ejecutadas al mismo tiempo, pero sólo la más rápida en responder será tomada en cuenta [15]. La representación más común del patrón discriminador es la que se muestra en la Figura 3.10 a), sin embargo, algunos autores consideran todas o un subconjunto de las representaciones de la Figura 3.10.

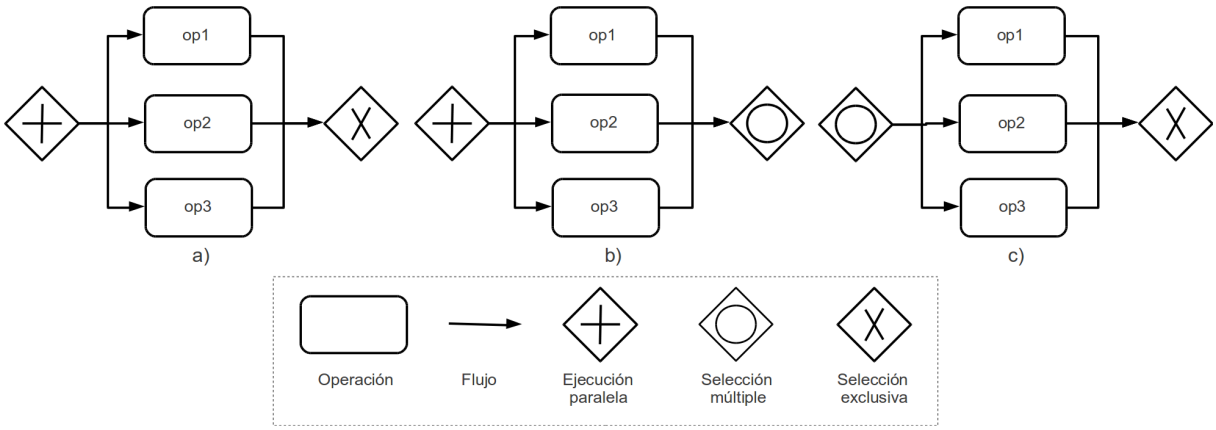

Figura 3.10: Representación gráfica del patrón discriminador
Fuente: Elaboración propia

La Figura 3.10 a) representa la ejecución de n operaciones en paralelo, de las cuales sólo una (generalmente la primera en responder) será tomada en cuenta para continuar con el flujo del ensamble. La Figura 3.10 b) representa la ejecución de n operaciones en paralelo, de las cuales m serán tomadas en cuenta para continuar con el flujo del ensamble; donde $1 \leq m \leq n$. La Figura 3.10 c) representa la ejecución de m operaciones en paralelo, de las cuales sólo una será tomada en cuenta para continuar con el flujo del ensamble.

Tabla 3.7: Fórmulas de agregación para el patrón OR
Fuente: Elaboración propia

Atributo de calidad	ID	Fórmula	Categoría	Fuentes
MTTF	1	$\frac{1}{\sum_{i=1}^{n} p_i \cdot \lambda(o_i)}$	Dependiente de la arquitectura, dependiente del uso, derivado	[16]
Disponibilidad	2	$\sum_{i=1}^{n} p_i \cdot Av(o_i)$	Dependiente de la arquitectura, dependiente del uso	[17]
"Uptime probability"	3	$max\left\{\prod_{y\in\mathbb{S}} y, \forall\mathbb{S} \in \mathcal{P}(\mathbb{T})\right\}$, $\mathbb{T} = \{Up(o_i) \mid 1 \leq i \leq n\}$	Dependiente de la arquitectura	[19]
Reputación	4	$min_{i=1}^{n}\{Rep(o_i)\}$	Dependiente de la arquitectura	[20]
"Throughput"	5	$min\{y \in \mathbb{S}, \forall\mathbb{S} \in \mathcal{P}(\mathbb{T})\}$, $\mathbb{T} = \{Tp(o_i) \mid 1 \leq i \leq n\}$	Dependiente de la arquitectura	[19]
Tiempo de ejecución	6	LB: $min\{y \in \mathbb{S}, \forall\mathbb{S} \in \mathcal{P}(\mathbb{T})\}$, UB: $max\{y \in \mathbb{S}, \forall\mathbb{S} \in \mathcal{P}(\mathbb{T})\}$, $\mathbb{T} = \{Xt(o_i) \mid 1 \leq i \leq n\}$	Dependiente de la arquitectura	[19]
Nivel de encriptación	7	$min\{y \in \mathbb{S}, \forall\mathbb{S} \in \mathcal{P}(\mathbb{T})\}$, $\mathbb{T} = \{Cryp(o_i) \mid 1 \leq i \leq n\}$	Dependiente de la arquitectura	[19]
Costo	8	$\frac{1}{n}\sum_{i=1}^{n} h_i \cdot Cost(o_i)$	Dependiente de la arquitectura, dependiente del uso	[20]
	9	LB: $min\left\{\sum_{y\in\mathbb{S}} y, \forall\mathbb{S} \in \mathcal{P}(\mathbb{T})\right\}$, UB: $max\left\{\sum_{y\in\mathbb{S}} y, \forall\mathbb{S} \in \mathcal{P}(\mathbb{T})\right\}$, $\mathbb{T} = \{Cost(o_i) \mid 1 \leq i \leq n\}$	Dependiente de la arquitectura	[19]

La Tabla 3.8 muestra ejemplos de fórmulas de agregación para el patrón discriminador. De las fórmulas que se encuentran en la Tabla 3.8, las fórmulas 1, 2, 3, 6, 7, 9, 10, 11, 12, 13 y 14 son dependientes de la arquitectura; las fórmulas 4, 8 y 15 son, además, dependientes del uso y la fórmula número 5 puede ser tanto dependiente del uso como derivada, ya que depende de la probabilidad de que la latencia, de la operación o_i, sea menor que la latencia de los demás componentes que se ejecutan en paralelo. La fórmula número 2 tiene la particularidad de que calcula la confiabilidad del ensamble con base en todos los posibles estados, pero sin tomar en cuenta la probabilidad de cada estado.

3.4.7. Componente DAG

El acrónimo DAG proviene de "Direct Aciclic Graph" y se trata de un patrón de composición que puede dividirse en un conjunto de rutas opcionales (XOR) y cada una puede

Tabla 3.8: Fórmulas de agregación para el patrón discrimidador
Fuente: Elaboración propia

Atributo de calidad	ID	Fórmula	Categoría	Fuentes
Confiabilidad	1	$1 - \prod_{i=1}^{n}(1 - Rel(o_i))$	Dependiente de la arquitectura	[18]
	2	$\sum_{i_1=0}^{1} ... \sum_{i_n=0}^{1} g(\sum_{j=1}^{J} s_{ij} - m) \cdot \prod_{j=1}^{J}((1 - s_{ij}) + (2s_{ij} - 1)Rel(c_{ij}))$	Dependiente de la arquitectura	[10]
"Uptime probability"	3	$\prod_{i=1}^{n} Up(o_i)$	Dependiente de la arquitectura	[19]
Reputación	4	$\frac{1}{n}\sum_{i=1}^{n} h_i \cdot Rep(o_i)$	Dependiente de la arquitectura, dependiente del uso	[20]
Fidelidad	5	$\sum_{i=1}^{n} t_i \cdot Fid(o_i)$	Dependiente de la arquitectura, dependiente del uso, derivada	[18]
"Throughput"	6	$min_{i=1}^{n}\{Tp(o_i)\}$	Dependiente de la arquitectura	[19]
	7	$min\{y \in \mathbb{S}, \forall \mathbb{S} \in \mathcal{P}(\mathbb{T})\}$, $\mathbb{T} = \{Tp(o_i) \mid 1 \leq i \leq n\}$	Dependiente de la arquitectura	[19]
Tiempo de ejecución	8	$\frac{1}{n}\sum_{i=1}^{n} h_i \cdot Xt(o_i)$	Dependiente de la arquitectura, dependiente del uso	[20]
	9	LB: $min\{y \in \mathbb{S}, \forall \mathbb{S} \in \mathcal{P}(\mathbb{T})\}$, UB: $max\{y \in \mathbb{S}, \forall \mathbb{S} \in \mathcal{P}(\mathbb{T})\}$, $\mathbb{T} = \{Xt(o_i) \mid 1 \leq i \leq n\}$	Dependiente de la arquitectura	[19]
	10	LB: $min_{i=1}^{n}\{Xt(o_i)\}$, UB: $max_{i=1}^{n}\{Xt(o_i)\}$	Dependiente de la arquitectura	[19]
Latencia	11	$min_{i=1}^{n}\{Lat(o_i)\}$	Dependiente de la arquitectura	[18, 10]
Nivel de encriptación	12	$min_{i=1}^{n}\{Cryp(o_i)\}$	Dependiente de la arquitectura	[19]
	13	$min\{y \in \mathbb{S}, \forall \mathbb{S} \in \mathcal{P}(\mathbb{T})\}$, $\mathbb{T} = \{Cryp(o_i) \mid 1 \leq i \leq n\}$	Dependiente de la arquitectura	[19]
Costo	14	$\sum_{i=1}^{n} Cost(o_i)$	Dependiente de la arquitectura	[10, 18, 19, 20]
	15	$\frac{1}{n}\sum_{i=1}^{n} h_i \cdot Cost(o_i)$	Dependiente de la arquitectura, Dependiente del uso	[19]

contener varias subrutas que se ejecutan en paralelo (AND) sin que alguna de estas rutas o subrutas entre en algún ciclo. Los componentes DAG son analizados empleando la noción de corrida (run). Una corrida es un subgrafo de un componente DAG, que puede ser interpretado como sus ejecuciones concurrentes, junto con la probabilidad de que esta corrida se ejecute [13].

La Figura 3.11 muestra dos representaciones del mismo componente DAG. La Figura 3.11 a) es la representación del ensamble tal como fue construido y la Figura 3.11 b) muestra cada una de las corridas del mismo ensamble. De acuerdo con la Figura 3.11 b), en cada ejecución del ensamble, se ejecutará exclusivamente una corrida, que consta de varios flujos de operaciones que se ejecutarán en paralelo. Para agregar la calidad de este tipo de ensambles,

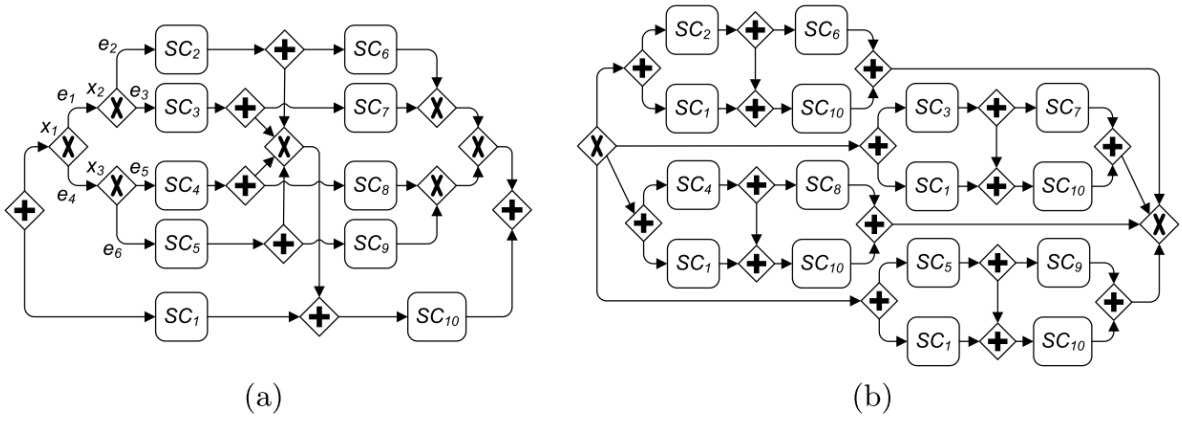

Figura 3.11: Representación gráfica del patrón DAG [13]

se han propuesto las fórmulas de la Tabla 3.9.

Además del patrón DAG se ha propuesto otro patrón con características similares, llamado *rutas paralelas entrelazadas* (RPE) [18]. La diferencia entre este patrón y el patrón DAG radica en que las corridas no necesariamente consisten en flujos paralelos, sino que pueden ser flujos secuenciales, sin embargo, es posible descomponerlo en forma similar para identificar cada una de las corridas del ensamble. Las fórmulas para agregar el patrón RPE se muestran en la Tabla 3.10. Como se puede ver en las Tablas 3.9 y 3.10, los atributos de calidad confiabilidad y costo, que son los únicos que se presentan en ambas tablas, comparten la misma fórmula para ambos patrones. Lo que confirma, aunque vagamente, que ambos patrones pueden agregarse en forma similar.

Tabla 3.9: Fórmulas de agregación para el patrón DAG
Fuente: Elaboración propia

Atributo de calidad	ID	Fórmula	Categoría	Fuentes
Confiabilidad	1	$\prod_{i=1}^{n} Rel(o_i)$	Dependiente de la arquitectura	[13]
Tiempo de ejecución	2	$CPA_{i=1}^{n}\{Xt(o_i)\}$	Dependiente de la arquitectura	[20]
Costo	3	$\sum_{i=1}^{n} Cost(o_i)$	Dependiente de la arquitectura	[13]

Las tres fórmulas que se presentan en la Tabla 3.9, entran en la categoría de dependiente de la arquitectura, entre ellas, la fórmula número 2, que tiene la particularidad de que sólo suma los tiempos de ejecución de los componentes que forman parte de la ruta crítica, es decir, de aquellos componentes que en conjunto forman el cuello de botella del ensamble y por lo tanto restringen el tiempo de ejecución mínimo del ensamble. De las fórmulas de la Tabla 3.10, la única que es dependiente del uso, es la número 2, ya que considera dos factores relacionados con el uso, la probabilidad de ejecución de cada operación (p_i) y la importancia de cada operación (w_i).

Tabla 3.10: Fórmulas de agregación para el patrón RPE
Fuente: Elaboración propia

Atributo de calidad	ID	Fórmula	Categoría	Fuentes
Confiabilidad	1	$\prod_{i=1}^{n} Rel(o_i)$	Dependiente de la arquitectura	[18]
Fidelidad	2	$\frac{1}{\sum_{i=1}^{n} p_i \cdot w_i} \cdot \sum_{i=1}^{n} p_i \cdot w_i \cdot Fid(o_i)$	Dependiente de la arquitectura, dependiente del uso	[18]
Latencia	3	$max_{i=1}^{n}\{Lat(o_i)\}$	Dependiente de la arquitectura	[18]
Costo	4	$\sum_{i=1}^{n} Cost(o_i)$	Dependiente de la arquitectura	[18]

3.4.8. Reducción Estocástica de Patrones de Flujo de Trabajo

En la mayoría de los casos, un ensamble de operaciones seguirá más de un patrón de flujo de trabajo. Por esa razón, para poder agregar la calidad de un ensamble de componentes de software, de acuerdo con los patrones de flujo de trabajo que lo definen, es necesario aplicar una técnica llamada reducción estocástica de flujos de trabajo, o stochastic workflow reduction (SWR). El SWR calcula la calidad agregada de un flujo de trabajo paso a paso. En cada paso, una regla de reducción es aplicada para reducir el flujo. Esto continúa hasta que sólo queda una tarea atómica en el flujo de trabajo. Dicha tarea contiene las métricas de calidad correspondientes al flujo de trabajo que está siendo analizado [38]. Las reglas de reducción consisten en la aplicación de fórmulas de agregación usando como criterio el atributo de calidad que se desea estimar y el patrón de flujo de trabajo que se va a reducir. Parte del ejemplo que se muestra a continuación fue tomado de [38]. La Figura 3.12 muestra el flujo de trabajo sobre el que vamos a aplicar SWR para estimar la calidad del ensamble.

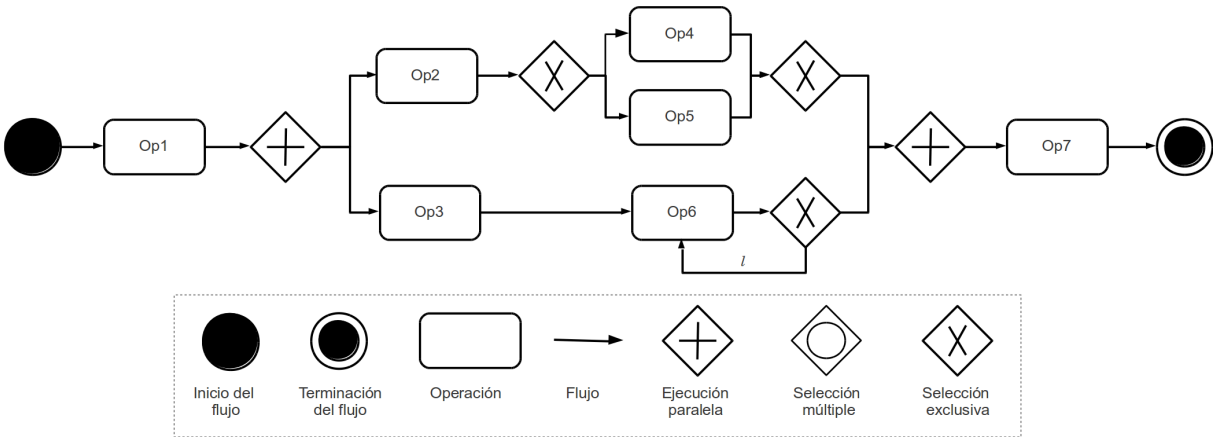

Figura 3.12: Ejemplo de flujo de trabajo con múltiples patrones de composición
Fuente: Elaboración propia

El flujo de la Figura 3.12 comienza con la ejecución secuencial de Op1. Seguido de esto Op2 y Op3 se ejecutan siguiendo el patrón AND, lo que desencadena dos coridas paralelas,

dentro de la corrida de Op2, hay una selección exclusiva entre los componentes Op4 y Op5, por otro lado, dentro de la corrida de Op3 hay un LOOP de l iteraciones de Op6. Ambas corridas paralelas se sincronizan para finalizar con Op7. Como se puede ver, hay patrones de flujo de trabajo anidados unos dentro de otros y el SWR deberá comenzar desde los flujos internos, ya que son los más simples. Los primeros dos candidatos son la selección exclusiva de Op4 y Op5 y el LOOP de Op6.

En este ejemplo vamos a estimar el costo del ensamble, empezando con la reducción de Op4 y Op5, cuyos costos y probabilidades de ejecución se muestran en la Tabla 3.11. En [38], se usa la siguiente fórmula para la reducción del patrón XOR:

$$Cost(a) = \sum_{i=1}^{n} p_i \cdot Cost(o_i)$$

$$Cost(a) = 300 \cdot 0{,}7 + 500 \cdot 0{,}3$$

$$Cost(a) = 210 + 150$$

$$Cost(a) = 360$$

Tabla 3.11: Costo y probabilidad de ejecución de Op4 y Op5
Fuente: Elaboración propia

Operación	Costo	Probabilidad
Op4	$300.00	70 %
Op4	$500.00	30 %

Una vez realizada la reducción de Op4 y Op5, pasamos a la reducción de Op6 que se ejecutará en varias iteraciones. Supongamos que el costo de Op6 es de $30.00 y que esta operación, se ejecutará 20 veces. La fórmula de agregación para la reducción del patrón LOOP será la siguiente:

$$Cost(a) = l \cdot Cost(o)$$

$$Cost(a) = 20 \cdot 30$$

$$Cost(a) = 600$$

Una vez realizado el SWR de los flujos más simples, el grafo se simplifica, haciendo parecer que lo que antes era un ensamble de operaciones, ahora es un componente atómico, como se muestra en la Figura 3.13

Ahora es posible hacer una reducción secuencial de cada una de las corridas paralelas, para luego hacer la reducción del flujo paralelo y finalmente la reducción del flujo completo, con lo que obtendremos una estimación de la calidad de todo el ensamble. Las Figuras 3.14, 3.15 y 3.16 muestran el resto del proceso de reducción. La reducción de ambos patrones,

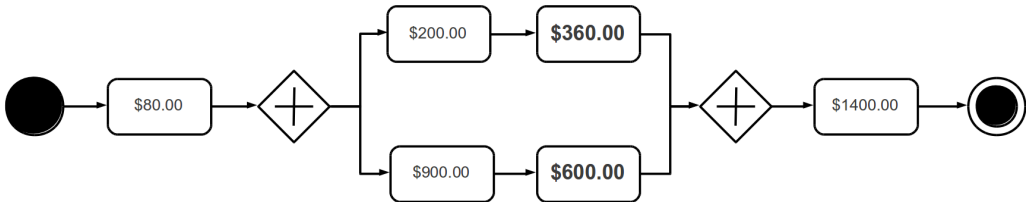

Figura 3.13: SWR de los patrones XOR y LOOP
Fuente: Elaboración propia

tanto secuencia como AND, se realiza aplicando la fórmula que se muestra a continuación:

$$Cost(a) = \sum_{i=1}^{n} Cost(o_i)$$

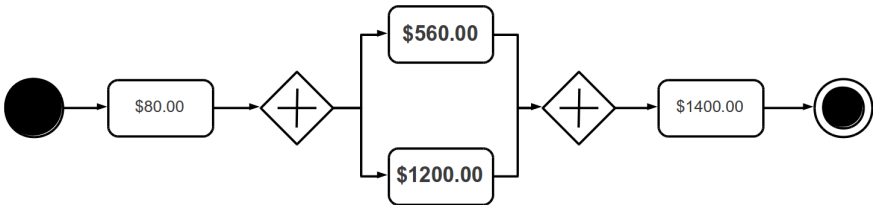

Figura 3.14: SWR del patrón secuencia en cada una de las corridas paralelas
Fuente: Elaboración propia

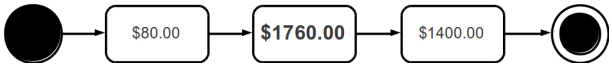

Figura 3.15: SWR del patrón AND
Fuente: Elaboración propia

Figura 3.16: SWR de la calidad del patrón secuencia
Fuente: Elaboración propia

Entre los trabajos donde se proponen fórmulas de agregación basadas en patrones de flujo de trabajo, existen algunos que extienden las fórmulas de agregación, agregando nuevas variables a las fórmulas que permiten adaptarlas a la arquitectura del software. En las próximas subsecciones se hablará de los trabajos que proponen fórmulas de agregación basadas en patrones de flujos de trabajo, y que están pensadas para software con tolerancia a fallos y Líneas de Producto de software.

3.4.9. Fórmulas de Agregación Considerando la Tolerancia a Fallos

En esta sección, se hizo mención de un patrón de flujo de trabajo que promueve la tolerancia a fallos, llamado discriminador. Como se explicó, este patrón consiste en varias operaciones

que se ejecutan en paralelo, de las cuales sólo un subconjunto deberá finalizar exitosamente su ejecución para continuar con el flujo del ensamble. A pesar de eso, las operaciones que no serán tomadas en cuenta, también deben ser ejecutadas, cosa que no pasa con el modelo propuesto en [31], donde los ensambles de software cuentan con un conjunto de tareas requeridas (I) y por cada tarea ($i \in I$) existe un conjunto de operaciones que son candidatas a realizarla (J_i). Es importante aclarar, que cada tarea será realizada exclusivamente por un servicio. El modelo propuesto en [31] se encuentra dentro del contexto de los servicios web, por lo que las operaciones son en realidad servicios disponibles en la World Wide Web. Debido a que este modelo no lleva a cabo la ejecución de un servicio a menos que sea requerido, las fórmulas de agregación propuestas incluyen una variable booleana que determina si un servicio ($j \in J$) deberá ser o no ejecutado. Dicha variable se expresa como (x_{ij}). Fuera de esta característica las fórmulas de agregación propuestas en [31] también toman como base el patrón de flujo de trabajo utilizado para el ensamble de los servicios. A continuación se muestra una de las fórmulas de [31] para la agregación de la confiabilidad en un ensamble que sigue el patrón XOR. Para interpretar esta fórmula tome como referencia la Tabla 3.13.

$$\sum_{l \in L} p_l \prod_{i \in IW_l} \sum_{j \in J_i} r_{ij} x_{ij}$$

La fórmula en apariencia es más compleja que las que se presentaron anteriormente, en parte, por la presencia de dos operadores de suma ($\sum$) y uno de producto ($\prod$). El primer operador de suma ($\sum_{l \in L}$), aplica para cada corrida del ensamble XOR (l), mientras que el operador de producto ($\prod_{i \in IW_l}$) aplica para cada una de las tareas (i) comprendidas dentro de l, es decir que esta fórmula asume que cada corrida del ensamble XOR puede comprender un conjunto de tareas, en otras palabras, un flujo anidado. Si asumimos que este flujo será convertido en una tarea atómica a través de la técnica SWR, el operador producto y la noción de corridas ya no serían requeridos dentro de la fórmula, por lo que podemos redefinir la fórmula de la siguiente manera sin que esta pierda su esencia.

$$\sum_{i \in IW} p_i \sum_{j \in J_i} r_{ij} x_{ij}$$

El segundo operador $\sum$ indica que se sumará el producto de la confiabilidad de todas las operaciones que realizan la tarea i por el factor que determina si una operación será ejecutada (x_{ij}). En [31] se indica que sólo será ejecutada una operación por cada tarea, por lo que, para todas las demás operaciones x_{ij} tomará el valor 0. Con base en este criterio, podemos concluir lo siguiente:

$$\sum_{j \in J_i} r_{ij} x_{ij} = r_i : r_{ij} | x_{ij} = 1$$

Lo que hemos hecho es sustituir esta suma por la expresión r_i, bajo el supuesto de que la suma de $r_{ij} \cdot x_{ij}$ es igual a r_{ij} donde $x_{ij} = 1$. Ahora hemos acotado un poco más la fórmula, lo que nos permite expresarla de la siguiente manera:

$$\sum_{i \in IW} p_i r_i$$

Con esto hemos comprobado que la fórmula que estamos analizando es equivalente a la primera fórmula de la Tabla 3.5 (fórmula 3.5.1), que como podemos ver, se trata de una suma ponderada con base en la probabilidad de que la tarea i sea ejecutada, lo que coloca a esta fórmula dentro de la clasificación de dependientes del uso, además de ser una fórmula dependiente de la arquitectura debido al factor de tolerancia a fallos (x_{ij}) y el hecho de basarse en un patrón de flujo de trabajo. Con esto, el lector puede apreciar más fácilmente la similitud entre esta y las demás fórmulas que se han presentado a lo largo de esta sección; lo que nos lleva a concluir que, lo que, en realidad, distingue a esta fórmula, de la fórmula 3.5.1, es, el factor de tolerancia a fallos x_{ij}. Lo que se acaba de demostrar con la fórmula de ejemplo, también ocurre con el resto de las fórmulas de agregación propuestas en [31], las cuales se muestran en la Tabla 3.12.

Las fórmulas de la Tabla 3.12 conservan la forma como fueron definidas en [31]. En la Tabla 3.13 se define la lista de expresiones aplicables a estas fórmulas, como apoyo para su interpretación. El formato y las expresiones de las fórmulas son notoriamente distintas a las que se usaron en las Tablas 3.3 a 3.10, sin embargo, las reglas para la agregación de la calidad son muy similares, como ya se ha demostrado.

En conclusión, las fórmulas de agregación propuestas en [31] también se basan en patrones de flujo de trabajo, pueden ser sometidas al SWR y pueden ser clasificadas según las categorías de [11].

En [17] y [16] también se proponen fórmulas de agregación considerando la tolerancia a fallos. En [17], la disponibilidad se evalúa como un atributo de calidad derivado de la tasa de fallos λ y la tasa de reparación μ, y se calcula con base en la siguiente fórmula.

$$\frac{\mu_i}{\lambda_i + \mu_i}$$

Como la técnica de [17] se basa en la arquitectura, toma en cuenta la cantidad de componentes de respaldo. Suponga que Av_R es la disponibilidad de una tarea redundante y b es el número de componentes de respaldo, con base en esta información, la disponibilidad se puede calcular a través de la siguiente fórmula.

$$Av_R = 1 - (1 - \frac{\mu}{\lambda + \mu})^b$$

Una vez calculada la disponibilidad de un componente redundante, ésta se puede agregar a la disponibilidad del resto del ensamble utilizando las fórmulas de agregación presentadas en las Tablas 3.3 a 3.10.

En [16] se presenta una propuesta similar, pero ahora el atributo de calidad a agregar es MTTF. La fórmula que se muestra a continuación sólo es válida para tareas que cuentan con un componente base y un componente de respaldo.

$$MTTF_R = \frac{3\lambda + \mu}{2\lambda^2}$$

La fórmula anterior es una fórmula derivada, ya que calcula el MTTF con base en la tasa de fallos y la tasa de reparación. También es una fórmula dependiente de la arquitectura, ya que toma en consideración la presencia de un componente de respaldo.

3.4.10. Fórmulas de Agregación para Líneas de Productos de Software

Las líneas de producto de software, o Software Product Line (SPLs) son utilizadas para crear productos de software adaptables, mediante la administración y ensamble de recursos reutilizables [30]. Los ensambles de componentes de SPLs se describen mediante modelos de características, como el que se muestra en la Figura 3.17. El término característica, puede referirse a uno de varios aspectos del software, como un componente, una operación, e incluso una variable. La notación de los modelos de características, permite conocer qué partes del ensamble de software son obligatorias u opcionales, también permite conocer la manera como serán utilizadas las partes del ensamble, por ejemplo, en una ejecución puede utilizarse exclusivamente una característica (Xor-Group), un subconjunto de las características (Or-Group) o todas las características (And-Group).

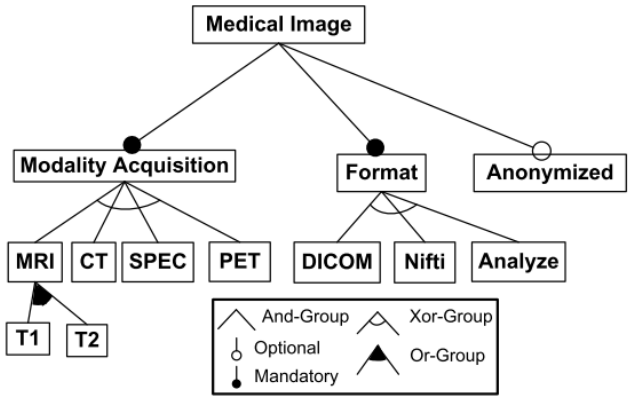

Figura 3.17: Ejemplo de modelo de características [1]

Las fórmulas de agregación propuestas en [27], toman en cuenta la notación de los modelos de características en sus fórmulas de agregación. El símbolo $\bullet$ de la fórmula que se muestra a continuación, hace referencia a aquellas características obligatorias del ensamble, mientras que el símbolo $\circ$ hace referencia a las características opcionales. La ausencia del símbolo $\circ$ en esta fórmula, significa que la disponibilidad se calcula multiplicando la disponibilidad de los componentes que están obligados a formar parte del ensamble, y por lo tanto, los componentes opcionales no son tomados en cuenta para calcular la disponibilidad.

$$\prod_{i=1}^{n} q_{av}(f_i) : \forall f_i \in f^{\bullet}$$

En la fórmula anterior, también se puede observar el uso del operador $\prod$ para agregar la disponibilidad, al igual que con las fórmulas de disponibilidad que se muestran en las Tablas 3.3 y 3.4. De acuerdo con la clasificación de [27], esta fórmula es adecuada para calcular la máxima disponibilidad de ensambles secuenciales y en paralelo. En consecuencia, estas fórmulas también se basan en patrones de flujo de trabajo y su valor agregado consiste en tomar en cuenta la variabilidad de los modelos de características, lo que las hace dependientes de la arquitectura. Las fórmulas propuestas en [27] se muestran en la Tabla 3.15 y el glosario de expresiones aplicable a estas fórmulas se muestra en la Tabla 3.14.

Tabla 3.15: Fórmulas de agregación propuestas en [27]
Fuente: Elaboración propia

Patrón	Atributo	ID	Fórmula	Categoría
Secuencia	Disponibilidad	1	LB: $\prod_{i=1}^{n} q_{av}(f_i) : \forall f_i \in f^{\bullet} \vee f^{\circ}$, UB: $\prod_{i=1}^{n} q_{av}(f_i) : \forall f_i \in f^{\bullet}$	Dependiente de la arquitectura
		2	LB: $min\left(\prod_{f_i \in F_{sub}} q_{av}(f_i) : \forall F_{sub} \in F_{C_k^n}\right)$, UB: $max\left(\prod_{f_i \in F_{sub}} q_{av}(f_i) : \forall F_{sub} \in F^{(*)}_{C_k^n} | F^{(**)}_{C_k^n}\right)$	Dependiente de la arquitectura
	"Throughput"	3	LB: $min(q_{tp}(f_i) : \forall f_i \in f^{\bullet} \vee f^{\circ})$ UB: $max(q_{tp}(f_i) : \forall f_i \in f^{\bullet} \vee f^{\circ})$	Dependiente de la arquitectura
		4	LB: $min(min(q_{tp}(f_i) : \forall F_{sub} \in F_{C_k^n}))$, UB: $max(max(q_{tp}(f_i) : \forall F_{sub} \in F^{(*)}_{C_k^n} | F^{(**)}_{C_k^n}))$	Dependiente de la arquitectura
	Latencia	5	LB: $\sum_{i=1}^{n} q_{rt}(f_i) : \forall f_i \in f^{\bullet}$, UB: $\sum_{i=1}^{n} q_{rt}(f_i) : \forall f_i \in f^{\bullet} \vee f^{\circ}$	Dependiente de la arquitectura
		6	LB: $min\left(\sum_{f_i \in F_{sub}} q_{rt}(f_i) : \forall F_{sub} \in F_{C_k^n}\right)$, UB: $max\left(\sum_{f_i \in F_{sub}} q_{rt}(f_i) : \forall F_{sub} \in F^{(*)}_{C_k^n} | F^{(**)}_{C_k^n}\right)$	Dependiente de la arquitectura
	Costo	7	LB: $\sum_{i=1}^{n} q_{pr}(f_i) : \forall f_i \in f^{\bullet}$, UB: $\sum_{i=1}^{n} q_{pr}(f_i) : \forall f_i \in f^{\bullet} \vee f^{\circ}$	Dependiente de la arquitectura
		8	LB: $min\left(\sum_{f_i \in F_{sub}} q_{pr}(f_i) : \forall F_{sub} \in F_{C_k^n}\right)$, UB: $max\left(\sum_{f_i \in F_{sub}} q_{pr}(f_i) : \forall F_{sub} \in F^{(*)}_{C_k^n} | F^{(**)}_{C_k^n}\right)$	Dependiente de la arquitectura
AND-DISC	Disponibilidad	9	LB: $\prod_{i=1}^{n} q_{av}(f_i) : \forall f_i \in f^{\bullet}$, UB: $\prod_{i=1}^{n} q_{av}(f_i) : \forall f_i \in f^{\bullet} \vee f^{\circ}$	Dependiente de la arquitectura
		10	LB: $min\left(\prod_{f_i \in F_{sub}} q_{av}(f_i) : \forall F_{sub} \in F_{C_k^n}\right)$, UB: $max\left(\prod_{f_i \in F_{sub}} q_{av}(f_i) : \forall F_{sub} \in F^{(*)}_{C_k^n} | F^{(**)}_{C_k^n}\right)$	Dependiente de la arquitectura
	"Throughput"	11	LB: $min(q_{tp}(f_i) : \forall f_i \in f^{\bullet} \vee f^{\circ})$ UB: $min(q_{tp}(f_i) : \forall f_i \in f^{\bullet} \vee f^{\circ})$	Dependiente de la arquitectura
		12	LB: $min(min(q_{tp}(f_i) : \forall F_{sub} \in F_{C_k^n}))$, UB: $min(min(q_{tp}(f_i) : \forall F_{sub} \in F^{(*)}_{C_k^n} | F^{(**)}_{C_k^n}))$	Dependiente de la arquitectura
	Latencia	13	LB: $max(q_{rt}(f_i) : \forall f_i \in f^{\bullet})$, UB: $max(q_{rt}(f_i) : \forall f_i \in f^{\bullet} \vee f^{\circ})$	Dependiente de la arquitectura

Continúa en la siguiente página

Tabla 3.15 – continuacón

Patrón	Atributo	ID	Fórmula	Categoría
		14	LB: $min(min(q_{rt}(f_i) : \forall F_{sub} \in F_{C_k^n}))$, UB: $max(max(q_{rt}(f_i) : \forall F_{sub} \in F_{C_k^n}^{(*)} \vert F_{C_k^n}^{(**)}))$	Dependiente de la arquitectura
	Costo	15	LB: $\sum_{i=1}^{n} q_{pr}(f_i) : \forall f_i \in f^{\bullet}$, UB: $\sum_{i=1}^{n} q_{pr}(f_i) : \forall f_i \in f^{\bullet} \vee f^{\circ}$	Dependiente de la arquitectura
		16	LB: $min\left(\sum_{f_i \in F_{sub}} q_{pr}(f_i) : \forall F_{sub} \in F_{C_k^n}\right)$, UB: $max\left(\sum_{f_i \in F_{sub}} q_{pr}(f_i) : \forall F_{sub} \in F_{C_k^n}^{(*)} \vert F_{C_k^n}^{(**)}\right)$	Dependiente de la arquitectura
OR-DISC	Disponibilidad	17	LB: $min\left(\prod_{f_i \in F_{sub}} q_{av}(f_i) : \forall F_{sub} \in F_{C_m^n}\right)$, UB: $max\left(\prod_{f_i \in F_{sub}} q_{av}(f_i) : \forall F_{sub} \in F_{C_m^n}\right)$	Dependiente de la arquitectura
		18	LB: $min\left(\prod_{f_i \in F_{sub}} q_{av}(f_i) : \forall F_{sub} \in F_{C_k^n}\right)$, UB: $max\left(\prod_{f_i \in F_{sub}} q_{av}(f_i) : \forall F_{sub} \in F_{C_k^n}^{(*)} \vert F_{C_k^n}^{(**)}\right)$	Dependiente de la arquitectura
	"Throughput"	19	LB: $min(q_{tp}(f_i) : \forall f_i \in f^{\bullet} \vee f^{\circ})$, UB: $max(min(q_{tp}(f_i) : f_i \in F_{sub}, \forall F_{sub} \in F_{C_m^n}))$	Dependiente de la arquitectura
		20	LB: $min(min(q_{tp}(f_i) : \forall F_{sub} \in F_{C_k^n}))$, UB: $max(max(q_{tp}(f_i) : \forall F_{sub} \in F_{C_k^n}^{(*)} \vert F_{C_k^n}^{(**)}))$	Dependiente de la arquitectura
	Latencia	21	LB: $min(max(q_{rt}(f_i) : f_i \in F_{sub}, \forall F_{sub} \in F_{C_m^n}))$, UB: $max(q_{rt}(f_i) : \forall f_i \in f^{\bullet} \vee f^{\circ})$	Dependiente de la arquitectura
		22	LB: $min(max(q_{rt}(f_i) : \forall F_{sub} \in F_{C_k^n}))$, UB: $max(max(q_{rt}(f_i) : \forall F_{sub} \in F_{C_k^n}^{(*)} \vert F_{C_k^n}^{(**)}))$	Dependiente de la arquitectura
	Costo	23	LB: $min\left(\sum_{f_i \in F_{sub}} q_{pr}(f_i) : \forall F_{sub} \in F_{C_m^n}\right)$, UB: $max\left(\sum_{f_i \in F_{sub}} q_{pr}(f_i) : \forall F_{sub} \in F_{C_m^n}\right)$	Dependiente de la arquitectura
		24	LB: $min\left(\sum_{f_i \in F_{sub}} q_{pr}(f_i) : \forall F_{sub} \in F_{C_k^n}\right)$, UB: $max\left(\sum_{f_i \in F_{sub}} q_{pr}(f_i) : \forall F_{sub} \in F_{C_k^n}^{(*)} \vert F_{C_k^n}^{(**)}\right)$	Dependiente de la arquitectura
AND-XOR	Disponibilidad	25	LB: $\prod_{i=1}^{n} q_{av}(f_i) : \forall f_i \in f^{\bullet}$, UB: $\prod_{i=1}^{n} q_{av}(f_i) : \forall f_i \in f^{\bullet} \vee f^{\circ}$	Dependiente de la arquitectura
		26	LB: $min\left(\prod_{f_i \in F_{sub}} q_{av}(f_i) : \forall F_{sub} \in F_{C_k^n}\right)$, UB: $max\left(\prod_{f_i \in F_{sub}} q_{av}(f_i) : \forall F_{sub} \in F_{C_k^n}^{(*)} \vert F_{C_k^n}^{(**)}\right)$	Dependiente de la arquitectura

Continúa en la siguiente página

Tabla 3.15 – continuacón

Patrón	Atributo	ID	Fórmula	Categoría
	"Throughput"	27	LB: $min(q_{tp}(f_i) : \forall f_i \in f^{\bullet} \vee f^{\circ})$ UB: $max(q_{tp}(f_i) : \forall f_i \in f^{\bullet} \vee f^{\circ})$	Dependiente de la arquitectura
		28	LB: $min(min(q_{tp}(f_i) : \forall F_{sub} \in F_{C_k^n}))$, UB: $max(max(q_{tp}(f_i) : \forall F_{sub} \in F^{(*)}_{C_k^n} \vert F^{(**)}_{C_k^n}))$	Dependiente de la arquitectura
	Latencia	29	LB: $max(q_{rt}(f_i) : \forall f_i \in f^{\bullet})$, UB: $max(q_{rt}(f_i) : \forall f_i \in f^{\bullet} \vee f^{\circ})$	Dependiente de la arquitectura
		30	LB: $min(min(q_{rt}(f_i) : \forall F_{sub} \in F_{C_k^n}))$, UB: $max(max(q_{rt}(f_i) : \forall F_{sub} \in F^{(*)}_{C_k^n} \vert F^{(**)}_{C_k^n}))$	Dependiente de la arquitectura
	Costo	31	LB: $\sum_{i=1}^{n} q_{pr}(f_i) : \forall f_i \in f^{\bullet}$, UB: $\sum_{i=1}^{n} q_{pr}(f_i) : \forall f_i \in f^{\bullet} \vee f^{\circ}$	Dependiente de la arquitectura
		32	LB: $min\left(\sum_{f_i \in F_{sub}} q_{pr}(f_i) : \forall F_{sub} \in F_{C_k^n}\right)$, UB: $max\left(\sum_{f_i \in F_{sub}} q_{pr}(f_i) : \forall F_{sub} \in F^{(*)}_{C_k^n} \vert F^{(**)}_{C_k^n}\right)$	Dependiente de la arquitectura
OR-XOR	Disponibilidad	33	LB: $min\left(\prod_{f_i \in F_{sub}} q_{av}(f_i) : \forall F_{sub} \in F_{C_m^n}\right)$, UB: $max\left(\prod_{f_i \in F_{sub}} q_{av}(f_i) : \forall F_{sub} \in F_{C_m^n}\right)$	Dependiente de la arquitectura
		34	LB: $min\left(\prod_{f_i \in F_{sub}} q_{av}(f_i) : \forall F_{sub} \in F_{C_k^n}\right)$, UB: $max\left(\prod_{f_i \in F_{sub}} q_{av}(f_i) : \forall F_{sub} \in F^{(*)}_{C_k^n} \vert F^{(**)}_{C_k^n}\right)$	Dependiente de la arquitectura
	"Throughput"	35	LB: $min(q_{tp}(f_i) : \forall f_i \in f^{\bullet} \vee f^{\circ})$ UB: $max(q_{tp}(f_i) : \forall f_i \in f^{\bullet} \vee f^{\circ})$	Dependiente de la arquitectura
		36	LB: $min(min(q_{tp}(f_i) : \forall F_{sub} \in F_{C_k^n}))$, UB: $max(max(q_{tp}(f_i) : \forall F_{sub} \in F^{(*)}_{C_k^n} \vert F^{(**)}_{C_k^n}))$	Dependiente de la arquitectura
	Latencia	37	LB: $min(q_{rt}(f_i) : \forall f_i \in f^{\bullet} \vee f^{\circ})$, UB: $max(q_{rt}(f_i) : \forall f_i \in f^{\bullet} \vee f^{\circ})$	Dependiente de la arquitectura
		38	LB: $min(min(q_{rt}(f_i) : \forall F_{sub} \in F_{C_k^n}))$, UB: $max(max(q_{rt}(f_i) : \forall F_{sub} \in F^{(*)}_{C_k^n} \vert F^{(**)}_{C_k^n}))$	Dependiente de la arquitectura
	Costo	39	LB: $min\left(\sum_{f_i \in F_{sub}} q_{pr}(f_i) : \forall F_{sub} \in F_{C_m^n}\right)$, UB: $max\left(\sum_{f_i \in F_{sub}} q_{pr}(f_i) : \forall F_{sub} \in F_{C_m^n}\right)$	Dependiente de la arquitectura
		40	LB: $min\left(\sum_{f_i \in F_{sub}} q_{pr}(f_i) : \forall F_{sub} \in F_{C_k^n}\right)$, UB: $max\left(\sum_{f_i \in F_{sub}} q_{pr}(f_i) : \forall F_{sub} \in F^{(*)}_{C_k^n} \vert F^{(**)}_{C_k^n}\right)$	Dependiente de la arquitectura
Ciclo arbitrario	Disponibilidad	41	LB: $q_{av}(f_i)^c : \forall f_i \in f^{\bullet} \vee f^{\circ}$, UB: $q_{av}(f_i)^c : \forall f_i \in f^{\bullet} \vee f^{\circ}$	Dependiente del uso, Dependiente de la arquitectura

Continúa en la siguiente página

Tabla 3.15 – continuacón

Patrón	Atributo	ID	Fórmula	Categoría
	"Throughput"	42	LB: $q_{tp}(f_i) : f_i \in f^{\bullet} \vee f^{\circ}$ UB: $q_{tp}(f_i) : f_i \in f^{\bullet} \vee f^{\circ}$	Dependiente de la arquitectura
	Latencia	43	LB: $cq_{rt}(f_i) : f_i \in f^{\bullet} \vee f^{\circ}$, UB: $cq_{rt}(f_i) : f_i \in f^{\bullet} \vee f^{\circ}$	Dependiente del uso, Dependiente de la arquitectura
	Costo	44	LB: $cq_{pr}(f_i) : f_i \in f^{\bullet} \vee f^{\circ}$, UB: $cq_{pr}(f_i) : f_i \in f^{\bullet} \vee f^{\circ}$	Dependiente del uso, Dependiente de la arquitectura
AND-AND	Disponibilidad	45	LB: $\prod_{i=1}^{n} q_{av}(f_i) : \forall f_i \in f^{\bullet}$, UB: $\prod_{i=1}^{n} q_{av}(f_i) : \forall f_i \in f^{\bullet} \vee f^{\circ}$	Dependiente de la arquitectura
		46	LB: $min\left(\prod_{f_i \in F_{sub}} q_{av}(f_i) : \forall F_{sub} \in F_{C_k^n}\right)$, UB: $max\left(\prod_{f_i \in F_{sub}} q_{av}(f_i) : \forall F_{sub} \in F^{(*)}_{C_k^n} \vert F^{(**)}_{C_k^n}\right)$	Dependiente de la arquitectura
	"Throughput"	47	LB: $min(q_{tp}(f_i) : \forall f_i \in f^{\bullet} \vee f^{\circ})$ UB: $min(q_{tp}(f_i) : \forall f_i \in f^{\bullet} \vee f^{\circ})$	Dependiente de la arquitectura
		48	LB: $min(min(q_{tp}(f_i) : \forall F_{sub} \in F_{C_k^n}))$, UB: $min(min(q_{tp}(f_i) : \forall F_{sub} \in F^{(*)}_{C_k^n} \vert F^{(**)}_{C_k^n}))$	Dependiente de la arquitectura
	Latencia	49	LB: $max(q_{rt}(f_i) : \forall f_i \in f^{\bullet})$, UB: $max(q_{rt}(f_i) : \forall f_i \in f^{\bullet} \vee f^{\circ})$	Dependiente de la arquitectura
		50	LB: $min(max(q_{rt}(f_i) : \forall F_{sub} \in F_{C_k^n}))$, UB: $max(max(q_{rt}(f_i) : \forall F_{sub} \in F^{(*)}_{C_k^n} \vert F^{(**)}_{C_k^n}))$	Dependiente de la arquitectura
	Costo	51	LB: $\sum_{i=1}^{n} q_{pr}(f_i) : \forall f_i \in f^{\bullet}$, UB: $\sum_{i=1}^{n} q_{pr}(f_i) : \forall f_i \in f^{\bullet} \vee f^{\circ}$	Dependiente de la arquitectura
		52	LB: $min\left(\sum_{f_i \in F_{sub}} q_{pr}(f_i) : \forall F_{sub} \in F_{C_k^n}\right)$, UB: $max\left(\sum_{f_i \in F_{sub}} q_{pr}(f_i) : \forall F_{sub} \in F^{(*)}_{C_k^n} \vert F^{(**)}_{C_k^n}\right)$	Dependiente de la arquitectura
XOR-XOR	Disponibilidad	53	LB: $min(q_{av}(f_i) : \forall f_i \in f^{\bullet} \vee f^{\circ})$, UB: $max(q_{av}(f_i) : \forall f_i \in f^{\bullet} \vee f^{\circ})$	Dependiente de la arquitectura
		54	LB: $min\left(\prod_{f_i \in F_{sub}} q_{av}(f_i) : \forall F_{sub} \in F_{C_k^n}\right)$, UB: $max\left(\prod_{f_i \in F_{sub}} q_{av}(f_i) : \forall F_{sub} \in F^{(*)}_{C_k^n} \vert F^{(**)}_{C_k^n}\right)$	Dependiente de la arquitectura
	"Throughput"	55	LB: $min(q_{tp}(f_i) : \forall f_i \in f^{\bullet} \vee f^{\circ})$ UB: $max(q_{tp}(f_i) : \forall f_i \in f^{\bullet} \vee f^{\circ})$	Dependiente de la arquitectura
		56	LB: $min(min(q_{tp}(f_i) : \forall F_{sub} \in F_{C_k^n}))$, UB: $max(max(q_{tp}(f_i) : \forall F_{sub} \in F^{(*)}_{C_k^n} \vert F^{(**)}_{C_k^n}))$	Dependiente de la arquitectura
	Latencia	57	LB: $min(q_{rt}(f_i) : \forall f_i \in f^{\bullet} \vee f^{\circ})$, UB: $max(q_{rt}(f_i) : \forall f_i \in f^{\bullet} \vee f^{\circ})$	Dependiente de la arquitectura

Continúa en la siguiente página

Tabla 3.15 – continuacón

Patrón	Atributo	ID	Fórmula	Categoría
		58	LB: $min(max(q_{rt}(f_i) : \forall F_{sub} \in F_{C_k^n}))$, UB: $max(max(q_{rt}(f_i) : \forall F_{sub} \in F^{(*)}_{C_k^n} \vert F^{(**)}_{C_k^n}))$	Dependiente de la arquitectura
	Costo	59	LB: $min(q_{pr}(f_i) : f_i \in f^{\bullet} \vee f^{\circ})$, UB: $max(q_{pr}(f_i) : f_i \in f^{\bullet} \vee f^{\circ})$	Dependiente de la arquitectura
		60	LB: $min\left(\sum_{f_i \in F_{sub}} q_{pr}(f_i) : \forall F_{sub} \in F_{C_k^n}\right)$, UB: $max\left(\sum_{f_i \in F_{sub}} q_{pr}(f_i) : \forall F_{sub} \in F^{(*)}_{C_k^n} \vert F^{(**)}_{C_k^n}\right)$	Dependiente de la arquitectura
OR-OR	Disponibilidad	61	LB: $min\left(\prod_{f_i \in F_{sub}} q_{av}(f_i) : \forall F_{sub} \in F_{C_m^n}\right)$, UB: $max\left(\prod_{f_i \in F_{sub}} q_{av}(f_i) : \forall F_{sub} \in F_{C_m^n}\right)$	Dependiente de la arquitectura
		62	LB: $min\left(\prod_{f_i \in F_{sub}} q_{av}(f_i) : \forall F_{sub} \in F_{C_k^n}\right)$, UB: $max\left(\prod_{f_i \in F_{sub}} q_{av}(f_i) : \forall F_{sub} \in F^{(*)}_{C_k^n} \vert F^{(**)}_{C_k^n}\right)$	Dependiente de la arquitectura
	"Throughput"	63	LB: $min(q_{tp}(f_i) : \forall f_i \in f^{\bullet} \vee f^{\circ})$, UB: $max(min(q_{tp}(f_i) : f_i \in F_{sub}, \forall F_{sub} \in F_{C_m^n}))$	Dependiente de la arquitectura
		64	LB: $min(min(q_{tp}(f_i) : \forall F_{sub} \in F_{C_k^n}))$, UB: $max(max(q_{tp}(f_i) : \forall F_{sub} \in F^{(*)}_{C_k^n} \vert F^{(**)}_{C_k^n}))$	Dependiente de la arquitectura
	Latencia	65	LB: $min(max(q_{rt}(f_i) : f_i \in F_{sub}, \forall F_{sub} \in F_{C_m^n}))$, UB: $max(q_{rt}(f_i) : \forall f_i \in f^{\bullet} \vee f^{\circ})$	Dependiente de la arquitectura
		66	LB: $min(max(q_{rt}(f_i) : \forall F_{sub} \in F_{C_k^n}))$, UB: $max(max(q_{rt}(f_i) : \forall F_{sub} \in F^{(*)}_{C_k^n} \vert F^{(**)}_{C_k^n}))$	Dependiente de la arquitectura
	Costo	67	LB: $min\left(\sum_{f_i \in F_{sub}} q_{pr}(f_i) : \forall F_{sub} \in F_{C_m^n}\right)$, UB: $max\left(\sum_{f_i \in F_{sub}} q_{pr}(f_i) : \forall F_{sub} \in F_{C_m^n}\right)$	Dependiente de la arquitectura
		68	LB: $min\left(\sum_{f_i \in F_{sub}} q_{pr}(f_i) : \forall F_{sub} \in F_{C_k^n}\right)$, UB: $max\left(\sum_{f_i \in F_{sub}} q_{pr}(f_i) : \forall F_{sub} \in F^{(*)}_{C_k^n} \vert F^{(**)}_{C_k^n}\right)$	Dependiente de la arquitectura

3.5. Resumen del Capítulo

En este capítulo se habló acerca de las aportaciones de otros autores con respecto a tres aspectos base para la estimación de la calidad de ensambles de componentes, que son la sintaxis, la semántica y las técnicas de estimación. La sintaxis, se refiere al formato que se usará para documentar la calidad de los componentes de software. Entre las opciones, hemos

visto que la más conveniente puede ser el formato XML, ya que puede ser interpretado tanto por personas, como por máquinas, lo que permitirá automatizar el proceso de estimación. La semántica se refiere al significado de las palabras, que en este caso es importante, ya que pueden existir diferentes interpretaciones acerca de un mismo atributo de calidad. Para mitigar cualquier posibilidad de malas interpretaciones acerca los atributos de calidad, estos deben ser definidos y corréctamente diferenciados unos de otros.

El tema central de este capítulo y de este estudio, en general, son las técnicas de estimación de la calidad en ensambles de componentes, que como se ha visto, todos los trabajos se concentran en la definición de fórmulas de agregación, aunque enfocadas en diferentes aspectos acerca de los ensambles de componentes de software cuya calidad se desea estimar. Hemos visto fórmulas que se basan, tanto en los atributos de calidad, como en otros factores tanto internos como externos que afectan la manera como se va a estimar la calidad de un ensamble. Esos factores pueden ser el uso, la dependencia hacia otros atributos de calidad de los componentes, la arquitectura y el contexto. También hemos visto fórmulas donde el patrón de composición determina la manera de estimar la calidad de los ensambles de software, razón por la cual estas fórmulas son consideradas dependientes de la arquitectura. Además de eso hemos sido testigos de otras aportaciones, que además de tomar en cuenta los patrones de composición, se enfocan en otras decisiones arquitectónicas como la tolerancia a fallos o los modelos de características de las líneas de productos de software (SPL).

Además de describir las aportaciones de otros autores, en este capítulo se han identificado relaciones entre los trabajos analizados. Para eso se han clasificado todas las fórmulas de agregación encontradas, tomando como base la clasificación propuesta en [11], misma que se ha incluido en las tablas de fórmulas de agregación presentadas a lo largo de este capítulo. En este capítulo también se simplificó una de las fórmulas propuestas en [31] para demostrar cómo sus fórmulas, aunque más complejas tienen las mismas bases que las fórmulas presentadas en tablas anteriores, sin embargo, a diferencia de muchas de ellas posee un factor de tolerancia a fallos distinto del que se usa en el patrón discriminador. Otras de las aportaciones de este capítulo son los glosarios de atributos de calidad y patrones de flujo de trabajo, que pueden servir como punto de partida para unificar la interpretación de estos conceptos en la definición de nuevas técnicas para estimar la calidad de ensambles de componentes de software.

Tabla 3.12: Fórmulas de agregación propuestas en [31]
Fuente: Elaboración propia

Patrón	Atributo	ID	Fórmula	Categoría
AND	Confiabilidad	1	$\prod_{l \in L} \prod_{i \in IW_l} \sum_{j \in J_i} r_{ij} x_{ij}$	Dependiente de la arquitectura
	"Throughput"	2	$min_{l \in L} \left(min_{i \in IW_l} \left(\sum_{j \in J_i} d_{ij} x_{ij} \right) \right)$	Dependiente de la arquitectura
	Tiempo de Ejecución	3	$max_{l \in L} \left(\sum_{i \in IW_l} \sum_{j \in J_i} e_{ij} x_{ij} \right)$	Dependiente de la arquitectura
	Costo	4	$\sum_{l \in L} \sum_{i \in IW_l} \sum_{j \in J_i} c_{ij} x_{ij}$	Dependiente de la arquitectura
Secuencia	Confiabilidad	5	$\prod_{i \in IS} \sum_{j \in J_i} r_{ij} x_{ij}$	Dependiente de la arquitectura
	"Throughput"	6	$min_{i \in IS} \left(\sum_{j \in J_i} d_{ij} x_{ij} \right)$	Dependiente de la arquitectura
	Tiempo de Ejecución	7	$\sum_{i \in IS} \sum_{j \in J_i} e_{ij} x_{ij}$	Dependiente de la arquitectura
	Costo	8	$\sum_{i \in IS} \sum_{j \in J_i} c_{ij} x_{ij}$	Dependiente de la arquitectura
XOR	Confiabilidad	9	$\sum_{l \in L} p_l \prod_{i \in IW_l} \sum_{j \in J_i} r_{ij} x_{ij}$	Dependiente del uso, Dependiente de la arquitectura
	"Throughput"	10	$\sum_{l \in L} p_l \cdot min_{i \in IW_l} \left(\sum_{j \in J_i} d_{ij} x_{ij} \right)$	Dependiente del uso, Dependiente de la arquitectura
	Tiempo de Ejecución	11	$\sum_{l \in L} p_l \sum_{i \in IW_l} \sum_{j \in J_i} e_{ij} x_{ij}$	Dependiente del uso, Dependiente de la arquitectura
	Costo	12	$\sum_{l \in L} p_l \sum_{i \in IW_l} \sum_{j \in J_i} c_{ij} x_{ij}$	Dependiente del uso, Dependiente de la arquitectura
OR	Confiabilidad	13	$\sum_{h \in H} p_h \prod_{l \in L^h} \prod_{i \in IW_l} \sum_{j \in J_i} r_{ij} x_{ij}$	Dependiente del uso, Dependiente de la arquitectura
	"Throughput"	14	$\sum_{h \in H} p_h \cdot min_{l \in L^h} \left(min_{i \in IW_l} \sum_{j \in J_i} d_{ij} x_{ij} \right)$	Dependiente del uso, Dependiente de la arquitectura
	Tiempo de Ejecución	15	$\sum_{h \in H} p_h \cdot max_{l \in L^h} \left(\sum_{i \in IW_l} \sum_{j \in J_i} e_{ij} x_{ij} \right)$	Dependiente del uso, Dependiente de la arquitectura
	Costo	16	$\sum_{h \in H} p_h \sum_{l \in L^h} \sum_{i \in IW_l} \sum_{j \in J_i} c_{ij} x_{ij}$	Dependiente del uso, Dependiente de la arquitectura
Componente DAG	Confiabilidad	17	$\sum_{l \in L} p_l \cdot \prod_{l \in L_a} \prod_{i \in IW_l} \sum_{j \in J_i} r_{ij} x_{ij}$	Dependiente del uso, Dependiente de la arquitectura
	"Throughput"	18	$\sum_{l \in L} p_l \cdot min_{l \in L_a} \left(min_{i \in IW_l} \left(\sum_{j \in J_i} d_{ij} x_{ij} \right) \right)$	Dependiente del uso, Dependiente de la arquitectura
	Tiempo de Ejecución	19	$\sum_{l \in L} p_l \cdot max_{l \in L} \left(\sum_{i \in IW_l} \sum_{j \in J_i} e_{ij} x_{ij} \right)$	Dependiente del uso, Dependiente de la arquitectura
	Costo	20	$\sum_{l \in L} p_l \cdot \sum_{l \in L_a} \sum_{i \in IW_l} \sum_{j \in J_i} c_{ij} x_{ij}$	Dependiente del uso, Dependiente de la arquitectura

Tabla 3.13: Glosario de expresiones para las fórmulas de agregación basadas en patrones de flujo de trabajo y una arquitectura con tolerancia a fallos
Fuente: Elaboración propia

Expresión	Significado
IW	Conjunto de tareas de un ensamble tipo AND, XOR, OR o DAG
IS	Conjunto de tareas de un ensamble tipo secuencia
i	Valor que hace referencia a una tarea del ensamble, $(1 \leq i \leq \#IW) \vee (1 \leq i \leq \#IS)$
J_i	Conjunto de operaciones que realizan una tarea i
j	Valor que hace referencia a una operación del ensamble, $(1 \leq j \leq \#J_i)$
L	Conjunto de corridas de un ensamble AND, XOR, OR o DAG
l	Valor que hace referencia a una corrida de un ensamble AND, XOR, OR o DAG, $1 \leq l \leq \#L$
H	Conjunto potencia de las posibles corridas en un ensamble OR, $H = \mathcal{P}(L)$
h	Valor que hace referencia a un conjunto de corridas que serán ejecutadas en paralelo en un ensamble OR, $1 \leq h \leq \#H$
r	Confiabilidad
d	"Throughput"
e	Tiempo de ejecución
c	Costo
x_{ij}	Valor que determina si una operación j será seleccionada para realizar la tarea i, $Dom(x_{ij}) = 0, 1$
p_l	En ensambles OR, XOR y DAG, es la probabilidad de que una corrida l sea ejecutada
p_h	En ensambles OR, es la probabilidad de que un conjunto de corridas h sean ejecutadas

Tabla 3.14: Glosario de expresiones para las fórmulas de agregación basadas en patrones de flujo de trabajo y modelos de características
Fuente: Elaboración propia

Expresión	Significado
n	Número de operaciones de un ensamble de software
i	Valor que hace referencia a una operación del ensamble, $(1 \leq i \leq n)$
f	Característica (todas las características de las fórmulas son operaciones)
q_{av}	Disponibilidad
q_{tp}	"Throughput"
q_{rt}	Latencia
q_{pr}	Costo
F	Conjunto de características del ensamble
$F_{C_k^n}$	Conjunto de todas las combinaciones permisibles de conjuntos de características, $F_{C_k^n} \subset \mathcal{P}(F)$
$F_{C_m^n}$	$F_{C_m^n} = \mathcal{P}(F_{C_k^n})$
$\bullet$	Se refiere a características obligatorias
$\circ$	Se refiere a características opcionales
$*$	Se refiere a conjuntos de características de tipo Or-Group
$**$	Se refiere a conjuntos de características de tipo Xor-Group

Capítulo 4

Análisis del Estado del Arte

Comenzaremos el análisis planteando la siguiente pregunta ¿De qué depende la estimación de la calidad en un ensamble de componentes de software? Como hemos visto en el estado del arte, en [11] se afirma que son las características de los atributos de calidad, las que marcan la pauta de cómo estimar la calidad en ensambles de software, y estos atributos de calidad son clasificados de acuerdo con las reglas necesarias para realizar esta estimación. Por otro lado, también hemos visto que un mismo atributo de calidad puede ser estimado de varias maneras, dependiendo del patrón de flujo de trabajo usado en el ensamble de componentes de software. En el capítulo anterior, se presentó un ejemplo de atributo de calidad diréctamente agregable propuesto en [11], donde el consumo de memoria de un ensamble de componentes se calcula sumando el consumo de memoria de cada uno de los componentes. Esto es cierto si los componentes del ensamble se ejecutan en forma secuencial o en paralelo, pero deja de serlo cuando los componentes se ejecutan de acuerdo con el patrón XOR, en cuyo caso, la estimación de la calidad de un ensamble de software se realizaría a través de una suma ponderada por la probabilidad de ejecución o tomando el consumo de memoria del componente que haga mayor uso de este recurso. En [11] no se hace mención del impacto del patrón de flujo de trabajo cuando se intenta medir atributos de calidad como el consumo de memoria, sino que simplemente se cataloga al consumo de memoria como un atributo de calidad que siempre habrá de ser diréctamente agregable; esto quizás ocurre, porque este trabajo no se enfoca en los servicios web, que es donde más se usan los patrones de flujo de trabajo. Otra posible razón es que no tiene como finalidad proponer fórmulas de agregación de la calidad para ensambles de componentes de software, sino establecer un conjunto de criterios que sirvan como apoyo para definir fórmulas de agregación a la medida del ensamble de componentes cuya calidad se desea estimar, sin importar el contexto de aplicación o el modelo de composición utilizado.

En las próximas secciones se harán algunas observaciones con respecto a la sintaxis para la especificación de interfaces, se hablará, también, acerca de las tendencias encontradas durante la revisión de la literatura y se intentará identificar relaciones entre los conceptos que se han estudiado, con relación a este tema. En las últimas secciones se discutirá acerca de qué tan completas son las fórmulas de agregación presentadas y qué criterios pueden hacer a unas fórmulas de agregación más adecuadas que otras.

4.1. Sintaxis

Se han encontrado pocos trabajos que analizan la manera como los atributos de calidad de los componentes deben ser especificados. Entre los trabajos encontrados, las propuestas que resultan más convincentes son aquellas que proponen extender la sintaxis del WSDL, de manera que permita especificar la calidad de los servicios web, junto con su funcionalidad. Esta propuesta es, de acuerdo con los trabajos que hablan al respecto, muy efectiva en este contexto, sin embargo, existen otros contextos que no hacen uso del WSDL, para los que habría que establecer un estándar parecido que permita agregar especificaciones de calidad. En el caso del WSQDL, que es una extención del WSDL muy completa, para especificar la calidad de servicios web, existe una desventaja, que es el almacenamiento de información que no corresponde al componente en sí, sino que son metadatos acerca de los atributos de calidad, como en el caso de la unidad de medida. Esto agrega complejidad al WSQDL, además de consumo de ancho de banda al transmitir toda esta información a través de la red. La principal desventaja es con respecto al mantenimiento, ya que si se desea cambiar la unidad de medida para un atributo de calidad y que este cambio aplique a todos los WSQDL, se deberá modificar cada WSQDL. Por esa razón, en este análisis, se propone utilizar un sistema de registros, como los que se proponen en [21] y [32]. El formato XML, como ya se mencionó es un formato muy viable para documentar información que deberá ser interpretada por un ordenador, por lo que se propone que estos registros sean provistos en este formato. La información contenida en los registros, debería basarse en la información de calidad que proporciona el WSQDL, que a pesar de la desventaja mencionada, proporciona información muy completa acerca de los atributos de calidad.

Aunque el sistema de registros ya no sea parte del WSDL, es recomendable extender el WSDL, de manera que permita conocer la ubicación del registro que contiene la especificación de calidad correspondiente al servicio web descrito, lo que permitiría recuperar la especificación de calidad del servicio web en forma automática, sin agregar demasiada complejidad al WSDL.

La existencia de registros, independientes del WSDL, también permitirá especificar la calidad de componentes que no se encuentren en el contexto de los servicios web.

En cuanto a la información contenida en los registros, como ya se dijo, es recomendable basarse en la información provista por el WSQDL, sin embargo, es posible que esta información no sea suficiente para algunos de los interesados en conocer la calidad de los componentes de software. En [27] se incluyen algunas fórmulas de agregación donde se obtienen dos valores, un valor optimista y un valor pesimista. Estos valores informan acerca de la mejor calidad y la peor calidad que el ensamble de software puede ofrecer, respectivamente. Esto le permite al usuario conocer el rango de valores de calidad que el software puede ofrecer, información que resulta muy útil cuando las restricciones del software establecen tiempos de respuesta o costos máximos que pueden tolerarse, por ejemplo. En otros casos, al usuario del software no le bastará con conocer la mejor, o la peor calidad que puede ofrecer el software, sino la calidad con mayor probabilidad, o la probabilidad de que la calidad se encuentre dentro de ciertos límites, según lo argumentado en [18]. Para poder ofrecer esta información al usuario del software, es necesario incluir varias entradas de calidad en el formato de especificación, y no sólo una, como se esperaba al principio de este estudio. Para que la estimación de calidad de ensambles de software sea más completa, se recomienda realizar la especificación de calidad

de acuerdo con la propuesta de [18], cuya propuesta también incluye la manera de agregar la calidad en ensambles de software, usando fórmulas de agregación como las descritas en el Capítulo 3.

Al principio de este estudio, se esperaba encontrar trabajos que hablaran acerca de la expecificación y estimación de la calidad en componentes compuestos, de los cuales se habló en el Capítulo 2. Lamentablemente, todos los trabajos encontrados se enfocaron en el concepto general de ensambles de software, sin referirse específicamente a componentes compuestos, como tal. En el caso de los ensambles de servicios web, el resultado suele ser otro servicio web que puede ser utilizado por terceros y también puede ser reutilizado para conformar otro ensamble de servicios web, esto convierte a los ensambles de servicios web tanto en productos de software terminados, como en componentes compuestos, pero esto es posible sólo gracias a SOA, por lo que en otros contextos, esto no sería tan fácil de lograr.

En el Capítulo 2 se mencionó que la especificación de la calidad de componentes compuestos, debe ser similar a la de los componentes atómicos, de manera que los componentes compuestos puedan ser integrados en un ensamble de software de la misma forma como se hace con los atómicos. Para fines de la agregación de la calidad, el principio debe ser el mismo, sin embargo, a diferencia de los componentes atómicos, la estimación de calidad de componentes compuestos, implica atravesar un proceso de SWR, para lo que se necesita conocer los distintos patrones de composición utilizados en el ensamble. Esto suma otro problema con respecto a la agregación de la calidad, ya que en ausencia de la información relacionada con la manera como están compuestos los componentes compuestos, sería imposible realizar la agregación de la calidad. En el caso de los servicios web, este problema está prácticamente resuelto, ya que esta información puede ser obtenida de a través de BPEL, que también se basa en el formato XML; pero nuevamente nos enfrentamos a la limitante de que esta solución sólo aplica para los servicios web, por lo que habrá que determinar una solución que permita recuperar los patrones de composición de componentes compuestos.

La recomendación para solucionar este problema, es utilizar un registro adicional, que permita conocer la estructura del ensamble y, con base en esta, realizar el SWR de la calidad correspondiente al componente compuesto, cuya calidad se desea agregar. Dado que este registro será utilizado únicamente para el SWR, la información requerida en este, no debería ser muy compleja. Para ilustrar esta propuesta, a continuación se presenta un ejemplo de la posible estructura de este registro para el flujo presentado en la Figura 3.12.

```
<componente_compuesto nombre="Componente 1" id="Comp7">

    <componentes>
        <componente id="Comp1">
            <operaciones>
                <operacion id="o1" alias="Op1"/>
                <operacion id="o2" alias="Op7"/>
            </operaciones>
        </componente>
        ...
        <componente id="Comp4">
            <operaciones>
```

```
                <operacion id="o1" alias="Op5"/>
            </operaciones>
        </componente>
    </componentes>

    <operaciones>
        <operacion_compuesta nombre="Operaci\'on 8" id="Op8">
            <patron:secuencia>
                <operacion id="Op1"/>
                <patron:and>
                    <patron:secuencia>
                        <operacion id="Op2"/>
                        <patron:xor>
                            <operacion id="Op4" probabilidad="0.6"/>
                            <operacion id="Op5" probabilidad="0.4"/>
                        </patron:xor>
                    </patron:secuencia>
                    <patron:secuencia>
                        <operacion: id="Op3"/>
                        <patron:loop iteraciones="20">
                            <operacion id="Op6"/>
                        <patron:loop>
                    </patron:secuencia>
                </patron:and>
                <operacion id="Op7"/>
            </patron:secuencia>
        </operacion_compuesta>
    </operaciones>

</componente_computeso>
```

4.2. Tendencias en las Técnicas de Estimación

Al principio de este capítulo se mencionó que algunos trabajos se concentran en aplicaciones dentro del contexto de los servicios web, mientras que otros trabajos se concentran en diferentes contextos, o evitan hablar acerca de un contexto específico, ya que consideran que su propuesta puede servir para varios contextos de aplicación; y en casos más ambiciosos, buscan proponer una solución que sea independientemente del contexto de aplicación.

Como se muestra en la Tabla 4.1, de los 21 trabajos que han hablado acerca de la estimación de la calidad en ensambles de componentes de software, 19 han estado dentro del contexto de los servicios web, lo que equivale al 90 % de los trabajos revisados. Esto podría deberse a que existe una mayor preocupación por estimar la calidad en ensambles de servicios web o que el contexto de los servicios web resulta una opción más viable para realizar estimaciones de calidad que el resto de los contextos, por tratarse de elementos de software disponibles a través de una red pública, de fácil composición y de fácil descubrimiento. Del

Tabla 4.1: Artículos por contexto de aplicación
Fuente: Elaboración propia

Contexto	Artículos	Fuentes
Múltiples contextos	1	[11]
Web mashups	1	[37]
Aplicaciones de escritorio	1	[34]
Servicios web	19	[27, 9, 18, 20, 38, 13, 15, 31, 7, 34, 10, 19, 2, 17, 6, 16, 26, 39, 29]

resto de los trabajos, uno se encuentra en el contexto de web mashups, otro aplica para múltiples contextos y, en el caso de [34], además de estar dentro del contexto de los servicios web, también aplica para aplicaciones de escritorio. De todos estos, los únicos trabajos que no mencionan el uso de patrones de flujo de trabajo son [11] y [37], que son los únicos que no entran en el contexto de los servicios web, lo que respalda la idea de que los patrones de flujo de trabajo están asociados a este contexto.

4.2.1. Patrones de Flujo de Trabajo

Aunque algunos autores no tomen en cuenta los patrones de composición para la definición de fórmulas de agregación, es posible asumir que estos autores han definido sus fórmulas con base en ensambles de software que sin intención de ello siguen algún patrón, como por ejemplo el patrón secuencial. Cuando se desarrolla un programa de software, este se construye con base en una secuencia lógica de instrucciones, donde cada instrucción se lleva a cabo únicamente cuando la intrucción anterior a concluido; esto ocurre de forma similar a como ocurre en el patrón secuencial. Esto puede provocar que la presencia del patrón secuencial, dentro del ensamble de componentes de software pase desapercibida, hasta el momento en que el desarrollador se vea en la necesidad de utilizar ensambles que sigan otros patrones, como XOR o AND. Al principio de este capítulo se analizó una fórmula propuesta en [11], que tiene como propósito agregar el consumo de memoria en un ensamble de componentes. El objetivo del análisis de esta fórmula es hacer evidente para el lector la existencia de patrones de composición en cualquier ensamble de componentes de software, aún cuando dicho patrón no sea reconocido explícitamente por el desarrollador del ensamble. También es importante reconocer que no todos los ensambles de software cuentan con una estructura que permita identificar fácilmente los patrones de composición utilizados. Este es otro punto a favor de los servicios web, ya que, para ellos, existen modelos de composición que se basan explícitamente en patrones de flujo de trabajo, lo que permite identificar cada uno de los patrones utilizados en el ensamble y facilita el SWR; tal es el caso del modelo de composición utilizado por el *Business Process Execution Language* (BPEL).

Como se puede observar en la Figura 4.1, el patrón secuencial es el que más se ha hecho presente en los trabajos que proponen fórmulas de agregación de la calidad. Esto suena lógico si se toma en cuenta que en los flujos de trabajo es muy común encontrar tareas que dependen unas de otras, es decir, que las salidas esperadas de una tarea serán usadas como

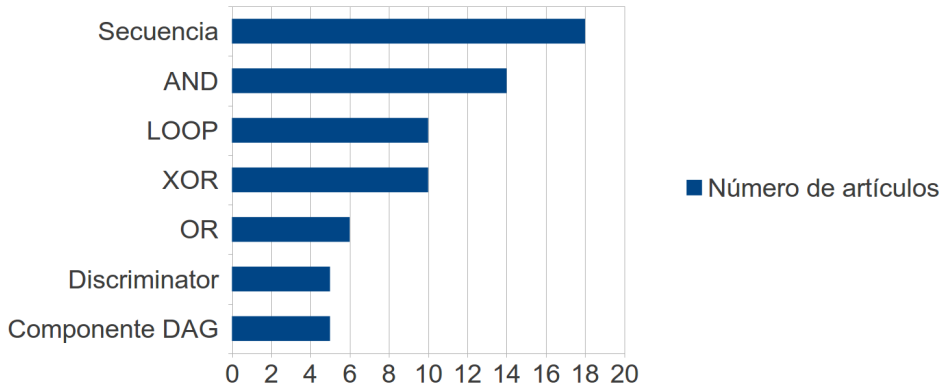

Figura 4.1: Número de artículos por patrón de flujo de trabajo
Fuente: Elaboración propia

entradas para la tarea siguiente. También es importante tomar en cuenta que si las tareas de un flujo de trabajo no siguen una secuencia lógica, pueden ofrecer resulados impredecibles y en muchas ocasiones indeseables. Estos criterios hacen que el patrón secuencial sea de gran importancia en la definición de flujos de trabajo, lo que explicaría una mayor preocupación por definir fórmulas de agregación para flujos de trabajo secuenciales, que por cualquier otro tipo de flujo. Otro factor que podría haber influido en la cantidad de trabajos que hacen uso de este patrón es que se trata de uno de los tipos de ensamble más sencillos, y las fórmulas de agregación definidas para este patrón, suelen ser igual de simples.

El segundo patrón más utilizado en la definición de fórmulas de agregación es el patrón AND, principalmente en el contexto de los servicios web, donde los servicios de un ensamble pueden alojarse en diferentes nodos de red, lo que permite que varios servicios puedan ser ejecutados en paralelo. La ejecución en paralelo es una técnica muy común para optimizar tiempos, siempre que un conjunto de operaciones sean independientes unas de otras, lo que explicaría la cantidad de fórmulas de agregación que se han definido para este patrón. Además de eso, las fórmulas de agregación definidas para el patrón AND, suelen ser casi tan simples como las fórmulas definidas para el patrón secuencia.

Los patrones XOR y LOOP han sido menos trabajados que los patrones secuencia y AND, sin embargo, también gozan de popularidad entre los trabajos que han definido fórmulas de agregación para ensambles de componentes de software. El que estos cuatro patrones se encuentren entre los más utilizados en las fórmulas de agregación de la calidad, no debe ser una sorpresa para quienes han definido flujos de trabajo, ya que son los cuatro patrones más usados en cualquier modelo de procesos, como *Business Process Model Notation* (BPMN) o BPEL.

Los patrones OR, Discriminator y DAG son los menos trabajados en la definición de fórmulas de agregación para ensambles de componentes de software; en parte, puede ser porque no son patrones muy comunes en la definición de procesos y por otra parte, debido a la complejidad que implica su agregación.

Los patrones utilizados en los modelos de características son otro tipo de patrones de composición que pueden ser usados en la definición de fórmulas de agregación para ensambles

de componentes de software en SPLs, como se demostró en [27]. Este tipo de patrones no se ha incluido en la gráfica puesto que se trata de patrones de naturaleza diferente a los patrones de flujo de trabajo y sólo hay un documento donde se definen fórmulas de agregación basadas en estos patrones. Entre los patrones que se estudian en [27] se encuentran características obligatorias, características opcionales, Or-Group y Xor-Group. Los primeros dos patrones hacen distinción de las características indispensables en cada uno de los productos de la línea y las que deberán ser agregadas únicamente en ciertos productos. Las características obligatorias son aquellas que se harán presentes en todos los productos, desde el más básico, hasta el más sofisticado, mientras que las características opcionales, probablemente, no se harán presentes en el producto más básico de la línea, pero sí en otros de los productos. Algunas de las fórmulas de [27] presentadas en la Tabla 3.15 hacen uso de estos dos patrones para determinar rangos de calidad, donde uno de los límites resulta de la calidad ofrecida únicamente cuando las características obligatorias se encuentran presentes y otro de los límites resulta de la calidad ofrecida por la presencia de todas las características del ensamble, tanto obligatorias, como opcionales. El patrón Xor-Group indica que de un conjunto de características sólo una se hará presente en cada producto de la línea, mientras que el patrón Or-Group indica que un subconjunto de las características se hará presente en cada producto de la línea, ya sea una, todas o ninguna de las características del grupo, lo que sin duda afectará la calidad obtenida en cada uno de los productos de la línea.

4.2.2. Atributos de Calidad

En la Figura 4.2 se muestra la cantidad de artículos que han definido fórmulas de agregación para los atributos de calidad rendimiento, fiabilidad, costo y seguridad. Estos cuatro atributos de calidad se encuentran en el mismo nivel de acuerdo con la taxonomía que se presenta en la Figura 3.4.

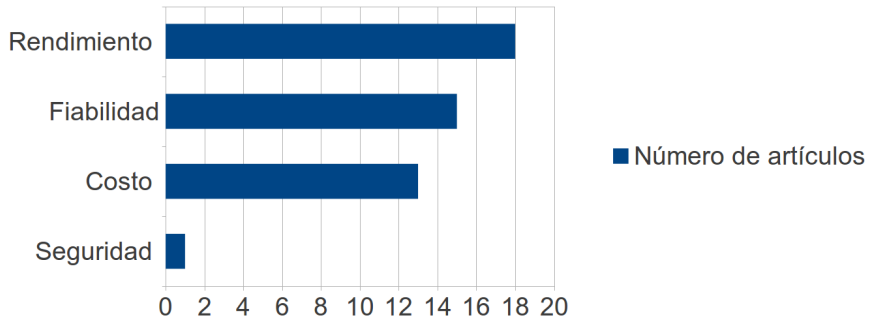

Figura 4.2: Número de artículos por atributo de calidad
Fuente: Elaboración propia

De los cuatro atributos de calidad, aquel para el que más trabajos han definido fórmulas de agregación es el rendimiento. Seguido de este, aunque también muy utilizados, se encuentran la fiabilidad y el costo; y por último se en cuentra seguridad, para el que tan solo un trabajo ha definido fórmulas de agregación.

La frecuencia con la que se han propuesto fórmulas de agregación para estimar el rendimiento de ensambles de software demuestra que hay una gran preocupación acerca de la

construcción de software de alto rendimiento. El rendimiento es quizás el atributo de calidad que más preocupa a los usuarios del software. En la mayoría de los casos, los usuarios no sabrán qué nivel de seguridad es apropiado para el sistema, ni tampoco sabrán comunicarle al desarrollador la fiabilidad esperada, pero probablemente sabrán cuánto tiempo están dispuestos a esperar para ver una respuesta por parte del sistema. Además de la preocupación del usuario hacia los tiempos, hay que tomar en cuenta las restricciones de hardware ya que si el usuario del software no cuenta con un equipo de hardware que cuente con la memoria requerida para la operación del software, entonces el sofware será inservible para este.

El poder predecir el rendimiento en un ensamble de software antes de que este sea construido, permitirá al desarrollador de software construir un producto de software que logre satisfacer las necesidades y expectativas del usuario sin necesidad de invertir tiempo y dinero excesivos.

Aunque la mayoría de los usuarios del software no sean conscientes acerca de sus expectativas con respecto a la fiabilidad, se harán conscientes de ello en cuanto usen un producto de software cuya tasa de fallos sea tan alta, que merme su productividad, o que no se encuentre disponible durante los periodos de tiempo requeridos; lo que provocará que el usuario no se sienta satisfecho con el software adquirido, afectando, también, la probabilidad de que otros usuarios deseen adquirir este software. Es por eso, que estimar la fiabilidad de un ensamble de software, antes de su construcción, resulta de gran importancia, para el desarrollador de software.

En la mayoría de los contextos de aplicación, la estimación del costo como atributo de calidad no cobrará mucho sentido, ya que los usuarios, regularmente, pagarán una renta mensual o comprarán el software para volverse propietarios de este y poder explotarlo ilimitadamente. En el contexto de los servicios web, la estimación del costo cobra gran importancia, ya que los servicios se encuentran disponibles en una red pública, por lo que, en la mayoría de los casos, el usuario no podrá hacerse propietario de los servicios web. Las políticas de cobro de los servicios web suelen depender del proveedor del servicio, quien es el que se encarga de que el software se encuentre operando y disponible para los usuarios. En este modelo de negocio, el proveedor del servicio web, puede elegir entre cobrar una renta periódica, cobrar por invocación del servicio o proveer el servicio en forma gratuita, lo que hace menos predecible el costo de un ensamble de servicios web e incrementa la preocupación por agregar el costo de del ensamble.

La estimación de la seguridad no ha sido un tema popular en la literatura y eso se debe, quizás, a que algunos trabajos donde se ha hablado acerca de este atributo de calidad, afirman que su estimación es muy complicada. En [11] y [22] se menciona que la seguridad es un atributo de calidad que depende en gran medida del contexto del sistema, que como ya se ha mencionado, depende a su vez de varios factores, tanto internos como externos del sistema, lo que dificulta asegurar un margen de error reducido al momento de estimar la seguridad. Particularmente, en [11] se afirma que es imposible derivar la seguridad de un ensamble con base en la seguridad de los componentes.

El único trabajo donde se proponen fórmulas de agregación para seguridad es [19], que se concentra en el nivel de encriptación del ensamble, el cual se calcula con base en el nivel de encriptación de los componentes que lo constituyen, lo que contradice la postura de [11] y [22]. Sin embargo, el nivel de encriptación es tan sólo una parte de lo que viene implícito

dentro de la seguridad, por lo que este trabajo no descarta la posibilidad de que lo que se afirma en [11] y [22], sea válido para otros aspectos más complejos de la seguridad.

A continuación se realizará este mismo análisis para los atributos de calidad que se encuentran dentro de los grupos de rendimiento y confiabilidad, para agregar mayor profundidad al análisis.

En este trabajo, se considera que los atributos de calidad de los cuales se hablará a continuación son incluso más representativos que sus atributos padre, dentro de la taxonomía. Por ejemplo, el concepto rendimiento resulta muy vago, dada su generalidad, a diferencia del concepto latencia, que da una idea más clara acerca de lo que se desea medir.

Atributos de Rendimiento

De acuerdo con la Figura 4.3, el atributo de rendimiento que más se ha trabajado, es la latencia, esto podría deberse a que, este atributo de calidad es muy evidente para el usuario, quien probablemente basará su nivel de satisfacción en el grado en el que el software cumpla con las restricciones establecidas para la latencia. Una desventaja al evaluar la latencia es que se trata de un atributo de calidad muy variable, ya que, depende de otros atributos de calidad del ensamble de software, entre los más relevantes, el tiempo de ejecución de los componentes y el ancho de banda entre nodos de red, lo que puede provocar que en muchas ejecuciones, la latencia tome valores muy alejados del valor estimado. Se asume que en un ambiente, donde todos estos factores sean controlados, la latencia de los componentes será hasta cierto punto, constante y se podrán hacer estimaciones más confiables acerca de la latencia de un ensamble de software; teoría que sería interesante de comprobar en un momento dado.

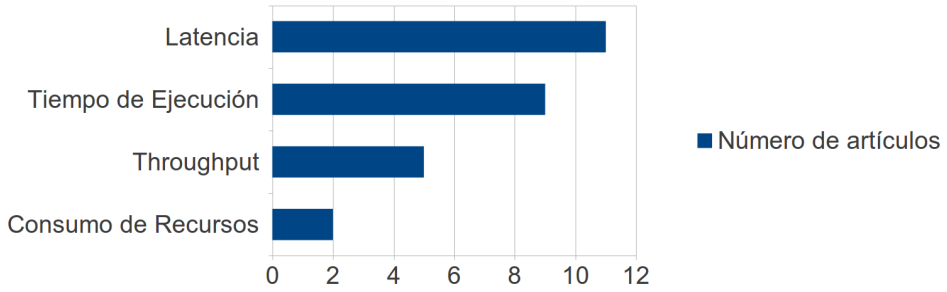

Figura 4.3: Número de artículos por atributo de rendimiento
Fuente: Elaboración propia

Como se acaba de mencionar, el tiempo de ejecución de los componentes es uno de los factores base para estimar la latencia de un ensamble de componentes de software. El tiempo de ejecución suele ser más controlado que la latencia, aunque como se menciona en [11] también depende de otros atributos de calidad y características relacionadas con la implementación como la existencia de tareas de mayor prioridad, tiempo de bloqueo y periodos de ejecución.

Actualmente existen herramientas "profile" para la mayoría de los ambientes de desarrollo, que permiten conocer el tiempo de ejecución de un componente en particular, despreciando el tiempo consumido en otras tareas del sistema, lo que permite medir el tiempo de ejecución en

forma controlada y precisa. Esta ventaja pudo ser un gran motivador para que el tiempo de ejecución sea el segundo atributo de rendimiento más trabajado en el tema de la estimación de ensambles de componentes de software.

El "throughput" es una referencia muy útil para conocer la carga de trabajo que el software puede soportar durante cierto periodo de tiempo, sin embargo, no es un atributo de calidad que el usuario pueda evaluar a simple vista, por lo que no tendrá un impacto directo en la satisfacción de este. Quizás por eso no ha sido un atributo de calidad muy tomado en cuenta en los trabajos relacionados con este tema.

Únicamente dos trabajos han propuesto fórmulas para estimar el consumo de recursos, uno de ellos se enfoca en el consumo de memoria [11] y otro en el ancho de banda [15]. Esto quizás se deba a que estos dos atributos de calidad son prácticamente imperceptibles para la mayoría de los usuarios, lo que explicaría que despierten un menor interés en los desarrolladores de software.

Atributos de Fiabilidad

El atributo de fiabilidad para el que más trabajos han propuesto fórmulas de agregación, es la confiabilidad que se refiere a la frecuencia con que ocurren fallos en el sistema (Ver Figura 4.4). Cuando el software presenta fallos con una frecuencia excesiva, impacta negativamente en la satisfacción del usuario, quien se percatará inmediatamente de que el software no está cumpliendo sus expectativas. Eso hace que la confiabilidad sea, junto con la latencia, uno de los dos atributos de calidad que más impacto tienen en la satisfacción del usuario.

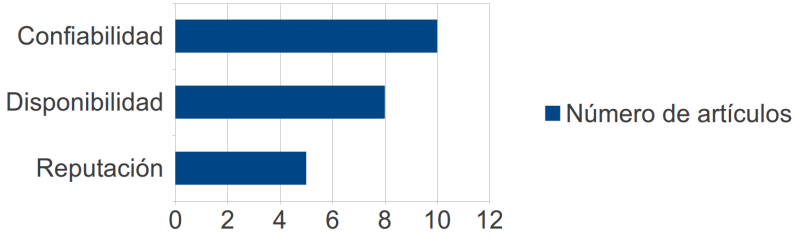

Figura 4.4: Número de artículos por atributo de fiabilidad
Fuente: Elaboración propia

Entre los trabajos que han propuesto fórmulas de agregación para la confiabilidad, un trabajo se ha referido a esta en términos de MTTF, mientras que los otros nueve han usado el término de confiabilidad, refiriéndose, en realidad a la probabilidad de éxito en cada ejecución del software.

El siguiente atributo de fiabilidad más trabajado es la disponibilidad. En uno de los trabajos, se definen fórmulas para agregarla en términos de "uptime probability", en otros seis en términos de disponibilidad y otro define fórmulas de agregación para accesibilidad, y para disponibilidad. Aunque no todos los trabajos dejan en claro a qué se refieren con disponibilidad, se infiere que en realidad estan hablando de "uptime probability".

Existe un menor número de trabajos que han definido fórmulas de agregación para la reputación, uno de estos refiriéndose a este atributo de calidad en términos de fidelidad y otros cuatro lo manejan simplemente como reputación.

4.2.3. Tipos de Fórmulas

En el Capítulo 3 se mencionó que las clasificaciones de atributos de calidad propuestas en [11] podrían ser más apropiadas para clasificar fórmulas de agregación que atributos de calidad. Esto podría justificarse con ayuda de las Tablas 3.3 a 3.10, donde se presentan casos de fórmulas de agregación para un mismo atributo de calidad, cuyas características corresponden a diferentes clasificaciones; por ejemplo, el costo, que cuando se estima en un ensamble que sigue los patrones AND y secuencia, las características de la fórmula de agregación, hacen parecer que se trata de una fórmula diréctamente agregable, aunque en realidad, para efectos de este estudio, se sitúa dentro de la clasificación de dependiente de la arquitectura por tratarse de una fórmula basada en patrones de flujo de trabajo. Por otra parte, cuando el patrón del ensamble es OR, XOR o LOOP, las características de la fórmula de agregación conciden con las de la clasificación de dependiente del uso.

El que una fórmula de agregación reuna ciertas caracteristicas parece no depender sólo del atributo de calidad, como podría ser interpretado, de acuerdo con lo que se dice en [11], sino que es resultado de tratar de estimar un atributo de calidad en particular, siguiendo un modelo de composición particular, ya sea que se trate de patrones de flujo de trabajo u otro tipo de patrones de composición.

Las fórmulas de agregación reunidas durante la revisión de la literatura fueron clasificadas de acuerdo con las categorías ya mencionadas. En total se lograron reunir 279 fórmulas, de las cuales, 277 han entrado en la categoría de dependientes de la arquitectura, que como se puede ver en la Figura 4.5, es la categoría que ha reunido más fórmulas de agregación, y corresponden, en su mayoría, a las fórmulas basadas en patrones de flujo de trabajo. La siguiente categoría con mayor número de fórmulas de agregación es dependiente del uso, con 80 fórmulas identificadas. Después se encuentran, derivada, con 10 fórmulas, y diréctamente agregable con 1 fórmula. Como se puede ver, no se han encontrado fórmulas de agregación que se puedan catalogar como dependientes del contexto.

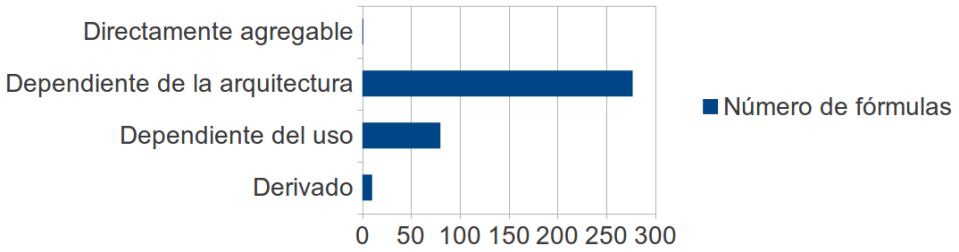

Figura 4.5: Número de formulas por clasificación
Fuente: Elaboración propia

Como se hizo notar en el capítulo 3, las fórmulas basadas en patrones de flujo de trabajo se encuentran dentro de la clasificación de dependientes de al arquitectura y suman 276 de las 279 que se analizaron en este estudio. De estas 276 fórmulas, 118 fórmulas no contienen elementos que hagan evidente su relación con la arquitectura, ni tampoco son dependientes del uso o derivadas, por lo que podrían dar la apariencia de ser diréctamente agregables. Una notable cantidad de estas fórmulas corresponde a los patrones secuencia y AND, lo que

respalda nuestro supuesto de que todas las fórmulas de agregación se ajustan a un patrón de composición, incluso cuando quien define la fórmula de agregación no sea consciente de eso.

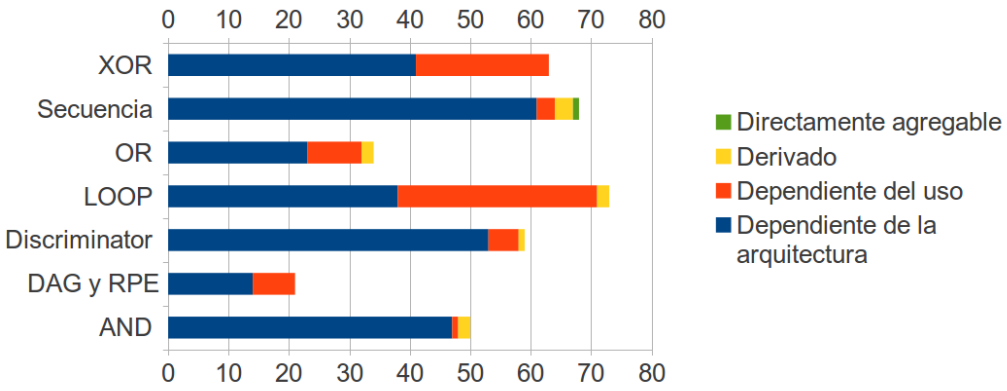

Figura 4.6: Número de formulas por clasificación y patrón de flujo de trabajo
Fuente: Elaboración propia

Las fórmulas dependientes de la arquitectura se observan con gran frecuencia en el patrón discriminador, que como ya se ha mencionado, es un patrón común en el software tolerante a fallos. Aunque no muchos trabajos han definido fórmulas de agregación para este patrón, los pocos trabajos que lo han hecho, han definido fórmulas para cada tipo de discriminador, lo que ha provocado que sea el segundo patrón con más fórmulas de agregación. Esto también se puede comprobar observando la Figura 4.6.

Las fórmulas dependientes del uso son más frecuentes en los patrones XOR y LOOP, que se caracterizan por ofrecer una calidad muy variable. En el caso de XOR, la calidad depende del componente que va a ser ejecutado, por lo que la manera más confiable de estimar la calidad de un ensamble, es con base en la probabilidad de que cada componente sea ejecutado, el cual es un factor que depende del uso. En el caso de LOOP, la calidad generalmente dependerá de cuántas veces sea ejecutado el componente, que también es un factor que depende del uso. Esto indica que también puede haber una relación entre las fórmulas dependientes del uso y los patrones XOR y LOOP, que después de los patrones AND y secuencia, son los más trabajados en la literatura, de acuerdo con la Figura 4.1.

Las fórmulas derivadas no parecen tener una relación directa con ningún patrón de los que se han analizado en este estudio, pero podría ser que dependan de algún atributo de calidad. Para poner a prueba esta teoría se ha incluido la Figura 4.7, que muestra la relación de fórmulas por atributo de calidad, indicando la proporción con que reunen características correspondientes a cada una de las clasificaciones. La gráfica incluye atributos de calidad que se encuentran a diferentes niveles de la taxonomía, pero que se considera que han sido los más representativos en este estudio.

Aunque quizás se podrían sacar algunas deducciones acerca de la relación entre atributos de calidad y clasificaciones, con base en la Figura 4.7, ninguna deducción es respaldada por algún hallazgo encontrado hasta el momento. En cuanto a las fórmulas derivadas, lo único observable es que sólo se han encontrado ejemplos de estas para la estimación de latencia, reputación y confiabilidad, siendo confiabilidad el atributo de calidad con el mayor número de fórmulas derivadas.

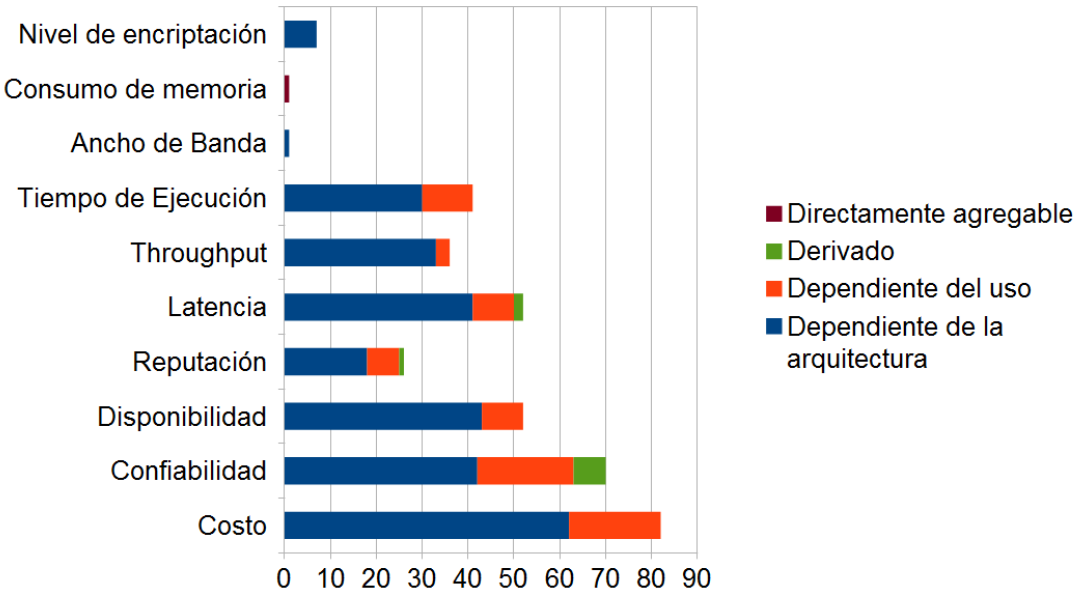

Figura 4.7: Número de formulas por clasificación y atributo de calidad
Fuente: Elaboración propia

Todo lo anterior puede significar, por una parte, que quizás, la clasificación propuesta en [11] no siempre depende de los atributos de calidad, sino también de los patrones de composición; y por otra parte, que los trabajos que han propuesto fórmulas de agregación basadas en patrones de flujo de trabajo pudieron haber omitido criterios importantes acerca de la estimación de la calidad de ensambles de componentes de software. Un ejemplo de esto es la falta de fórmulas derivadas en atributos como latencia y tiempo de ejecución.

4.2.4. Operadores de Agregación

Una característica de las fórmulas de agregación que sí ha demostrado tener cierta relación con los atributos de calidad es el operador de agregación. Cuando hablamos de operador de agregación, nos referimos a ciertos elementos dentro de la sintaxis de una fórmula de agregación que indican la manera como se agregarán los operandos. Estos pueden ser operadores aritméticos, como $\sum$ o $\prod$, una sintaxis como $\frac{x}{y}$ o x^y u otro tipo de operadores, como por ejemplo max o min, que son operadores comunes en la agregación de conjuntos de datos. Las Figuras 4.8 y 4.9, se han incluido para respaldar el análisis acerca de las tendencias con respecto a los operadores de agregación.

Entre las tendencias que se han observado, está la relación entre el operador producto ($\prod$) y los atributos confiabilidad y disponibilidad. Estos dos atributos de calidad suelen evaluarse con base en métricas de probabilidades. Cuando la calidad se expresa en términos de probabilidades, la probabilidad agregada de la ejecución de varios componentes se puede expresar como $P(a) = P(c_1) \cap P(c_2) \cap ... \cap P(c_n)$ y se calcula de la siguiente manera $P(a) = P(c_1) \cdot P(c_2) \cdot ... \cdot P(c_n)$. Esta regla explica la relación entre el operador producto y los atributos confiabilidad y disponibilidad, aún cuando no sea aplicable para todos los patrones

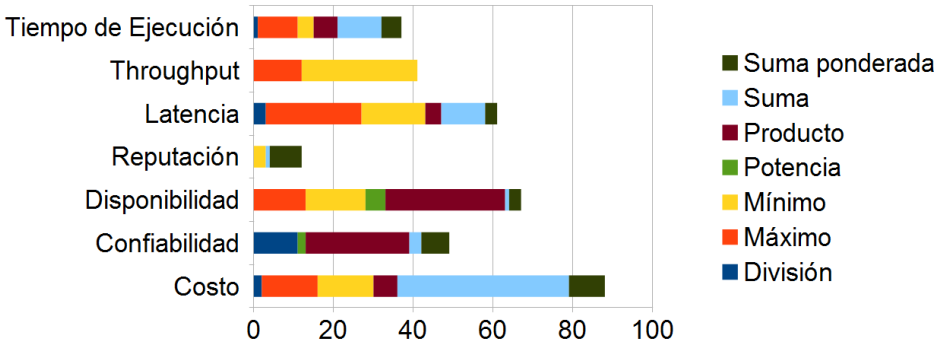

Figura 4.8: Número de formulas por operador de agregación y atributo de calidad
Fuente: Elaboración propia

de flujo de trabajo. Los patrones en los que se han encontrado más fórmulas de agregación que usan el operador producto, son secuencia, LOOP, y AND.

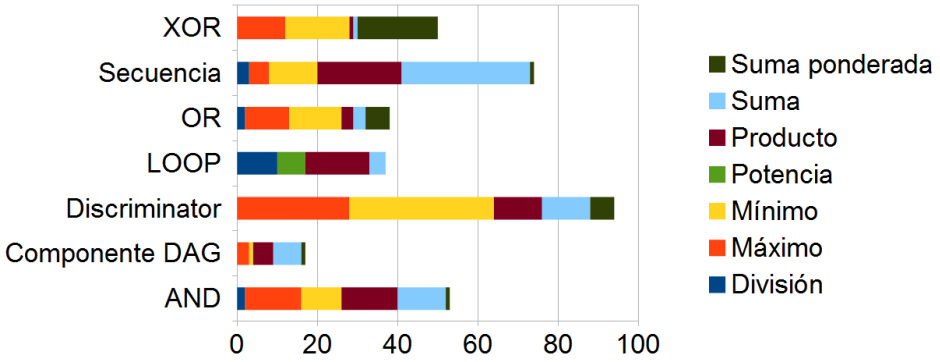

Figura 4.9: Número de formulas por operador de agregación y patrón de flujo e trabajo
Fuente: Elaboración propia

La suma ponderada se observa con mayor frecuencia en el patrón XOR, que como ya se mencionó, es un operador relacionado con el factor de uso. En muchas de las fórmulas para XOR, el factor de uso se expresa como la probabilidad de que cada componente de un ensamble sea ejecutado, y se usa para ponderar la calidad de cada componente antes de emplear el operador suma para concluir con la agregación. Aunque a simple vista, los patrones OR y discriminador, también parecen ser dependientes del uso, la suma ponderada se observa con menor frecuencia en ellos.

El operador suma ($\sum$) se observa con mayor frecuencia en el atributo costo y el patrón secuencia. Además, el 100 % de las fórmulas donde el atributo es costo y el patrón secuencia han usado el operador suma, lo que confirma la relación entre estos tres conceptos.

Los operadores potencia y división, se observan con mayor frecuencia en el patrón LOOP, que como ya se demostró utilizan estos dos tipos de operadores en fórmulas donde se requiere estimar el número de iteraciones que serán ejecutadas. En muchos de los casos las fórmulas son equivalentes, por lo que a pesar de usar distintos operadores, producen el mismo resultado.

Los operadores máximo (max) y mínimo (min) se observan con mucha frecuencia en el patrón discriminator, principalmente, en la intersección de este patrón con los atributos latencia y "throughput". En el caso de "throughput", el operador predominante es el operador min, mientras que en latencia, predomina el operador max, que sirven para calcular el peor "throughput" y la peor latencia del ensamble, respectivamente.

4.3. ¿Qué tan Completas son las Fórmulas de Agregación Existentes?

Para evaluar qué tan completas son las fórmulas de agregación, empezaremos hablando acerca de el rendimiento. La mayoría de las fórmulas de agregación para el tiempo de ejecución, que consideran los patrones AND y secuencia, aparentan ser diréctamente agregables, aunque como ya se mencionó, en este trabajo se les considera dependientes de la arquitectura por el simple hecho de basarse en patrones de flujo de trabajo, además, en [11] se afirma que el tiempo de ejecución depende de otros atributos de calidad de los componentes del ensamble de software. La postura del presente trabajo, es que ambos tienen algo de cierto. En el contexto de los servicios web, que es donde se desenvuelve la mayoría de los trabajos que han descrito fórmulas de agregación, el software reside en un único equipo de hardware, por lo que en muchos casos estimar el tiempo de ejecución, puede realizarse sin necesidad de tomar en cuenta cualquier otra característica de los componentes; sin embargo, aquellos contextos donde el software está hecho para ser instalado en diferentes equipos de hardware, será imposible agregar, correctamente, el tiempo de ejecución de un ensamble de componentes si no se consideran características como la velocidad de transmisión de datos, el tiempo de ejecución de los componentes y la prioridad de las tareas en cada componente. Las aplicaciones de escritorio son un ejemplo de este tipo de contextos, ya que el desarrollador de software no pude conocer con exactitud las condiciones de los equipos sobre los cuales se ejecutará el software.

Muchos de los trabajos que tratan de estimar confiabilidad y disponibilidad, suelen ignorar la tolerancia a fallos en sus fórmulas de agregación. Aunque la arquitectura de un ensamble de componentes de software no siempre está pensada para la tolerancia a fallos, no hay que pasar por alto las recomendaciones propuestas en [17] y [16], ya que la arquitectura con tolerancia a fallos es una de las medidas más comunes para incrementar la fiabilidad de un sistema de software, particularmente en términos de confiabilidad y disponibilidad.

La variabilidad en las líneas de producto de software, también deberá ser considerada en la estimación de la calidad de ensambles de software, cuando así se requiera. Afortunadamente, la propuesta de [27] se adapta muy fácilmente a las fórmulas de agregación propuestas por otros autores, ya que únicamente añade condiciones relativas a la variabilidad, dentro de las fórmulas de agregación.

4.4. ¿Cómo Identificar el Operador de Composición Más Adecuado?

En el Capítulo 3 se presentan algunas fórmulas de agregación de la calidad que a pesar de ser diferentes son empleadas para estimar la calidad del mismo atributo, en ensambles de componentes de software que siguen el mismo patrón de flujo de trabajo. Eso significa que ambas fórmulas pueden ser candidatas para estimar la calidad de un mismo ensamble de componente de software. En esta subsección se mencionarán algunos criterios que pueden ayudar a decidir, con base en el operador de agregación, qué fórmula es la más adecuada en ciertos casos.

A partir de ahora se analizarán los operadores de agregación, empezando por los operadores promedio y suma ponderada, para estimar atributos de calidad en ensambles tipo XOR. Aunque el promedio, suele ser una medida representativa de un conjunto de datos, estimar la calidad de un ensamble de componentes de software, promediando la calidad ofrecida por cada uno de los componentes no resultará de utilidad, a menos que la calidad promedio coincida con la calidad del componente con mayor probabilidad de ser ejecutado. A diferencia del promedio, la suma ponderada con base en la probabilidad de ejecución de cada componente ofrecerá una estimación más cercana a la realidad, en la mayoría de los casos, debido a que los componentes con mayor probabilidad de ejecución tendrán mayor impacto en la calidad del ensamble de software. La estimación será aún más confiable en aquellos casos donde las probabilidades tengan una distribución normal.

La suma ponderada ofrecerá al usuario un único valor, dándole cierta seguridad de que en la mayoría de las ejecuciones, la calidad será cercana a la calidad estimada, sin embargo, esta información no será de gran utilidad cuando el usuario desee poder medir esta seguridad, en cuyo caso lo recomendable sería usar el enfoque probabilístico descrito en [18]. Este enfoque probabilístico resulta particularmente útil cuando se desea estimar el costo de ensambles de software, ya que a diferencia de los atributos de rendimiento y fiabilidad, que pueden ofrecer infinidad de valores de calidad, los posibles costos de un ensamble de componentes de software suelen ser muy limitados. Asumiendo que un ensamble de tipo XOR contiene 3 componentes y cada componente tiene un único costo asignado, el ensamble únicamente tendrá 3 posibles costos. En este caso, una suma ponderada probablemente dará como resultado un costo relativamente aproximado al costo que se presentará con mayor frecuencia, pero que en el peor de los casos será menor que el costo con mayores probabilidades. Esta estimación puede resultar insatisfactoria para el usuario cuando este se percate de que la mayoría de las veces el costo de ejecución del ensamble es el mismo y que es considerablemente mayor que el costo estimado. En este caso, lo que probablemente sea más útil es informar al usuario acerca de la probabilidad de cada costo, de manera que este le encuentre sentido a los resultados obtenidos tras la ejecución del ensamble. Este es otro punto a favor de la propuesta descrita en [18]. Aún así, la suma ponderada sigue siendo una opción muy aceptable en el caso de otros atributos de calidad, pero este dato será más ilustrativo siempre que vaya acompañado de la probabilidad correspondiente a este resultado.

La suma ponderada también es más recomendable que la información relacionada con valores máximos y mínimos en aquellos casos donde lo que interesa al usuario es el valor con mayor probabilidad; sin embargo, como ya se expuso en la subsección anterior, si el usuario

Tabla 4.2: Información ofrecida por diferentes modelos de fórmulas de agregación
Fuente: Elaboración propia

Modelo	Tipo	Información
$min_{i=1}^{n} q(c_i)$	Mínimo	La mejor calidad ofrecida por el ensamble. Ej. La menor latencia en un ensamble XOR. La peor calidad ofrecida por el ensamble. Ej. El menor "throughput" en un ensamble secuencial.
$max_{i=1}^{n} q(c_i)$	Máximo	La mejor calidad ofrecida por el ensamble. Ej. La mayor confiabilidad en un ensamble XOR. La peor calidad ofrecida por el ensamble. Ej. La mayor latencia en un ensamble AND.
$\frac{1}{n} \cdot \sum_{i=1}^{n} q(c_i)$	Promedio	La calidad promedio entre las operaciones. Ej. La confiabilidad promedio entre los componentes del ensamble.
$\sum_{i=1}^{n} q(c_i) \cdot p(i)$	Suma ponderada	La calidad más probable del ensamble. Ej. La latencia en un ensamble XOR.
$l \cdot q$	Producto de l y q	La calidad de un ensamble LOOP cuando la calidad de la operación debe ser sumada por cada ejecución y se conoce el número de iteraciones. Ej. El costo en un ensamle LOOP.
$\sum_{i=1}^{n} l \cdot q(c_i) \cdot p(l)$	Suma ponderada del producto de l y q	La calidad más probable de un ensamble loop cuando la calidad de la operación debe ser sumada por cada ejecución y se desconoce el número de iteraciones pero se conoce la probabilidad de que la operación se ejecute l veces.
$q(1-n)^{-1}$	Potencia (probabilidad de seguir iterando)	La calidad más probable de un ensamble loop cuando la calidad de la operación debe ser sumada por cada ejecución y se desconoce el número de iteraciones, pero se conoce la probabilidad de permanecer en el ciclo.
q^l	Potencia con base q y exponente l	La calidad de un ensamble LOOP cuando la calidad de la operación debe ser multiplicada por cada ejecución y se conoce el número de iteraciones. Ej. La confiabilidad en un ensamble LOOP.
$\sum_{i=1}^{n} q(c_i)^l \cdot p(l)$	Suma ponderada de la potencia de q^l	La calidad más probable de un ensamble loop cuando la calidad de la operación debe ser multiplicada por cada ejecución y se desconoce el número de iteraciones, pero se conoce la probabilidad de que la operación se ejecute l veces.
$q^{(1-n)^{-1}}$	Potencia de q^l (probabilidad de seguir iterando)	La calidad más probable de un ensamble loop cuando la calidad de la operación debe ser multiplicada por cada ejecución y se desconoce el número de iteraciones, pero se conoce la probabilidad de permanecer en el ciclo.

desea estar informado acerca de la calidad más deficiente ofrecida por el ensamble de software, la mejor opción serán los valores máximos y mínimos. En este trabajo se recomienda que, en la medida de lo posible, se utilicen ambos operadores de agregación y se especifique la calidad con base en todos estos resultados. De esta forma, será posible realizar ensambles de componentes de software que cumplan con diferentes tipos de restricciones, ya sean basadas en la probabilidad o en casos pesimistas.

Estos criterios se resumen en la Tabla 4.2 donde se muestra la información proporcionada por varios modelos de fórmulas de agregación, de manera que cada fórmula pueda ser seleccionada de acuerdo con la información que se desea obtener acerca del ensamble. Para no repetir las fórmulas presentadas en el Capítulo 3, las fórmulas de la Tabla 4.2 se han expresado en una forma más general, por lo que la variable q hace referencia a la calidad de uno o varios componentes, sin referirse a un atributo de calidad en particular; aún así, esto no significa que las fórmulas sean adecuadas para cualquier atributo de calidad; simplemente son modelos que permitirán al lector conocer qué información es posible obtener a partir de cada operador de agregación.

4.5. Resumen del Capítulo

En este capítulo se han comparado las fórmulas de agregación propuestas en la literatura usando como criterio los conceptos abordados a lo largo de este estudio, poniendo en evidencia la posible existencia de patrones de composición en cualquier fórmula de agregación, aún cuando dicha fórmula no contenga elementos visibles que sirvan como evidencia para relacionarla con el patrón correspondiente. También se han identificado relaciones entre las clasificaciones de las fórmulas y algunos patrones de flujo de trabajo, pero sobre todo se ha hecho evidente la influencia compartida entre el patrón de flujo de trabajo y el atributo de calidad en la clasificación de la fórmula de agregación.

Como parte de este análisis, también se ha concluido que ninguna de las propuestas en cuanto a fórmulas de agregación se puede considerar completa, ya que se ha demostrado que algunos autores no toman en cuenta la existencia de patrones de composición, mientras que otros ignoran la necesidad de fórmulas derivadas en atributos de calidad como el tiempo de ejecución o la confiabilidad en un ensamble con tolerancia a fallos.

Finalmente se mencionaron algunos aspectos, relacionados con los operadores aritméticos usados en las fórmulas de agregación, que deberán considerarse al momento de elegir entre varias fórmulas para agregar la calidad de ensambles de componentes de software.

Con la información proporcionada en este trabajo, será posible continuar definiendo técnicas de estimación de la calidad en ensambles de componentes de software en forma más ordenada y completa, siempre que se tomen en cuenta todos los aspectos recolectados y expuestos en este estudio del estado del arte. Aquellos que no deseen desarrollar nuevas técnicas de estimación de la calidad pero deseen llevar a cabo estimaciones con base en las técnicas existentes, pueden usar este trabajo como guía para seleccionar la técnica de estimación que será empleada en sus proyectos. El comprender en su totalidad las fórmulas de agregación explicadas en este trabajo, permitirá, a quienes deseen llevar a la práctica estas técnicas, combinar partes de distintas fórmulas para ajustarlas a las necesidades de cada proyecto de desarrollo de software y, en el mejor de los casos, extenderlas para resolver problemas que no han sido abordados en la literatura.

Capítulo 5

Conclusiones y Trabajo Futuro

En el Capítulo 1 se planteó como primer objetivo de este estudio del estado del arte, la recolección de un conjunto de trabajos en donde se describan técnicas para estimar la calidad de ensambles de componentes de software a partir de la información de calidad provista por los componentes que constituyen dicho ensamble. Estos trabajos se describen en el Capítulo 3 de esta tesis, donde se dan a conocer las técnicas de estimación de la calidad de ensambles de componentes de software propuestas por varios autores junto con otros aspectos relacionados con el proceso de estimación, como son la sintaxis y la semántica. En total se recolectaron 37 artículos que datan de los años 1995 a 2012. De estos artículos, 21 aportaron fórmulas de agregación de la calidad en ensambles de componentes de software, de los cuales, 20 artículos contienen fórmulas basadas en patrones de flujo de trabajo.

Como segundo objetivo se propuso analizar y clasificar dichas técnicas de acuerdo con sus características principales. Dichas fórmulas fueron clasificadas en el Capítulo 3 y analizadas en el Capítulo 4. Como se inidicó antes, todas las técnicas analizadas proponen el empleo de fórmulas de agregación en conjunto con el SWR. La clasificación propuesta fue tomada de [11], que originalmente correspondía a los tipos de atributos de calidad. En este estudio se le ha dado un nuevo enfoque, donde las clasificaciones corresponden con el tipo de fórmula, el cual se ve influenciado por el atributo de calidad y el patrón de composición del ensamble de componentes de software. La clasificación resultante consta de las siguientes 5 categorías:

- Diréctamente Agregable
- Dependiente de la Arquitectura
- Derivada
- Dependiente del Uso
- Dependiente del Contexto

En este estudio se determinó que todas aquellas fórmulas que fueran definidas de acuerdo con algún patrón de composición serían dependientes de la arquitectura, por lo que cerca del 99 % de las formulas presentadas se encuentran dentro de esta clasificación, mientras que sólo una de las fórmulas ha sido tomada como diréctamente agregable, puesto que así se

manifiesta en [11], sin embargo, en este trabajo, se ha mencionado en varias ocasiones la teoría de que cualquier fórmula de agregación podría tener implícitos uno o varios patrones de composición, lo que pone en duda la existencia de fórmulas diréctamente agregables, de acuerdo con el criterio presentado en este estudio.

De las fórmulas encontradas, no se identificó una sola que, en apariencia fuera dependiente del contexto, probablemente debido a la complejidad implicada en esta categoría, según [11].

Además de esto, en el Capítulo 4, se realizó una comparación de las fórmulas de agregación, de acuerdo con el atributo de calidad para el cual fueron definidas y el patrón de composición usado en el ensamble. Como resultado de esta comparación se llegó a la conclusión de que existe una fuerte relación entre los patrones XOR y LOOP y las fórmulas dependientes del uso. También se encontró relación entre el patrón secuencia y el atributo costo con el operador suma. Los valores máximos y mínimos han sido utilizados en la mayoría de las fórmulas de agregación para el patrón discriminador, así como la suma ponderada para el patrón XOR.

Entre los patrones de flujo de trabajo, el que se ha hecho presente en más trabajos dentro de la literatura ha sido el patrón secuencia, seguido de AND, LOOP y XOR, respectivamente. Los patrones OR, discriminador y DAG no se han tomado en cuenta en muchos trabajos, ya que se trata de patrones menos comunes y mucho más complejos.

De los atributos de calidad, los atributos de rendimiento se han hecho presentes en la mayoría de los trabajos, siendo latencia el más mencionado y consumo de recursos el menos mencionado. Los atributos de fiabilidad y el costo también se han hecho presentes en un gran número de trabajos. El atributo de calidad para el que menos trabajos han definido fórmulas de agregación es la seguridad, que como se menciona en [11] resulta un atributo de calidad muy retador debido a su dependencia del contexto, sin embargo, se ha demostrado que el nivel de encriptación si puede ser estimado con respecto al nivel de encriptación de los componentes que constituyen un ensamble.

El último objetivo consistía en el análisis de las fortalezas y debilidades de las técnicas recolectadas, mismo que se realizó en el capítulo 4, secciones 4.3 y 4.4, donde se concluyó que no existe una propuesta que considere todos los aspectos necesarios para la estimación de la calidad en ensambles de componentes de software. Además de eso, se establecieron algunos criterios que deberán tomarse en cuenta para seleccionar una fórmula de agregación; entre ellos la necesidad de valores mínimos y máximos, cuando se desea conocer los peores escenarios posibles de calidad; la suma ponderada para incrementar la precisión en la estimación de la calidad; y la utilidad de agregar un enfoque probabilístico al SWR.

Finalmente, en respuesta a la pregunta planteada en el Capítulo 4 ¿De qué depende la estimación de la calidad de un ensamble de componentes de software? De acuerdo con lo que que se observó en la sección 4.1, Los dos principales criterios para definir una técnica de estimación son el atributo que se desea estimar y los patrones de composición utilizados en la construcción del ensamble. El patrón de composición seguramente tendrá una mayor influencia en la mayoría de las características de la fórmula de agregación, salvo por el operador de agregación, que en la mayoría de los casos, dependerá, por igual, del atributo de calidad y del patrón de composición. Además de esto, la arquitectura añadirá variables a la fórmula de agregación, generalmente, sin alterar la estructura básica, determinada por el patrón de composición y el atributo de calidad. También se debe tomar en cuenta que algunos atributos de calidad son derivados por naturaleza, como es el caso del tiempo de ejecución y el MTTF,

por lo que será necesario incluir la derivación de la calidad de estos atributos antes, o durante la fórmula de agregación.

Con esto llegamos a la conclusión de que las técnicas de estimación de la calidad en ensambles de componentes de software existentes, permiten realizar dicha estimación en forma automática, sin embargo hasta el momento no se ha encontrando una técnica lo suficientemente robusta, que asegure una estimación precisa considerando todas las variables que pudieran influir en la calidad de distintos ensambles de software.

5.1. Trabajo por Realizar

Como ya se mencionó, la gran mayoría de los trabajos relacionados se han enfocado en servicios web, sin embargo, este es sólo uno de los contextos de aplicación donde se requiere realizar estimaciones de calidad, por lo que aún existe mucho trabajo por realizar con respecto al tema de la estimación de la calidad de ensambles de componentes.

En cuanto a la estimación de la calidad en el contexto de los servicios web, los próximos trabajos deberán usar la información encontrada para comprobar su efectividad en la estimación de la calidad de los servicios web; especialmente en lo que corresponde al rendimiento, ya que aunque las fórmulas de agregación para estimar atributos como latencia, tiempo de ejecución y "throughput" han sido respaldadas por muchos trabajos, la mayoría de ellos no comparte la visión de [11] acerca de atributos de calidad como el tiempo de ejecución, cuyas reglas de estimación obligan a tomar en consideración otros atributos de calidad de los componentes que constituyen el ensamble, de las cuales ya se ha hablado anteriormente. En [11], también es el único trabajo donde se ha hablado acerca del impacto del número de solicitudes simultáneas, en el rendimiento, que como se mencionó anteriormente, pudieran prolongar la respuesta por parte del componente, debido al tiempo que permanecen las solicitudes en cola de espera. Aunque en [11] el enfoque no es hacia los servicios web, es necesario identificar qué partes de su propuesta tienen aplicación en este contexto.

Los próximos trabajos también deberán definir reglas de estimación para otros contextos y arquitecturas, como por ejemplo la arquitectura orientada a eventos. En este tipo de arquitecturas, la latencia de los componentes dependen de la frecuencia con que estos envían y reciben eventos, además de los tiempos de ejecución de cada componente. Para aclarar este punto pondremos el siguiente ejemplo: supongamos que tenemos un sistema que cuenta con un sensor de temperatura dentro de los refrigeradores donde se almacenan carnes en un supermercado. Cada 2 minutos, el sensor envía un evento al manejador de eventos a través de la red local, indicando la temperatura actual del refirgerador. El software de monitoreo de temperatura solicita cada 2 minutos la temperatura del refirgerador y en caso de detectar que la temperatura se encuentra por encima del límite establecido, enciende una alarma en el área de mantenimiento, para que el problema con el refrigerador se resuelva antes de que ocurran pérdidas a causa del incremento de temperatura. En teoría el tiempo máximo que debería tardar la alarma en encenderse depués de que la temperatura del refrigerador exceda del límite establecido, debería ser cercano a 4 minutos, a pesar de que ambos componenes sean ejecutados en paralelo. En la Figura 5.1 se intenta ilustrar este ejemplo. El ejemplo, con fines ilustrativos, se encuentra incompleto, ya que no se toma en cuenta, el tiempo de transmisión a través de la red y el tiempo de ejecución de cada uno de los componentes,

que también podría incrementar el tiempo transcurrido entre el incremento de temperatura y el encendido de la alarma, sobre todo, en casos donde la cola de eventos pudiera volverse demasiado larga a causa de otros sensores que también envían eventos al administrador. Otro aspecto, a tomar en cuenta, es la prioridad del monitor de temperatura con respecto a otras tareas, si es que el dispositivo donde se aloja este componente tuviera otras tareas asignadas.

Así como la estimación de la latencia en ensambles de componentes que usan la arquitectura orientada a eventos requiere consideraciones adicionales. Otras arquitecturas aún no exploradas en este tema, seguramente representarán retos que no pueden ser superados con las fórmulas de agregación propuestas hasta el momento.

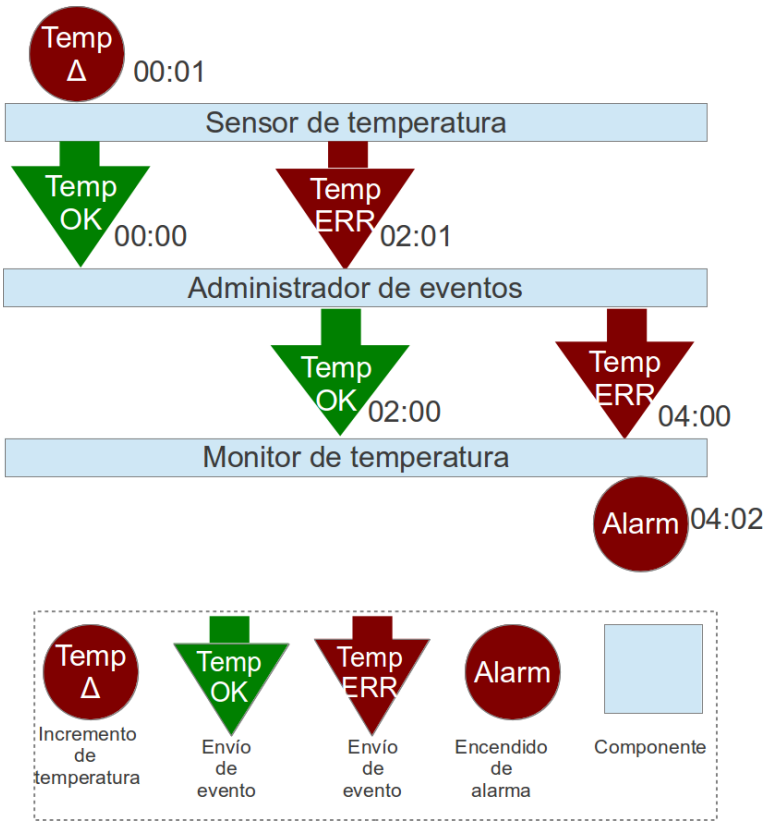

Figura 5.1: Monitoreo de temperatura con una arquitectura orientada a eventos
Fuente: Elaboración propia

Los trabajos futuros también deben evaluar la manera de calcular los atributos de calidad relacionados con las líneas de tiempo, como latencia y tiempo de ejecución, en ciertos entornos de desarrollo. Está por ejemplo el caso de java, donde la máquina virtual ejecuta procesos adicionales, como el recolector de basura para borrar de la memoria aquellos objetos que ya no son requeridos. Otro problema con respecto a las aplicaciones de java, es que, en ocasiones, la latencia de los componentes es muy prolongada, durante las primeras ejecuciones y en ocasiones, se logra estabilizar hasta después de varias ejecuciones, lo que provoca resultados muy irregulares al momento de medir este atributo de calidad. Una solución a estos problemas sería modificar la configuración de la máquina virtual de java, de manera que se inhiba la ejecución del recolector de basura, cosa que también podría provocar un desbordamiento de la memoria, además de que al llevar el software a producción la latencia estimada no

correspondería con la latencia real. Por esta razón es importante evaluar hasta qué punto se debe controlar el ambiente de ejecución para medir la calidad de los componentes de software.

En el futuro, también se debe determinar la unidad de medida idonea para los atributos de rendimiento, ya que en ciertos contextos, los componentes serán ejecutados en dispositivos con diferentes recursos de hardware. Esto provoca que la latencia, tiempo de ejecución y "throughput" sean diferentes en cada dispositivo. Esto es equivalente a usar partes del cuerpo humano como unidades de medida, como se hacía en la antigüedad; ya que cada individuo es distinto, también lo es el tamaño de su cuerpo y por lo tanto sus medidas también resultarán distintas. Así como se han definido unidades de medida estándar como lo son, actualmente, el pié, la pulgada, el métro, etcétera, quizás también habría que establecer una unidad de medida estándar para el rendimiento del software, de manera que sirva como un buen punto de comparación independientemente de los recursos de hardware destinados a cada componente de software.

El crear una unidad de medida universal para el rendimiento, es un proyecto ambicioso, ya que no es algo que una persona o un grupo de personas independientes deban realizar. Es necesario determinar una base de medición, como puede ser un equipo con ciertos recursos de hardware. Posteriormente se deberá establecer una escala para calificar equipos con diferentes recursos de hardware y finalmente, determinar la manera como se calcularán los tiempos, o en su caso, el "throughput", a través de la calificación del software y del hardware donde este software será implementado. En [4] se presenta un proceso parecido a lo que se está proponiendo, donde la unidad de medida, es conocida como RDSEFF, como se mencionó en el Capítulo 3. Aunque la propuesta resulta muy convincente, aún existe trabajo por realizar, para que los usuarios del software puedan contar con una unidad de medida universal, que a través de cierta herramienta de software, les permita conocer la latencia, el tiempo de ejecución y el "throughput", en unidades de tiempo, para un hardware específico.

En los trabajos futuros, también se deberá analizar el impacto de otros patrones de composición, distintos a los patrones de flujo de trabajo, en la agregación de la calidad de ensambles de sofware. En arquitectura de software, se observa la existencia de dos grupos de patrones, llamados *patrones arquitectónicos* y *patrones de diseño*. Entre los patrones de diseño se incluyen algunos que son homólogos a los patrones de flujo de trabajo que se mencionan en esta tesis. La existencia de patrones de flujo de trabajo entre los patrones de diseño abre la posibilidad de que existan otros patrones de diseño que también influyan en la calidad de un ensamble de software, sin embargo, esto no se ha explorado en los trabajos que fueron revisados durante este estudio, o por lo menos, en ninguno de los trabajos se ha concluido acerca de esto. En el caso de los patrones arquitectónicos, se encontró un trabajo que habla acerca del impacto de la arquitectura en capas sobre la calidad de un ensamble de software [25]. En este trabajo, se afirma que el impacto de la calidad debe ponderarse de acuerdo con la capa en la que se encuentran los componentes, considerando que las capas inferiores deben tener mayor peso que las capas superiores, sin embargo, no se han encontrado fórmulas de agregación de la calidad que incluyan entre sus variables la capa donde se sitúan los componentes del ensamble.

Bibliografía

[1] Mathieu Acher and Philippe Collet. Managing variability in workflow with feature model composition operators. *Software Composition*, 2010.

[2] Mohammad Alrifai and Thomas Risse. Combining global optimization with local selection for efficient QoS-aware service composition. *Proceedings of the 18th international conference on World wide web - WWW '09*, page 881, 2009.

[3] M Barbacci, MH Klein, TA Longstaff, and CB Weinstock. Quality Attributes. (December), 1995.

[4] Steffen Becker, Heiko Koziolek, and Ralf Reussner. The Palladio component model for model-driven performance prediction. *Journal of Systems and Software*, 82(1):3–22, January 2009.

[5] Y Benkaouz and M Erradi. A Distributed Protocol for Privacy Preserving Aggregation. *Networked Systems*, 2013.

[6] A Bhuvaneswari. QoS Considerations for a Semantic Web Service Composition. *European Journal of Scientific Research*, 65(3):403–415, 2011.

[7] Gerardo Canfora and Massimiliano Di Penta. A lightweight approach for QoS-aware service composition. 2004.

[8] C Cappiello, M Matera, M Picozzi, F Daniel, and A Fernandez. Quality-Aware Mashup Composition : Issues , Techniques and Tools. 2010.

[9] Jorge Cardoso. Applying Data Mining Algorithms to Calculate the Quality of Service of Workflow Processes. *Intelligent Techniques and Tools for Novel System*, 2008.

[10] Jorge Cardoso, Amit Sheth, John Miller, Jonathan Arnold, and Krys Kochut. Quality of service for workflows and web service processes. *Web Semantics: Science, Services and Agents on the World Wide Web*, 1(3):281–308, April 2004.

[11] Ivica Crnkovic, Magnus Larsson, and Otto Preiss. Concerning Predictability in Dependable Component- Based Systems : Classification of Quality Attributes. pages 257–278, 2005.

[12] Scott G Decker and Jim Dager. Software Product Lines Beyond Software Development. *11th International Software Product Line Conference (SPLC 2007)*, pages 275–280, September 2007.

[13] Marlon Dumas and L García-Bañuelos. Aggregate quality of service computation for composite services. *Service-Oriented Computing*, 2010.

[14] Thomas Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall, 2005.

[15] M Oriol Hilari. Quality of Service (QoS) in SOA Systems. A Systematic Review. 2009.

[16] Tao Hu, Minyi Guo, Song Guo, Hirokazu Ozaki, Long Zheng, Kaori Ota, and Mianxiong Dong. MTTF of Composite Web Services. *International Symposium on Parallel and Distributed Processing with Applications*, pages 130–137, September 2010.

[17] Tao Hu, Song Guo, Minyi Guo, Feilong Tang, and Mianxiong Dong. Analysis of the Availability of Composite Web Services. *2009 Fourth International Conference on Frontier of Computer Science and Technology*, pages 231–237, December 2009.

[18] San-Yih Hwang, Haojun Wang, Jian Tang, and Jaideep Srivastava. A probabilistic approach to modeling and estimating the QoS of web-services-based workflows. *Information Sciences*, 177(23):5484–5503, December 2007.

[19] MC Jaeger. Qos aggregation for web service composition using workflow patterns. *Enterprise Distributed Object Computing Conference, 2004. EDOC 2004. Proceedings. Eighth IEEE International*, 2004.

[20] MC Jaeger. QoS aggregation in Web service compositions. *Proceedings of IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE-05)*, 2005.

[21] Kamil Ježek, Přemek Brada, and Petr Štěpán. Towards Context Independent Extra-functional Properties Descriptor for Components. *Electronic Notes in Theoretical Computer Science*, 264(1):55–71, August 2010.

[22] Magnus Larsson. *Predicting Quality Attributes in Component-based Software Systems*. Number 8. 2004.

[23] KK Lau, Ioannis Ntalamagkas, CM Tran, and T Rana. (Behavioural) design patterns as composition operators. *Component-Based Software Engineering*, pages 232–251, 2010.

[24] Kung-Kiu Lau and Tauseef Rana. A Taxonomy of Software Composition Mechanisms. *2010 36th EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 102–110, September 2010.

[25] Chengpu Li, R Pooley, and Xiaodong Liu. Ontology-Based Quality Attributes Prediction in Component-Based Development. 2010.

[26] Jiang Ma and Hao-peng Chen. A Reliability Evaluation Framework on Composite Web Service. *2008 IEEE International Symposium on Service-Oriented System Engineering*, (2007):123–128, December 2008.

[27] Bardia Mohabbati, D Gašević, and M Hatala. A quality aggregation model for service-oriented software product lines based on variability and composition patterns. *Service-Oriented Computing*, pages 1–18, 2011.

[28] G. Pour. Towards component-based software engineering. *Proceedings. The Twenty-Second Annual International Computer Software and Applications Conference (Compsac '98) (Cat. No.98CB 36241)*, page 599, 1998.

[29] Sidney Rosario and Albert Benveniste. Probabilistic qos and soft contracts for transaction-based web services orchestrations. *Services Computing, IEEE Transactions on (Volume:1 , Issue: 4)*, 2008.

[30] M Rosenmüller and Norbert Siegmund. Code generation to support static and dynamic composition of software product lines. *GPCE '08 Proceedings of the 7th international conference on Generative programming and component engineering*, 2008.

[31] Dieter Schuller, Artem Polyvyanyy, Luciano Garcia-banuelos, Stefan Schulte, and Luciano Garc. Optimization of Complex QoS-Aware. (December):452–466, 2011.

[32] S Sentilles and P Štěpán. Integration of extra-functional properties in component models. *Component-Based Software Engineering*, 2009.

[33] Approved September. IEEE standard glossary of software engineering terminology. 121990, 1990.

[34] Rafael Tolosana-calasanz, Omer F Rana, and A Ba. Automating Performance Analysis from Taverna Workflows. 5282:1–15, 2008.

[35] Perla Velasco, Mbe Koua, and Christophe Ndjatchi. Deriving functional interface specifications for composite components. *Software Composition*, pages 1–17, 2011.

[36] Perla Velasco Elizondo and Kung-Kiu Lau. A catalogue of component connectors to support development with reuse. *Journal of Systems and Software*, 83(7):1165–1178, July 2010.

[37] Jingbo Xu, Hailong Sun, Xu Wang, Xudong Liu, and Richong Zhang. Optimizing pipe-like mashup execution for improving resource utilization. *2012 Fifth IEEE International Conference on Service-Oriented Computing and Applications (SOCA)*, pages 1–4, December 2012.

[38] Tao Yu, Yue Zhang, and Kwei-Jay Lin. Efficient algorithms for Web services selection with end-to-end QoS constraints. *ACM Transactions on the Web*, 1(1):6–es, May 2007.

[39] Liangzhao Zeng, Boualem Benatallah, Marlon Dumas, Jayant Kalagnanam, and Quan Z. Sheng. Quality driven web services composition. *Proceedings of the twelfth international conference on World Wide Web - WWW '03*, page 411, 2003.

Anexos

CIMAT

CENTRO DE INVESTIGACIÓN EN MATEMÁTICAS, A.C.

BIBLIOTECA

AUTORIZACION
PUBLICACION EN FORMATO ELECTRONICO DE TESIS

El que suscribe

Autor(s) de la tesis: David Alfredo Barredo Hernández

Título de la tesis: Análisis y Clasificación de Técnicas Basadas en Patrones de Flujo de Trabajo para la Estimación de la Calidad de Ensambles de Componentes de Software

Institución y Lugar: Centro de Investigación en Matemáticas A.C. Unidad Zacatecas, Zacatecas, Zac.

Grado Académico: Licenciatura ( ) Maestría (✓) Doctorado ( ) Otro ( ) ----------

Año de presentación: 2013

Área de Especialidad: Ingeniería de Software

Director(es) de Tesis: Dra. Perla Velasco Elizondo y Dra. Alejandra García Hernández

Correo electrónico: davofredo@cimat.mx, davofredo@gmail.com

Domicilio: 53B #379 entre 56 y 58, fraccionamiento Fco. de Montejo, C.P. 97203 Mérida, Yucatán, México

Palabra(s) Clave(s): Estimación de la calidad, componentes de software, ensamble de componentes, componentes compuestos

Por medio del presente documento autorizo en forma gratuita a que la Tesis arriba citada sea divulgada y reproducida para publicarla mediante almacenamiento electrónico que permita acceso al público a leerla y conocerla visualmente, así como a comunicarla públicamente en la Página WEB del CIMAT.

La vigencia de la presente autorización es por un periodo de 3 años a partir de la firma de presente instrumento, quedando en el entendido de que dicho plazo podrá prorrogar automáticamente por periodos iguales, si durante dicho tiempo no se revoca la autorización por escrito con acuse de recibo de parte de alguna autoridad del CIMAT

La única contraprestación que condiciona la presente autorización es la del reconocimiento del nombre del autor en la publicación que se haga de la misma.

Atentamente

David Alfredo Barredo Hernández

Nombre y firma del tesista

CALLE JALISCO S/N MINERAL DE VALENCIANA APDO. POSTAL 402
C.P. 36240 GUANAJUATO, GTO., MÉXICO
TELÉFONOS (473) 732 7155, (473) 735 0800 EXT. 49609 FAX. (473) 732 5749
E-mail: biblioteca@cimat.mx