ICT-UNPA-26-2011
ISSN: 1852-4516
Aprobado por Resolución Nº 0509/11-R-UNPA

SEPARACIÓN AVANZADA DE CONCERNS PARA DESARROLLAR APLICACIONES GRID

Natalia B. Trejo, Sandra I. Casas
natalia.trejo@gmail.com, scasas@unpa.edu.ar

Unidad Académica Río Gallegos – Universidad Nacional de la Patagonia Austral
Lisandro La Torre 1070 – 0054-2966-4423113 – Río Gallegos – Santa Cruz – Argentina
Marzo 2011

RESUMEN

El desarrollo de aplicaciones para Grid computing, puede beneficiarse si se aplican enfoques emergentes de Ingeniería de Software basados en la Separación Avanzada de Concerns. En el presente trabajo analizamos los más representativos, de qué manera se aplicaron en el área de Grid computing. Los enfoques abordados incluyen Desarrollo de Software Dirigido por Modelos, Desarrollo de Software Basado en Componentes, Desarrollo de Software Orientado a Aspectos, Ingeniería de Línea de Productos y Desarrollo de Software Orientado a Features. Se analizan y comparan estos enfoques de Separación Avanzada de Concerns para determinar de qué manera y en qué grado cada uno de ellos satisfacen las necesidades que se requieren para desarrollo de aplicaciones para Grid computing.

Palabras claves: servicios Grid, Separación Avanzada de Concerns, aplicaciones Grid, Desarrollo de Software Orientado a Features

INTRODUCCIÓN

La computación en Grid se refiere al entorno de computación conectado mediante Internet en el que los recursos de cómputo y de datos están dispersos geográficamente en diferentes dominios administrativos con diferentes políticas de seguridad y de uso. Los recursos son inherentemente heterogéneos, comprendiendo ordenadores, estaciones de trabajo, clústeres, supercomputadoras, software de aplicación, equipamiento científico, datos, almacenamiento, redes, sensores, etc. Con las tecnologías Grid se pueden construir aplicaciones a gran escala para ejecutarse sobre entornos Grid (Foster et al., 2001). Las aplicaciones Grid son, generalmente, aplicaciones distribuidas, heterogéneas y multitareas donde los componentes son tareas que pueden ser implementadas por aplicaciones arbitrarias. De esta forma, el desarrollo, implementación y ejecución de aplicaciones en estos entornos plantea importantes desafíos debido, por ejemplo, a la comunicación y coordinación entre las tareas componentes de las aplicaciones y a las fallas diversas y condiciones de error que pueden ocurrir durante la ejecución.

El desarrollo de una aplicación Grid a partir de la composición de diferentes servicios, puede enfocarse a partir de la Separación de Concerns (SoC), al considerar dichos servicios como distintas funcionalidades independientes y reutilizables. Así, distintas composiciones de servicios conforman diferentes aplicaciones Grid, lo que requiere a priori una composición correcta entre sus interfaces y restricciones (requisitos no funcionales, tales como las relacionadas

con la QoS[1], etc.), libre de interacciones no compatibles. Las interacciones no compatibles entre los servicios en principio pueden exhibir un comportamiento impredecible o no deseado, en el peor de los casos provocarán la interrupción de la ejecución de la aplicación. Las interacciones de este tipo además de imposibilitar la reutilización de los servicios, generan problemas en la evolución, mantenimiento y escalabilidad de las aplicaciones Grid.

El objeto del presente trabajo es realizar un estudio del estado del arte con respecto a los enfoques de ingeniería de software relacionados con SoC que se han utilizado en el desarrollo de aplicaciones para Grid computing, realizando una comparación de los resultados alcanzados en cada uno de ellos. A partir de las características particulares de las aplicaciones para entor-

[1] QoS: Quality of Service

nos Grid, describiremos cómo los enfoques pueden aportar sus beneficios en la creación y mantenimiento de estas aplicaciones complejas, distribuidas y altamente cambiantes.

El documento se organiza como sigue: la Sección 1 describe el concepto de aplicación para Grid computing y sus características particulares haciendo especial referencia a los workflows de servicios Grid. La Sección 2 describe los nuevos enfoques de Ingeniería de Software relacionados con la Separación Avanzada de Concerns. La Sección 3 presenta de que manera se aplicaron estos enfoques a Grid computing. La siguiente sección muestra los resultados logrados y los desafíos que aún restan ser abordados para el desarrollo y mantenimiento de aplicaciones para Grid computing. Por último presentamos las principales conclusiones.

1 APLICACIONES PARA GRID COMPUTING

1.1. GRID COMPUTING

Grid computing (Foster y Kesselman, 2004), como nuevo paradigma de computación distribuida, permite gestionar y utilizar de forma segura y coordinada recursos heterogéneos que pueden encontrarse distribuidos geográficamente y pertenecer a diferentes dominios administrativos. Los usuarios obtienen una visión unificada de dichos recursos y pueden acceder a ellos de manera transparente, esto facilita el trabajo colaborativo a gran escala. Grid computing permite alcanzar de forma segura y económica capacidades computacionales para ejecutar eficientemente aplicaciones intensivas en datos ó computación.

Por otro lado, el paradigma de Computación Orientada a Servicios (Service Oriented Computation, SOC) (Huhns y Singh, 2005) surge para facilitar el desarrollo de aplicaciones distribuidas a gran escala, independientes de la plataforma y a bajo costo, facilitando la automatización de procesos de negocio a través de elementos básicos de construcción llamados servicios. La arquitectura de software para este tipo de aplicaciones es conocida como Arquitectura Orientada a Servicios (Service Oriented Achitecture, SOA), en la cual los servicios Web proveen un marco de implementación.

Los servicios Web son componentes de software diseñados para proveer operaciones específicas (*servicios*) que se encuentran accesibles usando tecnologías estándares de Internet como XML y protocolos de red a la vez que son independientes de la plataforma (SOAP (W3C Consortium, 2007), Web Services (W3C Consortium, 2004), XML (W3C Consortium, 2008), HTTP (Fielding et al., 1999)). Son de interés para Grid computing ya que permiten implementar la infraestructura Grid a través de la comunicación entre componentes Grid distribuidos y también como interfaz uniforme para que los usuarios puedan acceder a los recursos Grid. Un escenario para la utilización de servicios Web es usarlos como front-end de aplicaciones existentes que operen sobre una plataforma Grid, esto se realiza encapsulando el código de la aplicación para producir un servicio Web y accediendo a ella mediante la interfaz de dicho servicio.

1.2. SERVICIOS WEB Y SERVICIOS GRID

Los servicios Web normalmente son *stateless*, es decir, que no mantienen información entre una invocación y la siguiente. En las aplicaciones para Grid computing es de utilidad que los servicios Web sean *stateful*, es decir, que conserven información entre invocaciones y entre invocaciones realizadas por diferentes clientes para permitir que una secuencia de acciones a ejecutarse pueda depender de acciones y resultados anteriores. Dicha información o estado se almacena en uno o varios recursos, los que se encuentran separados del servicio Web, el que actua como su front-end. Esta extensión de servicios Web para Grid computing se conoce como servicios Grid o servicios basados en WSRF (Web Service Resource Framework (Foster et al., 2004)) que a su vez se fundamenta en el estándar OGSA (Open Grid Services Architecture (Foster et al., 2005)).

SOA implica que los servicios de software distribuidos pueden ser libremente compuestos ó integrados a través de interfaces claramente definidas, esto se realiza a través de la orquestación o la coreografía (Peltz, 2003). Como los servicios Grid se basan en los servicios Web, la creación de aplicaciones Grid basadas en servicios Grid utilizará la orquestación de servicios. La integración se logra a través de una aplicación central, el sistema de gestión de workflows Grid, que permite modelar las interacciones entre los servicios que han sido compuestos a través de sus interfaces. El motor de workflows Grid toma la especificación de workflows y controla el orden de ejecución y el intercambio de mensajes.

1.3. WORKFLOWS GRID

Los desarrolladores de aplicaciones para Grid computing y los usuarios que despliegan y ejecutan aplicaciones en entornos Grid, precisan herramientas Grid en el nivel de aplicación (Foster y Kesselman, 2004). Es decir, software que se construya y utilice la infraestructura Grid proveyendo nuevas funcionalidades y abstracciones de alto nivel para facilitar las tareas de los desarrolladores y usuarios de Grid computing. Estas herramientas se sitúan sobre otros servicios Grid, en (Foster y Kesselman, 2004) se presenta una taxonomía de ellas, entre las que se encuentran aquellas relacionadas con los entornos de ejecución de aplicaciones Grid, en particular las conocidas como workflows.

Un workflow (Workflow Management Coalition, 1993) puede ser definido como la automatización de un proceso, formado por varios participantes, llamados sub-procesos, donde la información o datos se transfieren entre ellos basándose en un conjunto de reglas predefinidas para lograr un objetivo. Desde el punto de vista de Grid computing, una tarea en un Grid puede componerse a su vez de varias sub-tareas que frecuentemente son interdependientes.

Si consideramos los servicios Grid que forman la infraestructura Grid en muchos middleware, como Globus Toolkit (Globus Alliance, 1996), un nuevo servicio Grid puede estar formado a partir de la combinación de dos o más servicios Grid ya existentes. En este caso, un workflow define el orden de ejecución de estos servicios dentro del nuevo servicio Grid y los mensajes intercambiados entre ellos. Por lo tanto podría definirse un workflow Grid, como la automatización de una colección de servicios mediante la coordinación a través de dependencias de control y datos para resolver un problema específico ó generar un nuevo servicio Grid.

De esta forma, la clave para aprovechar los beneficios de la computación Grid es la capacidad de integrar los servicios básicos para crear aplicaciones Grid de alto nivel. Los lenguajes de workflows permiten la composición de tales servicios. Con ellos, la aplicación de más alto nivel puede ser modelada en forma abstracta (Deelman et al., 2004; Hernández et al., 2004; Neubauer et al., 2006), donde los nodos representan tareas mientras que las aristas, relacionando tales nodos, representan dependencias entre tareas, flujo de datos o control de flujo. Asimismo, las tareas pueden ser realizadas por los servicios Grid básicos.

Grid computing se utiliza ampliamente en comunidades científicas relacionadas con bioinformática, astronomía, física de altas energías, etc. Tales aplicaciones requieren la orquestación de diversos recursos Grid y la ejecución de complejos workflows. Las ventajas de los workflows son diversas (Jia y Rajkumar, 2005), permitiendo la construcción de aplicaciones dinámicas a partir de recursos heterogéneos distribuidos y provenientes de diferentes dominios administrativos.

Al existir esta afinidad natural entre workflows y Grid computing y al ser los servicios Grid extensión de los servicios Web, existe un creciente interés en el estudio de cómo los nuevos y emergentes paradigmas de IS pueden mejorar el desarrollo de aplicaciones basadas en servicios Grid.

1.4. ESPECIFICACIÓN Y COMPOSICIÓN DE SERVICIOS GRID

El Open Grid Forum (Open Grid Forum, 2005) ha adoptado el estándar OGSA para el modelamiento de los recursos del Grid, a través de servicios Grid. Dichos servicios están construidos en base a la tecnología de servicios Web. Globus Toolkit, a partir de su versión 4 (GT4), comienza a utilizar WSRF, el estándar para implementar servicios Grid que cumplen con los requerimientos OGSA, debido a que es una solución que, además de cumplir con los requerimientos de los servicios Grid, se mantiene fiel a los fundamentos de servicios Web.

WSRF hace una distinción explícita entre *servicio* y *recursos con estado* actuando bajo ese servicio. WSRF define los medios por los cuales un servicio Web y un recurso con estado se componen. WSRF denomina a esta composición WS-Resource. De acuerdo con WSRF, un recurso con estado tiene los datos que representan al mismo descriptos en un documento XML (llamado *Resource Properties*) que es conocido y accedido por uno o más servicios Web. Es importante notar que para los clientes, el servicio y los recursos son vistos como una misma entidad a través de una especificación WSDL que constituye la interfaz del mismo. Dichos clientes nunca tratarán directamente con instancias de los recursos sino que lo harán implícitamente a través de las interacciones con el servicio que cumple con la especificación WSRF. La implementación de WSRF implícitamente pasa la información de identificación del recurso cuando ocurre una interacción de mensajes entre un cliente y un WS-Resource. El cliente no tiene que incluir explícitamente un identificador de recursos en la solicitud.

WSDL permite que a través de un formato XML se describan servicios Web y servicios Grid. WSDL describe la interfaz publica de los servicios Web/Grid. Está basado en XML y describe la forma de comunicación, es decir, los requisitos del protocolo y los formatos de los mensajes necesarios para interactuar con los servicios listados en su catálogo. Las operaciones y mensajes que soporta se describen en abstracto y se vinculan después al protocolo concreto de red y al formato del mensaje. Así, WSDL se usa a menudo en combinación con SOAP y XML

Schema. Un programa cliente que se conecta a un servicio Web/Grid puede leer el WSDL para determinar qué funciones están disponibles en el servidor. Los tipos de datos especiales se incluyen en el archivo WSDL en forma de XML Schema. El cliente puede usar SOAP para hacer la llamada a una de las funciones listadas en el WSDL. Lo importante a tener en cuenta es que el único contacto entre los servicios Grid y sus usuarios es la interfaz de servicios. Esas interfaces de servicios son definidas por el WSDL existente.

Los usuarios de aplicaciones Grid frecuentemente modifican dichas aplicaciones para añadir, remover o modificar los servicios Grid sobre los que se basan tales aplicaciones, lo cual demuestra la volatilidad de los requisitos funcionales y no funcionales de dichas aplicaciones. Esta situación puede conducir a interacciones no deseables que afecten negativamente la calidad de rendimiento de la aplicación completa y también afecten la satisfacción del usuario.

Business Process Execution Language for Web Services (BPEL4WS) es el lenguaje utilizado para la definición y ejecución de procesos de negocio utilizando servicios Web. Permite, de una manera relativamente fácil y simple, componer una serie de servicios Web en nuevos servicios compuestos denominados procesos de negocio (business processes). Existen trabajos que intentan adaptar este lenguaje para aplicarlo a workflows Grid (Turner y Tan, 2006; Dörnemann et al., 2007). Sin embargo, el problema principal de las especificaciones generadas para workflows Grid, es que carecen de soporte para verificar la validez de la composición al no permitir realizar algún tipo de validación formal. Sin embargo, se considera que la verificación de la validez de la composición de servicios Grid es una propiedad fundamental para la generación de una especificación correcta.

Se describen a continuación, algunos enfoques de Separación Avanzada de Concerns que recientemente han sido aplicados al desarrollo de aplicaciones para Grid computing.

2 ENFOQUES DE SEPARACIÓN DE CONCERNS

2.1. MOTIVACIÓN

La complejidad de las aplicaciones de software se ha incrementado en la última década a raíz de los avances tecnológicos en materia de hardware y comunicaciones y de los nuevos requerimientos que surgen de las organizaciones modernas. Las técnicas de desarrollo de software convencionales, han demostrado incapacidad para abordar estos desarrollos, haciendo necesario el estudio de nuevos enfoques relacionados con SoC y ASoC (Separación Avanzada de Concerns) que puedan dar un soporte adecuado a la construcción de tales aplicaciones. En este espectro de aplicaciones complejas pueden citarse las aplicaciones Grid, cuyos usuarios finales suelen ser científicos e ingenieros que a través de sistemas distribuidos basados en entornos Grid, ejecutan aplicaciones que hacen utilización intensivas de datos o computación. En este marco, una aplicación Grid puede corresponderse con un workflow (Workflow Management Coalition, 1993) consistente en la orquestación (composición) del conjunto de servicios Grid que lo conforman, exhibiendo un nivel de interoperabilidad adecuado entre los mismos y que permiten su ejecución en un entorno Grid según las restricciones (no funcionales) necesarias.

A continuación se describen los conceptos que motivan y constituyen el estado actual del conocimiento en el tema.

2.2. SEPARACIÓN DE CONCERNS

La Separación de Concerns (Hürsch y Lopes, 1995) se encuentra en el centro de la Ingeniería de Software (IS) desde hace más de una década, como principio fundamental para el desarrollo de software. En su forma más general, se refiere a la capacidad de identificar, encapsular y manipular aquellas piezas de software que son relevantes para un concepto, objetivo, tarea o propósito particular. Los concerns son la principal motivación para la organización y la descomposición del software en partes más pequeñas, manejables y comprensibles, cada una de las cuales es responsable de uno o más concerns.

Según (Filman et al., 2005) un concern es "Algo en lo que se tiene interés a lo largo del desarrollo de un sistema". Es decir, que un concern es un tema de interés, una parte del problema a resolver, o una característica o propiedad que se debe considerar. Este concepto se utiliza para referirse tanto a propiedades funcionales como a las no funcionales de un sistema. Ejemplos de concerns que expresan estas propiedades son: seguridad, tolerancia a fallos o sincronización. Si se siguen aproximaciones tradicionales, el comportamiento correspondiente a un concern queda entrelazado (*tangled*) con el de otros concerns del sistema. Por otra parte, para llegar a expresar sus objetivos, este comportamiento queda o puede quedar disperso (*scattered*) a lo largo del mismo.

Con una adecuada SoC se pretende reducir la complejidad del software y mejorar la comprensión, promover la trazabilidad dentro y a través de los artefactos y durante todo el ciclo de vida del software; facilitar la reutilización, la adaptación no invasiva, la personalización, la evolución y simplificar la integración de componentes.

Un objetivo fundamental del área de Separación Avanzada de Concerns, ASoC (Brichau et al., 2002) es superar la tiranía de la descomposición dominante, de la que adolecen muchos de los artefactos de notaciones modernos, sólo permitiendo la separación de ciertos tipos de concerns (por ejemplo, los datos en los enfoques orientados a objetos). Otros tipos importantes de concerns se solapan, interactúan y atraviesan los módulos dominantes, y no pueden ser encapsulados eficazmente. A su vez, el software relacionado con ciertos concerns se dispersa a través de varios módulos, y se mezcla con software relacionado con otros concerns. Los enfoques de ASoC específicamente tratan estas características claves del software complejo y a gran escala.

En los últimos años, en el campo de la investigación se han desarrollado mecanismos avanzados de modularización que ayudan a superar una serie de problemas asociados con la inadecuada SoC. Por ejemplo, se pueden mencionar, la programación adaptativa (Lieberherr, 1996), componentes plug-and-play adaptativos (Mezini y Lieberherr, 1998), programación orientada a aspectos (Kiczales et al., 1997), filtros de composición (Aksit et al., 1992), módulos conceptuales (Baniassad y Murphy, 1998), ingeniería de requerimientos basada en features (Griss, 2000), programación generativa (Batory et al., 2000a,b; Czarnecki y Eisenecker, 2000), separación multidimensional de concerns e hiperespacios (Tarr et al., 1999), modelado por roles (Andersen y Reenskaug, 1992), programación orientada a sujetos (Harrison y Ossher, 1993), programación orientada a variación (Lieberherr, 1996), viewpoints (Nuseibeh et

al., 1994), desarrollo de software orientado a features (Apel y Kaestner, 2009), entre otros. Cada enfoque explora un punto diferente en el espacio de soluciones y trata diferentes subgrupos de problemas. De hecho, existe una multitud de enfoques posibles para ASoC (muchas aún sin explorar), y cada uno tiene ventajas y desventajas, pero los enfoques ASoC tienen en común la capacidad de definir y manipular los concerns *tradicionales* (ortogonales) y los concerns superpuestos ó *transversales* (no ortogonales).

A continuación presentamos la descripción de algunos de estos enfoques de ASoC que han sido ó pueden ser relevantes para el desarrollo y mantenimiento de software para Grid computing.

2.3. DESARROLLO DE SOFTWARE DIRIGIDO POR MODELOS

El Desarrollo de Software Dirigido por Modelos (Model-Driven Software Development, MDSD) es un enfoque de desarrollo de software que trata de mejorar la adaptabilidad de los sistemas ante los cambios, no sólo de los requisitos, sino también los tecnológicos, mediante la definición de modelos a diferentes niveles. Es una aproximación basada en el modelado de sistemas software, la transformación de modelos y la generación final del sistema a partir de ellos. Sin embargo, al ser sólo una aproximación, no define técnicas a utilizar ni fases del proceso ni ningún tipo de guía metodológica, sólo proporciona una estrategia general a seguir en el desarrollo del software. En este paradigma los modelos se consideran elementos de primer orden: un modelo se crea con un propósito específico y contiene sólo la información necesaria para llevar a cabo dicho propósito. Así, distintos modelos contemplan el desarrollo y manipulación de los diversos aspectos del sistema en estudio. La transformación empieza en etapas tempranas y el ingeniero de software completa cada modelo obtenido de un modo más o menos manual. El proceso de transformación de un modelo en otro juega un papel fundamental a lo largo del desarrollo de un sistema bajo este paradigma. MDSD soporta también distintos niveles de abstracción, relacionados en mayor o menor medida con modelos dependientes de la plataforma de implementación. La gran ventaja de la utilización de esta aproximación es que los modelos no sólo son documentos, sino que realmente son los elementos centrales del desarrollo que finalmente se transforman en el producto final.

Model Driven Architecture (MDA) (Soley, 2000), propuesta del OMG (Object Management Group (MDA, 1997)), es un marco de trabajo para el desarrollo software dirigido por modelos. MDA se basa en la separación entre la especificación de la funcionalidad esencial del sistema y la implementación de dicha funcionalidad; le da mayor importancia al modelado conceptual y al papel de los modelos como elementos de primer orden en el diseño, desarrollo e implementación del software y a la definición de las transformaciones entre ellos. En función del nivel de abstracción, MDA considera diferentes tipos de modelos:

- Los requisitos del sistema son recogidos en los *Modelos Independientes de Computación* (*Computation Independent Models, CIM*). En este nivel se modela la lógica de negocio o la especificación de los requisitos de un modo independiente de la computación.

- Los *Modelos Independientes de la Plataforma* (*Platform Independent Models, PIM*) representan la funcionalidad del sistema abstrayéndose de la plataforma final. En este nivel se modela el sistema independientemente de cualquier plataforma software/hardware. El modelo presentado en el CIM es refinado en este nivel; para ello, se transforma un modelo indepen-

diente de la computación en otro independiente de plataforma. El nivel PIM permite centrarse en la especificación de la funcionalidad del sistema, olvidándose de todo detalle relativo a la plataforma en la que será implementado.

- Los *Modelos Específicos de la Plataforma* (*Platform Specific Models, PSM*) combinan las especificaciones contenidas en un PIM con los detalles de la plataforma elegida. A partir de los distintos PSM se pueden generar automáticamente distintas implementaciones del mismo sistema. En este nivel se introducen los detalles específicos de una plataforma en los modelos que han sido especificados en niveles anteriores. PSM es un refinamiento del nivel PIM anterior.

La definición de los niveles supone una mejora en el desarrollo de sistemas complejos; sin embargo, a veces, los modelos y las transformaciones entre ellos se vuelven excesivamente grandes.

2.4. DESARROLLO DE SOFTWARE BASADO EN COMPONENTES

El paradigma del Desarrollo Software Basado en Componentes (Component Based Software Development, CBSD) (Szyperski, 1998) supone adoptar un enfoque diferente al tradicional, ya que su objetivo es evitar abordar el desarrollo de los sistemas desde cero, sino hacerlo mediante la aplicación de técnicas de reutilización de partes del software ya desarrolladas y probadas. Estos elementos reutilizables se denominan *componentes*. A partir de una discusión inicial sobre el término componente, qué es y qué no es, se ha llegado a una definición comúnmente aceptada, debida a Szyperski (Szyperski, 1998), quien afirma que “Un componente software es una unidad de composición binaria que especifica un conjunto de interfaces que el componente ofrece al entorno y un conjunto de dependencias que el componente tiene del entorno”.

Además, un *componente software* en general se desarrolla de forma independiente, para ser compuesto posteriormente con otros componentes. Hoy se considera que un componente es una unidad de composición reutilizable. Su reutilización en diferentes contextos hace necesario que se tenga que prestar especial atención a la descripción de sus interfaces. Esto es así porque mediante ellas el componente se comunica con los otros con los que se relaciona, proporcionando servicios al resto y recibiendo del entorno los servicios que ofrecen otros componentes. Los lenguajes de definición de interfaces deberían proporcionar información que permita manejar los componentes como cajas negras que encapsulan una cierta funcionalidad a la que no sería necesario acceder.

CBSD se basa en tres conceptos, a saber:
1. La encapsulación, ocultamiento de detalles de implementación y configuración del componente. Un proveedor de componentes puede ofrecer su funcionalidad en la forma de un componente de caja negra que los consumidores pueden utilizar en varias aplicaciones. Este nivel de abstracción alto facilita el desarrollo basado en componentes y lo hace menos propenso a errores.

2. El soporte para composición de componentes permite el diseño y despliegue de sistemas complejos formados por subsistemas más pequeños organizados jerárquicamente. De esta

forma se puede reconfigurar un subsistema en particular sin comprometer la aplicación completa.

3. La descripción de la arquitectura del sistema, utilizando un Lenguaje de Descripción de Arquitectura (Architecture Description Language, ADL) así como los detalles de despliegue referidos a la ubicación de cada componente y a cómo interactuar con otros componentes, que potencialmente pueden encontrarse ubicados remotamente.

Existen dos tipos de componentes: componentes *primitivos* que ejecutan una única tarea (normalmente tienen que ser tan simples como sea posible para permitir su reuso) y los componentes *compuestos* consistentes de un número de sub-componentes.

2.5. DESARROLLO DE SOFTWARE ORIENTADO A ASPECTOS

El desarrollo de Software Orientado a Aspectos (Aspect-Oriented Software Development, AOSD) (Kiczales et al., 1997) es un paradigma de desarrollo de software que tiene como objetivo modularizar los llamados *crosscutting concerns*. Los *crosscutting concerns* han sido considerados como una de las principales características responsables de la complejidad del software. El término se refiere a propiedades del software que no pueden ser efectivamente modularizadas usando técnicas tradicionales, como las orientadas a objetos (por *crosscutting* se puede entender que rompe las barreras de encapsulación). AOSD busca modularizar los sistemas de software para aislar funcionalidades de soporte o secundarias de las funcionalidades básicas del sistema, es decir, aplicar SoC para obtener una mejor modularización del sistema de software.

Esta separación permite la construcción de software de calidad más modularizado, facilitando la reutilización, el mantenimiento y la evolución de los sistemas. En general, un *aspecto* puede considerarse como *un elemento con comportamiento extra que se ejecuta en un momento dado durante la ejecución de una aplicación*. Un aspecto según (Filman et al., 2005) es “una unidad modular diseñada para encapsular el comportamiento de un concern específico, siendo la naturaleza de esta descomposición diferente de otras descomposiciones usadas anteriormente en aproximaciones convencionales”. También se ha definido un *aspecto* como *una unidad para modularizar una propiedad transversal en el sistema (crosscutting concern)*.

Relacionado con el principio de SoC está el problema de los *intereses o propiedades transversales* (crosscutting concerns). Una propiedad transversal es (Filman et al., 2005) aquella cuya implementación queda dispersa (*scattered*) a lo largo de varios elementos del sistema.

Además, la implementación dispersa supone o puede suponer la obtención de un código enmarañado (*tangled*). Por eso, el término *propiedades transversales* (crosscutting concern) se suele describir en términos de dispersión (*scattering*) y enmarañamiento (*tangling*). Es decir, el código transversal se refiere a la dispersión y el enmarañamiento del código de los intereses o propiedades del sistema que surge debido a una pobre modularización del mismo. De esta manera, el propósito de la definición de aspectos es llegar a modularizar el comportamiento de una propiedad específica (*concern*), evitando los problemas de dispersión (*scattering*) y enmarañamiento (*tangling*).

Para cada fase de desarrollo hay diferentes propuestas para realizar la separación de propiedades que resultan más o menos adecuadas en función de las características del sistema a construir (Martínez, 2008). Las propuestas varían en función de cómo y cuándo se identifican los aspectos; algunas siguen el modelo simétrico y otras el asimétrico (Harrison et al., 2002); unas definen los aspectos como componentes o conectores; y otras definen conceptos nuevos como vistas (*view*) asociadas a los aspectos o concerns. Además, diferentes propuestas varían en el modo en que se realiza la composición de los aspectos: se puede hacer de forma estática -en tiempo de compilación-, de forma dinámica -en tiempo de carga- ó en tiempo de ejecución. Por otra parte, los puntos de intercepción pueden hacer referencia a elementos internos de clases y componentes (modelos invasivos) o sólo al comportamiento externo (modelos no invasivos), siendo un *punto de intercepción* o punto de enlace (*join point*) un punto en la ejecución de un módulo del sistema en el que se puede insertar un cierto concern. En este caso, el hilo de ejecución es interceptado para que se ejecute un módulo aspectual (*aspecto*), si se dan ciertas condiciones.

2.6. INGENIERÍA DE LÍNEA DE PRODUCTOS

Ingeniería de Línea de Producto o Desarrollo de Software de Línea de Productos (Product Line Engineering, PLE ó Software Product Line Development, SPL) (Weiss y Lai, 1999) es un enfoque que aplica una forma especializada de reutilización de software mediante el empleo planificado de artefactos básicos de software (*core assets*) dentro del ámbito de un conjunto de productos relacionados, dichos artefactos de software básicos se utilizan en la producción de un producto en una línea de productos de software. Es decir, se crea una colección de sistemas de software que son similares a partir de un conjunto de activos de software usando un mismo medio de producción.

El procesos de ingeniería se divide en dos equipos de trabajo (Piattini y Garzás, 2007). El primer equipo se encarga de la Ingeniería de Dominio ó *core asset development* (Clements y Northrop, 2001). Este equipo es responsable de desarrollar los elementos comunes al dominio: estudiar el dominio, definir su alcance (requisitos) dentro del mercado objetivo de la SPL, definir las *features*, implementar los *core assets* reutilizables y su mecanismo de variabilidad, y establecer cómo es el plan de producción. Se recogen iterativamente los requisitos comunes para toda la SPL. Estos requisitos se expresan típicamente en forma de features (que ayudan no sólo a que los clientes puedan distinguir unos productos de otros, sino que dirigen asimismo el desarrollo del código que implemente también esas características).

El segundo equipo se encarga de la Ingeniería de Producto ó *product development* (Clements y Northrop, 2001). Sus cometidos incluyen desarrollar los productos para clientes concretos, a partir de los recursos basados en requisitos concretos de clientes. Para ello, este segundo equipo utiliza los recursos creados por el equipo anterior. Utiliza la infraestructura creada por el equipo de dominio para construir productos concretos para clientes que lo demanden. Durante el desarrollo del producto, las principales tareas son las de configuración y las de composición. Las de configuración *instancian* los elementos comunes, resolviendo los puntos de variación. Aquí se pueden utilizar mecanismos como los *#ifdefs*, o bien, técnicas manuales donde se rellena con código plantillas de funciones (*stubs*). Por otro lado, la composición implica la escritura de código alrededor de estos elementos comunes (*glue code*) para facilitar su interconexión.

Esta división formal entre dominio y producto es complementada por algunos autores (Clements y Northrop, 2001) con una parte dedicada a la gestión encargada de asignar, coordinar y supervisar los recursos. Esta gestión se realiza tanto en el nivel técnico (gestión de proyectos concretos) como en el nivel organizativo (estructura organizativa adecuada a los objetivos propuestos). Los artefactos de gestión creados como planificaciones también forman parte de los elementos comunes (*core assets*).

2.7. DESARROLLO DE SOFTWARE ORIENTADO A FEATURES

El desarrollo de Software Orientado a Features (Feature-Oriented Software Development, FOSD) (Apel y Kaestner, 2009) es un paradigma reciente para la construcción, personalización y síntesis de sistemas de software a gran escala, donde el concepto central es la *feature*, que puede traducirse como atributo o característica. Una feature se define como una unidad de funcionalidad de un sistema de software que satisface un requerimiento, representa una decisión de diseño, proporciona una opción potencial de configuración. El software se descompone en términos de las features que provee, para construir software bien estructurado que pueda adaptarse a las necesidades del usuario y al escenario de aplicación. A partir de un conjunto de features comunes se puede generar una variedad de sistemas de software diferentes que comparten features comunes y difieren en otras, de manera similar al enfoque SPL (Sección 2.6).

Apel y Kaestner (2009) presentan a FOSD como paradigma integral de desarrollo de software, donde el concepto de feature se utiliza para estructurar el diseño y el código de un sistema de software. Las features son las unidades básicas de la reutilización de este enfoque, y las variantes de un sistema de software varían en las features que proporcionan. El software se genera en forma eficiente y correcta sobre la base de un conjunto de artefactos de features y de la selección de features que realiza el usuario. Las fases del proceso FOSD pueden resumirse en:

- Análisis del dominio: el objetivo es identificar y capturar las variantes y similitudes de un dominio en la forma de un *modelo de features*.

- Diseño de dominio y especificación: se especifican las propiedades estructurales y comportamentales de las features usando un lenguaje de modelado formal o informal.

- Implementación del dominio: el objetivo es establecer un mapeo uno–a–uno entre las features que surgen durante el análisis del dominio y las features que surgen en el nivel de implementación.

- Configuración y generación del producto: la visión es que siguiendo la selección de features que realice el usuario, automáticamente se genera el sistema de software que cumpliría con los requisitos de calidad.

3 ENFOQUES APLICADOS A GRID COMPUTING

En esta sección presentamos algunos de los trabajos más representativos que aplicaron los enfoques de ASoC, descriptos en la sección anterior, en el desarrollo de aplicaciones para Grid computing.

3.1. DESARROLLO DE SOFTWARE DIRIGIDO POR MODELOS

Hlaoui y BenAyed (2009) proponen emplear el enfoque MDA para realizar la composición semiautomática y semántica de workflows de aplicaciones Grid a partir de servicios Grid existentes.

El propósito del trabajo es modelar y componer interactivamente aplicaciones basadas en workflows de servicios Grid sin considerar descripciones de bajo nivel del entorno Grid. Para ello utiliza el enfoque MDA en el desarrollo de las aplicaciones basándose en las descripciones semánticas y sintácticas de los servicios Grid y en descripciones abstractas que se obtienen de la extensión de la notación de diagramas de actividades del Lenguaje Unificado de Modelado, UML. La técnica consiste en modelar el workflow de servicios Grid usando el lenguaje de diagrama de actividades de UML correspondiente al Modelo Independiente de la Plataforma (PIM), según el enfoque MDA. El proceso de composición de servicios Grid se realiza utilizando una descripción ontológica del dominio el problema a la vez que se ofrece al usuario una interfaz gráfica basada en la extensión de UML. Se define un Lenguage Específico del Dominio (Domain Specific Language, DSL) usando los mecanismos de extensión de UML para componer workflows a partir de los servicios Grid y transformar el workflow abstracto resultante en un Modelo Específico de la Plataforma (PSM) para ser ejecutado. Una actividad en el diagrama de actividades de UML representa una operación del servicio Grid, mientras que los flujos de objetos representan los tipos de resultados que fluyen de una actividad a otra. Los enlaces entre operaciones representan los flujos de control entre operaciones. A través de la notación UML se provee al usuario de un modelo abstracto y comprensible de la aplicación Grid. Se diseña el modelo de workflow que, una vez validado, se transforma en un modelo específico para ser implementado y ejecutado. Finalmente el modelo de workflow se implementa usando un formato específico y se almacena para se reusado posteriormente en el desarrollo de otras aplicaciones Grid futuras.

Para aplicar el enfoque los autores combinan diferentes herramientas con los diagramas de actividades de UML y estándares de la Web Semántica como Web Ontology Languague (OWL) y el lenguaje de consultas para ontologías RDQL. Los autores sólo consideran el enfoque MDA aplicado a los requerimientos funcionales del dominio del problema. Las pruebas del modelo propuesto se realizaron sobre una aplicación de ejemplo que realiza análisis de polución de tráfico de una ciudad. La etapa en la fase de desarrollo en la que se aplicó corresponde al diseño. No existe ninguna condición en cuanto a plataforma de implementación, lenguaje de programación para poder aplicar el enfoque.

Como puntos a mejorar, los autores proponen realizar el mapeo de los modelos UML que representan el workflow abstracto a una descripción concreta de workflow para ser ejecutada en un motor de workflow. Además haría falta incluir las restricciones no funcionales del dominio

de la aplicación Grid, ya que el trabajo realizado no lo contempla. Como punto positivo a destacar, el trabajo propone utilizar el enfoque MDA realizando la extensión de los diagramas de actividades de UML para proveer una notación visual y facilitar el análisis de la composición de servicios Grid a través de workflows.

De la misma forma, los autores de (Smith et al., 2006), proponen emplear el enfoque MDA en la composición de workflows de aplicaciones Grid a partir de servicios Grid existentes.

El objetivo es usar el enfoque MDA para desarrollar aplicaciones Grid orientadas a servicios, minimizando la intervención del usuario para transformar los modelos PIM a PSM y PSM a código para un entorno Grid orientado a servicios. La técnica consiste en la división de la capa PSM del modelo en componentes específicos de Grid (Grid PSM sublayer) y componentes específicos del dominio del problema (Business PSM sublayer). Luego se utiliza un *UML Grid Profile* que facilita las transformaciones entre PIM, PSM y el código tanto para los diseñadores como para desarrolladores y usuarios. La separación de concerns en la capa PSM se refleja en la capa de código mediante el uso de anotaciones Java, permitiendo que el mismo código se ejecute en diferentes dominios intercambiando esas anotaciones y por tanto desacoplando el código de la aplicación del middleware Grid.

Los autores aplican el enfoque a nivel teórico generando el modelo correspondiente. Solamente se consideran los requisitos funcionales del dominio del problema. El trabajo utiliza un aplicación cliente/servidor de ejemplo consistente en un cliente que accede a un único servicio que ejecuta algoritmos para descifrar claves de sistemas Unix. La utilización del enfoque involucró las etapas de diseño e implementación del ciclo de desarrollo de software. Los autores no impusieron ningún tipo de pre-requisito para utilizar el enfoque.

El modelo propuesto no incluye los requerimientos no funcionales de la aplicación. Los autores no explicitan claramente qué mecanismos de validación ni verificación utilizan al diseñar aplicaciones basadas en workflows de servicios Grid. Sin embargo, existe una clara separación de concerns al adaptar el enfoque MDA para soportar el desarrollo de aplicaciones basadas en servicios Grid. La división de la capa PSM permite transformaciones más simples entre las otras capas y separa la lógica del negocio de los concerns específicos de Grid permitiendo que cada experto en el dominio trabaje en su propio campo.

3.2. DESARROLLO DE SOFTWARE BASADO EN COMPONENTES

Los autores de (Basukoski et al., 2008) diseñaron un método genérico para desarrollar aplicaciones basadas en Grid Component Model, GCM (CoreGRID, 2007).

El objetivo del trabajo es proveer un método genérico para desarrollar aplicaciones basadas en GCM. GCM provee varios niveles de administradores/gestores autonómicos en los componentes, que se ocupan de las características no funcionales de los programas componentes. Los componentes GCM tienen dos tipos de interfaces: una funcional y otra no funcional. La interfaz funcional alberga todos los puertos que contribuyen a implementar las características funcionales de los componentes, es decir, las características que contribuyen directamente al cómputo de los resultados esperados por el componente. La interfaz no funcional contiene todos los puertos necesarios para soportar la actividad del gestor de componentes en la implementación de las características no funcionales, es decir, todas aquellas características que

contribuyen a la eficiencia del componente en la consecución de la característica esperada (funcional), pero los resultados no están directamente involucrados con resultados reales de cómputo. Cada componente GCM puede contener uno o más gestores, interactuando con otros gestores en otros componentes a través de sus interfaces no funcionales. Estos gestores interactúan tanto con los otros gestores y componentes internos del mismo componente utilizando mecanismos adecuados proporcionados por la implementación de componentes GCM. Dichos gestores están presentes en cada componente que gestionan todos los aspectos relacionados con Grid que contribuyen a la correcta ejecución del componente en la arquitectura Grid de destino.

Se generó un entorno integrado diseñado para permitir el desarrollo de aplicaciones Grid basado en componentes llamado Grid Integrated Development Environment (GIDE) para Eclipse. Este entorno provee soporte para los desarrolladores de aplicaciones a través de una herramienta que les permite realizar la composición de la aplicación de manera gráfica y asegurarles el acceso a las funcionalidades del middleware y acceso a los fuentes si fuera necesario. El enfoque aplicado contempla tanto los requisitos funcionales como no funcionales de la aplicación. Se utilizó un caso de estudio de desarrollo de aplicación de un sistema de identificación biométrico que los autores consideran como un proceso de negocio o un workflow. Los autores aplicaron el enfoque a la etapa implementación del ciclo de desarrollo de software. No existe pre-requisito alguno para poder aplicar el enfoque.

Puede ocurrir que la utilización del enfoque genere gran cantidad de artefactos que deben ser mantenidos (descripción ADL, representación diagramática, ficheros fuentes de cada componente, etc.). Además no contempla mecanismos para realizar la validación y verificación de los workflows generados. Es un framework que solamente soporta un solo lenguaje de programación para las aplicaciones Grid. Como punto a destacar, diremos que el ciclo de desarrollo es corto, generando aplicaciones que son altamente portables. El framework de componentes basados en GCM puede dar soporte a las propiedades dinámicas de las aplicaciones.

3.3. DESARROLLO DE SOFTWARE ORIENTADO A ASPECTOS

Sobral y Monteiro (2008) proponen diferentes soluciones para facilitar la gridificación de aplicaciones científicas secuenciales codificadas en Java aplicando el paradigma AOSD a través de AspectJ (Kiczales et al., 2001).

El trabajo apunta a diseñar un DSL para Grid computing, utilizando AspectJ para permitir la paralelización/gridificación de aplicaciones científicas secuenciales sin realizar cambios en el código original. La estrategia consiste en tomar un programa secuencial y aplicar un conjunto de transformaciones especificadas por el programador (abstracciones, programación paralela) para generan la versión paralela del mismo código. Los autores intentan automatizar estas transformaciones de código proporcionando un método de paralelización que no sea invasivo sino modular y que pueda *conectarse (plugged)* en el código existente. Se diseña un DSL para localizar los puntos a gridificar o paralelizar y colocarlos en módulos que puedan ser *(un)plugged* ó incorporados en el código original.

Los autores aplicaron el enfoque a través de un prototipo que a su vez se sustenta en el uso de otras herramientas desarrolladas por ellos. Sólo los requisitos funcionales fueron contemplados en la aplicación secuencial a gridificar. Para demostrar la aplicación del prototipo, se uti-

lizó un ejemplo simple consistente en el benchmark RayTracer que representa una imagen de sesenta figuras esféricas. El enfoque fue aplicado en la etapa de implementación del ciclo de desarrollo de software y sólo se puede utilizar en código realizado en Java.

Cómo crítica, podemos decir que el enfoque se aplica sólo en la fase de construcción de software a nivel de código, no se detalla cómo se aplicaría en el desarrollo de una aplicación Grid orientada a servicios. Sólo se utiliza para gridificar aplicaciones secuenciales desarrolladas en Java. No se consideran aspectos no funcionales relacionados con la QoS, por ejemplo, que pueden beneficiar el rendimiento de la aplicación secuencial que se gridifica. Como punto a favor podemos decir que el trabajo propone herramientas y métodos para que aplicaciones secuenciales puedan aprovechar el potencial de un entorno Grid, permitiendo que la aplicación primero se paralelice y luego se gridifique aplicando el enfoque AOSD.

En (Sousa et al., 2008), los autores presentan un framework completo para la gridificación de aplicaciones secuenciales y paralelas mediante la aplicación de técnicas AOP[2] evitando modificar los sistemas heredados de manera invasiva.

El objetivo del trabajo es presentar un framework llamado AspectGrid que permite agregar o quitar características relacionadas con Grid en aplicaciones o sistemas heredados. Se puede utilizar el framework para transformar código secuencial en una versión paralela o distribuida. La técnica consiste en que las aplicaciones científicas se gridifiquen a través de un conjunto de componentes de software opcionales que pueden ser integrados para satisfacer las necesidades de la aplicación ó plataforma de destino. Aplicando las técnicas de AOP en la gridificación de aplicaciones científicas para Grid computing, un aspecto se corresponde con un servicio para especificar los puntos de ejecución donde el comportamiento específico del aspecto puede ser integrado en el sistema. El framework se divide en diferentes servicios: (1) servicios proveedores de tareas correspondientes a los códigos específicos de dominio que generarán las tareas a ejecutarse en los recursos de cómputo; (2) servicios de ejecución que proporcionan los recursos computacionales donde se ejecutarán las tareas; (3) servicios centrales del framework que envían las tareas a los servicios de ejecución disponibles y que se pueden ampliar con características opcionales para cumplir con necesidades específicas de la aplicación; (4) servicios propios del entorno Grid que incluyen paralelización, distribución de carga, tolerancia a fallos, ejecución y monitoreo remotos.

Partiendo del framework desarrollado a nivel conceptual, se realiza la gridificación de una aplicación de ejemplo. El framework se utiliza para otorgar a las aplicaciones heredadas características no funcionales en cuanto a permitir la ejecución paralela, tolerancia a fallos, ejecución remota, etc., de códigos legados. Las pruebas se realizaron utilizando una aplicación de ejemplo sencilla que realiza el cálculo del conjunto de Mandelbrot. Los autores aplicaron el enfoque en la etapa de implementación del ciclo básico de desarrollo de software. Como pre-requisito la aplicación existente debe consistir en una aplicación en Java.

Como crítica podemos decir que sólo contempla la aplicación del enfoque AOSD en sistemas heredados codificados en lenguaje Java y que no se consideran los requerimientos no funcionales propios de la aplicación. Los servicios que pueden conectarse en la aplicación secuencial no incluyen características relacionadas con aspectos de seguridad ni autenticación los cuales son propiedades de todo sistema Grid. Como punto a favor, podemos decir que el beneficio del uso del enfoque AOSD es que la naturaleza no invasiva del framework desarro-

[2] AOP: Aspect-Oriented Programming

llado implica que los cambios no son irreversibles en el código fuente y que no se introduce ninguna sobrecarga en el código secuencial original, ya que los módulos de gridificación pueden conectarse en cualquier momento durante y después del proceso de gridificación. La gridificación además es de grano fino y no requiere que el código fuente tenga que cumplir con convenciones de código específicas.

3.4. INGENIERÍA DE LÍNEA DE PRODUCTOS

Bashroush y Perrott (2005) aplican el enfoque SPL en el desarrollo de una familia de productos de software basada en servicios Grid, específicamente dentro del proyecto GeneGrid.

El trabajo apunta a estudiar de qué forma SPL puede ser usada en el dominio de Grid computing y en el diseño de servicios Grid. La técnica consiste en considerar como variabilidad en los servicios Grid a la elección de la tecnología y plataforma, las interfaces de descripción, las políticas de seguridad, etc. Se divide el proceso de desarrollo de servicios Grid en cuatro etapas: (1) Gestión de variabilidad y modelado de features: se identifican las features del sistema y se analizan las similitudes y variabilidades entre los diferentes servicios Grid de la familia de productos que se desea modelar, utilizando técnicas como FODA (Kang et al., 1990). (2) Desarrollo de activos (assets): los componentes que forman los servicios, los servicios en sí, documentación, arquitectura, que formarán la familia de productos. Se pueden usar dos enfoques: top-down (diseñar la descripción de los servicios y luego construir los componentes) ó bottom-up (diseñar los componentes a partir de las similitudes identificadas en la etapa anterior y luego componerlos para formar los servicios). (3) Evaluación y prueba: de los servicios diseñados contra un conjunto pre-establecido de atributos de calidad y funcionales para luego ser implementados a nivel de unidad y realizar las pruebas a nivel de servicios. (4) Gestión de evolución: para que las modificaciones y actualizaciones se introduzcan usando los assets existentes, y los cambios se introduzcan a nivel de modelo de features y arquitectura primero y luego se propaguen afectando el código a través de las especificaciones de componentes, documentación, etc.

El enfoque sólo se aplica a nivel teórico, usando como ejemplo un proyecto de Grid para aplicar SPL. Los requerimientos considerados se relacionan con características funcionales de la aplicación. Los autores utilizan un ejemplo sencillo y simplificado del proyecto GeneGrid consistente en diversos portales Grid, workflows y aplicaciones Grid. Las etapas del ciclo de desarrollo abarcadas fueron análisis y diseño. No se requiere ninguna condición previa que cumplir por parte de la aplicación en la que desee aplicar el enfoque según estos autores.

Como comentario, podemos decir que el enfoque SPL se aplica de manera incompleta en la generación de aplicaciones para Grid computing. Se utilizan las primeras dos etapas del enfoque SPL en el desarrollo de la aplicación para Grid computing en un nivel abstracto, es decir, en un ejemplo que no es una aplicación basada en workflow de servicios o para desarrollar un servicio Grid. Como punto a destacar es que este trabajo constituye unos de los primeros trabajos que plantea los conceptos de familias de servicios Grid y evolución de servicios a partir del enfoque SPL.

El trabajo de Acher et al. (2008) propone mejorar la reutilización de servicios involucrados en el dominio del procesamiento de imágenes médicas. La variabilidad de los servicios Grid se refiere tanto a las variabilidades funcionales como a las no funcionales.

En el área del procesamiento de imágenes médicas, Grid computing permite construir modelos específicos para cada paciente y reducir el tiempo de cómputo que puede resultar crítico en la práctica clínica. Se diseñan servicios de procesamiento de imágenes basados en SOA para permitir abstraer la heterogeneidad de la plataforma técnica y de los servicios subyacentes. Asimismo los workflows permitirían diseñar algoritmos compuestos y complejos basados en esos servicios. El propósito del trabajo consiste en analizar la variabilidad de los aspectos funcionales y no funcionales del dominio del procesamiento de imágenes médicas y proponer una arquitectura en la cual los servicios se organicen dentro de una arquitectura de línea de productos y donde los metamodelos den soporte para organizar la información necesaria. La estrategia es proponer una línea de productos de servicios para describir las principales variaciones en las especificaciones funcionales y no funcionales de los servicios de procesamiento de imágenes médicas así como en las posibles flujos de información y procesos en un workflow Grid.

Los autores propusieron metamodelos para representar la variabilidad de los requerimientos funcionales así como mecanismos para representar aquellos relacionados con QoS. Asimismo han desarrollado un trabajo teórico proponiendo una arquitectura para que los proveedores de servicios y los expertos en workflows capturen las similitudes y diferencias de los servicios legados, que les permitan construir servicios de manera eficiente según esas similitudes y diferencias y para usar una línea y seleccionar los servicios apropiados de acuerdo a criterios funcionales y no funcionales. De esta forma, el trabajo considera tanto los requerimientos funcionales como no funcionales del dominio del problema. Se utiliza un caso de estudio consistente en un workflow para procesamiento de imágenes médicas para aplicar los metamodelos y la arquitectura con miras a implementar en servicios y datos a gran escala. La etapa del ciclo de desarrollo involucrada fue el análisis. Los autores no definieron ningún tipo de restricción para que el enfoque pueda ser utilizado en el desarrollo de cualquier otra aplicación.

Como principal punto a estudiar se encuentra el relacionado con la interacción entre variabilidades de requerimientos funcionales y de QoS de modo tal de componerlos para obtener una arquitectura que ofrezca todas las prestaciones posibles al seleccionarlas de una línea de productos. La arquitectura aún no contempla mecanismos para realizar la validación y verificación de los workflows que se pueden generar al seleccionar diferentes servicios de una línea de productos de servicios. El trabajo refleja un claro análisis del dominio de servicios Grid para el procesamiento de imágenes y de los mecanismos que permiten definir restricciones de QoS relacionadas con los servicios en sí mismos o las necesidades propias de los usuarios. De esta manera no sólo se modelan atributos funcionales de los servicios sino también aquellos relacionados con la QoS del procesamiento de imágenes médicas para que los planificadores Grid y los motores de workflows puedan seleccionar servicios según esas restricciones.

3.5. DESARROLLO DE SOFTWARE ORIENTADO A FEATURES

El trabajo de Trejo et al. (2010) pretende modelar los servicios Grid aplicando FOSD y utilizar, en la etapa de diseño y especificación del dominio, la técnica de Diseño por Contratos (DbC) (Meyer, 1992, 1997) para detectar y resolver el problema de interacciones entre features (Feature Interaction Problem, FIP) al componer servicios Grid.

El objetivo es presentar la aplicación del enfoque FOSD en el desarrollo de servicios Grid y en las aplicaciones Grid basadas en workflows de servicios Grid. Se propone el modelo de features para servicios Grid y se extiende la especificación WSDL para controlar el problema de interacción entre features entre servicios Grid utilizando DbC. La estrategia consiste en que el servicio Grid se representa a través del modelo de features donde la raíz es el servicio Grid y las features corresponden a los elementos que forman la interfaz WSDL y también incluyendo al documento de Resources Properties expresado como features del servicio Grid. Este modelo luego se extiende para incorporar features basadas en DbC que representan características relacionadas con la QoS y con dependencias de control y datos con el objetivo de servir a la verificación durante el proceso de composición de servicios Grid.

Se crearon modelos tanto del servicio Grid como de la extensión de la especificación WSDL que incluye elementos basados en DbC para controlar las interacciones entre servicios Grid. El modelo de features de un servicio Grid permite modelar tanto las features funcionales como las restricciones no funcionales. El trabajo aplica el enfoque a un ejemplo simple consistente en un servicio que realiza las operaciones matemáticas elementales. Las etapas de ciclo de desarrollo de software que se abarcaron fueron el análisis y diseño. No se requiere ninguna condición que deban cumplir las aplicaciones para desarrollarse con la estrategia para aplicar este enfoque.

Como crítica, podemos mencionar que el trabajo muestra resultados preliminares y aún se encuentra aplicado en un nivel teórico. Sin embargo, podemos destacar que aplicando FOSD a Grid computing se pueden atacar las interacciones entre servicios Grid de los cuales sólo se disponen sus interfaces. Utilizando FOSD se puede pensar en definir cada servicio Grid con un modelo de features y luego poder definir una familia de servicios Grid relacionados que permitan generar diferentes aplicaciones basadas en las combinaciones de estos servicios pero que también puedan detectarse interacciones no deseables en tiempo de diseño de workflows.

Acher et al. (2010) han realizado un aproximación al modelado de la variabilidad de servicios Grid que se combinan en un workflow utilizando la primera fase según el paradigma de FOSD.

Dentro del contexto de workflows Grid para aplicaciones de procesamiento de imágenes médicas, los autores definen familias de servicios para definir líneas de productos de servicios que se combinan de diferentes formas en los workflows. Definen los puntos que varían en los servicios y los describen usando modelos de features para crear familias de workflows definidas como composiciones de estos modelos de features. Aplican una técnica composicional para razonar acerca de la compatibilidad entre los servicios conectados y asegurar la consistencia del workflow completo. La técnica consiste en que los servicios se organizan sobre una arquitectura de línea de producto y se utilizan modelos de features para estructurar la información necesaria en términos de variabilidad de cada servicio. De esta forma se define una familia de servicios como un conjunto de concerns que exhiben variabilidad y que se representan con uno o varios modelos de features. Se propone un método para componer servicios parametrizados mediante workflows utilizando SPL y features. Esto se realiza para proveer mecanismos que permitan a los proveedores de servicios Grid, como científicos investigadores o expertos en workflows Grid, capturar las similitudes y variabilidades en servicios parametrizados que son ofrecidos en un entorno Grid. Otra razón es para proveer soporte para componer y adaptar servicios de tal forma que los consumidores de ellos puedan asegurar la consistencia de los workflows resultantes con propiedades bien definidas. El workflow se crea componiendo familias de servicios y luego se configura la línea de producto workflow resul-

tante. Los modelos de features se usan para describir las features comunes y variables de un servicio adaptado a las necesidades del consumidor del servicio. Los aspectos variables del servicio parametrizado se describen usando distintos modelos de features, donde cada uno de ellos describe puntos variables en una dimensión particular. Se usa un conjunto de operadores de composición para insertar nuevos concerns con variabilidad dentro de la descripción del servicio y para combinar modelos de variabilidades de los servicios conectados o compuestos. Además los modelos de features se usan para razonar en ese nivel, sobre la dependencia de los servicios especificada por el usuario e identificada como activa en el workflow.

Se han generado metamodelos de servicios Grid y de workflows. Se utilizaron modelos de features para representar el concepto de variabilidad en el contexto del dominio del problema y también se han diseñado mecanismos para evaluar y asegurar la consistencia de los workflows a partir de la combinación de estos modelos. Los requerimientos que se han contemplado en el trabajo se relacionan con las especificaciones funcionales y restricciones no funcionales del dominio del problema del procesamiento de imágenes médicas. Los tests se realizaron sobre un caso de estudio que involucra una familia de workflows que pueden parametrizarse para realizar el procesamiento de imágenes biomédicas. Las etapas del ciclo de desarrollo de software abarcadas por este trabajo fueron el análisis y diseño. No se requiere ninguna condición a cumplir por parte de la aplicación para que pueda aplicarse este enfoque.

Como observación, podemos decir que el modelo no maneja restricciones entre modelos de features tanto sin son internas como entre varios modelos de features, pero si los concerns relacionados con estos modelos de features no son independientes, aparece el concepto de FIP. Las interacciones entre features a pesar de perjudicar el desarrollo incremental y modular, no ha sido contemplado en el trabajo para no anular los beneficios que propone el mismo. Como punto a favor, los autores aplican los principios de SoC. Definen un concern como un término a usarse en un sentido amplio desde requerimientos de alto nivel hasta aspectos de implementación de bajo nivel, referirse a propiedades medibles o al comportamiento del sistema. Se asocian y describen las concerns de acuerdo a elementos precisos definidos en un servicio y se generan varios modelos de variabilidad en lugar de uno solo para que la descripción de la variabilidad de un servicio sea modularizada y donde cada uno de esos módulos se refiera a un concern claramente identificado. Utilizando SoC, los autores permiten que el usuario pueda detectar y corregir las inconsistencias en el workflow diseñado mediante la elección de diferentes servicios y saber cuales son los que están provocándola. Las estrategias pueden ser: seleccionar otro servicio, identificar y corregir el concern modificando la descripción de la variabilidad de los servicios o configurando de otra forma a los mismos servicios.

3.6. COMBINACIONES DE ENFOQUES

Los autores de (Li et al., 2006) combinan los enfoques de MDSD con AOSD para crear un framework para el desarrollo de servicios Grid.

Los autores crearon un Model-Driven Aspect Framework (MDAF) para facilitar el desarrollo de aplicaciones basadas en servicios Grid combinando los enfoques MDA y AOP. La estrategia consiste en que la salida UML en formato XMI (XML Metadata Interchange) tomada de una herramienta CASE se utiliza como entrada del framework para generar una suite de ficheros fuentes y configuraciones relacionadas con la aplicación. El código específico relacionado con Grid se modulariza en aspectos individuales desacoplándose del código propio de la apli-

cación. De esta forma el código específico de Grid puede evolucionar sin afectar la lógica de negocio la que consiste en clases Java independientes de Grid y que pueden ser compiladas, testeadas y reusadas fuera de un entorno Grid determinado sin tener que realizar ninguna modificación. Los desarrolladores usan notaciones UML simples para especificar los concerns Grid en el nivel de modelo y no precisan ser expertos en la implementación de los aspectos de Grid en diferentes middleware Grid.

Se genera el framework MDAF para desarrollar aplicaciones basadas en servicios Grid. Este framework solamente considera los requerimientos funcionales de la aplicación. Se utiliza como ejemplo una aplicación para realizar full-text search de documentos web donde existe un componente cliente que accede a un servicio Grid. Las etapas del ciclo de desarrollo de software involucradas fueron análisis, diseño e implementación. No se definió condición previa para poder aplicar el enfoque.

Como crítica podemos decir que no se contempla la transformación de los modelos UML para soportar la arquitectura orientada a servicios de Grid computing y que la implementación de la aplicación sólo se genera para un único lenguaje de programación. Otro punto para comentar es que se aplica el framework para desarrollar servicios conectados a través de workflows pero no toma en cuenta la interacciones entre los servicios Grid que se generan. Como punto a destacar, decir que el trabajo permite la separación de la lógica del negocio de los detalles específicos de Grid a nivel de modelo y de código con lo que se facilita el desarrollo, prueba y mantenimiento de las aplicaciones basadas en servicios Grid. Al especificar los servicios usando notación UML se facilita el proceso de diseño a los desarrolladores.

4 ANÁLISIS COMPARATIVO

4.1. CRITERIOS DE COMPARACIÓN

En esta sección presentaremos algunos criterios para facilitar el estudio comparativo de los trabajos que han implementado los enfoques de ASoC en el desarrollo des aplicaciones para Grid computing.

- Nivel de abstracción: los enfoques pueden aplicarse a nivel teórico o de modelado o a nivel aplicado generando entornos de desarrollo, herramientas ó prototipo que puede ser reutilizado

- Requisitos: el enfoque aplicado puede considerar requerimientos funcionales y/o no funcionales de la aplicación Grid

- Validación empírica: los resultados experimentales de los modelos ó herramientas desarrolladas pueden haberse aplicado a: (a) ejemplo sencillo (involucra un servicio); (2) ejemplo mediano (involucra más de un servicio); (3) caso de estudio de mayor complejidad (involucra más de un servicio combinados de diferentes formas); (4) a gran escala

- Para aplicaciones: el enfoque se puede aplicar (1) para gridificar aplicaciones existentes (secuenciales ó paralelas); (2) para crear nuevas aplicaciones basadas en servicios Grid (existe un aplicación cliente que accede a un servicio Grid); (3) para crear nuevas aplicaciones basadas en workflows de servicios Grid

- Etapa: a qué fase del ciclo básico de desarrollo de software los autores del trabajo aplican el enfoque

4.2. EVALUACIÓN DE ENFOQUES APLICADOS

Tomando como base los criterios introducidos en la sección anterior, compararemos los diferentes enfoques de ASoC descriptos en la Sección 3. A continuación, la Tabla 1, resume los trabajos más recientes que aplicaron enfoques ASoC al desarrollo de software para Grid computing y la Tabla 2 presenta el resultado de la aplicación de los criterios.

Tabla 1: Lista de trabajos comparados que aplicaron enfoques de ASoC

Enfoque	Referencia	Breve descripción del objetivo	Sección
MDSD	(Hlaoui y BenAyed, 2009)	Modelar y componer interactivamente aplicaciones basadas en workflows utilizando descripciones semánticas y sintácticas de servicios Grid y descripciones abstractas provenientes de la extensión de la notación de diagramas de actividades de UML.	3.1
	(Smith et al., 2006)	Desarrollar aplicaciones Grid orientadas a servicios transformando los modelos PIM a PSM y PSM a código para un entorno Grid orientado a servicios.	
CBSD	(Basukoski et al., 2008)	Proveer un método genérico para desarrollar aplicaciones basadas en Grid Component Model.	3.2
AOSD	(Sobral y Monteiro, 2008)	Diseñar un DSL para Grid computing, utilizando AspectJ para poder realizar la paralelización ó gridificación de aplicaciones científicas secuenciales sin realizar cambios en el código original.	3.3
	(Sousa et al., 2008)	Presentar el framework AspectGrid que permite agregar o quitar características relacionadas con Grid a aplicaciones o sistemas heredados. Se puede utilizar también para transformar código secuencial en una versión paralela o distribuida.	
SPL	(Bashroush y Perrott, 2005)	Estudiar de qué forma SPL puede ser usada en el dominio de Grid computing y en el diseño de servicios Grid.	3.4
	(Acher et al., 2008)	Proponer una línea de productos de servicios para describir las principales variaciones en las especificaciones funcionales y no funcionales de los servicios de procesamiento de imágenes médicas así como en los posibles flujos de información y procesos en un workflow Grid.	
FOSD	(Trejo et al., 2010)	Presentar la aplicación del enfoque FOSD en el desarrollo de servicios Grid y sus	3.5

		interacciones en las aplicaciones Grid basadas en workflows de servicios Grid.	
	(Acher et al., 2010)	Dentro del contexto de workflows Grid para aplicaciones de procesamiento de imágenes médicas, los autores proponen familias de servicios para definir líneas de productos de servicios que se combinan de diferentes formas en los workflows.	
MDSD+AOSD	(Li et al., 2006)	Crear un Model-Driven Aspect Framework para facilitar el desarrollo de aplicaciones basadas en servicios Grid combinando los enfoques MDA y AOP.	3.6

Tabla 2: Comparación de enfoques de ASoC aplicados a Grid Computing

Enfoque	Referencia	Abstracción	Requisitos	Pruebas	Para aplicaciones	Etapa
MDSD	(Hlaoui y BenAyed, 2009)	modelado	funcionales	ejemplo simple	basadas en workflows de servicios	diseño
	(Smith et al., 2006)	modelado	funcionales	ejemplo simple	basadas en workflows de servicios	diseño, implementación
CBSD	(Basukoski et al., 2008)	aplicado	funcionales y no funcionales	caso de estudio	basadas en servicios	implementación
AOSD	(Sobral y Monteiro, 2008)	aplicado	funcionales	ejemplo simple	secuenciales existentes	implementación
	(Sousa et al., 2008)	modelado	no funcionales	caso de estudio	existentes	implementación
SPL	(Bashroush y Perrott, 2005)	modelado	funcionales	ejemplo simple	basadas en servicios	análisis, diseño
	(Acher et al., 2008)	modelado	funcionales y no funcionales	caso de estudio	basadas en workflows de servicios	análisis
FOSD	(Trejo et al., 2010)	modelado	funcionales y no funcionales	ejemplo simple	basadas en workflows de servicios	análisis, diseño
	(Acher et al., 2010)	modelado	funcionales y no funcionales	caso de estudio	basadas en workflows de servicios	análisis, diseño
MDSD+AOSD	(Li et al., 2006)	aplicado	funcionales	ejemplo simple	basadas en servicios	análisis, diseño, implementación

5 DISCUSIÓN

Como se observa en la Tabla 2, la mayoría de los trabajos que aplicaron enfoques ASoC lo hicieron a nivel teórico, sin llegar a desarrollar una herramienta o framework para facilitar el desarrollo de aplicaciones para Grid computing. Las excepciones son los trabajos (Sobral y Monteiro, 2008; Li et al., 2006; Basukoski et al., 2008), que aportaron algún tipo de herramienta ó framework, sin embargo las mismas se encuentran disponibles para generar códigos en un solo lenguaje de programación, o sirven para gridificar aplicaciones secuenciales que serán ejecutadas en entornos Grid. De estos trabajos solamente (Basukoski et al., 2008) considera tanto los requisitos funcionales como las restricciones no funcionales y también la volatilidad de los requisitos en el desarrollo de la aplicación para Grid computing. Además este último trabajo es el único que considera el desarrollo de aplicaciones basadas en workflows de servicios Grid.

Por otro lado, la mayoría de los enfoques emergentes de IS aplicados a Grid computing no se utilizan para modelar requerimientos funcionales y requerimientos no funcionales que las aplicaciones Grid deben satisfacer. Sin embargo, los trabajos que consideran requisitos funcionales y no funcionales de las aplicaciones son los relacionados con los enfoques CBSD (Sección 2.4), SPL (Sección 2.6) y FOSD (Sección 2.7). A diferencia del resto de trabajos, éstos permiten que el desarrollador de aplicaciones para Grid computing pueda definir restricciones no funcionales, principalmente relacionadas con QoS antes que la aplicación sea efectivamente lanzada en la plataforma Grid de destino. Para ciertas aplicaciones Grid, los requisitos no funcionales son tan importantes como los funcionales, por ejemplo aquellas aplicaciones científicas o médicas, donde características relacionadas con el rendimiento de la aplicación, la exactitud de los resultados, la confidencialidad de los datos, tiempo de respuesta, costo de ejecución de la aplicación, etc. también deben poder ser definidos durante el desarrollo de aplicación Grid.

De los trabajos comparados, ninguno de ellos excepto (Trejo et al., 2010), analizan y/o discuten escenarios que reflejen posibles problemas de interoperabilidad entre los servicios Grid que se combinan para satisfacer las necesidades el dominio del problema.

De los trabajos comparados, la mayoría de los autores utilizaron ejemplos simples consistentes en pocos servicios, excepto los trabajos de (Basukoski et al., 2008; Sousa et al., 2008; Acher et al., 2008, 2010) que utilizaron casos de estudio basados en workflows de servicios Grid con varios servicios intercambiando información y siguiendo un flujo de control determinado. Esto demuestra que los resultados experimentales de los enfoques aplicados al desarrollo de aplicaciones para Grid computing se encuentran en una fase temprana y por lo tanto es un campo de investigación activo y desafiante por las características propias de estas aplicaciones (altamente complejas y a gran escala, requisitos altamente variables, configuraciones globales de la plataforma de despliegue y ejecución, gran nivel de interoperabilidad entre sus componentes, entre otras).

Es interesante observar que sólo la minoría de estos trabajos aplicaron el enfoque ASoC de manera integral y completa en el desarrollo de una aplicación Grid. Los trabajos de ASoC aplicados a Grid computing son relativamente recientes considerando que la tecnología Grid tiene sus orígenes a mediados de la década del 90. Esto demuestra la necesidad de afrontar con apropiadas abstracciones de programación y eficientes entornos de resolución de problemas la *crisis del software* por la que atraviesa el desarrollo de aplicaciones para entornos Grid.

6 CONCLUSIONES

Los científicos e ingenieros, como usuarios finales en sistemas distribuidos basados en entornos Grid, construyen cada vez aplicaciones más complejas para manejar y procesar grandes conjuntos de datos y ejecutar experimentos científicos sobre los recursos distribuidos. Tales escenarios de aplicación requieren medios para componer y ejecutar flujos de trabajo complejos. Por lo tanto es deseable desarrollar mecanismos que permitan manejar estas características durante todas las fases del proceso de construcción de estas aplicaciones a partir de workflows de servicios Grid.

En este trabajo hemos desarrollado un survey de los principales enfoques ASoC aplicados al desarrollo de aplicaciones Grid. Cada una de ellos ha sido presentado indicando sus principales aportes y finalmente se ha realizado una comparación de lo que estos proponen. A partir de lo investigado podemos concluir que aún falta mucha investigación en este ámbito para poder presentar casos de estudio a gran escala y con un proceso de desarrollo de software integral de este tipo de aplicaciones.

REFERENCIAS

Acher, M., Collet, P., Lahire, P. y France, R. (2010). Managing variability in workflow with feature model composition operators. In *SC'10: Proceedings Of The 9th International Conference on Software Composition*, pp. 17–33. Berlin, Heidelberg: Springer-Verlag.

Acher, M., Collet, P., Lahire, P. y Montagnat, J. (2008). Imaging services on the Grid as a product line: Requirements and architecture. In *SPLC (2): Software Product Lines, 12th International Conference*, pp. 137–142. Lero Int. Science Centre, University of Limerick, Ireland.

Aksit, M., Bergmans, L. y Vural, S. (1992). An object-oriented language-database integration model: The composition-filters approach. In *ECOOP '92: Proceedings of the European Conference on Object-Oriented Programming*, pp. 372–395. London, UK: Springer-Verlag.

Andersen, E. P. y Reenskaug, T. (1992). System design by composing structures of interacting objects. In *ECOOP '92: Proceedings of the European Conference on Object-Oriented Programming*, pp. 133–152. London, UK: Springer-Verlag.

Apel, S. y Kaestner, C. (2009). An overview of Feature-oriented software development. *Journal of Object Technology*, 8(4): 1–3.

Apel, S., Kaestner, C. y Lengauer, C. (2008). Research challenges in the tension between features and services. In *SDSOA '08: Proceedings of the 2nd International Workshop on Systems Development in SOA Environments*, pp. 53-58. NY, USA: ACM.

Baniassad, E. L. A. y Murphy, G. C. (1998). Conceptual module querying for software reengineering. In *ICSE '98: Proceedings Of the 20th International Conference on Software Engineering*, pp. 64–73. Washington, DC, USA: IEEE Computer Society.

Bashroush, R. y Perrott, R. (2005). Using a software product line approach in designing grid services. http://www.allhands.org.uk/2005/proceedings/papers/499.pdf.

Basukoski, A., Buhler, P., Getov, V., Isaiadis, S. y Weigold, T. (2008). Methodology for component-based development of Grid applications. In *CBHPC'08: Proceedings Of The 2008 Compframe/HPC-GECO Workshop On Component Based High Performance*, pp. 1:1–1:6. New York, NY, USA: ACM.

Batory, D., Chen, G., Robertson, E. y Wang, T. (2000a). Design wizards and visual programming environments for Genvoca generators. *IEEE Transaction Software Engineering*, 26(5):441–452.

Batory, D. S., Johnson, C., MacDonald, B. y Heeder, D. (2000b). Achieving extensibility through product-lines and domain-specific languages: A case study. In *ICSR-6: Proceedings of the 6th International Conerence on Software Reuse*, pp. 117–136. London, UK: Springer-Verlag.

Brichau, J., Glandrup, M., Clarke, S. y Bergmans, L. (2002). Advanced separation of concerns. In A. Frohner (Ed.) *Object-Oriented Technology* , vol. 2323 of *Lecture Notes in Computer Science*, pp. 191–212. Springer Berlin/Heidelberg.

Clements, P. y Northrop, L. (2001). *Software product lines: practices and patterns*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

CoreGRID (2007). Deliverable DPM 04 Basic Features of the Grid Component Model (assessed). Tech. Rep. Project No. FP6-004265, Institute on Programming Model - European Research Network on Foundations, Software Infrastructures and Applications for large scale distributed, GRID and Peer-to-Peer Technologies.

Czarnecki, K. y Eisenecker, U. W. (2000). *Generative programming: methods, tools, and applications*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co.

Deelman, E., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Patil, S., Su, M.-H., Vahi, K. y Livny, M. (2004). Pegasus: Mapping scientific workflows onto the Grid. In M. D. Dikaiakos (Ed.) *Grid Computing*, vol. 3165 of *Lecture Notes in Computer Science*, pp.131–140. Springer Berlin – Heidelberg.

Dörnemann, T., Friese, T., Herdt, S., Juhnke, E. y Freisleben, B. (2007). Grid Workflow Modelling using Grid-Specific BPEL extensions. In *Proceedings of German e-Science Conference 2007*, pp. 1–9.

Fielding, R. T., Gettys, J., Mogul, J. C., Nielsen, H. F., Masinter, L., Leach, P. J. y Berners-Lee (1999). RFC 2616 Hypertext Transfer Protocol – HTTP/1.1. http://tools.ietf.org/html/rfc2616.

Filman, R. E., Elrad, Z., Clarke, S. y Aksit, M. (2005). *Aspect-Oriented Software Development*. Addison-Wesley Professional.

Foster, I., Frey, J., Graham, S., Tuecke, S., Czajkowski, K., Ferguson, D., Leymann, F., Nally, M., Sedukhin, I., Snelling, D., Storey, T., Vambenepe, W. y Weerawarana, S. (2004). Modeling stateful resources with web services v.1.1. http://www-106.ibm.com/developerworks/library/ws-resource/ws-modelingresources.pdf.

Foster, I. y Kesselman, C. (2004). *The Grid: Blueprint for a new computer infrastructure*. San Francisco, USA: Morgan Kaufmann, 2nd ed.

Foster, I., Kesselman, C. y Tuecke, S. (2001). The anatomy of the Grid: Enabling scalable virtual organizations. *International Journal of High Performance Computing Applications*, 15(3):200–222.

Foster, I., Kishimoto, H., Savva, A., Berry, D., Djaoui, A., Grimshaw, A., Horn, B., Maciel, F., Siebenlist, F., Subramaniam, R., Treadwell, J. y Reich, J. V. (2005). The Open Grid Services Architecture, Version 1.0. http://www.gridforum.org/documents/GWD-I-E/GFD-I.030.pdf.

Globus Alliance (1996). Globus Toolkit. http://www.globus.org/toolkit/.

Griss, M. L. (2000). Implementing product-line features with component reuse. In *ICSR-6: Proceedings of the 6th International Conerence on Software Reuse*, pp. 137–152. London, UK: Springer-Verlag.

Harrison, W. y Ossher, H. (1993). Subject-oriented programming: a critique of pure objects. In *OOPSLA '93: Proceedings Of The Eighth Annual Conference On Object-Oriented Programming Systems, Languages, And Applications*, pp. 411–428. New York, NY, USA: ACM.

Harrison, W. H., Ossher, H. L., Tarr, P. L. y Harrison, W. (2002). Asymmetrically vs. symmetrically organized paradigms for software composition. Tech. rep., Research Report RC22685, IBM Thomas J. Watson Research.

Hernández, F., Bangalore, J. G. y Reilly, K. (2004). A graphical modeling environment for the generation of workflows for the Globus Toolkit. In *ICS 2004: Workshop on Component Models and Systems for Grid Applications, 18th Annual ACM International Conference on Supercomputing*. New York, NY, USA: ACM Press.

Hlaoui, Y. B. y BenAyed, L. J. (2009). An interactive composition of workflow applications based on UML activity diagram. *Special Issue on ICIT 2009 Conference - Applied Computing. UbiCC Journal*, 4(3):599–608.

Huhns, M. N. y Singh, M. P. (2005). Service-oriented computing: Key concepts and principles. *IEEE Internet Computing*, 9:75–81.

Hürsch, W. L. y Lopes, C. V. (1995). Separation of concerns. Tech. rep., College of Computer Science, Northeastern University.

Jia, Y. y Rajkumar, B. (2005). A taxonomy of workflow management systems for Grid computing. *Grid Computing*, 3(3-4):171–200.

Kang, K., Cohen, S., Hess, J., Novak, W. y Peterson, S. (1990). Feature-Oriented Domain Analysis (FODA). Feasibility Study. Tech. Rep. CMU/SEI-90-TR-21, ESD-90-TR-222, Software Engineering Institute, Carnegie Mellon University.

Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J. y Griswold, W. (2001). Getting started with AspectJ. *Communication*. ACM , 44:59–65.

Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.-M. y Irwin, J. (1997). Aspect-oriented programming. In M. Aksit, y S. Matsuoka (Eds.) *ECOOP'97 – Object-Oriented Programming*, vol. 1241 of *Lecture Notes in Computer Science*, pp. 220–. Springer Berlin / Heidelberg.

Li, W.-J., Huang, C.-W., Chen, Q.-C. y Bian, H. (2006). A model-driven aspect framework for Grid service development. In *APSCC'06: Proceedings of the 2006 IEEE Asia-Pacific Conference on Services Computing*, 2006.

Lieberherr, K. J. (1996). *Adaptive Object-Oriented Software: The Demeter Method with Propagation Patterns*. PWS Publishing Company, Boston. ISBN 0-534-94602-X.

Martínez, A. N. (2008). *Marco de trabajo para el desarrollo de arquitecturas software orientado a aspectos*. Ph.D. thesis, Universidad de Extremadura.

MDA (1997). Model Driven Architecture. http://www.omg.org/mda.

Meyer, B. (1992). Applying "design by contract". *Computer*, 25(10):40–51.

Meyer, B. (1997). *Object-Oriented Software Construction (2nd Edition)*. Prentice Hall, 2nd ed.

Mezini, M. y Lieberherr, K. (1998). Adaptive plug-and-play components for evolutionary software development. *SIGPLAN Not.*, 33(10):97–116.

Neubauer, F., Hoheisel, A. y Geiler, J. (2006). Workflow-based Grid applications. *Future Generation in Computer System*, 22(1):6–15.

Nuseibeh, B., Kramer, J. y Finkelstein, A. (1994). A framework for expressing the relationships between multiple views in requirements specification. *IEEE Transaction Software Engineering*, 20(10):760–773.

Open Grid Forum (2005). Open Grid Forum. http://www.ogf.org/.

Peltz, C. (2003). Web services orchestration and choreography. *Computer*, 36:46–52.

Piattini, M. G. y Garzás, J. (2007). *Fábricas de Software: experiencias, tecnologías y organización*. RA-MA.

Smith, M., Friese, T. y Freisleben, B. (2006). Model driven development of service-oriented Grid applications. In *AICT-ICIW '06: Proceedings of the Advanced International Conference*

on Telecommunications and International Conference on Internet and Web Applications and Services, p. 139. Washington, DC, USA: IEEE Computer Society.

Sobral, J. y Monteiro, M. (2008). A domain-specific language for parallel and Grid computing. In *DSAL '08: Proceedings of the 2008 AOSD workshop on Domain-specific Aspect languages*, pp. 1–4. New York, NY, USA: ACM.

Soley, R. (2000). Model driven architecture, white paper. http://www.omg.org/cgi-bin/doc?omg/00-11-05.

Sousa, E., Gonçalves, R., Neves, D. y Sobral, J. (2008). Non-invasive gridification through an Aspect-oriented approach. In *2nd Iberian Grid Infrastrucutre Conference (Ibergrid'08)*, pp. 323–336.

Szyperski, C. (1998). *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley Professional.

Tarr, P., Ossher, H., Harrison, W. y Sutton, S. M., Jr. (1999). N degrees of separation: multi-dimensional separation of concerns. In *ICSE '99: Proceedings Of The 21st International Conference On Software Engineering*, pp. 107–119. New York, NY, USA: ACM.

Trejo, N., Casas, S. y Rossi, G. (2010). Modeling Grid services using FOSD. In *Workshop de Sistemas Distribuidos y Paralelismo, Jornadas Chilenas de Computación*. Chile.

Turner, K. J. y Tan, K. L. L. (2006). Orchestrating Grid services using BPEL and Globus Toolkit 4. In *7th PGNet Symposium*, pp. 31–36. Liverpool, UK: School of Computing, Liverpool John Moores University.

W3C Consortium (2004). Web Services Architecture. http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/.

W3C Consortium (2007). SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). http://www.w3.org/TR/2007/REC-soap12-part1-20070427/.

W3C Consortium (2008). Extensible Markup Language (XML) 1.0 (Fifth Edition). http://www.w3.org/TR/2008/REC-xml-20081126/.

Weiss, D. y Lai, C. (1999). *Software Product-Line Engineering: A Family-Based Software Development Process*. Addison-Wesley.

Workflow Management Coalition (1993). WfMC. http://www.wfmc.org