Serie Científica de la Universidad de las Ciencias Informáticas
http://publicaciones.uci.cu/index.php/SC | seriecientifica@uci.cu
No. 3, Vol. 4, Año: 2011
ISSN: | RNPS:

Tipo de artículo: Artículo original
Temática: Ingeniería de software – Computación paralela
Recibido: 24/03/2011 | Publicado: 31/03/2011

Propuestas de Ingeniería de Software para la Computación de Alto Rendimiento

Software Engineering Proposals for High Performance Computing

Laritza Cabrera Barroso[1], Enrique Soler Castillo[2]

[1] Geitel. Departamento de Gestión Documental y Archivística. Universidad de las Ciencias Informáticas, Carretera a San Antonio de los Baños, km 2 ½, Torrens, Boyeros, La Habana, Cuba. CP19370
[2] Departamento de Lenguajes y Ciencias de la Computación. Universidad de Málaga. España.

lcabrera@uci.cu, esc@lcc.uma.es

Resumen
La computación científica se ha convertido en un área de enorme relevancia dentro de las ciencias de la computación. El número de usuarios interesados en el uso de técnicas computacionales para resolver problemas científicos, está creciendo notablemente. Asimismo, la complejidad y el elevado coste computacional de dichos problemas, han impulsado el desarrollo de entornos avanzados para la Computación paralela y distribuida de Alto Rendimiento. Gracias a ello, científicos e ingenieros han logrado abordar situaciones a gran escala que resultarían intratables sin el uso de estas novedosas tecnologías. En el presente artículo, se exponen los principios básicos de la computación paralela y se realiza un estudio de los modernos paradigmas que han sido aplicados a la Computación de Alto Rendimiento, con el fin de liberar al programador de la mayoría de los detalles en la explotación del paralelismo. Se presentan además, las más relevantes y novedosas tecnologías desarrolladas bajo la filosofía de estos paradigmas, haciendo especial énfasis en las propuestas basadas en componentes software.
Palabras clave: Componente software, computación científica, computación de alto rendimiento, computación paralela, computación distribuida.

Abstract:
Scientific computing has become an area of great relevance in computer science. The number of users interested in using computational techniques to solve scientific problems, is growing significantly. Also, the complexity and high computational cost of these problems have prompted the development of advanced environments for parallel and distributed High Performance Computing. As a result, scientists and engineers have achieved to solve large-scale situations that would be intractable without the use of these novel technologies. In this paper, we present the basic principles of parallel computing and a study of modern paradigms that have been applied to the High Performance Computing, in order to free the programmer from most of the details on the exploitation of parallelism. It also presents the most relevant and innovative technologies developed under the philosophy of these paradigms, with particular emphasis on component-based proposals.

Keywords: *Distributed computing, high performance computing, scientific computing, parallel computing, software component.*

1. Introducción

El acelerado desarrollo en las distintas ramas de la ciencia y la ingeniería, exigen el diseño de novedosas técnicas de computación de altas prestaciones, que permitan el procesamiento de grandes cantidades de datos, reduciendo los tiempos de respuesta y posibilitando el tratamiento de problemas complejos. La computación científica, área de enorme relevancia dentro de las ciencias de la computación, es la disciplina que aborda la construcción de modelos matemáticos y métodos numéricos para resolver, eficientemente, problemas científicos y de ingeniería usando ordenadores.

Las aplicaciones científicas tienen actualmente un fuerte impacto económico y social. Por ejemplo, la investigación acerca del genoma humano realizada por el *Internacional Human Genome Sequencing Consortium* abre nuevas posibilidades en Bioinformática. Asimismo, avances en la física y química computacional, la astrofísica, la simulación medioambiental y en muchos otros dominios, han permitido el estudio de numerosos procesos.

Debido a los fuertes requisitos de computación y memoria, los problemas abordados en la computación científica se consideran intratables sin el uso de ordenadores paralelos. Dichos ordenadores se caracterizan por disponer de un conjunto de procesadores, capaces de trabajar de forma colaborativa en la realización de una tarea computacional. Muchos de los problemas que se intentan resolver en el campo de los Sistemas Inteligentes, necesitan de una elevada potencia de cálculo, que sólo pueden ser abordados por medio de la implementación de algoritmos paralelos, que permitan aprovechar distintos tipos de arquitecturas hardware en las que se dispone de más de un procesador.

La programación de aplicaciones científicas paralelas ha estado tradicionalmente basada en mecanismos de bajo nivel de abstracción, con el objetivo de lograr un control preciso del rendimiento. Aunque han podido desarrollarse aplicaciones muy eficientes, el bajo nivel de abstracción requiere conocimientos profundos de los mecanismos de comunicación y sincronización. Afortunadamente, después de muchos años en los que la construcción de aplicaciones de altas prestaciones, implicaba el conocimiento de un lenguaje o biblioteca de funciones específicos, actualmente existen estándares que permiten la construcción de aplicaciones, ejecutables en distintos entornos de multicomputadores, aprovechando de forma transparente las características diferenciales de las distintas arquitecturas.

La creciente demanda de aplicaciones de alto rendimiento, no sólo a nivel científico, sino a nivel industrial, comercial y técnico, unido a la complejidad inherente al desarrollo de las mismas, han impulsado la construcción de entornos avanzados para la Computación paralela y distribuida de Alto Rendimiento (CAR). Novedosas prácticas de la ingeniería de software han sido utilizadas con el propósito de disminuir el coste asociado al desarrollo de este tipo de aplicaciones.

El presente artículo tiene como objetivo, la realización de un informe del estado del arte sobre las diferentes propuestas existentes en ingeniería del software para la Computación de Alto Rendimiento, haciendo particular énfasis en las propuestas basadas en componentes software. Para ello se exponen los principios básicos de la Computación Paralela y se realiza un estudio de los modernos paradigmas orientados al desarrollo de aplicaciones de alto rendimiento, referenciando las más relevantes y novedosas tecnologías en cada caso. Se establece además, la clasificación y comparativa de las principales propuestas basadas en componentes software.

2. Computación Paralela

La computación paralela permite el desarrollo de aplicaciones que aprovechan el uso de múltiples procesadores de manera colaborativa, con el fin de resolver un problema común. El objetivo fundamental que persiguen estas técnicas, es reducir el tiempo de ejecución de una aplicación mediante el empleo de múltiples procesadores. Estas propuestas abarcan desde estaciones de trabajo con varios procesadores, hasta superordenadores con cientos o miles de ellos, incluyendo las redes de área local, clúster y sistemas empotrados.

2.1 Clasificación de Arquitecturas Paralelas

En 1966 Michael J. Flynn (Flynn, 1966) propuso un modelo de clasificación de computadoras en cuatro categorías. Esta organización se basa en la pluralidad de instrucciones y datos que la computadora utiliza para procesar información.

Puede haber secuencias de instrucciones sencillas o múltiples y secuencias de datos sencillas o múltiples.

- *Single Instruction - Single Data (SISD),* clasifica los sistemas con un único procesador tal como se describe en la arquitectura tradicional de von *Neumann.*
- *Single Instruction - Multiple Data (SIMD),* la misma instrucción se ejecuta de forma sincronizada por varios procesadores que operan sobre datos distintos.
- *Multiple Instructions - Single Data (MISD),* es la clase menos extendida, ya que múltiples flujos de instrucciones suelen requerir diferentes flujos de datos.
- *Multiple Instructions - Multiple Data (MIMD),* es la categoría a la que pertenecen la mayor parte de los sistemas paralelos actuales. Existen múltiples procesadores autónomos, cada uno ejecutando sus propias instrucciones sobre sus datos.

Las arquitecturas del tipo MIMD suelen clasificarse según la organización de su memoria en dos grupos. Los sistemas SMP *(symmetric shared-memory multiprocessor),* disponen de un conjunto de procesadores, todos conectados a una única memoria central. Esta arquitectura es también conocida como UMA *(uniform memory access),* por la relación simétrica en el acceso a memoria de todos los procesadores. En el otro grupo, los multiprocesadores de memoria distribuida, constituidos por una colección de nodos conectados por una red de alta velocidad, donde cada nodo tiene su propio procesador, memoria local y sistema de entrada y salida.

Desde el punto de vista de ingeniería de software, estas arquitecturas han dado lugar a nuevos modelos de programación que separan las propiedades de alto nivel de aquellas de más bajo nivel. Estos modelos se agrupan en dos clases: Modelos de Memoria Compartida y Modelos de Memoria Distribuida.

En los modelos de memoria compartida, el programador debe especificar las actividades de un conjunto de procesos que se comunican a través de la lectura y escritura asíncrona de datos en un espacio de memoria común. Aspectos como la distribución de datos, no necesitan ser programados, ya que todos los procesos utilizan una misma memoria. Debe controlarse, en cambio, el acceso concurrente a los datos usando mecanismos de sincronización, como puede ser semáforos o variables de condición. En los modelos de memoria distribuida, los procesos únicamente acceden a una memoria local y deben emplear otros mecanismos, como el paso de mensajes o llamada a procedimiento remoto RPC *(Remote Procedure Call),* para intercambiar la información. Estos modelos proporcionan generalmente mayor rendimiento que los de memoria compartida.

Otras clasificaciones que suelen ser usadas para caracterizar los modelos de programación son: Paralelismo de datos vs Paralelismo de tareas y Paralelismo Explícito vs Paralelismo Implícito.

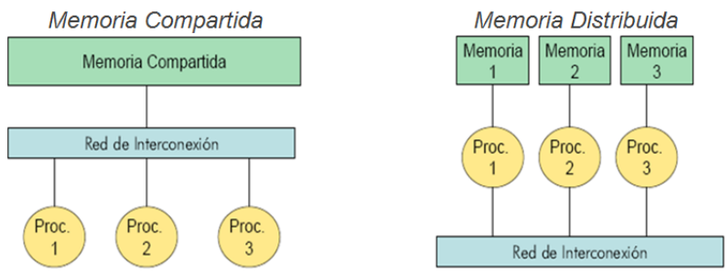

Figura 1. Modelos de Programación Paralela.

Entre las principales herramientas de programación paralela figuran: *OpenMP*, estándar unificado y portable de paralelismo en entornos de memoria compartida y *MPI (Message Passing Interface)*, librería de rutinas para el desarrollo de programas paralelos basados en paso de mensajes. Aunque este tipo de herramientas ha probado su efectividad en cuanto a la implementación de aplicaciones, son de muy bajo nivel para permitir el diseño y construcción de grandes aplicaciones distribuidas con el nivel de abstracción necesario.

3. Técnicas modernas de Ingeniería de Software para la CAR

El principal obstáculo para la difusión de la programación paralela es precisamente su complejidad. Desde el punto de vista de la Ingeniería de Software, novedosas propuestas han sido desarrolladas con el fin de disminuir el coste asociado al desarrollo de software científico. La Programación Paralela Estructurada, Programación Basada en Componentes y Programación Orientada a Aspectos, constituyen modernos paradigmas que liberan al programador de la mayoría de los detalles en la explotación del paralelismo. Asimismo, alternativas como las infraestructuras *Grid*, y más recientemente los sistemas *Cloud*, aparecen como Paradigmas de Computación Distribuida, que permiten el acceso transparente y seguro a enormes sistemas en red distribuidos de forma geográfica.

3.1 Paradigmas de Computación Distribuida

El uso de arquitecturas de computación paralelas ha permitido el desarrollo de importantes áreas en múltiples campos de la ciencia básica y aplicada. Sin embargo, el elevado coste de los sistemas paralelos tradicionales, ha limitado su uso a grandes industrias y centros de investigación. Hace algunos años, el uso de redes de computadores de bajo coste representa una alternativa práctica y económica a los grandes sistemas. Alternativas como las infraestructuras *Grid*, y más recientemente los sistemas *Cloud*, aparecen como Paradigmas de Computación Distribuida que cambian el modo de utilizar los computadores, permitiendo el acceso transparente y seguro a enormes recursos computacionales.

El surgimiento de los multicomputadores ha dado lugar a la computación basada en *clusters* de PCs, que ofrecen un buen ratio coste/prestaciones para la ejecución de aplicaciones paralelas. Una extensión al concepto de computación basada en *clusters* de PCs, es la llamada computación basada en Intranet, que permite el uso de los equipos de una organización para la ejecución de trabajos secuenciales o paralelos. Sin embargo, cuando el ámbito de las aplicaciones atraviesa las fronteras de una única organización, surgen una serie de problemas que no aparecen en la computación basada en Intranet. El hecho de trabajar con diferentes dominios de administración, que pueden estar sujetos a diversas políticas de seguridad, gestión y localización, supone un nuevo desafío para las estrategias de computación. Por lo tanto, la necesidad de una infraestructura de computación distribuida, que aglutine recursos de diferentes centros geográficamente dispersos, da lugar al surgimiento de la Computación *Grid (*Malla o Grilla).

La principal idea que subyace a las tecnologías *Grid*, es ofrecer una interfaz homogénea y estándar para acceder a los recursos.

3.1.1 Computación *Grid*

Ian Foster (Foster & Kesselman, The Grid 2: *Blueprint for a new Computing Infrastructure*, 2004) define *Grid* como, sistema que coordina recursos, que no están sujetos a un control centralizado, usando interfaces y protocolos estándares, abiertos y de propósito general para proveer calidades de servicios no triviales. Definiciones más recientes hacen hincapié en la capacidad de combinar los recursos de las diferentes organizaciones con un objetivo común.

La tecnología *Grid* han alcanzado gran popularidad y su espectro de aplicación abarca múltiples áreas de la ciencia y la ingeniería. En ese sentido, existen varios proyectos relacionados con el desarrollo de *middlewares Grid*, como *Polder*, desarrollado en la Universidad de *Amsterdam*, o Condor, desarrollado en la Universidad de *Wisconsin*. Sin embargo, en los últimos años, Globus Toolkit (Foster & Kesselman, Globus: A metacomputing infrastructure toolkit, 1997) se ha convertido en el estándar de facto para el despliegue de *Grids* computacionales. Esta herramienta de libre distribución permite la creación de *Grids*, facilitando la compartición de recursos computacionales de manera segura, a través de diferentes organizaciones, sin sacrificar la autonomía local de los diferentes dominios de administración que integren el *Grid*. Incluye servicios y librerías para la monitorización y gestión de recursos, así como para la seguridad y gestión de ficheros.

La Computación *Grid* ha sido muy usada en la e-Ciencia, como apoyo en la búsqueda de soluciones a problemáticas reales del planeta, gracias a la ejecución de proyectos que serían imposibles de realizar sin el poder computacional de una *Grid*.

Un ejemplo de proyectos científico desarrollado sobre esta tecnología es, *Seti@home* (Seti@home), cuya meta consiste en el procesamiento de señales de radio para la búsqueda de posibles pruebas de inteligencia extraterrestre. Este proyecto fue el primer intento de computación distribuida realizado con éxito y en el que actualmente participan más de 4 millones de voluntarios de todo el mundo. La plataforma BOINC *(Berkley Open Infraestructure for Network Computing)*, tecnología *GRID* sobre la que funciona *Seti@home*, es considerada como un cuasi-superordenador, por su alto rendimiento computacional medio, de hasta 2 PetaFLOPS.

3.1.2 Computación en Nubes

La Computación en Nubes *(Clouds Computing)* es otro paradigma de Computación Distribuida, asociado al suministro de infraestructuras de computación. Existe cierta ambigüedad entre la clara definición de esta propuesta y la Computación *Grid*, debido a que ambas comparten visiones similares: reducir los costes informáticos y aumentar la flexibilidad y fiabilidad mediante el uso de hardwares operados por terceros.

En (Vaquero, Merino, & Caceres, 2009) se define Nubes como, conjunto de recursos virtuales, fácilmente usables y accesibles (tales como hardware, plataformas de desarrollo y / o servicios). Estos recursos pueden ser reconfigurados dinámicamente, contemplando asimismo la utilización óptima de los recursos. Este conjunto de recursos suele ser explotado por un modelo de pago por uso, en el que las garantías son ofrecidas por el proveedor de infraestructuras.

Desde el punto de vista científico, la interpretación más popular de Computación en Nubes es la de Infraestructura como un servicio *(Infrastructure as a Service (IaaS))*, que proporciona medios genéricos para el *hosting* y suministro del acceso a la infraestructura y su software de funcionamiento.

Mientras en Computación *Grid*, un proyecto grande es dividido entre múltiples computadoras para hacer uso de sus recursos, la Computación en Nubes, permite que múltiples pequeñas aplicaciones sean ejecutadas al mismo tiempo.

La virtualización de hardware, es una de las ventajas que convierten a este paradigma en una atractiva alternativa para los científicos de la computación. Esta propuesta constituye un avance significativo para el despliegue automático y escalable de software científico complejo, posibilitando además, mejorar significativamente la utilización de los recursos compartidos.

3.2 Programación Paralela Estructurada

La propuesta de programación paralela estructurada o basada en esqueletos (Pelagatti, 1998) (Cole, 1989), consiste en capturar estilos algorítmicos reutilizables en abstracciones de alto nivel. Al programador se le ofrece un conjunto de esqueletos para estructurar el paralelismo de la aplicación.

Muchos programas paralelos se basan en colecciones de procesos que, lejos de interactuar de forma aleatoria y poco predecible, lo hacen de una manera estructurada y siguiendo ciertos patrones que suelen repetirse en otros programas. La idea de la programación basada en esqueletos consiste en abstraer estos patrones que suelen repetirse y crear librerías de esqueletos algorítmicos, o formas genéricas y reutilizables de computación e interacción, que pueda ser ofrecida al programador como herramienta para definir el paralelismo a alto nivel. En este enfoque, desarrollar un programa consiste en seleccionar e instanciar el grupo de esqueletos, que represente la solución paralela adecuada al problema que se pretende resolver.

Algunos de los sistemas de esqueletos más populares son: *El Sistema de Cole*, lenguaje paralelo explícito de alto nivel *P3L: Pisa Parallel Programming Language*, los lenguajes de coordinación *SCL* y *DIP (Domain Interaction Patterns)* y las librerías *eSkel* y *Muskel*.

A pesar de las ventajas que promete la programación basada en esqueletos, ésta sigue siendo actualmente una tecnología poco extendida, comparada con otros enfoques.

3. 3 Programación Basada en Componentes

El enfoque de componentes proporciona una forma de descomposición del software en unidades que son conceptualmente manejables y que interactúan de una manera específica y de fácil comprensión. La programación orientada a componentes, o POC (Heineman & Council, 2001) se puede entender como una evolución natural de la programación orientada a objetos para sistemas abiertos, los cuales se caracterizan por ser concurrentes, reactivos, extensibles, distribuidos y por permitir que sus componentes heterogéneos ingresen o abandonen el sistema de forma dinámica.

Las tecnologías de componentes software soportan el desarrollo de un estilo particular que incluye: componentes, modelo de componentes y plataformas de componentes. En (Szypersky, 1998) se define componentes software como: unidad de composición de aplicaciones, con un conjunto de interfaces y requisitos, que ha de poder ser desarrollado, adquirido, incorporado al sistema y compuesto con otros componentes de forma independiente, en tiempo y espacio. El modelo de componentes especifica reglas de diseño a los desarrolladores y el *framework* o plataforma de componentes, provee un conjunto de servicios para soportar e implementar dichas reglas.

Ventajas como la reutilización, configuración, flexibilidad y alto nivel de abstracción en el diseño, implementación, despliegue y mantenimiento de aplicaciones, hacen que este paradigma sea especialmente aplicado a la computación científica de alto rendimiento. Sin embargo los estándares de componentes más populares como: OMG CCM, *Microsoft COM+ y Sun Enterprise JavaBeans*, son poco efectivos a la hora de cumplir requisitos de software científico, ya que carecen de abstracciones que permitan capturar conceptos de alto rendimiento en la arquitectura de

componentes. En este sentido, grandes esfuerzos se dedican actualmente al desarrollo de modelos y arquitecturas de componentes para la Computación de Alto Rendimiento.

3.3.1 Modelos y Arquitecturas de componentes

CORBA (OMG, 2003), ha sido utilizado como mecanismo básico de componentes encargado de proveer entornos avanzados de programación paralela. *PACO++: Parallel CORBA Object* (Pérez, Priol, & Ribes, 2004) es una implementación portable del concepto de objeto CORBA paralelo. Esta especificación permite la integración de código SPMD *(Single Program-Multiple Data)* en objetos paralelos, cada uno representado por una colección de objetos CORBA idénticos. La capa de software PACO++ que se introduce entre el código del usuario y el código del *stub/skeleton*, tiene como objetivo capturar las llamadas CORBA para gestionar las invocaciones sobre los objetos paralelos. Esta realiza la distribución de datos de los argumentos y ejecuta las operaciones sobre los objetos correspondientes. Una ventaja significativa de PACO++ con respecto a otras propuestas basadas en CORBA, es que no se requiere modificaciones en el IDL *(Interface Definition Language)* ni el ORB *(Object Request Broquer).*

El modelo de componentes *CCA (Common Component Architecture)* (Armstrong, y otros, 2006) es una fecunda propuesta llevada a cabo por CCA Forum, grupo de investigadores de laboratorios nacionales e instituciones académicas, con el propósito de definir una arquitectura de componentes estándar para la Computación de Alto Rendimiento. En esta especificación se describen los mecanismos de composición y conjunto de servicios asociados a un marco de trabajo *(framework).* El objetivo de CCA es lograr la interoperabilidad entre lenguajes, de forma tal que una misma aplicación pueda usar componentes escritos en lenguajes como C, C++, *Fortran* 77, *Fortran* 90 o *Phyton.* SIDL *(Scientific IDL)* es el lenguaje utilizado para describir las interfaces de los componentes y Babel el compilador asociado para generar todo el código que une las llamadas entre lenguajes distintos. *Ccaffeine*, XCAT, *SCIRun*; son marcos de trabajo compatibles con CCA que exploran diferentes áreas de la computación científica.

Recientemente, varias propuestas han sido concebidas como resultado de la integración de esqueletos software y tecnología de componentes. Ejemplos de entornos de programación desarrollados bajo esta filosofía son:

ASSIST (Vanneschi, 2002) y *SBASCO* (Díaz, Rubio, Soler, & Troya, 2004).

ASSIST, centrado en la programación de alto nivel y productividad de software, se beneficia del uso de componentes, e intenta superar algunas limitaciones del enfoque clásico de esqueletos. Permite al programador el diseño de aplicaciones en forma de grafo genérico de componentes paralelos, representados a través de una construcción de alto nivel llamada; "módulo paralelo" o *ParMod (Parallel Module).* El concepto de *ParMod* puede entenderse como un tipo genérico y configurable de esqueleto paralelo. Es un nuevo paradigma que además de expresar la semántica de varios esqueletos, como casos particulares, es capaz de expresar computaciones más complejas y de un carácter menos estructurado.

SBASCO (Skeleton- BAsed Scientific COmponents), es un entono de programación orientado al desarrollo eficiente de aplicaciones numéricas paralelas y distribuidas. Esta aproximación proporciona interoperabilidad, capacidad de programación de alto nivel, composición y portabilidad de rendimiento. Los llamados componentes científicos encapsulan las tareas de computación que resuelven el problema numérico, las cuales pueden ser secuenciales o paralelas. El uso de esqueletos permite establecer la estructura de paralelismo interna de un componente, definiendo un alto nivel de interacción y comunicación entre sus tareas.

AspectSBASCO, es una extensión del modelo *SBASCO*, donde se adicionan conceptos de programación orientada a aspectos. Los aspectos son tratados como componentes que implementan funcionalidades transversales *(cross-*

cutting). Esta aproximación multi-paradigma provee un alto nivel de paralelismo, reusabilidad, interoperabilidad y separación clara de competencias.

La computación *Grid* por su parte, plantea importantes retos desde el punto de vista de los modelos de programación, ya que consiste en la programación de aplicaciones que se ejecutan sobre recursos heterogéneos a gran escala, que evolucionan dinámicamente. *GCM (Modelo de Componentes Grid)* (Baude, y otros, 2009), extensión del Modelo de Componentes Jerárquico, Fractal (Bruneton, Coupaye, Leclercq, Quéma, & Stefani, 2006), es una aproximación de componentes que facilita la programación de aplicaciones *Grid*. Como este tipo de aplicaciones generalmente son resultado de la composición de muchos componentes paralelos (generalmente idénticos), en el modelo se han incorporado mecanismos de composición, que soportan la comunicación colectiva en un conjunto de componentes. El impacto de las interfaces colectivas asociadas con las jerarquías, es que cualquier componente de un sistema puede ser considerado como, y transformado en un componente paralelo de una forma muy natural.

En cierto sentido, la programación de componentes aparece como una composición espacial describiendo la conexión entre componentes, mientras la programación de servicios promueve una composición temporal que expresa la planificación y flujo de control entre los servicios. En el contexto de la *Grid*, ambos enfoques han sido utilizados por separado. En (Bouziane, Pérez, & Priol, 2008) se presenta el *Modelo de Componentes Espacio-Temporal STCM* que combina ambos enfoques, logrando un fuerte acoplamiento de código y eficiente uso de los recursos.

Como una extensión de este modelo, recientemente se ha presentado el *Modelo de Esqueletos Espacio-Temporal SKTM* (Aldinucci, Bouziane, Pérez, & Danelutto, 2009). Esta propuesta combina tecnología de componentes y servicios según propone el modelo STCM e incorpora conceptos de esqueletos algorítmicos. Proporciona mayor facilidad de diseño y separación de los aspectos funcionales y no funcionales en el sistema, así como la portabilidad de aplicaciones a diferentes contextos de ejecución. Este modelo ofrece elevados niveles de abstracción que permiten expresar el comportamiento funcional de una aplicación a través de su ensamblado. Componentes STKM constituyen una extensión de componentes GCM, implementados sobre SCA (Arquitectura de Componentes y Servicios), donde dichos componentes se muestran como servicios provistos por implementaciones de usuarios.

Sin dudas, la ingeniería de software basada en componentes brinda una acertada perspectiva para enfrentar la complejidad del software científico moderno. Todas estas propuestas, en general, plantean el desarrollo y composición de un conjunto componentes científicos de alto rendimiento, desde diferentes técnicas o estilos de programación. En la tabla 1, se clasifican y comparan los modelos y arquitecturas de componentes referenciadas, especificando en cada caso, tipo de componentes, elementos definidos para el ensamblado o composición de los mismos, así como los modelos de programación aplicados.

Tabla1. Propuestas basadas en componentes software.

Propuesta	**Clasificación**	**Tipo de componentes**	**Composición de componentes**	**Modelos de programación**
PACO++	Modelo de Objetos Paralelos.	-Objeto CORBA paralelo -Objeto CORBA estándar	- librerías MPI o PVM entre instancias del objeto paralelo. -Capa software Paco ++ -Proxy (Interface Manager, Output Manager)	Programación Basada en Componentes(PBC)
CCA	Modelo de Componentes para la Computación Científica.	Componentes científicos Paralelos y distribuidos.	Patrón de interfaz proporcionada/requerida. *Puertos* - elementos finales de conexión.	Programación Basada en Componentes(PBC)
ASSIST	Entorno de desarrollo de aplicaciones paralelas.	Módulos ASSIST paralelos y secuenciales.	-Canales entrada/salida. -Librerías de memoria compartida-distribuida -Objetos CORBA	- PBC - Programación Basada en Esqueletos(PBE)
SBASCO	Modelo de Componentes para el desarrollo de aplicaciones numéricas paralelas y distribuidas.	Componentes científicos paralelos o secuenciales.	Lenguaje de composición basado en el uso de esqueletos paralelos.	- PBC - PBE
Aspect-SBASCO	Extensión del modelo SBASCO orientado a la programación de aspectos.	(SCs) – Componentes Científicos (ACs) – Componentes Aspectos	SC–SC: primitivas de flujo de datos. SC–AC: llamadas a métodos y controlada por Conectores de Aspecto (ACNs).	- PBC - PBE - Programación Orientada a Aspectos(POA)
GCM	Modelo de Componentes Grid.	Paralelos y distribuidos Primitivos o Compuestos	Lenguaje de descripción de arquitectura (ADL). Composición jerárquica y colectiva a partir de Interfaces *Multicast* y *Gathercast.*	- PBC
STCM	Modelo de Componentes Espacio-Temporal.	Task-Component (Componente-tarea)	Modelo de composición basado en AGWL (Abstract Grid Workow lenguage) y GCM ADL con soporte de puertos temporales.	-PBC - Programación Orientada a Servicios(POS)
STKM	Modelo de Esqueletos Espacio-Temporal.	Task-Component Skeleton construct (construcción esqueleto)	Dependencia espacial y temporal.	- PBC - POS - PBE

Al mismo tiempo que la ciencia computacional avanza hacia aplicaciones más realistas, multi-físicas y multi-escala, se torna cada vez más difícil la selección y ajuste de todos los componentes de una aplicación determinada. En este sentido, no existe una correcta estrategia de solución que pueda abarcar todo este espectro de manera eficiente. Interfaces de componentes comunes, junto a la interoperabilidad de lenguajes de programación y composición

dinámica, son características claves de la tecnología de componentes. Los principales retos de investigación van dirigidos fundamentalmente a; cómo, en tiempo de ejecución, tomar las mejores decisiones en cuanto a confiabilidad, precisión y rendimiento.

Los principales esfuerzos realizados en cuanto a Calidad de Servicios (QoS), de las propuestas basadas en componentes, de forma general van dirigidos hacia; la reconfiguración dinámica de componentes en tiempo de ejecución, creación de infraestructuras de medición y análisis de rendimiento, infraestructuras de control para el reemplazo dinámico de componentes y toma de decisiones de dominio específico, mecanismos para la auto-adaptación de componentes a entornos en evolución y requisitos cambiantes, composición colectiva de grupos de componentes y gestión automática de la composición.

Estas son solo una representación de las muchas propuestas basadas en componentes software que han sido desarrolladas para enfrentar la complejidad de las aplicaciones científicas. El Desarrollo basado en componentes ha resultado en una tecnología ampliamente usada para la computación de alto rendimiento. Extiende los beneficios del diseño orientado a objetos y proporciona metodologías de codificación e infraestructura de apoyo para mejorar la extensibilidad, fiabilidad y mantenimiento de software. Actualmente existe una marcada tendencia a combinar este paradigma con otros enfoques, aprovechando las bondades y solucionando las limitaciones de uno y otro paradigma.

Según muestra esta sección, eficientes propuestas han sido desarrolladas bajo esta filosofía.

3.4 Programación Orientada a Aspectos

En el diseño de sistemas software existen conceptos o propiedades que afectan todos los módulos del sistema de forma transversal y no pueden ser encapsulados en una única unidad. La programación orientada a aspectos o POA (Elrad, Filman, & Bader, 2001), propone una solución para el manejo modular de estas propiedades transversales, denominadas *cross-cutting concerns*, permitiendo la descomposición del sistema en base a múltiples dimensiones.

El software científico suele recurrir al procesamiento paralelo para resolver problemas de gran escala típicos en ciencias e ingeniería. Las aplicaciones, normalmente, se basan en librerías como MPI o PVM, que ofrecen al desarrollador una colección de primitivas y tipos de datos para la implementación del paralelismo. Un problema muy común en esta clase de aplicaciones es la presencia del llamado “código enmarañado” debido a la mezcla, en los mismos bloques de código, de sentencias que expresan paralelismo con otras que describen un modelo matemático. La programación orientada a aspecto, es una herramienta útil para la separación de competencias y puede suponer una solución a este problema. Permite separar aspectos como la sincronización, distribución, seguridad, administración de memoria, entre otros, de las funcionalidades básicas del sistema, según muestra la Figura 2.

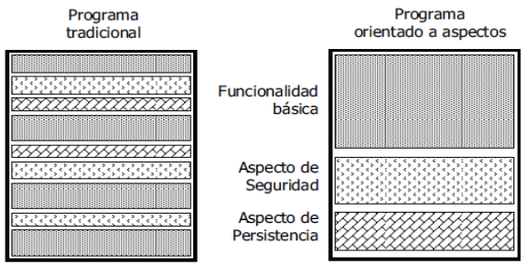

Figura2. Diseño tradicional y orientado a aspectos.

Las ventajas que propone este paradigma se resumen en modularidad, capacidad de evolución y reutilización de código. Los conceptos se implementan en unidades separadas, resultando más fácil integrar nuevas funcionalidades mediante la implementación de aspectos que podrían añadirse al sistema sin necesidad de rediseñar toda la aplicación. A través de esta clara separación de conceptos se obtienen diseños menos acoplados que facilitan la reutilización.

3. 4.1 Principales tecnologías

AspectJ (Kiczales, Hilsdale, Hugunin, Kersten, Palm, & Griswold, 2001), lenguaje de propósito general, es una extensión orientada a aspectos de Java para la definición y manejo de aspectos. Actualmente constituye la propuesta más popular y robusta de desarrollo orientado a aspectos. El código *AspectJ* es compilado es el estándar *bytecode* de Java.

Con un enfoque similar, el lenguaje *AspectC++* (Spinczyk, Gal, & Schröde, 2002) ofrece una alternativa para la programación orientada a aspectos en C y C++.

Actualmente muy pocas son las propuestas que han aplicado programación orientada a aspectos a la Computación de Alto Rendimiento. Algunas aproximaciones van dirigidas fundamentalmente al uso de lenguajes de propósito general (principalmente *AspectJ*) con diversos fines. Ejemplo de ello son las propuestas de Harbulot y Gurd. En (Harbulot & Gurd, 2004), se propone la separación de la estructura de paralelismo que atraviesa un conjunto de aplicaciones Java mediante el uso de *AspectJ*. En (Harbulot & Gurd, 2006), se presenta una extensión al modelo de puntos de unión de *AspectJ* que permite la paralelización de los bucles sin tener que rediseñar la aplicación. En (Sobral, 2006) se presenta una metodología para el desarrollo modular de programas paralelos siguiendo un enfoque incremental, sobre la base de componer distintos aspectos de paralelismo como son concurrencia, partición, distribución y optimización.

Por otro lado, la integración de aspectos y componentes software ha resultado en interesantes propuestas que logran aprovechar las ventajas de ambos paradigmas. La representación de los aspectos como componentes reutilizables y el uso de un lenguaje de componentes como lenguaje base, son los principales retos que enfrenta este enfoque. En el área de la computación de alto rendimiento, son realmente escasos los esfuerzos encaminados al desarrollo de tecnologías que integren aspectos y componentes. En este sentido, *AspectSBASCO* (Díaz, Romero, Rubio, Soler, & Troya, 2009), propuesta analizada en la sección anterior, se presenta como un poderoso entorno de programación que combina aspectos, componentes y esqueletos software. La idea de esta propuesta es lograr un alto nivel de modularidad y reutilización de código en aplicaciones científicas paralelas.

4. Conclusiones

La complejidad que supone el desarrollo de aplicaciones científicas, ha motivado en gran medida importantes avances en el campo de la CAR. Mediante este artículo, se presentan las principales y más novedosas prácticas de la ingeniería de software en esta área. La Programación Paralela Estructurada, el Desarrollo Basado en Componentes y la Programación Orientada a Aspectos, constituyen modernos paradigmas de programación, que implementan abstracciones de alto nivel para software científico complejo. Asimismo, la Computación *Grid* y Computación en Nubes, se presentan como alternativas efectivas y económicas, que permiten el acceso transparente, seguro y barato a enormes recursos computacionales distribuidos geográficamente. A lo largo del artículo, se ha hecho un análisis de las novedosas tecnologías que aplican los principios de estos paradigmas a la CAR.

Especialmente la programación Basada en Componentes, ha resultado una técnica muy utilizada en esta área. La idea de descomposición del problema en partes más simples y reutilizables, brinda incalculables ventajas para el desarrollo y mantenimiento de las aplicaciones científicas. Se ha establecido la clasificación y comparación de los principales modelos y arquitecturas de componentes desarrollados para la construcción de aplicaciones científicas. Muchas de estas propuestas han surgido como resultado de la integración de la tecnología de componentes con otros paradigmas, fundamentalmente, con la programación Paralela Estructurada. Este paradigma posibilita la programación de alto nivel, basada en la implementación de esqueletos que logran ocultar todos los detalles de la arquitectura y ejecución paralela. Siendo así, los programadores pueden centrar su esfuerzo en las tareas computacionales de la aplicación, en

lugar de dedicarlo al control y coordinación del paralelismo. De igual forma que en los componentes científicos queda encapsulada la experiencia de sus desarrolladores, permitiendo así componer aplicaciones más complejas, los esqueletos software permiten reutilizar estructuras de paralelismo definidas y disponer de un conjunto prediseñado de soluciones paralelas que han sido optimizadas y depuradas, aumentando la productividad y minimizando las probabilidades de errores de programación.

La Programación Orientada a Aspectos por su parte, logra resolver problemas de código mezclado y diseminado que aparecen en algoritmos paralelos, permitiendo la fácil separación de conceptos. De esta forma se obtienen diseños menos acoplados que facilitan la reutilización.

Las tendencias actuales de usar estas tecnologías de forma unificada tienen como objetivo, mejorar el software científico respecto a otros estilos clásicos de programación, en términos de un mayor rendimiento, modularidad, facilidad de desarrollo, adaptabilidad, reutilización y fiabilidad.

5. Agradecimientos

La siguiente contribución es resultado del Programa Oficial de Formación de Másteres en Ingeniería de Software e Inteligencia Artificial desarrollado por la Universidad de Málaga, España, en conjunto con la Universidad de las Ciencias Informáticas.

Referencias

- ALDINUCCI, M., BOUZIANE, L., PÉREZ, CH. and DANELUTTO, M. "STKM on SCA: a Unified Framework with Components, Workflows and Algorithmic Skeletons," LNCS. Vol. 5704, 2009, p. 678--690.
- ARMSTRONG, R., *et al*. "The CCA Component Model for High-Performance Scientific Computing," Concurrency and Computation: Practice and Experience. Vol. 18, 2006, p. 215 – 229.
- BAUDE, F. *et al*., "GCM: a Grid Extension to Fractal for Autonomous Distributed Components", Ann. Telecommun, No. 64, 2009, p. 5–24.
- BOUZIANE, L.; PÉREZ, CH., and PRIOL, T. "A Software Component Model with Spatial and Temporal Compositions for Grid infrastructures," Lecture Notes in Computer Science. Vol. 5168, agosto de 2008, p. 698-708.
- BRUNETON, E.; COUPAYE, T.; LECLERCQ, QUÉMA, M. V. and J. B., STEFANI. "The Fractal Component Model and its Support in Java," Softw. Pract. Exper. No. 36, 2006, p. 1257–1284.
- COLE, M. Algorithmic Skeletons: Structured Management of Parallel Computation. MIT Press, 1989.
- DÍAZ, M.; RUBIO, B.; SOLER, E. and TROYA, J.M. "SBASCO: Skeleton-Based Scientific Components", in 12th Euromicro Conference on Parallel, Distributed and Network based Processing, A Coruña, Spain, 2004, p. 318 – 324.
- DÍAZ, M.; ROMERO, S.; RUBIO, B.; SOLER, E. and TROYA, J. M. "Adding Aspect-Oriented Concepts to the High-Performance ComponentModel SBASCO," in 17th International Conference on Parallel, Distributed and Network-Based Processing (PDP 2009), Weimar, Germany, 2009, p. 21-27.
- ELRAD, T. FILMAN, R.; and BADER, A. "Aspect-Oriented Programming: Introduction", Communications of the ACM, Vol. 44(10), 2001, p. 29 – 32.
- FLYNN, M.J. "Very High-Speed Computing Sysem", *Proceeding of the IEEE*. Vol. 54, No. 12, 1966. p. 1901.
- FOSTER, I. and KESSELMAN, C. "Globus: A Metacomputing Infrastructure Toolkit," *International Journal*

of Supercomputer Applications. Vol. 2, 1997. p. 115–128.

- FOSTER, I. and KESSELMAN, C. *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kauffman, 2004.
- HARBULOT, B. and GURD, J. "Using AspectJ to Separate Concerns in Parallel Scientific Java Code, en Proc. of," in the Third International Conference on Aspect-Oriented Software Development, Lancaster, UK, 2004, p. 122 – 131.
- HARBULOT, V. and GURD, J. "A Join Point for Loops in AspectJ, en Proc. of the," in 5th International Conference on Aspect-Oriented Software Development, Bonn, Germany, 2006, p. 63 – 74.
- HEINEMAN, G. and COUNCIL, W. Component-Based Software Engineering. Addison-Wesley, 2001.
- KICZALES, G. *et al*. "An Overviewof AspectJ," in Europe Conference on Object-Oriented Programming, Budapest, Hungary, 2001, p. 327 – 353.
- OMG, "Common Object Request Broker Architecture (CORBA/IIOP)," Technical Report 2003.
- PELAGATTI, S. Structured Development of Parallel Programs. Taylor and Francis, 1998.
- PÉREZ, C.; PRIOL, T. and RIBES, A. "PaCO++: A Parallel Object Model for High Performance Distributed Systems," in 37th Hawaii International Conference on System Sciences, Hawaii, USA, 2004, p. 274a.
- Seti@home. [en línea]. Disponible en: [http://setiathome.ssl.berkeley.edu/].
- SOBRAL, J. L. "Incrementally Developing Parallel Applications with AspectJ", in Parallel and Distributed Processing Symposium, 2006. 20th International, 2006, p. 10.
- SPINCZYK, O.; GAL, A. and SCHRÖDE, W. "AspectC++: An Aspect-Oriented Extension to C++", in 40th International Conference on Tools Pacific, Sydney, 2002, p. 53 – 60.
- SZYPERSKY, C. Component Software. Beyond Object-Oriented Programming. Addison-Wesley, 1998.
- VAQUERO, L. M.; MERINO, L. R. and CÁCERES, J. M. "A Break in the Clouds: Towards a Cloud Definition", ACM SIGCOMM Computer Communication Review. Vol. 39, No. 1, enero de 2009.
- VANNESCHI, M. "The Programming Model of ASSIST, an Environment for Paralleland and Distributed Portable Applications". Parallel Computing. Vol. 28, 2002, p. 1709 – 1732.