UNIVERSIDAD POLITÉCNICA DE VALENCIA

Tesis de Máster en Computación Paralela y Distribuida

Plug-in Globus para WINGS

Autora: Elizabeth Martí Garcia
Dirigido por: Vicente Hernández García
Tutor: Miguel Caballer Fernández

Grupo de Redes y Computación de Altas Prestaciones
Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
2009

Índice General

1

Motivación y Objetivos

1.1 Motivación

Las aplicaciones científicas y tecnológicas son un punto clave en el campo de la ingeniería. Cada día es más común utilizar la ingeniería como motor de resolución de problemas de la vida cotidiana. Estamos hablando de campos tan dispares como pueden ser la medicina, la meteorología o la arquitectura, entre otros. Se trata de programas informáticos que resuelven situaciones muy complejas, lo cual implica una carga computacional alta, cada vez más en ascenso, y un tiempo de ejecución elevado.

Otro tema a tener en cuenta además de la velocidad de ejecución, es el volumen de datos que utilizan las aplicaciones; grandes cantidades de datos que actúan como entradas y salidas del proceso y que el usuario tiene que manejar.

Al coste computacional que supone la propia ejecución, hay que sumarle muchas veces el coste añadido de realizar visualizaciones complejas de los resultados, hacer simulaciones lo más realistas posibles, generar datos complejos, etc.

Estos factores hacen que sea dificíl conseguir unos tiempos de ejecución razonables utilizando los computadores actuales. Por lo tanto la necesidad de este tipo de aplicaciones es conseguir máquinas más rápidas, con más memoria y más espacio en disco que sean capaces de soportar las ejecuciones.

Debido a todo esto empieza a aplicarse en el ámbito científico la tecnología Grid para obtener resultados óptimos al ejecutar aplicaciones pesadas. Actualmente el término Grid va unido al de Workflow [1], ya que la unión de estas dos tecnologías ofrece grandes beneficios.

Dicha unión da lugar a lo que se conoce como Grid Workflow, que no es

más que un sistema workflow estándar basado en Grid. Una definición más exacta y estandarizada de lo que sería un Grid Workflow, es la que da *Rajkumar Buyya* [2]: "Un Grid Workflow puede definirse como la composición de servicios de aplicaciones Grid que se ejecutan en recursos heterogéneos y distribuidos en un orden bien definido, para cumplir unos objetivos específicos".

Los sistemas Workflow son grandes aliados en el mundo de la ingeniería ya que agilizan el proceso de ejecución de aplicaciones pesadas, proporcionando un flujo de trabajo controlado y coordinado por el sistema. Se consigue la automatización de todas las actividades y procesos de trabajo, y no solo la automatización de algunas tareas individuales.

Pero además de agilizar el proceso de ejecución, que ya supone un logro importante, el tiempo de ejecución de las tareas en sí también tiene que ser razonable. Es por ello que tenemos que buscar un sistema que pueda obtener una velocidad de ejecución óptima cuando se someten aplicaciones con una carga computacional alta.

Es aquí donde entra en acción la tecnología Grid como infraestrauctura capaz de trabajar con sistemas con altas necesidades de escalabilidad. Con esta tecnología tenemos la sensación de estar trabajando con un gran computador, cuyas prestaciones en cuanto a potencia, memoria y espacio en disco se adaptan a las necesidades de nuestras aplicaciones.

Algunos de los principales motivos para utilizar la tecnología Grid unida al concepto Workflow son los siguientes:

- Pueden construirse aplicaciones a través de la planificación de recursos distribuidos.
- La utilización de recursos localizados en dominios particulares consigue un aumento del rendimiento y una reducción del coste de ejecución.
- Es posible obtener capacidades/habilidades de procesamiento específicas en las ejecuciones, gracias al uso de múltiples dominios administrativos.
- Se promueven las colaboraciones inter-organizacionales a través de la integración de varios equipos que gestionan diferentes partes del Workflow.
- Extiende los conceptos de Workflow a la gestión de procesos industriales, cientifícos y de ingeniería.

- Proporciona facilidad de uso a los ingenieros y expertos del ámbito de la empresa que no tienen conocimientos de programación en Grid ni de las herramientas necesarias para ello.

- Estos sistemas permiten trabajar en equipo desde diferentes lugares físicos.

Una vez analizadas las necesidades actuales en el sector que nos ocupa, nos planteamos la creación de una aplicación que aprovechara las ventajas de los dos conceptos introducidos. Se consigue así que todo el proceso de ejecución sea transparente para el usuario, gracias a las ventajas de los workflows, y por otro lado se utiliza toda la funcionalidad que ofrece la tecnología Grid.

Para conseguir este objetivo hemos creado un plug-in Globus para añadirlo a WINGS [3]. WINGS se define como un motor workflow orientado a capacidades multi-grid y a fácil extensibilidad. Debido a la arquitectura modular que presenta WINGS, el plugin creado se añade al sistema como un componente más sin tener que hacer modificaciones adicionales. Tanto las características como la arquitectura de WINGS se estudiaran más detalladamente en el capítulo cuatro.

1.2 Objetivos

Los objetivos que nos hemos marcado en la realización de esta tesis de máster, se resumen en los siguientes puntos:

- Analizar las tecnologías existentes que puedan ser utilizadas en el desarrollo del trabajo.
- Estudiar las necesidades del sector al que nos dirigimos y del usuario potencial de nuestra aplicación.
- Utilización de una herramienta propia que automatice el proceso de ejecución de aplicaciones pesadas.
- Que la herramienta diseñada pueda ser ejecutada desde diferentes máquinas sin importar la ubicación geográfica de las mismas.
- Hacer el proceso de ejecución lo más transparente posible al usario. Sólo se requiere su intervención para definir los parámetros de ejecución y cargar el fichero xml que los contiene, quedando luego a la espera de los resultados.

1.3 Estructura del Documento

El documento está dividido en siete capítulos, a través de los cuales hemos tratado de plasmar los aspectos más relevantes del trabajo desarrollado.

En el primer capítulo se expone la motivación que nos ha llevado a desarrollar esta tesis de máster y los objetivos que se pretenden conseguir con él, además de una pequeña introducción al caso que nos ocupa.

En el segundo capítulo nos adentramos en el concepto de Grid, y para ello empezamos con un pequeño resumen de los antecedentes de las tecnologías Grid. Dicho resumen nos llevará a la introducción del tercer capítulo donde nos centramos en la definición de Grid. Abarcaremos los conceptos más importantes que hay que conocer a la hora de utilizar esta tecnología (organizaciones virtuales, servicios web, middleware Grid, GT4, ...).

El capítulo cuarto está dedicado a WINGS. Empezaremos con una definición genérica de WINGS y de su arquitectura, para acabar con un análisis de su comportamiento y utilidad cuando se usa conjuntamente con Grid.

En en el capítulo quinto se aborda la implementación del plug-in creado, y todo lo que ello conlleva. Revisaremos los detalles de implementación, el middleware utilizado y la funcionalidad conseguida con el plugin implementado.

En el capítulo sexto se presenta un caso práctico de utilización del plug-in implementado. Se analiza la utilidad de nuestra aplicación en un caso de estudio concreto, estudiando los tiempos de ejecución obtenidos con el fin de evaluar el comportamiento de la herramienta desarrollada.

Y todo ello nos lleva a exponer, en el capítulo séptimo, los trabajos futuros que nos planteamos llevar a cabo, tanto para mejorar la aplicación desarrollada, como para incluir en la misma nuevos conceptos que amplíen su funcionalidad y campos de uso.

1.4 Introducción

En este apartado vamos a realizar una pequeña introducción a la aplicación creada, con el fin de aclarar todos los conceptos a los que haremos referencia a lo largo del documento.

El enfoque de esta tesis parte del concepto de *Workflow* [4], o *Flujo de trabajo*, concepto muy conocido y extendido en el ámbito de los negocios donde, las exigencias del mercado y la presión de la competencia obligan a las organizaciones a ser más eficaces y eficientes en todos los servicios que ofrecen al cliente: producción, servicios internos y control. Según la coalición *Workflow Management Coalition (WFMC)* [5] el término *Workflow*, podría definirse como:
"La automatización de un proceso de empresa, total o parcial, en el cual documentos, información o tareas son pasadas de un participante a otro a los efectos de su procesamiento, de acuerdo a un conjunto de reglas establecidas".

Los sistemas Workflow ayudan a manejar los procesos de negocios asegurando que las tareas con mayor prioridad serán llevadas a cabo:

- tan pronto como sea posible,
- por la persona correcta, y
- en el orden correcto.

Para llevar a cabo las funciones del Workflow, se definen los *Sistemas de Gestión del Workflow (WFMS)*[1], o Motores Workflow. Su misión consiste en proporcionar el entorno para la activación, gestión y ejecución de los procesos del Workflow a través de software.
Constituyen una herramienta capaz tanto de modelar formalmente los procesos (incluidos agentes, datos y aplicaciones) que son llevados a cabo en una organización, como de ejecutar estos procesos.
Estos sistemas integran y coordinan aplicaciones heterogéneas distribuidas que colaboran entre sí para llevar a cabo procesos.

Las operaciones principales que realizan los WFMS son las siguientes:

- Modelado y representación de los procesos de flujo de trabajo y las actividades que los contituyen.
- Selección de los procesos para su activación, en respuesta a una petición del usuario o a sucesos clave.

[1] Workflow Management Systems

- Programación de las actividades para los agentes y tareas resultantes.
- Monitorización y adaptación de los procesos de ejecución.

Así mismo, los WFMS se caracterizan por tener tres puntos de funcionalidad:

- *Funciones en tiempo de construcción (Build-time functions)*, encargadas de la definición y modelado de un proceso y de las actividades que lo componen.
- *Funciones de control en tiempo de ejecución (Run-time control functions)*, cuya misión es controlar el proceso en el entorno de ejecución, llevando a cabo todas las tareas o actividades definidas como parte del mismo.
- *Funciones de interacción en tiempo de ejecución (Run-time interaction)*, que interactúan con los usuarios o aplicaciones externas para que los participantes del proceso puedan llevar a cabo sus tareas.

En 1995 la WFMC (Workflow Management Coalition) propone un modelo de referencia para los WFMS a través del documento *The Workflow Reference Model* [6]. Pretende de este modelo establecer una arquitectura base sobre la cual desarrollar aplicaciones workflow.
Este modelo de referencia toma una amplia visión de la administración de Workflows, que pretende adaptar la gran variedad de técnicas de implementación y entornos operacionales que caracterizan a esta tecnología. A pesar de dicha variedad, todos los WFMS presentan características comunes que definen una base para desarrollar la capacidad de integración e interoperabilidad entre los diferentes productos.
En definitiva, el objetivo de la WMFC es definir estándares y guías globales para el desarrollo de WFMS; brinda un marco general para la construcción de los mismos sin describir en detalle ningún WFMS en particular.

En la imagen se muestran los componentes e interfaces que componen un sistema de gestión de workflows.

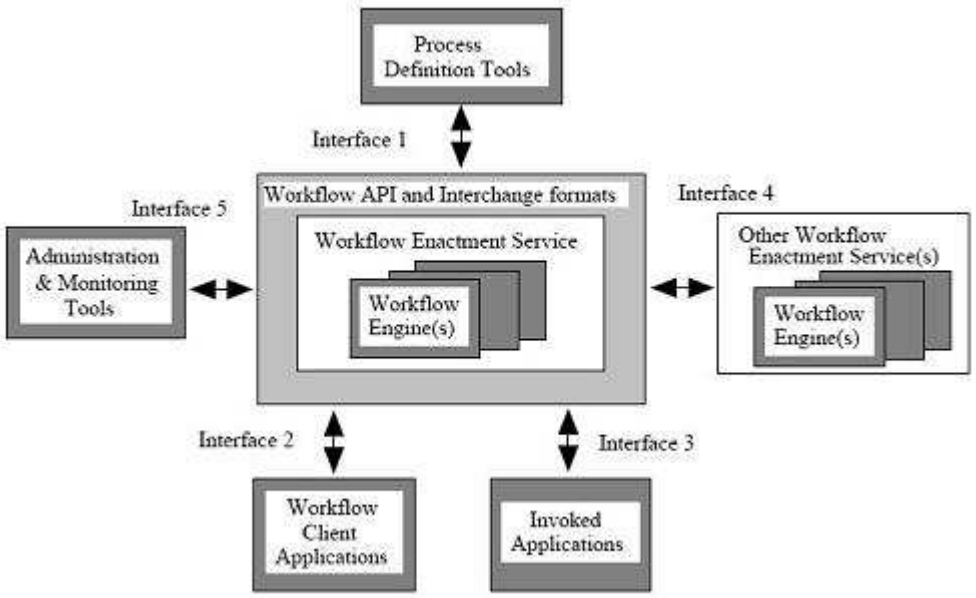

Figura 1.1: Modelo de Referencia de un WFMS. Componentes e interfaces.

El módulo central, como se puede ver en la imagen, es el *Workflow Enactment Service o servicio de ejecución del Workflow*, el cual se encarga de crear, gestionar y ejecutar cada una de las instancias del modelo de flujo de trabajo. Dentro de este componente se encuentra el motor del WFMS, *(Workflow Engine)*, que proporciona la ejecución de cada instancia del Workflow. En caso de estar en un entorno de ejecución de flujo de trabajo distribuido, pueden existir diferentes motores de flujo de trabajo que controlen distintas partes de la ejecución del proceso. La comunicación de este componente con el resto se realiza a través de lo que la WFMC denomina *WAPI* (Workflow APIs) [7], es decir, la interfaz para la programación de aplicaciones de flujo de trabajo.

Otro de los componentes del sistema es el *Process Definition Tools*, o herramientas de definición de proceso, a través de las cuales se puede modelar, describir y documentar un determinado flujo de trabajo o proceso de negocio. Dichas herramientas pueden ser informales como, por ejemplo, lenguaje natural, lápiz y papel, o se puede tratar de herramientas más formales y sofisticadas (interfaces gráficas con un modelo bien definido). Se debe especificar la lógica del proceso, las actividades o tareas que lo componen, los participantes humanos, las aplicaciones invocadas, los datos utilizados, etc.

Estos dos elementos descritos se comunican a través de la *interfaz 1*, que se encarga del intercambio de información entre el componente que permite

la definición del proceso y el propio servicio de ejecución del flujo de trabajo. Esta interfaz hace necesaria la definición de un metamodelo básico, en el que se identifique el conjunto mínimo de entidades para la definición de un proceso, permitiendo el intercambio de información entre ambos componentes.

Encontramos también en el modelo de referencia el módulo *Workflow Client Applications*, o aplicaciones del cliente Workflow, que representan las entidades de software utilizadas por el usuario final en aquellas actividades que requieren participación humana para su realización. Si este componente se separa de lo que es el propio componente de ejecución, es necesaria la *interfaz 2* para definir y manejar claramente el concepto de lista de trabajos (worklist), como una cola de trabajo asignado a un usuario o a un grupo de usuarios por el propio motor de ejecución del flujo de trabajo.

El componente denonimado *Invoked Applications*, aplicaciones invocadas, representa software o aplicaciones ya existentes, que el sistema de gestión del Workflow puede utilizar para la realización de ciertas actividades, teniendo en cuenta que, en principio, dichas aplicaciones se pueden encontrar en cualquier plataforma o lugar de la red. Para permitir la comunicación entre las aplicaciones y el servicio de ejecución del flujo de trabajo se utiliza la *interfaz 3*. La comunicación no se realiza sólo a nivel de invocación sino también de transformación de datos en formatos entendibles por ambos componentes.

La interoperabilidad entre WFMS está representada por el componente *Other Workflow Enacment Services*, siendo la *interfaz 4* la que permite las comunicaciones con el servicio de ejecución. En este caso, la WFMC ha desarrollado un conjunto de escenarios de interoperabilidad, que van desde la conexión a nivel de actividades simples hasta todo un completo intercambio de definición de procesos y datos[8].

El componente *Administration & Monitoring Tools*, o herramientas de administración y monitorización, permite que distintos servicios de ejecución compartan las mismas funciones de administración y monitorización del sistema, como pueden ser el control de los recursos, la gestión de usuarios y la supervisión del estado de todo el proceso.

Llegados a este punto es importante remarcar que el concepto de flujo de trabajo no es aplicable únicamente a los procesos de negocio, a pesar de ser los procesos más comentados cuando se habla de Worflows. También se puede adaptar a otros tipos de procesos, así como sus técnicas de gestión que son aplicables a una gran variedad de problemas de coordinación y control.

Una definición más general, no centrada en el mundo de los negocios, es la de Rusinkiewicz y Sheth (1994)[9]:
"Workflow es un conjunto de actividades que abarca la ejecución coordinada de múltiples tareas desarrolladas por diferentes entidades procesadoras para llegar a un objetivo común."
Una tarea o proceso representa la definición de un trabajo a realizar y la entidad procesadora es la encargada de realizar dicho trabajo. La naturaleza de esta entidad no se indica en la definición, por tanto puede ser una persona, un ordenador, una máquina o cualquier otro recurso capaz de llevar a cabo el trabajo. Es esta característica la que abre la posibilidad de usar la tecnología Workflow para modelar cualquier tipo de proceso que implique la colaboración entre personas o entre personas y máquinas, y no solamente los procesos pertenecientes al ámbito de los negocios.
Tomando como base estas características propias del Workflow, surge la idea de unir dos tecnologías: Workflow y Grid, dando lugar al concepto de Grid Workflow.
La definición estándar dada por *Rajkumar Buyya*[2], la hemos presentado al inicio de esta memoria, pero otra definición igual de válida es la ofrecida por *Geoffrey Fox*[10] según el cual un Grid Workflow es:
"La automatización de procesos, que implica la preparación de un conjunto de servicios Grid, agentes y actores que deben de estar combinados para resolver un problema o definir un nuevo servicio."

En la arquitectura Grid, un sistema de Workflow es un tipo de middleware Grid a nivel de aplicación, capaz de soportar el modelado, rediseño y ejecución de procesos pesados/de gran escala en una amplia variedad de complejas aplicaciones científicas y de negocios como pueden ser, por ejemplo, modelado del clima, astrofísica, física de alta energía, biología estructural y química, cirugía médica, recuperación de desastres, banca internacional, seguros, modelado y control de stocks de mercado internacional.

Un Grid Workflow puede representarse a través de un grafo[2], donde los nodos representan las actividades y las uniones simbolizan las dependencias entre ellas, llamadas flujos (flows). La verificación se basa normalmente en una extensión de un tipo de método formal. Tanto la verificación como la validación de un Grid Workflow deben permitirnos identificar cualquier violación de la corrección en la especificación del Workflow y consecuentemente su eliminación a tiempo.

[2]grid workflow graph

Las principales diferencias entre el uso clásico del workflow y su aplicación en el ámbito de las tecnologías Grid son principalmente:

- **Mantenimiento de estado:** Esta característica básica dentro del entorno de los servicios Grid, y que lo diferencia de los clásicos servicios Web, debe ser contemplada en los workflows Grid.
- **Fiabilidad:** Los recursos Grid pueden fallar durante la ejecución de un workflow. Esto debe ser tenido en cuenta disponiendo de una tolerancia a fallos avanzada (workflow checkpointing, control de procesos,...).
- **Rendimiento:** Uno de los objetivos de la computación en Grid es la de obtener altas prestaciones. Por tanto los workflow deben permitir la utilización del paralelismo, selección de recursos, balanceo de carga, ...
- **Flujos de datos muy grandes:** En los entornos Grid, tanto la cantidad de datos a tratar como el tamaño de los flujos pueden llegar a ser muy grandes.

En la siguiente figura podemos ver un modelo de referencia correspondiente a un sistema de gestión de worflow para un entorno Grid (Grid WFMS)[11].

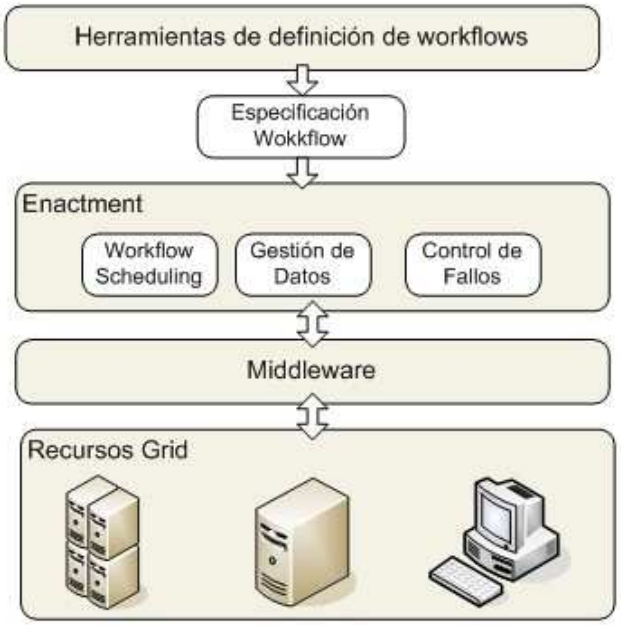

Figura 1.2: Modelo de Referencia de un Grid WFMS

El modelo presentado mantiene la estructura original de un WFMS, pero se añaden ciertos componentes necesarios para trabajar con procesos Grid. Al igual que ocurría con la definición de los Workflows genéricos, el proceso de trabajo de un Grid Workflow se divide en tres fases: tiempo de construcción, instanciación en tiempo de ejecución y ejecución. La primera fase se refiere a la definición y modelado de las tareas del workflow y sus dependencias. Mientras que las funciones en tiempo de ejecución gestionan las ejecuciones del workflow y las interacciones con recursos Grid para procesar las aplicaciones Workflow.

El elemento principal del modelo es el *Grid Workflow Enacment Service* o Servicio de ejecución del Grid Workflow. El objetivo de este componente es diseñar un motor de ejecución, que pueda recibir y procesar peticiones workflow procedentes de los clientes, que posteriormente serán ejecutadas en nodos Grid controlados por un planificador como puede ser GRAM, incluido en Globus Toolkit. Este elemento es esencialmente un demonio que acepta conexiones de entrada de los clientes. Cada nueva conexión genera un nuevo thread o hilo de ejecución - el Job Manager, que gestionará todas las comunicaciones con el cliente.

Desde el punto de vista de la arquitectura, el núcleo de ejecución puede definirse como un sistema de dos capas. La capa superior, *capa de control*, es responsable de obtener la definición del proceso, interpretarla, navegar por el correspondiente grafo y elegir las actividades a lanzar. El nivel inferior, o *capa de interacción*, se encarga de gestionar la conexión de los recursos y de requerir el cumplimiento de las tareas.

Los recursos Grid están conectados al Workflow Enacment Service (WFES) a través del middleware Grid, un componente distribuido mucho más complejo encargado de interactuar con los recursos, permitir el acceso a los mismos, gestionar la seguridad y el cumplimiento de otras tareas. Como ejemplo concreto podríamos considerar Globus Toolkit.

Las herramientas de definición y modelado de aplicaciones que se utilizan en la fase de construcción, tienen como misión modelar el sistema en forma de grafo, de manera que este grafo pueda traducirse posteriormente a lenguaje XML generando un fichero que será el argumento de entrada de la aplicación.

2

Antecedentes de la Tecnología Grid

2.1 Antecedentes

Es importante conocer cómo nació la tecnología Grid (Grid Computing) y de por qué se creó desde la perspectiva de la evolución de la computación avanzada. Como todos los avances en la humanidad, surge como respuesta a un problema, para el cuál las soluciones existentes no eran suficientes o no eran viables considerando los recursos disponibles para resolverlo.

El problema en cuestión era la gran demanda, tanto de computación como de espacio y gestión de almacenamiento, requeridos por un gran número de aplicaciones que gestionaban grandes cantidades de datos y que tenían que hacerlo de forma eficiente.

Se necesitaba una solución para este problema que preocupaba a los investigadores, y ésta debería ser eficiente en el uso de sus limitados recursos.

Las diferentes iniciativas adoptadas para resolver el problema se pueden agrupar en:

- Cluster Computing,
- Intranet Computing,
- Internet Computing,
- Peer-to-peer Computing,

- Collaborative & Computing Portals,
- Infraestructuras Grid,

2.1.1 Cluster Computing

Debido al bajo precio y alto rendimiento de los actuales sistemas PC, estaciones de trabajo y redes de interconexión, la construcción de un cluster se presenta frecuentemente como la mejor alternativa a la adquisición de un equipo multiprocesador. Sin embargo, es importante aclarar que los sistemas basados en la replicación de equipos también presentan inconvenientes como su mayor dificultad de mantenimiento y la necesidad de un paradigma de programación basado en memoria distribuida.
Esto supone que para desarrollar programas únicamente se pueden utilizar modelos de programación basados en granjas de procesadores, modelo de alta productividad, o en paso de mensajes como MPI (Message Passing Interface) o PVM (Parallel Virtual Machine), modelo de alto rendimiento, y no se pueden utilizar de forma eficiente modelos de memoria compartida, como multithreading o directivas tipo OpenMP.
Por otro lado, el rendimiento de estos sistemas es menor que el ofrecido por los sistemas multiprocesador para aplicaciones de grano fino. Los clusters habitualmente están gestionados por herramientas software que se encargan de ejecutar las aplicaciones de los usuarios sobre las distintas máquinas, en función de diferentes criterios de planificación fijados por el administrador.
Estas herramientas de gestión de recursos pueden ser sistemas integrados de planificación como MOSIX o gestores de colas batch como PBS (versión libre), PBS-Pro (versión comercial), LSF o SGE, y realizan una planificación dedicada asumiendo, en principio, la disponibilidad constante de los recursos que gestionan.

2.1.2 Intranet Computing

Algunas de las herramientas descritas en la sección anterior permiten también gestionar un sistema heterogéneo de equipos de forma oportunista, de modo que se pueda utilizar toda la potencia computacional distribuida que se encuentra habitualmente desaprovechada. Estas herramientas realizan una planificación basada en la suposición de que los recursos no están dedicados, y por tanto no tienen porqué estar disponibles durante toda la ejecución de los trabajos. Se usan normalmente para coordinar los recursos de procesamiento en la Intranet de la empresa o del centro de investigación sin salir de su dominio de administración.

Por tanto, el objetivo de Intranet Computing es unir la potencia computacional desaprovechada de los recursos hardware distribuidos dentro de un único dominio de administración, para alcanzar rendimientos semejantes a los proporcionados por los sistemas de alto rendimiento comerciales con un coste diferencial prácticamente nulo.
Tanto estas herramientas, como las correspondientes a Cluster Computing, suelen instalarse junto con otros servicios que permiten compartir ficheros, como NFS, y usuarios, como NIS. Además, si la política de seguridad así lo exige se pueden instalar mecanismos de encriptación y autenticación basados por ejemplo en NIS+ y secure NFS, que usan Kerberos o Diffie-Hellman.
Intranet Computing es la alternativa más eficiente para aprovechar la capacidad de procesamiento disponible en los equipos ociosos de la red, por medio de la ejecución de trabajos independientes, normalmente ejecuciones paramétricas.
Ejemplos de herramientas de gestión de trabajos paramétricos son Fura, AppLES, Nimrod, Condor y productos de empresas como Entropía, Avaki, United Devices o Parabon. Existen también herramientas que permiten tanto planificación dedicada como oportunista, como por ejemplo SGE o LSF.

2.1.3 Internet Computing

Extender la colaboración entre sistemas distribuidos al ámbito de Internet presenta serios problemas adicionales. En Internet la seguridad es un factor clave y además el ancho de banda de la red puede hacer muy ineficiente el uso de la computación distribuida por medio de los paradigmas anteriores. Cluster e Intranet Computing suelen ser efectivos y eficientes a pequeña escala. A mayor escala han surgido nuevas aproximaciones que aprovechan los tiempos ociosos de los sistemas conectados a Internet para resolver problemas científicos de interés que se pueden expresar de forma paramétrica, en un modo Seti@home [12].
SETI@home es un experimento científico que utiliza ordenadores conectados a Internet para la búsqueda de inteligencia extraterrestre. Este proyecto permite a cualquier persona con un ordenador y conexión a Internet formar parte de la búsqueda. El usuario solo tiene que descargar un programa cliente que funciona a modo de salvapantallas, utilizando los recursos inutilizados de nuestro ordenador para analizar los datos que previamente hemos descargado y que provienen directamente de Arecibo, el gigantesco telescopio que se utiliza en este experimento para captar ondas procedentes del espacio.

Las alternativas actuales permiten el uso de los procesadores de sistemas

remotos donados por organismos o particulares para ejecuciones paramétricas. Este tipo de aplicaciones es muy común en determinados ámbitos de aplicación, como la bioinformática. Actualmente no hay ninguna empresa que pague a aquellos que "ceden" su procesador, esto es, todavía no se ha creado un mercado de recursos de cálculo. De hecho, según reflejan los últimos estudios de mercado, todas las empresas que comercializan este tipo de aplicaciones, como Entropía, Parabon o United Devices, hacen negocio principalmente en la Intranet de las empresas debido a problemas de seguridad por ambas partes:

- Las empresas que quieren comprar procesador son reacias a que sus códigos y datos propietarios viajen por Internet y se instalen en máquinas "desconocidas". Los propietarios de las máquinas no se fían de que estos programas que se ejecutan no están "contaminados" con algún virus.

- Además, cuando el "donador" de recursos es una empresa nos encontramos con problemas derivados de la disparidad en políticas de seguridad, procedimientos de explotación de recursos,etc. Por otro lado, no debemos olvidar que el bajo ancho de banda proporcionado por Internet actualmente limita el tipo de trabajo susceptible de usar esta tecnología. Deben ser trabajos paramétricos, esto es que se pueden dividir fácilmente en trabajos totalmente independientes, que requieran bastante uso de procesador, para evitar que se tarde más en enviarlos que en ejecutarlos, que no requieran prácticamente memoria, para que el usuario de los sistemas no se vea afectado por su ejecución, y que no tengan casi entrada/salida, para evitar la costosísima transmisión de ficheros de gran tamaño.
  Por otro lado, debido a la poca fiabilidad y robustez de los sistemas donde se invocan las ejecuciones, el problema debe contar con un gran número de conjuntos de datos de entrada.

Otro modelo de computación, también en investigación en Internet, es *On-Demand Computing*, computación bajo demanda. Las aplicaciones que usan este paradigma se programan para que usen recursos específicos de forma remota. Un ejemplo es Netsolve que permite delegar la ejecución de funciones de librerías matemáticas típicas que requieren gran potencia computacional a servidores remotos. Otra herramienta semejante es Ninf.

2.1.4 Peer-to-peer Computing

Las soluciones de computación indicadas anteriormente se basan en el modelo cliente/servidor. Existe un cliente que realiza peticiones y un servidor que

las atiende. Este modelo centralizado suele presentar cuellos de botella para sistemas masivos.
La solución es un modelo peer-to-peer donde los nodos de la red actúan simultanéamente como clientes y servidores. Las redes peer-to-peer aprovechan, administran y optimizan el uso del ancho de banda de los demás usuarios de la red por medio de la conectividad entre los mismos, obteniendo más rendimiento en las conexiones y transferencias que con algunos métodos centralizados convencionales, donde una cantidad relativamente pequeña de servidores provee el total del ancho de banda y recursos compartidos para un servicio o aplicación.

Este modelo es muy común en la compartición de ficheros en entornos distribuidos, por ejemplo Napster o Gnutella. También es comúnmente usado en telefonía VoIP para hacer más eficiente la transmisión de datos en tiempo real.

2.1.5 Collaborative & Computing Portals

Los portales colaborativos (Collaborative Portals) permiten la interacción entre grupos de investigación, empresas, proveedores y entidades colaboradoras de forma fácil y eficiente. A mayor nivel de colaboración aumenta la productividad y se reducen los costes.
Por otro lado, los portales de computación (Computing Portal) son una nueva tecnología cuyo objetivo es ahorrar tiempo y dinero proporcionando un único punto de acceso seguro, basado en navegador web, para invocar servicios como la ejecución de aplicaciones en plataformas de alto rendimiento.

3

Grid Computing

Tanto Cluster como Intranet Computing son paradigmas que permiten obtener gran capacidad de procesamiento dentro de un único dominio de administración. Alguna de estas herramientas, como SGE Enterprise o Condor Flocking, también permite la interconexión de varios departamentos o dominios de administración, siempre que estos estén también gestionados internamente con la misma herramienta. Sin embargo, ninguna de las herramientas anteriores permite compartir recursos distribuidos en diferentes dominios de administración, cada uno con sus propias políticas de seguridad y gestión de recursos.

Las tecnologías descritas en los puntos anteriores son casos especiales de un nuevo paradigma de computación distribuida que actualmente está revolucionando no sólo la computación de altas prestaciones sino Internet en general. Esta nueva tendencia, referida de forma general por el término Grid, supone un cambio radical en la colaboración de sistemas conectados a Internet, y en particular en la computación de altas prestaciones, debido a su enorme potencial respecto al intercambio y gestión de recursos.
Gracias a la tecnología Grid se pueden evitar muchos de los inconvenientes indicados anteriormente, como la seguridad, y desarrollar infraestructuras que no sólo compartan sus procesadores sino además su capacidad de almacenamiento y aplicaciones. De hecho, muchas de las empresas indicadas anteriormente en Intranet e Internet Computing están portando sus tecnologías propias para usar como base la tecnología Grid, en particular el Globus Toolkit.
Es importante resaltar que la tecnología Grid no pretende sustituir las tecnologías anteriores, ya que su ámbito de aplicación es diferente. El objetivo

de la tecnología Grid es unir de forma desacoplada los recursos de diferentes dominios de administración, respetando sus políticas de seguridad y herramientas de gestión internas.

Las alternativas presentadas como antecedentes de la tecnología Grid, a pesar de representar avances significativos no suponen cambios revolucionarios, ya que consisten en manipular la tecnología existente, principalmente el modelo cliente/servidor y la arquitectura de protocolos TCP/IP para unificar recursos distribuidos. Por otro lado, no debemos olvidar que también el almacenamiento y la entrada/salida son problemas de gran importancia en computación de altas prestaciones. Existen variantes distribuidas para resolver las necesidades de almacenamiento (por ejemplo NFS, Webfs, PVFS...) y opciones para que la computación y el intercambio de información por la red se realice de forma segura, garantizando autenticación tanto del cliente como del servidor, confidencialidad e integridad (por ejemplo SSL, NIS+, Kerberos...).

La tecnología Grid surge como un tipo de sistema distribuido que permite compartir, seleccionar y agregar recursos distribuidos, independientemente de su localización, a través de múltiples dominios administrativos basados en su disponibilidad, capacidad, rendimiento, coste y requerimientos de calidad de servicios. Además, para cada dominio de administración se respeta su política interna de seguridad y su software de gestión de recursos en la Intranet.

Los fundamentos de esta tecnología son básicamente tres:

1. **Compartición de recursos a gran escala:** la idea fundamental es poder compartir una serie de recursos entre los posibles usuarios, de manera que sea igual de sencillo poder acceder a una infraestructura local que a una localizada en cualquier parte del mundo. Es más, el uso debe poder llevarse a cabo utilizando muchos recursos distintos distribuidos geográficamente. Entre los recursos que podemos compartir se pueden distinguir:

    - Computadores: proporcionando potencia de cálculo para realizar las computaciones necesarias, es el recurso básico al que queremos acceder.
    - Redes de comunicaciones: que permitirán interconectar el resto de recursos para que la ejecución de las aplicaciones sea posible.

- Instrumentos: generalmente instrumentos de carácter científico, que son necesarios en algunas aplicaciones. Ejemplos de estos instrumentos pueden ser cabinas de visualización, microscopios electrónicos, radio-telescopios, etcétera.
- Datos: que deban ser compartidos por una comunidad para lograr sus objetivos, como datos para simulaciones nucleares, variables meteorológicas para la obtención de las predicciones, o cadenas de adn para ser procesadas. Además estos datos pueden tener requerimientos de privacidad, como pueden ser los expedientes médicos y datos asociados (mamografías, escáneres, etc.) a pacientes que pueden ser accedidos por una comunidad médica para analizarlos.

2. **Organización de recursos distribuidos de varias organizaciones:** Como hemos dicho, estos recursos pueden pertenecer a organizaciones distintas que tengan sus propios administradores locales, aplicando las políticas que son necesarias y adecuadas para cada organización. Por lo tanto el control sobre los mismos es limitado y no centralizado.

3. **Recursos heterogéneos:** los recursos a compartir son heterogéneos, ya que distintas organizaciones pueden disponer de multitud de elementos para compartir que soporten diferentes protocolos, herramientas, etcétera. El Grid debe tener en cuenta este punto para proveer una infraestructura común que pueda interoperar estos recursos.

El nacimiento de la tecnología Grid para computadores traía una serie de ventajas que otros sistemas no ofrecían.

1. **Seguridad**, para pertenecer a un grid se deben seguir una serie de protocolos que garantizan la seguridad del sistema. En contraste con los sistemas peer-to-peer donde la participación es anónima y poco segura. Adicionalmente, debe garantizarse la seguridad de las organizaciones virtuales (VOs) que forman el Grid, proporcionando un intercambio de recursos controlado, seguro y flexible.

2. **Confiabilidad** a diferencia de otros sistemas, donde los procesos deben ser replicados múltiples veces, evaluando y comparando sus resultados para asegurar su fiabilidad, en un grid las organizaciones participantes son confiables, por lo tanto se asume que, en cierto modo, no hay datos corruptos.

3. **Coordinación**, los miembros de un grid aportan siempre más recursos, es decir, es un esfuerzo coordinado donde todos aportan y todos usan los recursos disponibles.

4. **Escalabilidad**, un grid es altamente escalable ya que se pueden agregar cada vez más recursos incrementando las capacidades del mismo. Esto es principalmente porque, por la naturaleza colaborativa y coordinada del grid, los miembros definen los recursos y los tiempos en que estos están disponibles para el grid, así existe un dinamismo en el cual los servicios prestados por una institución pueden estar o no disponibles en un momento determinado y el grid está preparado para ello.

El término *Grid Computing* apareció a mediados de los años 90 en los trabajos de Ian Foster y Carl Kesselman.

La palabra *Grid* en inglés significa malla y hace referencia a la red eléctrica. Para obtener electricidad, simplemente debemos conectarnos a cualquier punto de la malla electrica (power grid) sin preocuparnos por las plantas generadoras que interconectadas brindan este servicio, es decir, no sabemos de dónde proviene la electricidad que usamos, simplemente la aprovechamos. Este fue el concepto que adaptaron Carl Kesselman e Ian Foster en su libro *The Grid: Blueprint for a new computing infrastructure* publicado en 1998.

La idea que presentaron en su publicación era la de crear una red mundial de laboratorios proveedores de poder de cómputo y capacidad de almacenamiento, asi como lo hacen las plantas eléctricas para proveer la electricidad, y que el acceso a ellos fuese fácil, permitiendo a todos aprovechar la capacidad de este gran conjunto heterogéneo de sistemas sin necesidad de preocuparse por la interconexión, ya que no es necesario saber qué nodos de esta red nos proveen el servicio.

Pero la idea de Grid va más allá, no sólo se trata de compartir ciclos de CPU para realizar cálculos complejos sino que se busca la creación de una infraestructura de computación distribuida. Esta ardua tarea involucra labores de definición de la arquitectura general, de interconexión de diferentes redes, de definición de estándares, de desarrollo de procedimientos para la construcción de aplicaciones, etc.

Hemos hablado mucho de lo que ofrece y de las posibilidades del grid computing pero todavía no hemos definido de manera concreta el término. Según Ian Foster y Carl Kesselam: *"Un grid computacional es una infraestructura hardware y software que proporciona un acceso seguro, consistente, ubicuo y barato a capacidades computacionales de alto nivel".*

El concepto de infraestructura se utiliza porque un grid es un conjunto de recursos (ciclos de CPU, datos, sensores, etc.), y todos esos recursos necesitan

una interconexión hardware y un control software para que estén ensamblados en un grid. Esta infraestructura debe proporcionar a los usuarios un servicio seguro a todos los niveles: capacidad de cómputo, de integridad de datos, de seguridad de acceso, etc. El servicio debe ser consistente, basado en estándares, y de esta manera el acceso y las operaciones sobre el grid estarán definidas por dichos estándares, evitando la heterogeneidad. La idea de ubicuidad no es tanto la posibilidad de acceder a cualquier recurso del grid como que el grid llega a cualquier sitio, de esta manera uno se asegura que una vez conectado desde cualquier punto puede extraer del grid toda la potencia que requiera. Además el acceso y uso del grid debe tener un coste económico que le haga atractivo para que su utilización se universalice.

Ian Foster define una lista de control basada en tres puntos para plasmar lo que sería un sistema Grid. Según él, un Grid es un sistema que:

1. coordina recursos que no están sujetos a un control centralizado,
2. utiliza protocolos e interfaces abiertos, de propósito general, y
3. proporciona calidades de servicios no triviales.

3.1 Funcionamiento del Grid

Podemos describir el funcionamiento del Grid del siguiente modo:

- El Grid descansa sobre un software, denominado *middleware*, que asegura la comunicación transparente entre diferentes ordenadores repartidos por todo el mundo.
- El segundo elemento es un motor de búsqueda que no sólo encontrará los datos que el usuario necesite, sino también las herramientas para analizarlos y la potencia de cálculo necesaria para utilizarlas.
- Al final del proceso, el Grid distribuirá las tareas de computación a cualquier lugar de la red en la que haya capacidad disponible y enviará los resultados al usuario.

Este funcionamiento se sostiene en cinco pilares básicos:

1. La posibilidad de compartir recursos.
2. La seguridad − acceso seguro.
3. El uso eficiente de los recursos.
4. Redes de comunicación fiables que acorten las distancias.
5. Estándares abiertos.

3.1.1 Compartición de recursos

Esta es la idea que está detrás de Grid: poder utilizar recursos remotos que nos permitan realizar tareas que no podríamos abordar en nuestra máquina o centro de trabajo.
La idea va más allá del simple intercambio de ficheros; se trata del acceso directo a software, ordenadores y datos remotos, así como acceso y control de otros dispositivos (sensores, telescopios, etc). Pero debemos hacer frente a un hecho: los recursos pertenecen a muchas personas distintas. Por tanto, nos encontramos con dominios administrativos diferentes, en los que se ejecuta software heterogéneo, y sometidos a las más diversas políticas de control de acceso y seguridad.
Este es un punto crucial de la tecnología Grid: no se trata de conseguir algo por nada o de ofrecer nuestros recursos de computación de forma altruista. Más bien se trata de crear una situación entre los propietarios de recursos de computación donde todos los implicados puedan apreciar las ventajas de

compartirlos, en la que haya mecanismos que aseguren la confianza entre los usuarios, y estableciendo las condiciones de uso de sus recursos: cuándo y qué puede hacerse en ellos.

3.1.2 Acceso seguro

Como en cualquier otra aplicación la seguridad es esencial y se centra en los siguientes aspectos:

- *Política de Acceso:* Tanto los que ofrecen sus recursos como los que los utilizan deben definir cuidadosamente qué es lo que van a compartir, a quién se permite el acceso y bajo qué condiciones.
- *Autenticación:* Es necesario un mecanismo para establecer la identidad de un usuario o de un recurso concreto.
- *Autorización:* También hace falta un procedimiento para determinar si una determinada operación es consistente con las relaciones que se han definido previamente de cara a compartir recursos.

La tecnología Grid necesita una forma eficiente de recopilar una serie de información:

- ¿Quién está autorizado a utilizar el GRID?.
- ¿Qué recursos está autorizado a utilizar?.
- ¿Quién da fe de que un usuario es quien dice ser?.
- ¿Cuáles son las políticas de uso de los diferentes recursos?.

Todos esos elementos pueden cambiar de un día para otro, lo que significa que, para que funcione de forma eficiente el Grid debe ser extremadamente flexible, capaz de adaptarse a todos los cambios y además contar con un mecanismo de ”contabilidad” eficiente.

3.1.3 Uso eficiente de recursos

El tercer aspecto fundamental en la tecnología Grid es el uso eficiente de los recursos. Es aquí donde radica el verdadero interés del Grid. No importa la cantidad de recursos de los que uno disponga; siempre habrá usuarios haciendo cola para utilizarlos. Se necesitan mecanismos para repartir el trabajo de forma automática y eficiente entre una gran cantidad de recursos, reduciendo las colas de espera.

La idea se parece mucho a las colas en las cajas de un supermercado. Todo el mundo se dirige a la cola más corta; más vacía. Lo que ocurre es que cuando nos colocamos en una, la persona que va delante ha elegido un producto cuyo código de barras no puede leer el escáner y averiguar el precio del artículo se convierte en una odisea. Así que lo que realmente necesitamos saber es, no sólo cuál es la cola más corta, sino también cuánto tiempo va a tardar cada persona en pasar por la caja.

En el Grid, en principio, tendremos información sobre los diferentes trabajos que se han enviado y, ya que todo se está ejecutando en ordenadores, podemos calcular cuál sería la asignación óptima de recursos. Para ello existe un software que lleva a cabo este trabajo y que, en general, gestiona la actividad del Grid. Este software recibe el nombre de *middleware*, y hablaremos de él más adelante.

3.1.4 Redes de comunicación rápidas y fiables

La existencia de conexiones de alta velocidad es lo que hace posible el Grid a escala mundial. Hace diez años hubiese sido ingenuo tratar de enviar grandes cantidades de datos a través del mundo para que se pudiesen procesar más rápido en otros ordenadores. El tiempo que se tardaba en transferirlos anularía el beneficio de un procesamiento más rápido.

3.1.5 Estándares abiertos

El quinto y último punto es el de los estándares. El objetivo es conseguir que las aplicaciones que se ejecuten en un Grid puedan funcionar en cualquier otro. Debido a que la naturaleza última de Grid es compartir recursos, es comprensible que la existencia de estándares abiertos redunde en beneficio de todos los agentes participantes.
Actualmente, los estándares de Grid los desarrolla el Global Grid Forum, y un estándar, conocido como OGSA (Open Grid Services Architecture), aparece como la referencia clave para los proyectos de desarrollo Grid.

En esencia, los principales proyectos relacionados con Grid se están desarrollando en base a una serie de protocolos y servicios que ofrece el Globus Toolkit (una infraestructura de código abierto desarrollada por la Globus Alliance) proporcionando un conjunto de herramientas para implementar los servicios y capacidades básicas para construir un Grid, tales como la seguridad, la localización y gestión de recursos y las comunicaciones mediante una serie de programas que implementan estos servicios.

Muchos de los protocolos y funciones definidas por el Globus Toolkit son similares a los que existen actualmente para redes y sistemas de almacenamiento, aunque optimizados para Grid. Además, las herramientas se pueden integrar por separado en los programas de software existentes, para ir acomodándolos a los requisitos de Grid y se encuentra a disposición general bajo un acuerdo de licencia *open source*.

Esto permite a todos los interesados utilizarlo libremente y al mismo tiempo añadir mejoras.

3.2 Organizaciones Virtuales (VOs)

Antes de empezar a analizar la arquitectura de la tecnología Grid, vamos a centrarnos en un concepto fundamental, que aparecerá a lo largo del documento y que es importante dejar claro para poder entender la filosofía de Grid.

El problema real que subyace al concepto Grid es compartir recursos de forma coordinada y resolver problemas en organizaciones virtuales de naturaleza dinámica, y formadas por más de una institución. La acción de compartir está referida al acceso directo a computadoras, software, datos, y otros recursos, para la resolución de problemas en forma colaborativa y estrategias de emergencia en el ámbito industrial, científico e ingenieril.
Esta compartición es altamente controlada, por parte de proveedores y consumidores de recursos, definiéndose de forma clara y concisa qué se está compartiendo, quienes están autorizados a compartir, y las condiciones bajo las cuales se concreta la acción de compartir.

Un conjunto de individuos y/o organizaciones definidos a partir de las anteriores reglas, forman lo que se denomina una **Organización Virtual (VO)**. Los distintos tipos de Organizaciones Virtuales pueden variar en sus propósitos, alcance, tamaño, duración, estructura y comunidad. Sin embargo se pueden identificar requerimientos comunes, particularmente la necesidad de que las relaciones de compartir sean altamente flexibles, presentando estas relaciones, sofisticados y precisos niveles de control, como son utilizados los recursos compartidos, sobre control de acceso y aplicación de políticas a nivel local y global. Además se ofrece la posibilidad de compartir varios recursos, desde archivos, datos, software, hasta sensores y redes.

Los recursos se agrupan dinámicamente para resolver problemas concretos, formando organizaciones virtuales, ofreciendo este enfoque un significativo nivel de paralelismo y/o balanceo de carga. Una cuestión importante para entender la computación Grid, es comprender que Grid esta por encima de todos los mecanismos que permiten compartir recursos, y surge como interrogante, qué grupos tendrán el suficiente incentivo de compartir recursos y qué recursos están dispuestos a compartir. En general, los recursos serán compartidos cuando hay un considerable incentivo de hacerlo, porque:

- el recurso es muy costoso,
- el recurso es escaso,
- compartir posibilita interacciones humanas, que de otra manera son

difíciles de alcanzar.

En todos los casos los participantes de las VOs están regidos por políticas que regularizan la forma de compartir recursos:

- El propietario de cada recurso es quien determina la disponibilidad, que está sujeta a restricciones sobre cuando, cómo, y qué puede ser realizado sobre el recurso.

- Los consumidores de recursos pueden especificar restricciones sobre las propiedades de los recursos con los que están preparados para trabajar, por ejemplo la necesidad sobre propiedades de seguridad.

- La implementación de tales restricciones requiere mecanismos para expresar políticas, para establecer la identidad de un consumidor o un recurso (autenticación), y para determinar si una operación es consistente con las relaciones de compartir establecidas (autorización).

- Las relaciones de compartición pueden variar dinámicamente a lo largo del tiempo en términos de los recursos implicados, la naturaleza del acceso permitido, y los participantes a quienes es permitido el acceso. Estas relaciones no necesariamente implican un conjunto de individuos explícitamente nombrados, pueden definirse implícitamente a través de políticas que gobiernan el acceso a los recursos, como por ejemplo, una organización puede permitir el acceso a cualquiera que pueda demostrar ser ”cliente” o ”estudiante”.

- La naturaleza dinámica de las relaciones de compartición requiere mecanismos para descubrir y caracterizar la naturaleza de las relaciones que existen en un momento del tiempo concreto.

- No son simplemente relaciones cliente-servidor, sino compañero a compañero: los proveedores pueden ser consumidores, y las relaciones pueden existir entre cualquier subconjunto de participantes. El mismo recurso puede ser utilizado en diferentes formas, dependiendo de las características impuestas sobre la compartición y el objetivo de compartición. Debido a la escasez de conocimientos previos sobre com serán utilizados los recursos; medidas de rendimiento, expectativas y límites que pueden ser parte de las condiciones impuestas sobre el uso o compartición del recurso.

3.3 Arquitectura Grid

La arquitectura de la tecnología Grid se basa en un modelo de capas formado por cinco niveles, cada uno de los cuales ejecuta una determinada función. Como ocurre en este tipo de modelos, las capas más altas están más cerca del usuario, mientras que las capas inferiores lo están de las redes de comunicación.

Figura 3.1: Arquitectura de la tecnología Grid

Si recorremos las capas desde la base, la primera con la que nos encontramos es la *capa de red* (Network Layer), cuya misión es asegurar la conexión entre los recursos que forman el Grid. En el nivel justo superior está la *capa de recursos* (Resources Layer), constituida por los dispositivos que son parte del Grid: ordenadores, sistemas de almacenamiento, catálogos electrónicos de datos e incluso sensores que se conecten directamente a la red.
En la zona superior está la *capa de middleware* (Middleware Layer), encargada de proporcionar las herramientas que permiten que los distintos elementos (servidores, almacenes de datos, redes, etc.) participen de forma coordinada en un entorno Grid unificado.

En la capa superior de la arquitectura, está la *capa de aplicación* (Application & Serviceware Layer) donde se incluyen todas las aplicaciones de los usuarios, portales y herramientas de desarrollo que soportan esas aplicaciones. Esta es la capa que ve el usuario.
En las arquitecturas más comunes del Grid, esta capa proporciona el llamado

serviceware, que recoge las funciones generales de gestión tales como la contabilidad del uso del Grid que hace cada usuario.

El serviceware está en la capa superior por ser un elemento con el que interactúa el usuario, mientras que el middleware está en una capa "oculta" de la que no debe preocuparse.

Para poder utilizar la arquitectura descrita, las aplicaciones que se desarrollen para ser ejecutadas en un PC concreto tendrán que adaptarse para poder invocar los servicios adecuados y utilizar los protocolos correctos. Igual que las aplicaciones que inicialmente se crearon para funcionar aisladamente se adaptan para poder ser ejecutadas en un navegador Web, el Grid requerirá que los usuarios dediquen cierto esfuerzo a "gridificar" sus aplicaciones.
Sin embargo, una vez adaptadas, miles de usuarios podrán usar las mismas aplicaciones, utilizando las capas de middleware para adaptarse a los posibles cambios en la arquitectura del Grid.

3.4 Middleware Grid

La tecnología grid es posible gracias al ″middleware grid″, el software especial que permite la integración de todos los distintos tipos de recursos que participan en él. Por ello podríamos decir que el middleware es el auténtico cerebro del Grid y sus pasos de actuación son los siguientes:

- Encontrar el lugar conveniente para ejecutar la tarea solicitada por el usuario.
- Optimizar el uso de recursos que pueden estar muy dispersos.
- Organizar el acceso eficiente a los datos.
- Se encarga de la autenticación de los diferentes elementos.
- Se ocupa de las políticas de asignación de recursos.
- Ejecutar las tareas.
- Monitorizar el progreso de los trabajos en ejecución.
- Gestionar la recuperación frente a fallos.
- Avisar cuando se haya terminado la tarea y devuelve los resultados.

El ingrediente fundamental del middleware son los metadatos, que contienen, entre otros aspectos, toda la información sobre el formato de los datos y dónde se almacenan (a veces en varios sitios distintos). El middleware está formado por muchos programas software. Algunos de esos programas actúan como agentes y otros como intermediarios, negociando entre sí, de forma automática, en representación de los usuarios del Grid y de los proveedores de recursos. Los agentes individuales presentan los metadatos referidos a los usuarios, datos y recursos. Los intermediarios se encargan tanto de las negociaciones entre máquinas para la autenticación y autorización de los usuarios como de definir los acuerdos de acceso a los datos y recursos y, en su caso, el pago por los mismos.
Cuando queda establecido el acuerdo, un intermediario planifica las tareas de cómputo y supervisa las transferencias de datos necesarias para acometer cada trabajo concreto. Al mismo tiempo, una serie de agentes supervisores especiales optimizan las rutas a través de la red y monitorizan la calidad del servicio. Por supuesto, todo esto ocurre en un intervalo de tiempo muchísimo menor que el que llevaría a los seres humanos realizar las mismas tareas manualmente.

Además, el middleware grid permite someter peticiones para ejecutar trabajos en el Grid; trabajos que se ejecutan en cualquier lugar de la red. Otros componentes software como el ″resource broker″ y el ″replica manager″ complementan el middlware y distribuyen los trabajos en los nodos que parecen ser los más adecuados para ellos. Para el usuario del middleware estos detalles están ocultos, para éste sólo el resultado es relevante: el trabajo es ejecutado en otro sitio.

El concepto de middleware grid puede verse como el sistema operativo de un supercomputador virtual que en realidad consiste en varios nodos independientes situados en diferentes emplazamientos, pero que aparecen ante el usuario como una única unidad computacional.

Un aspecto importante a tener en cuenta es la seguridad. Lo que un usuario puede hacer depende de su autorización, la cual se determina en función de su pertenencia a alguna VO. La autorización se realiza la mayor parte de las veces a través de una infraestructura de certificados estándar.

Otro aspecto que necesita ser manejado es el siguiente: cada institución tiene control sobre su dominio, cada institución puede tener diferentes sistemas y políticas. El middleware Grid se usa para vencer esta heterogeneidad y para proporcionar un interfaz en el límite del dominio de administración. De manera figurada, podemos describir el problema como un reloj de arena: en un extremo hay un conjunto variado de recursos y en el otro extremo hay varias organizaciones virtuales que tienen sus propias aplicaciones. Las aplicaciones pueden obtener acceso a los recursos heterogéneos mediante un pequeño conjunto de interfaces bien definidas; el middleware grid.

Entre las funcionalidades del middleware podemos destacar:

- Asignación eficiente de recursos.
- Ejecución de trabajos y posterior transferencia de resultados.
- Almacenamiento, registro y posterior localización y acceso a datos.
- Proveer mecanismos de seguridad: autenticación, autorización ....
- Monitorización.

Hay muchas soluciones middleware, pero algunas de las más populares son las que listamos a continuación: gLite (Lightweight Middleware for Grid Computing), Globus Toolkit 4, LCG, UNICORE6 (Uniform Interface to Computing Resources), OGSA-DAI, ProActive/GCM Parallel Suite, GRelC y Alchemi, entre otras.

3.5 Globus Toolkit

Globus Alliance, también conocida como Proyecto Globus, es un programa formado por un conjunto de organizaciones que tienen el objetivo común de construir herramientas que permitan la creación de sistemas Grid. Está formado por instituciones como: la Universidad de Chicago, la Universidad de Edimburgo, el Centro Nacional para las Aplicaciones de Supercomputación (NCSA), el Instituto de Ciencias de la Infomación del Sur de California y empresas como HP, IBM, Intel, Sun, Nortel, Univa y Cisco Systems.

El objetivo de este proyecto es generar software de código abierto que se utilizará en la producción de actividades científicas, de ingeniería y comerciales.

El principal producto de la *Globus Alliance* es el *Globus Toolkit*, considerado como el estándar de facto en la computación grid y formado por un conjunto de herramientas desarrolladas bajo licencia open source. La primera versión se publicó en 1998 y actualmente está disponible la versión 4.

Globus Toolkit es una colección de componentes software, APIs y librerías que permiten la creación y ejecución de aplicaciones distribuidas, y la construcción de un Grid. Actualmente, Globus se ha convertido en el estándar de facto para la computación distribuida y será el soporte sobre el que se desarrollarán aplicaciones grid. Globus consta de tres componentes fundamentales: gestor de recursos, servicio de información y gestor de datos; todos ellos construidos sobre una infraestructura de seguridad basada en certificados GSI (Grid Security Infrastructure).

Los componentes anteriores, ya sea de forma independiente o conjunta, facilitan el acceso transparente y seguro a recursos distribuidos geográficamente en diferentes dominios de administración, además de servir como herramientas básicas para implementar las fases de la planificación de trabajos grid tales como: descubrimiento, selección y preparación de recursos; y envío, monitorización, migración y finalización de trabajos.

Inicialmente las versiones 1(GT1) y 2 (GT2), se basaban en un uso intensivo de interfaces propietarias para la comunicación entre los diferentes servicios desplegados. Sin embargo, estas interfaces fueron implementadas utilizando protocolos estándar como LDAP (Lightweight Directory Access Protocol), FTP (File Transfer Protocol), etc. La versión GT2 sentó las bases de la computación Grid tal y como la conocemos en la actualidad. Su rápida aceptación e incorporación dentro de numerosos proyectos de investigación

relacionados con las tecnologías Grid, le convirtió en el estándar de facto para el despliegue de Grids computacionales.
El lanzamiento de la versión 3 (GT3) trató de ser un acercamiento al uso de protocolos abiertos que fracasó estrepitosamente, ya que la implementación de los servicios web ofrecida no respetó ningún estándar perteneciente a una arquitectura orientada a servicios. Este inconveniente produjo el rápido desarrollo e implantación de la versión 4 (GT4), versión que está actualmente en vigor y que ha integrado satisfactoriamente las tecnologías orientadas a servicios por medio de los servicios web. Empleando estos protocolos estándar, GT4 ha definido su arquitectura e interfaces, dotando de una interacción mucho más flexible a todos sus servicios.

La aparición de GT4 produjo una escisión en cuanto a la evolución del desarrollo de las herramientas basadas en Globus, ya que la nueva versión es incompatible con las anteriores. De este modo, los servicios de versiones anteriores que no están basados en los servicios web cobraron la denominación de Pre-WS (pre-Web Services). Sin embargo, la gran difusión que tuvo GT2 junto con la confianza que infunde a los usuarios, hace que los servicios pre-WS aún se mantengan por compatibilidad.

3.5.1 GT 2

Globus Toolkit 2 fue el resultado del proyecto Globus y se convirtió en el estándar de facto para poner en marcha grids computacionales.

Los pilares básicos del Globus Toolkit, desde la versión GT2 son:

- GRAM (Grid Resource Allocation Management),
- MDS (Monitoring and Discovering Services),
- GridFTP (Grid File Transfer Protocol),
- GSI (Globus Security Infrastructure).

1. Gestión de recursos - GRAM

La gestión de recursos grid se define como el proceso de identificar requerimientos, hacer matching de recursos con las aplicaciones, asignar estos recursos, planificarlos y monitorizarlos a través del tiempo para lanzar las aplicaciones sobre el grid lo más eficientemente posible.

La arquitectura de gestión de recursos de Globus permite el acceso transparente, unificado y seguro a los distintos gestores de recursos locales de cada centro o institución. Los principales componentes de esta arquitectura son: el lenguaje de especificación de recursos (RSL) y el gestor de asignación de recursos (GRAM).

El *lenguaje de especificación de recursos (RSL, Resource Specification Language)* es un pilar básico para comunicar peticiones de recursos entre componentes: de aplicaciones a resource brokers, co-allocators de recursos y gestores de recursos. En cada fase de este proceso, la información sobre los requerimientos es codificada en una expresión RSL por la aplicación o por los servicios de más alto nivel como resource brokers o co-allocators, de tal manera que esta expresión puede ir refinándose conforme se va pasando a través de las diferentes entidades hasta el nivel más bajo, en el que todos los requerimientos son fijos.
La información sobre la disponibilidad de los recursos y sus características puede obtenerse vía un sistema de información, otro componente de un sistema grid que comentaremos en el siguiente punto.

Por otro lado, *GRAM* representa el nivel más bajo de la arquitectura globus de gestión de recursos que implementan los resource manager locales.

Está compuesto por un conjunto de protocolos para enviar, monitorizar y terminar un trabajo y es responsable de varias acciones:

1. Procesar las especificaciones RSL que representan peticiones de recursos, ya sea para denegar la petición o para crear uno o más procesos ("trabajos") que satisfagan la petición.

2. Permitir la monitorización y gestión remota de trabajos creados como respuesta a estas peticiones.

3. Actualizar periódicamente el servicio de información de MDS, con información referente a la disponibilidad y capacidades actuales de los recursos que gestiona.

GRAM está pensado para hacer de interfaz entre los sistemas de planificación global y los gestores locales de recursos, de manera que una petición que llega al resource manager local deberá ser lo suficientemente concreta para que se pueda satisfacer.
Los gestores locales no tienen que representar un único host, sino que más bien representan un servicio que da acceso a varios recursos computacionales. Por lo tanto, *GRAM* necesita tener interfaces con estos sistemas locales, de manera que se pueda mapear entre las especificaciones RSL y los recursos que están disponibles. En la actualidad se disponen de interfaces con sistemas de gestores locales como son Condor, PBS, Easy, Fork, LoadLeveler, LSF y NQE.
Estas interfaces implementan una serie de métodos y funciones que constituyen el API de *GRAM*, y que juegan un papel importante en la gestión de recursos.
Este API provee funciones para lanzar y cancelar trabajos, y para preguntar sobre el estado de los mismos; si está ejecutándose o si está aceptado o en cola. Cuando lanzamos un trabajo, se nos devuelve un manejador de tarea (job handle) único, y éste puede ser utilizado para monitorizar y controlar el progreso del mismo.
Adicionalmente, en la petición de envío del trabajo puede requerirse que el progreso del mismo sea comunicado asíncronamente a una dirección URL de callback. Estos identificadores pueden pasarse a otros procesos, y los callbacks no tienen que ser dirigidos obligatoriamente al proceso que envió el trabajo.
Esto hace posible el envío de trabajos por terceras partes en nombre de la aplicación, como un gestor de más alto nivel o un co-allocator.

La implementación de esta arquitectura es la siguiente:

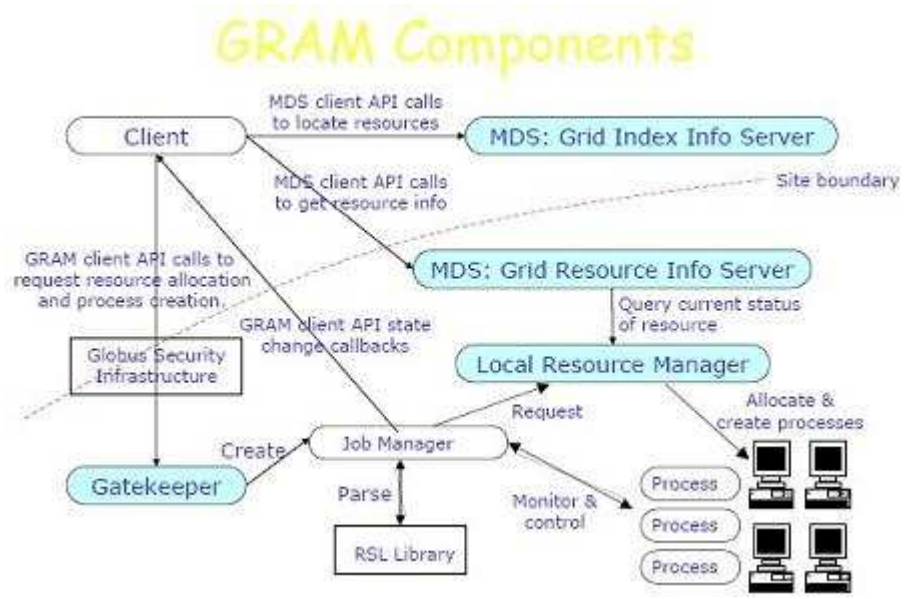

Figura 3.2: Arquitectura de *GRAM*

Los principales componentes son la librería GRAM cliente, el gatekeeper, la librería de parsing de RSL, el jobmanager (gestor de trabajos) y el GRAM reporter.

La *librería GRAM cliente* es usada por el usuario, aplicación o quien actúe en su nombre para enviar el trabajo. Interactúa con el gatekeeper de un sitio remoto realizando una autenticación mútua a través de los métodos GSI que se mencionarán posteriormente.
Envía una especificación de recursos, una posible petición de callback, y algunos componentes más que no están directamente relacionados con la gestión de recursos.
El *gatekeeper GRAM* es un componente bastante simple que debe estar ejecutándose como servicio remoto, y que hace básicamente tres cosas: realizar la autenticación mutua del usuario y los recursos a través de los servicios más básicos de GSI, determinar el usuario local bajo el cual se ejecutará el programa (también realizado en conjunción con estos servicios GSI), y ejecutar un jobmanager que se encargará de ejecutarse bajo ese usuario efectivo y se hará cargo de cada petición.

Las dos primeras tareas son realizadas, como ya hemos comentado, por la infraestructura de seguridad de Globus, que maneja la autonomía y el substrato heterogéneo en el dominio de la seguridad. Para iniciar el jobman-

ager, el gatekeeper debe autentificarse como un usuario privilegiado. Aún así, debido a la efectividad de la interfaz con GSI, el gatekeeper ha pasado numerosas revisiones por parte de organizaciones que han validado su código para su ejecución por parte de grandes centros de recursos.
El mapeo de usuarios remotos a usuarios locales minimiza la cantidad de código que se debe ejecutar en modo root. Además nos permite delegar en el sistema local.

El jobmanager es responsable de crear los procesos necesarios para satisfacer la petición del usuario. Esta tarea básicamente tiene que ver con realizar la petición correspondiente de recursos al sistema de gestión de recursos local (PBS, LSF, ...). Pero en caso de no existir ningún sistema local de gestión se podría utilizar el método *fork* para dicho fin.

Una vez creados los procesos, el jobmanager también es responsable de monitorizar y verificar su estado, notificando mediante callbacks los cambios de estado por los que pasa la petición, e implementando operaciones de control como las de terminación del trabajo. Un jobmanager terminará cuando el trabajo del cual es responsable haya terminado.

El jobmanager está formado por dos elementos:

1. Los componentes comunes (Common Components): trasladan los mensajes recibidos, del gatekeeper y el cliente en sí a una API interna que es implementada por el componente específico de la máquina. También translada las peticiones de callback del componente especifico a través de la API interna en mensajes al gestor de la aplicación.

2. El componente específico de la máquina (Machine-Specific Component): implementa la API interna en el entorno local. Esto incluye llamadas al sistema local, mensajes al monitor del recurso y preguntas al sistema de información (MDS).

Como hemos mencionado anteriormente, el jobmanager se encarga de monitorizar el estado de los trabajos que va lanzando. Los estados que puede alcanzar un trabajo son los siguientes:

- **Unsubmitted:** El trabajo no ha sido enviado al planificador. Para este estado nunca es enviado un callback, se utiliza en caso de que el jobmanager sea parado y vuelto a arrancar antes de que el trabajo sea enviado.

- **StageIn:** El jobmanager está recibiendo (stage-in) el ejecutable, la entrada o datos del trabajo. Los trabajos que no necesitan esta información no pasan por este estado.
- **Pending:** El trabajo ha sido enviado al planificador local, pero los recursos aún no han sido reservados para el trabajo.
- **Active:** El trabajo ya dispone de todos sus recursos y la aplicación se está ejecutando.
- **Suspendido:** El trabajo ha sido parado temporalmente por el planificador. Sólo algunos planificadores pueden causar este estado en un trabajo.
- **StageOut:** El jobmanager está enviando los ficheros de salida que ha producido el trabajo a almacemiento remoto. Los trabajos que no realizan este envío de datos no pasan por este estado.
- **Done:** El trabajo se ha completado satisfactoriamente.
- **Failed:** El trabajo terminó antes de completarse correctamente, debido a un error o a la cancelación por parte del usuario o del sistema.

El último componente que nos queda comentar es el *Gram Reporter*, que se encarga de almacenar en el sistema de información las características del planificador o gestor local de recursos (como por ejemplo el número de colas que existen, si éste soporta reserva, ...) y el estado del mismo (número de nodos, cúales de ellos estan ocupados y cuáles disponibles, trabajos activos, etcétera).

2. Servicio de información - MDS

El sistema de información de Globus es el *Monitoring and Discovery System (MDS)*, que usa el protocolo LDAP para la consulta uniforme de la información referente a los sistemas en el Grid.

Los servicios de información se utilizan para indexar, publicar y buscar información relativa a los recursos y servicios disponibles en cada nodo del Grid. Son utilizados por el resto de servicios de Globus que publican información a través de ellos.

El comportamiento del servicio es el siguiente: el Grid Resource Information Service (GRIS), provee de manera uniforme la búsqueda de recursos obteniendo como resultado la configuración actual, las capacidades, el estado y las prestaciones de cada recurso del Grid. La información suministrada por cada GRIS se agrupa en el Grid Index Information Service (GIIS), que ofrece una imagen conjunta y coherente de los recursos del Grid, es decir, acepta mensajes de registro de los GRIS y une esas fuentes de información en un espacio unificado de información.

En esta arquitectura de sistemas de información se hace frente a los requerimientos únicos de los entornos grid, y cuenta con dos elementos básicos:

- Una gran colección distribuída de proveedores de información genéricos que dan acceso a la información sobre entidades individuales, a través de operaciones o gateways, a otras fuentes de información. La información es estructurada siguiendo un modelo estándar de datos, tomado de LDAP: una entidad es descrita por un conjunto de "objetos" compuestos de pares atributo-valor.

- Servicios de alto nivel, que recopilan, gestionan, indexan y responden a la información suministrada por los anteriores proveedores de información. Se distinguen en particular servicios agregados de directorio, que facilitan el descubrimiento y monitorización de VOs implementando vistas y búsquedas tanto genéricas como especializadas, para una colección de recursos. Otros servicios de alto nivel pueden usar esta información además de otra directamente obtenida por proveedores.

Un proveedor de información es definido como un servicio que habla dos protocolos básicos: el *Grid Information Protocol (GRIP)* que se usa para acceder a información sobre las entidades, mientras que el *Grid Registration Protocol (GGRP)* se utiliza para notificar al directorio agregado servicios sobre la disponibilidad de la información.

3. Gestión de datos - GridFTP

GridFTP es el protocolo que se propuso para todas las transferencias de datos en entornos Grid. Extiende el protocolo FTP estándar con facilidades como transferencias particionadas, autoadaptación de las transferencias y seguridad basada en Globus.

Las aplicaciones científicas y de ingeniería que utilizan grandes volúmenes de datos requieren la transferencia de grandes cantidades de información (terabytes) entre sistemas de almacenamiento distribuidos geográficamente y acceso remoto a grandes conjuntos para las aplicaciones de usuario.

Desafortunadamente, muchos de estos sistemas de almacenamiento utilizan protocolos incompatibles o inéditos, que ni siquiera han sido publicados, por lo tanto, requieren el uso de las librerías del cliente para el acceso a los datos. Estos protocolos y librerías del cliente particionan la información entre los diferentes sistemas de almacenamiento, haciendo necesarios múltiples métodos de acceso a la información. Para vencer estas incompatibilidades entre protocolos, se desarrolló un protocolo universal de transferencia y acceso a los datos: GridFTP.

Como ya hemos mencionado anteriormente, GridFTP extiende el protocolo FTP y proporciona un amplio conjunto de herramientas que son las que ofrecen los diferentes sistemas de almacenamiento Grid en uso actualmente.

Los proveedores de almacenamiento consiguen una base de usuarios más amplia ya que sus datos están disponibles para cualquier usuario, y los usuarios amplian el rango de sistemas de almacenamiento a los que pueden acceder.

Es importante destacar el porqué de la elección del protocolo FTP. Además de ser el protocolo más comúnmente utilizado para transferencias de datos en internet, es el candidato idóneo para cubrir las necesidades de los sistemas Grid:

- FTP es un protocolo IETF estándar, ampliamente implementado y bien entendido. Por ello, hay una gran base de código y habilidades sobre las que trabajar.
- Proporciona una arquitectura bien definida para las extensiones del protocolo y posibilita el descubrimiento dinámico de las extensiones soportadas por una implementación particular.
- Numerosos grupos de desarrolladores de software Grid han añadido extensiones a través de IETF (Internet Engineering Task Force), muchas de las cuales serán particularmente útiles en entornos Grid.
- Además de transferencias cliente/servidor, el protocolo FTP también soporta transferencias directas entre dos servidores, mediadas por un cliente contra terceros (third party client).

La funcionalidad del protocolo GridFTP incluye ciertas características que son soportadas por las extensiones de FTP que ya han sido estandarizadas

(RFC 959) [13] pero que rara vez son implementadas en sistemas actuales. El resto de funciones son extensiones nuevas del protocolo FTP.

- **Soporte para Kerberos y GSI (Grid Security Infrastructure):** Contar con una autenticación robusta y flexible, integridad y confidencialidad, son tres puntos críticos a tener en cuenta en la transferencia o acceso a ficheros. GridFTP debe soportar la autenticación a través de GSI y Kerberos, con varios niveles de confidencialidad y/o integridad de datos controlados por el usuario. Para ofrecer esta funcionalidad, el protocolo implementa los mecanismos de autenticación definidos por el estándar RFC 2228 (FTP Security Extensions)[14].

- **Control "third-party" de la transferencia de datos:** Para gestionar grandes conjuntos de datos para comunidades distribuidas, el protocolo GridFTP proporciona control *third-party* de las transferencias de datos entre los servidores de almacenamiento. Las operaciones *third-party* permiten, a un usuario o aplicación en una ubicación, monitorizar y controlar una operación de transferencia de datos entre otras dos ubicaciones: el origen y destino de la transferencia. Dicho usuario no participa en la trasferencia, solo la gestiona.

- **Transferencia de datos paralela:** En las conexiones de banda ancha, la utilización de múltiples streams TCP en paralelo (incluso entre el mismo origen y destino) puede aumentar el ancho de banda total por encima del conseguido utilizando un único stream TCP. GridFTP soporta transferencias de datos paralelas a través de la extensión tanto de los comandos FTP como de los canales de datos.

- **Transferencia de datos en franjas (Striped Mode):** Los datos se dividen en franjas entre los múltiples servidores. GridFTP incluye extensiones que inician transferencias en modo "striped", las cuales utilizan múltiples streams TCP para transferir datos que están particionados entre varios servidores. Este modo de transferencia proporciona importantes mejoras en el ancho de banda superiores a las conseguidas con las transferencias paralelas.

- **Transferencia de ficheros parcial:** Ciertas aplicaciones pueden obtener más beneficio transfiriendo porciones de ficheros que con la transferencia completa del mismo, por ejemplo, los análisis de física de alta energía requieren acceso a un subconjunto relativamente pequeño de ficheros pertenecientes a una base de datos masiva y orientada a objetos físicos. Lo máximo que permite el protocolo FTP estándar es la transferencia del resto de un fichero empezando en un offset particular, es decir,

a partir de cierta posición dentro del fichero. Sin embargo, GridFTP proporciona comandos que soportan la transferencia de subconjuntos o regiones arbitrarias de un fichero.

- **Negociación automática del tamaño de buffer TCP:** El uso de parámetros óptimos a la hora de elegir el tamaño del buffer para las conexiones TCP puede mejorar considerablemente el rendimiento de las transferencias de datos. Sin embargo, realizar el ajuste de estos parámetros manualmente puede suponer un error sobretodo cuando se trata de usuarios no expertos, o simplemente puede que el usuario ni siquiera se lo plantee y por tanto no lo haga. GridFTP extiende los comandos del protocolo FTP para que soporte ambas opciones: negociación manual y automática del tamaño del buffer TCP para ficheros grandes y para grandes conjuntos de ficheros pequeños.

- **Soporte para la transferencia de datos fiable:** Este tipo de transferencias son importantes para muchas aplicaciones que manejan datos. Los métodos de recuperación de fallos son necesarios para manejar fallos como puede ser: redes transitorias y apagones en los servidores. El protocolo FTP estándar incluye herramientas básicas para el restablecimiento de transferencias fallidas que no están ampliamente implementadas. GridFTP aprovecha dichas herramientas y las extiende para cubrir el nuevo protocolo para los canales de datos.

A pesar de todas las ventajas que supone el uso de GridFTP, también hay que destacar las limitaciones que presenta su implementación. Dichas limitaciones harán que en las versiones actuales de Globus Toolkit se haya optado por otros protocolos de gestión de datos que citaremos posteriormente.

Algunas de las limitaciones más importantes que presenta GridFTP aparecen listadas a continuación:

1. Debido a que GridFTP no es un protocolo orientado a servicios web, no es capaz de utilizar los servicios web implicados en el despliegue de SOAP, WSDL, etc.

2. Para usar GridFTP, es obligatorio para el cliente mantener un socket abierto para la conexión con el servidor durante todo la transferencia de datos. En esta situación, cuando el socket de conexión con el servidor es interrumpido en algún instante, la recuperación de los datos no es factible ya que la información requerida para ello está oculta en la memoria de la máquina cliente. Sin embargo, si se utilizaran copias globales (globus-url-copy), los fallos en la máquina remota podrían ser convenientemente manejados durante la transferencia de datos.

3. Aunque GridFTP es una extensión del estándar FTP que incorpora seguridad basada en Globus, concretamente en GSI (Grid Security Infrastructure), a diferencia de FTP, esta implementación no está integrada en el navegador del lado cliente.

4. Un FTP estándar se construye directamente sobre varios navegores (browsers), por lo tanto, la barra de dirección del navegador puede ser usada integramente para buscar, subir y descargar ficheros utilizando una url FTP como puede ser (ftp://server:port/path).

5. El acceso a los servidores GridFTP requiere la atenticación del usuario utilizando GSI antes de utilizar un programa de línea de comandos como UberFTP. Por ello, para usar GridFTP, es necesario instalar y configurar Globus Toolkit en la máquina cliente.

4. Seguridad - GSI

Globus Toolkit usa GSI (Globus Security Infrastructure) para habilitar la autentificación y para tener una comunicación segura a través de una red abierta.

Las motivaciones principales de GSI son:

- Comunicación segura entre los elementos del grid.
- Soporte de registro único (single sign-on), incluyendo delegación de credenciales para realizar cómputos que involucren múltiples recursos y sitios. La funcionalidad *Single Sign-On (SSO)* es un procedimiento de autenticación que habilita al usuario para acceder a varios sistemas con una sola instancia de identificación.
- Ofrecer seguridad a través de diversos dominios administrativos. No existe un sistema de seguridad centralizado.

La seguridad es uno de los pilares fundamentales sobre los que se tiene que establecer un grid y todos los servicios superiores, y debe tenerse muy en cuenta ya que la filosofía de compartición de recursos tiene muchos problemas asociados. Los recursos pueden ser valiosos y por lo tanto se debe permitir el acceso sólo cuando se desee y a las entidades que se desee.
También los problemas a resolver pueden ser sensibles a la privacidad, así como los datos que estos problemas requieren o generan.

Por lo general, los recursos están situados en distintos dominios administrativos, por lo cual cada uno tiene sus propias políticas de acceso, procedimientos, mecanismos de seguridad, etcétera.
Para tener en cuenta estos puntos, la implementación de los servicios de seguridad tiene que estar públicamente disponible; lo que está relacionado con que los protocolos sean estándar, bien probados, y comprendidos por la comunidad.
El conjunto de recursos puede ser elevado, dinámico e impredecible, por lo que no estamos hablando de simple autorización y autenticación en un entorno cliente/servidor, sino que se necesita un método de delegación de credenciales de servicio a servicio, por el cual sea posible que se pueda autorizar y autenticar en nuestro nombre bajo unas circunstancias determinadas y en un entorno controlado.

La característica de autorización que aporta GSI introduce el concepto de Organización Virtual (VO), que como ya hemos mencionado en otras ocasiones, es una agrupación de usuarios y recursos dedicados a una actividad común.
Este concepto nos permiten estructurar, jerarquizar y organizar el Grid (los nodos deciden "soportar" las VOs de su interés). En consecuencia, un usuario que quiera trabajar en una infraestructura grid, debe ser miembro de, al menos, una VO que cubra sus intereses y podra usar los recursos que ofrezca dicha VO. La información de los miembros de las Organizaciones Virtuales se guarda en VO-servers (componente no incluido en Globus).

Los requerimientos en cuanto a seguridad son varios, dependiendo del punto de vista que adoptemos. Para un usuario las características que se deben satisfacer son las siguientes:

- Que sea fácil de usar, con comandos sencillos y con pocos pasos manuales.
- Que sólo se requiera un único log-in, ya que no es cómodo introducir las credenciales cada vez que se necesita autenticación o autorización en un recurso.
- Que se utilice un modelo basado en la credibilidad del usuario.
- Que sea posible la utilización de proxies y agentes.

Desde el punto de vista del propietario de los recursos, los puntos más importantes a tener en cuenta serían:

- Poder especificar políticas de control de acceso local, que sean flexibles y versátiles para poder plasmar las necesidades de cada entorno.
- Que sea posible el auditar y controlar, para hacer frente a posibles brechas en la seguridad de sus sistemas y tener un histórico de accesos.
- Integración con los sistemas de seguridad locales como AFS, Kerberos.
- Protección contra otros recursos comprometidos.

Desde el punto de vista del desarrollador que necesita la utilización de estos servicios, se requiere:

- Disponibilidad de un API/SDK con métodos para autenticar, hacer flexible la protección de mensajes, proveer métodos para la delegación, etc.
  - A través de llamadas directas (GSS-api).
  - O integrada en los servicios de nivel superior: GlobusIO, Condor-G, mpich-g2.

Para ello se integran en Globus todas estas necesidades en lo que se llama *Globus Security Infraestructura (GSI)*. Es un conjunto de protocolos y APIs que hacen frente a las necesidades planteadas, y que están basados en una serie de protocolos estándar, extendiéndolos. En particular se utilizan protocolos de autenticación de clave pública para la autenticación de usuarios y recursos, y para la protección de mensajes.

Para la identificación se usan certificados X.509[15], que se basan en una infraestructura de clave pública (PKI) [16].
Explicar en profundidad esta infraestructura está fuera de los propósitos de esta memoria, pero básicamente la infraestructura de clave pública (PKI) permite conocer que una determinada clave pertenece a un determinado usuario (o entidad). Está basada en los principios de la encriptación asimétrica, en la que cada entidad posee un par de claves, una pública y otra privada, de manera que los datos encriptados con una clave sólo pueden ser desencriptados con la otra. La clave pública es conocida, mientras que la privada es secreta y sólo debe ser conocida por la entidad.
En la PKI, la clave pública es encapsulada en un certificado X.509 que identifica inequívocamente a la entidad para la que se emitió. Además del nombre único del dueño del certificado (Distinguished Name o DN), éste contiene otra información como la fecha de caducidad, el emisor y una firma emitida

por este último.

El emisor del certificado será una Autoridad de Certificación (CA), que a través de su clave privada firmará dicho certificado, dando fé de su autenticidad, que siempre se puede comprobar a través de la clave pública de esta Autoridad.
Uno de los requerimientos principales para que sea posible la utilización en sistemas Grid, es que exista la posibilidad de hacer un único log-in y de utilizar la delegación de credenciales. Para ello es necesario la utilización de credenciales proxy (proxy credentials), que permitan que una entidad A otorgue a otra entidad B el derecho de ser autorizada con otros como si fuera la primera entidad A. De esta manera un proxy podrá ser creado a partir de un certificado normal X.509 o a partir de otro proxy, con el propósito de aplicar proxys restringidos según las necesidades.
Además se podrá aplicar delegación, que consiste en la posibilidad de creación remota de un proxy credencial de segundo nivel. Los proxies pueden ser restringidos, en el sentido de que se pueden delegar un subconjunto de derechos, lo que es deseable para tener restricciones de grano más fino. Después cada servicio en el que se quiere autenticar puede decidir si lo permite o no, por ejemplo como luego veremos, el Job Manager requiere un full proxy sin limitaciones, mientras que GridFTP permite un proxy con o sin limitaciones.
GSI soporta además el Standard GSS-API, implementando interfaces que soportan autenticación, delegación, integridad y confidencialidad de mensajes, lo que permite soportar aplicaciones como SSH o GridFTP.

Una cuestión aparte es la autorización, que no es llevada a cabo directamente por GSI. Es decir una vez que una entidad se ha identificado como tal y sabemos inequívocamente quien es, el problema de la autorización se basa en permitir o no el uso de ciertos recursos a esta entidad autenticada.

En la idea más simple, se mantiene en cada recurso que se quiere autenticar un fichero ”gridmapfile” que contiene los Distinguished Names de aquellos certificados que se aceptan, por lo que cada vez que se accede a un servicio de este recurso se comprueba si el proxy con el que se está accediendo está incluido, permitiendo el acceso si es así o rechazándolo en caso contrario.

En la práctica un usuario de un grid deberá disponer de un certificado de usuario X.509 expedido por una autoridad de certificación aceptada por éste grid. Con éste certificado le será posible contactar con los servicios de recepción de trabajos, pero realmente lo que se hace en el cliente es obtener un proxy a partir del certificado de usuario que será el que realmente se envíe

en la petición del servicio.

Básicamente, en el servidor se obtiene información del usuario a partir de este proxy, su DN que es el que se utiliza para ver si el usuario está finalmente autorizado o no a utilizar el servicio. Si este servicio necesita la comunicación con otros, se utilizará la delegación de este proxy como método para autenticarse en éstos nuevos servicios.

Según lo que hemos visto, se puede decir que un intercambio de mensajes GSI tiene tres fases, que ocurren en el siguiente orden:

1. *Establecimiento del contexto*, donde las dos partes se autentican mutuamente y establecen un contexto para proteger los futuros intercambios de mensajes.

2. *Delegación*, donde el cliente puede delegar sus credenciales en el servicio para que posteriormente se usen en su nombre.

3. *Solicitud específica*, en la cual se intercambia la información de la aplicación, opcionalmente protegida por el contexto establecido en la primera fase.

3.5.2 GT 4

En los desarrollos actuales de middleware, y más concretamente de Globus Toolkit, se han extendido algunos conceptos de la tecnología grid que han conducido a una búsqueda de convergencia con los conceptos de provisión de servicio y, en definitiva, con los Web Services.

GT4 es un software de base, no orientado a usuarios finales, sino a desarrolladores de aplicaciones e integradores de sistemas. Esta nueva versión se centra en la especificación e implementación del Web Service Resource Framework (WSRF), que consiste en una extensión de los servicios web actuales para darles características como estado, tiempo de vida, etc.

En la siguiente figura podemos ver los componentes que incluye. Distinguimos dos tipos de componentes: componentes WS y componentes pre-WS. Los primeros están orientados a servicios web y los aporta GT4, mientras que los segundos pertenecen a GT2 y como ya hemos mencionado se mantienen por compatibilidad.

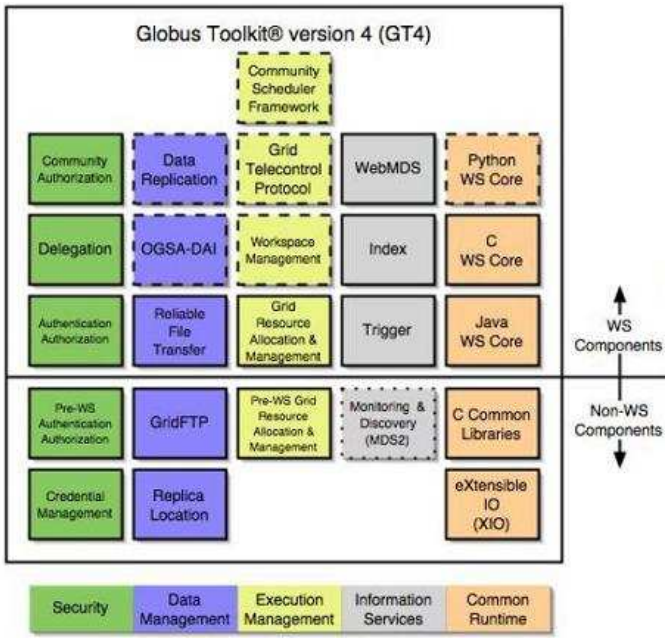

Figura 3.3: Componentes Globus Toolkit 4

- **WS Authentication & Authorization:** Contiene las librerías que hacen posible la autenticación y la autorización para los componentes WS. Esto incluye funcionalidades como autenticación basada en X.509,

en GSI SecureConversation, protección de mensajes, y todo un framework de autorización con varios mecanismos.

- **OGSA-DAI:** El objetivo de OGSA-DAI es proveer métodos uniformes para descubrir, acceder e integrar recursos de datos heterogéneos, principalmente bases de datos, en el Grid. Las bases de datos oficialmente admitidas son:
  - MySQL 3.2.x / 4.0.x
  - DB2 8
  - Oracle 9i / 10g
  - Xindice 1.0
  - SQL Server 2000
  - PostgreSQL 7.x

  Adicionalmente, aunque no de forma oficial, muchas bases de datos con el interfaz JDBC, ODBC o XML:DB, que admiten el SQL o el XPath, pueden publicarse usando OGSA-DAI. Los métodos de envío de datos admitidos son SOAP cliente a fichero, Streams HTTP(S), GridFTP y FTP a servidor separado.

- **Reliable File Transfer (RFT):** Este servicio tiene un rol similar al planificador batch en una máquina. Se le envía un trabajo de transferencia de datos, que podría consistir en 10000 o 100000 ficheros, con la información de los tamaños de buffer, streams,etc. La petición, junto con la información necesaria para reiniciar (en el caso de ser necesario) y las notificaciones, son almacenados en una base de datos. El servicio entonces ejecuta una transferencia GridFTP a terceros por el usuario.

- **WS Grid Resource Allocation and Management (WS GRAM):** WS GRAM ofrece un único interfaz para pedir y usar recursos remotos para la ejecución de trabajos. GT4 incorpora hasta dos implantaciones de GRAM:
  - Basada en un protocolo Pre-WS no propietario (Pre-WS GRAM).
  - Construida usando interfaces WS (WS GRAM).

- **WS Monitoring and Discovery System (MDS4):** Se trata de un monitor de nivel de Grid basado en WSRF, usado para descubrimiento y manejo de recursos. Sirve para obtener un gran rango de información relativa a los recursos básicos y colas, y puede servir de interfaz para

monitorización de sistemas cluster como Ganglia.
Cada servicio basado en WSRF ofrece bastante información de monitorización sobre sí mismo, de tal manera que permite a WS MDS usar sus datos.
Igual que cualquier sistema de monitorización Grid, MDS4 ofrece un servicio de indexado donde los datos son recogidos y almacenados. Además, MDS4 almacena información en la base de datos Xindice, basada en XML, y tiene un interfaz web para poder ver los datos fácilmente.

- **Phyton-C-Java WS Core:** Un contenedor proporciona el entorno de ejecución para un servicio web. Muchos de los componentes GT4 son servicios web y, por tanto, necesitan ejecutarse dentro de un contenedor.
  GT4 proporciona contenedores de servicios web para desarrollar y gestionar servicios escritos en Java, C y Python.

Web Services

Un servicio web (Web service) es un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones. Distintas aplicaciones de software desarrolladas en lenguajes de programación diferentes, y ejecutadas sobre cualquier plataforma, pueden utilizar los servicios web para intercambiar datos tanto en redes de ordenadores como en Internet. La interoperabilidad se consigue mediante la adopción de estándares abiertos. Las organizaciones OASIS y W3C (World Wide Web Consortium) son los comités responsables de la arquitectura y reglamentación de los servicios web. Para mejorar la interoperabilidad entre distintas implementaciones de servicios Web se ha creado el organismo WS-I (Web Services Interoperability Organization), encargado de desarrollar diversos perfiles para definir de manera más exhaustiva estos estándares.

Las principales ventajas de los servicios web son las siguientes:

1. Aportan interoperabilidad entre aplicaciones de software independientemente de sus propiedades o de las plataformas sobre las que se instalen.

2. Fomentan los estándares y protocolos basados en texto, que hacen más fácil acceder a su contenido y entender su funcionamiento.

3. Al apoyarse en HTTP, los servicios web pueden aprovecharse de los sistemas de seguridad firewall sin necesidad de cambiar las reglas de filtrado.

4. Permiten que servicios y software de diferentes compañías ubicadas en diferentes lugares geográficos puedan ser combinados fácilmente para proveer servicios integrados.

5. Permiten la interoperabilidad entre plataformas de distintos fabricantes por medio de protocolos estándar y abiertos. Las especificaciones son gestionadas por una organización abierta, la W3C, por tanto no hay secretismos por intereses particulares de fabricantes concretos y se garantiza la plena interoperabilidad entre aplicaciones.

Nos vamos a centrar en los principales estándares para el desarrollo de servicios web: SOAP, WSDL, y UDDI.

- **SOAP (Simple Object Access Protocol) [17]:** Es el protocolo de comunicaciones basado en XML utilizado para el intercambio de información entre aplicaciones software distribuidas; independientemente del sistema operativo subyacente, de los lenguajes de programación o de los modelos de orientación a objetos. En su descripción, SOAP se presenta como un protocolo ligero para el intercambio de información estructurada en un entorno distribuido y descentralizado. Por otro lado, no define ningún modelo de programación ni la semántica de una implementación específica, como podría ser una API, sino que define un mecanismo sencillo para el empaquetado de datos dentro de mensajes basados en esquemas XML. Gracias a la interoperabilidad ofrecida por SOAP, los sitios Web podrán ofrecer servicios Web accesibles al resto de aplicaciones, sin necesidad de la intervención humana. De este modo, una aplicación podrá ensamblar o combinar varias de estas soluciones y obtener un valor añadido. Solamente mediante un analizador XML y un servidor HTTP, es posible la generación de un objeto de negociación SOAP. Al igual que en CORBA, en los servicios web aparece el concepto de ORB, como sistema que asiste en la invocación remota, gestiona tiempos de vida, etc. SOAP es más que un protocolo extensible. Ningún protocolo puede garantizar un funcionamiento tan uniforme en cualquier tipo de arquitectura, plataforma o infraestructura de red. Así, establece mecanismos para el envío de mensajes de tipo RPC. Los mensajes SOAP son básicamente transmisiones unidireccionales con el contenido formateado a partir de esquemas XML, con espacios de nombres adecuados para poder llevar a cabo llamadas remotas.

- **WSDL (Web Services Description Language) [18]:** Es un lenguaje que está basado en XML y que permite la descripción de los servicios web desplegados. WSDL se utiliza también para la localización y ubicación de estos servicios en Internet. Un documento WSDL no es más que un documento XML que describe ciertas características propias de un servicio web, así como su localización y aquellos parámetros y métodos que soporta. Para tal fin, utiliza elementos XML como:

  - <portType> para las operaciones que proporciona el servicio web.
  - <message> para los mensajes que utiliza el servicio web.
  - <types> para los tipos de datos que utiliza el servicio web.
  - <binding> para los protocolos de comunicaciones que utiliza el servicio web.

  Por lo tanto, un documento WSDL representa un servicio como un conjunto de lo que se denomina puertos. Cada puerto tendrá un enlace específico al conjunto abstracto de operaciones y mensajes que el servicio implementará. Las operaciones representan el conjunto de servicios que se publican y que serán accesibles por la red, mientras que los mensajes incluyen los parámetros de entrada y salida a cada una de estas operaciones.

  La estructura de un documento WSDL es semejante a la siguiente:

  <definitions>
  <types>
  los tipos de datos...
  < /types>
  <message>
  las definiciones del mensaje...
  < /message>
  <portType>
  las definiciones de operación...
  < /portType>
  <binding>
  las definiciones de protocolo...
  < /binding>
  < /definitions>

  Así, WSDL se usa a menudo en combinación con SOAP y XML Schema. Un programa cliente que se conecta a un servicio web 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.

- **UDDI (Universal Description Discovery & Integration) [19]:** una vez definido el servicio web, necesitamos darlo a conocer para que el resto de aplicaciones puedan localizarlo. La especificación UDDI describe como un proveedor puede publicar la existencia de un servicio web en un directorio.

  A través de un conjunto de llamadas a API XML basadas en SOAP, se puede interactuar con UDDI tanto en tiempo de diseño como de ejecución para descubrir datos técnicos de los servicios que permitan invocarlos y utilizarlos. De este modo, UDDI sirve como infraestructura para una colección de software basado en servicios Web. UDDI es uno de los estándares básicos de los servicios web cuyo objetivo es ser accedido por los mensajes SOAP y dar paso a documentos WSDL, en los que se describen los requisitos del protocolo y los formatos del mensaje solicitado para interactuar con los servicios web del catálogo de registros.

Para invocar a un servicio web se dan los siguientes pasos:

1. El cliente localiza el servicio web mediante el uso de un registro UDDI o cualquier otro sistema que permita revisar las descripciones WSDL de los servicios web.

2. El registro UDDI responde devolviendo una dirección URI (Universal Resource Identifier) que apunta a uno de los servidores que contiene el servicio web. Los web services suelen usar URIs porque suelen estar dentro de un contenedor Web. Un URI tendría una estructura similar a esta: http://sitio.com/aplicacion/servicio-web-xxxx.

3. El cliente sabe donde está el servicio web mediante el URI, pero no sabe como invocarlo, para lo cual pregunta al servidor web como hacerlo.

4. El servidor web le contesta con un fichero WSDL que aporta los detalles para hacer la invocación. Más concretamente, describe el interfaz del servicio web.

5. Una vez conocido el dónde y el cómo, se hace la invocación. Se puede hacer de varias formas, pero la más común es usar SOAP (Simple Object Access Protocol), para las llamadas remotas de servicios (programas) mediante mensajes codificados y encapsulados en XML.

6. El servicio web contesta con la respuesta SOAP en un mensaje escrito en gramática XML.

Arquitectura de los Servicios Web

La arquitectura de los servicios web se sitúa en relación con varios componentes y tecnologías que comprenden una pila de servicios web o una implementación completamente funcional. Las implementaciones válidas incluyen un subconjunto o parte de esta pila de servicios, pero al menos tienen que proporcionar los componentes dentro de la arquitectura básica. Los componentes y tecnologías que extienden la arquitectura básica están representados dentro de la arquitectura extendida.

La arquitectura básica incluye tecnologías de servicios web capaces de:

- Intercambiar mensajes.
- Describir servicios web.
- Publicar y descubrir las descripciones de los servicios web.

Dicha arquitectura define la interacción entre los agentes software como un intercambio de mensajes entre los proveedores de servicios y los solicitantes de servicios. Los solicitantes son agentes software que hacen una petición para la ejecución de un servicio, mientras que los proveedores proporcionan servicios y son los responsables de la publicación de una descripción del servicio o servicios que ofrecen. Descripción de servicios que el solicitante tiene que ser capaz de encontrar.

La arquitectura básica modela las interaccciones entre tres roles: el proveedor de servicios, el solicitante de servicios y un organismo de descubrimiento de servicios. Las interacciones implican operaciones de publicación, descubrimiento y enlace. Estos roles y operaciones actúan sobre el módulo de software de los web services y su descripción. En un escenario típico, un proveedor de servicios presenta un módulo software accesible a través de la red (una implementación de un servicio web), y define una descripción del servicio que proporciona el web service y lo publica a través del organismo de descubrimiento de servicios. Por otro lado, el solicitante de servicios utiliza una operación de descubrimiento para recuperar la descripción del servicio localmente o desde el organismo de descubrimiento, y usa dicha descripción para interactuar con el proveedor e invocar la implementación del servicio web.

Proveedores y solicitantes se comunican a través de uno o más modelos de intercambio de mensajes, MEP(Message Exchange Patterns), los cuales definen la secuencia de uno o más intercambios de mensajes entre ellos.

Una descripción de servicio se presenta a través de un servicio de descubrimiento, en el cual un proveedor publica la descripción, y desde el cual el solicitante descubre la descripción. Dicha descripción incluye el tipo de datos y la estructura de la información que identifica el MEP y contiene la dirección del proveedor del servicio web.

La arquitectura extendida ofrece un soporte para MEP que agrupa mensajes básicos dentro de interacciones de alto nivel, detalles como soporte para seguridad, transacciones, privacidad y otras, pueden ser representadas en mensajes (módulos SOAP). Esta extensión se construye sobre la arquitectura básica utilizando los mecanismos de ampliación inherentes en las tecnologías básicas.

Figura 3.4: Arquitectura de los servicios web

El mecanismo de descripción de servicios es uno de los elementos clave de la arquitectura de los servicios web. La descripción completa de un servicio se encuentra en dos documentos separados: un documento NASSL (Network Accessible Service Specification Language) y otro documento WDS (Well-Defined Service). NASSL es un lenguaje de definición de interfaces basado en XML para servicios de red, y se utiliza para especificar la información operacional asignada a un servicio web, como puede ser la interfaz del servicio, los detalles de implementación, el protocolo de acceso y los nodos de contacto.

El documento WDS por su parte describe la información no operacional del servicio, como la categoría, la descripción, la fecha de expiración, además de información de negocios del proveedor (nombre, compañía, dirección e información de contacto). Este documento es complementario al documento NASSL correspondiente, juntos describen completamente el servicio y permiten su localización e invocación por parte de los solicitantes de servicios.

Web Services Resource Framework (WSRF)

WSRF es un framework genérico y abierto para el modelado y el acceso a recursos dinámicos usando servicios web. Mientras los servicios web no necesitan utilizar WSRF para implementar aplicaciones que gestionen el estado, WSRF define una serie de convencionalismos para el manejo del estado ofreciendo así la posibilidad a las aplicaciones de descubrir, examinar e interactuar con recursos dinámicos de manera estándar e interoperable. Al igual que otras tecnologías basadas en servicios web (WSDL, SOAP), WSRF especifica una sintaxis de máquina legible, pero su semántica está sólo descrita en texto plano en las especificaciones y no utiliza ninguna tecnología de semántica (como RDF, OWL, WSMO).

En pocas palabras, estas especificaciones permiten al programador declarar e implementar las asociaciones entre un servicio web y uno o más recursos dinámicos.

El estándar WSRF define una serie de especificaciones para la creación de herramientas interoperables, entre las que podemos destacar:

- WS-Resource: Describe la relación entre un servicio web y la entidad que se encargará de salvaguardar su estado, también llamada recurso. La especificación describe los mecanismos necesarios para el acceso y modificación de dichos recursos por medio de interfaces basadas en servicios web.

- WS-ResourceProperties: Define los mecanismos mediante los cuales se establecerán y publicarán las propiedades de un recurso de modo semántico y cómo serán declaradas por parte del interfaz del servicio.

- WS-ResourceLifeTime: Describe los mecanismos necesarios para cubrir el ciclo de vida de un recurso, definiendo diferentes estrategias para la destrucción de recursos.

- WS-Notifications: Incluye especificaciones que permiten que determinados componentes, de una infraestructura Grid, sean informados de

la ocurrencia de cualquier evento producido en el sistema global. Para ello, se establecen mecanismos de publicación y suscripción, de modo que en el sistema existirán componentes que publicarán listados de temas sujetos a notificaciones. A estos temas podrán suscribirse el resto de agentes de la infraestructura, recibiendo solamente información relacionada con los eventos asociados al mismo.

Open Grid Services Architecture (OGSA)

OGSA (Open Grid Services Architecture) describe una arquitectura para entornos de programación orientados a servicios en el campo de la ciencia y los negocios, desarrollado dentro del GGF (Global Grid Forum). Está basada en otras tecnologías de web services, sobretodo en WSDL y SOAP, pero intenta ser en gran parte agnóstica en relación al nivel de transporte de datos. En pocas palabras, OGSA trata de ser una arquitectura de computación e interacción distribuida basada en servicios, que asegura interoperabilidad en sistemas heterogéneos donde diferentes tipos de recursos pueden comunicarse y compartir información. Ha sido descrita como un refinamiento de las emergentes arquitecturas de web services, especialmente diseñadas para soportar requisitos Grid.

Según el Global Grid Forum la arquitectura OGSA es el modelo de industria para la computación grid basada en estándares. El término "Open" se refiere tanto al proceso de desarrollo de estándares como a los estándares en sí. El hecho de que OGSA sea orientado a servicios (service-oriented) se debe a que reparte la funcionalidad entre servicios interactivos bajamente acoplados que están en consonancia con los estándares de web services aceptados en el mercado. Y por último, el término arquitectura define los componentes, sus organizaciones e interacciones, y la filosofía de diseño global.

Esta arquitectura utiliza WSDL para conseguir auto-descripción, servicios descubridores y protocolos interoperables, con extensiones para soportar interfaces de coordinación múltiple y cambio de administración.

OGSA especifica tres características que un servicio web debe tener antes de ser clasificado como un servicio grid. Lo primero que se tiene que cumplir es que el servicio debe ser una implementación de algún tipo de servicio. Las extensiones usadas en OGSA incluyen el concepto de tipo de servicio o "sercicio type", el cual permite describir familias de servicios definidas por una colección de puertos de tipos especificados. La segunda característica dice que el servicio tiene que tener asociado un manejador de servicios grid (GSH - Grid Services Handle). El GSH es un tipo de URI grid para instanciar a los

servicios. No se trata de un link directo a la instancia del servicio, sino que está ligado a una referencia a servicio Grid (GSR - Grid Service Reference). Dicha referencia se refiere al documento WSDL para la instancia del servicio con la entrada "instanceOf" requerida y otras extensiones OGSA. La idea es que el manejador proporcione una manera constante de localizar la referencia (GSR) actual creada para la instancia del servicio, ya que la referencia podría cambiar si la instancia cambia o se actualiza. Y la tercera propiedad para catalogar un servicio como Grid, es el hecho de que cada instancia de un servicio Grid debe implementar un puerto llamado "GridService", que tiene asociadas tres operaciones:

1. Operación FindServiceData: Permite a un cliente descubrir más información sobre el estado del servicio, el entorno de ejecución y detalles semánticos adicionales que no están disponibles en la referencia GSR. Se trata de una propiedad importante que se añade a los servicios y que puede ser usada por los usuarios como una manera estándar de aprender más acerca del servicio.

2. Operación Destroy: Permite a un cliente autorizado destruir la instancia.

3. Operación SetTerminationTime: Se utiliza para extender el tiempo de vida de una servicio.

Además de los puertos GridService requeridos, OGSA define un conjunto estándar de puertos de servicios. Estos puertos definen las propiedades básicas requeridas por todos los sistemas distribuidos: mensajería, descubrimiento, creación de instancias y gestión del tiempo de vida. La gestión de los mensajes corre a cargo de los puertos de notificación (NotificationSource y NotificationSink). El propósito de este servicio es proporcionar un sistema simple de publicación-subscripción similar a Java JMS, pero basado en mensajes XML. Hay que destacar, que el servico de mensajes podría estar implementado por encima de diferentes sistemas comerciales y de investigación existentes. Este hecho permite un aprovechamiento máximo de las tecnologías existentes.

Entre los puertos de servicios Grid básicos o estándares ofrecidos por OGSA encontramos:

- Servicio HandleMap: Proporciona el mapeo entre el manejador de servicio Grid (GSH) y la referencia actual al servico Grid (GSR). Se podría considerar como un DNS para manejadores y referencias, pero desafortunadamente lo que no está definido es el tipo de puerto que se usa

para añadir/eliminar enlaces al/desde el servicio. Esto se puede hacer a través del sistema de notificación.

- Servicio Registry: Permite asociar los metadatos de una instancia de servicio a un registro. El puerto de registro permite además "desregistrar" servicios. Sólo hay un modo recomendado para extraer información del servicio de registro y dicho modo es utilizando el método FindServiceData en el puerto GridService.

- Servicio Factory: Es un servicio que puede ser usado para crear instancias de otros servicios. En las aplicaciones Grid el servicio factory puede crear instancias de servicios de aplicaciones pasajeras. La interacción con este servicio consiste en proporcionarle información del tiempo de vida y un documento XML que describa los datos específicos de la aplicación.

OGSA especifica ocho categorías de servicios, cada uno de ellos esencial para coordinar el trabajo de las aplicaciones que interactúan con recursos disponibles en un entorno compartido seguro. Los usuarios finales necesitan diferentes subconjuntos de estos servicios para satisfacer los requerimientos de sus casos de uso particulares, permitiendo modularidad y flexibilidad en el despliegue.

Las ocho categorías de servicios de alto nivel especificadas por OGSA son:

1. Servicios de infraestructura, que permiten la comunicación entre recursos dispares (ordenadores, recursos de almacenamiento, aplicaciones, etc.), eliminando las barreras asociadas a la utilización compartida de recursos.

2. Servicios de gestión de recursos, hacen posible la monitorización, la reserva, el desplieqgue y la configuración de recursos Grid basándose en los requerimientos de calidad de servicio.

3. Servicios de datos, permiten el movimiento de datos donde sea necesario; gestionan las copias replicadas, la ejecución de peticiones y las actualizaciones, y transforman los datos en nuevos formatos si se requiere.

4. Servicios de contexto, describen los recursos requeridos y las políticas de uso para cada cliente que utilize el Grid, permitiendo la optimización del recurso basada en los requerimientos del servicio.

5. Servicios de información, proporcionan producción y acceso eficiente a la información sobre el Grid y sus recursos, incluyendo el estado y la disponibilidad de un recurso concreto.

6. Servicios de auto-gestión, tratan de automatizar lo máximo posible la gestión del sistema para reducir costes y complejidad.

7. Servicios de seguridad, hacen cumplir las políticas de seguridad dentro de organizaciones virtuales, promoviendo la compartición segura de recursos y la apropiada autenticación y autorización de usuarios.

8. Servicios de gestión de ejecución, permiten que tanto acciones simples como complejas de un workflow sean ejecutadas, incluyendo ubicación, previsiones y gestión del ciclo de vida de las tareas.

Open Grid Services Infrastructure (OGSI)

OGSI es la implementación de la infraestructura que define OGSA. Es lo que se llama el ″middleware″ o la plataforma para ejecutar servicios Grid. OGSI no determina que middleware concreto debe usarse y existen implementaciones OGSI en java, .Net, etc. Se puede utilizar tecnología Java para crear y gestionar servicios Grid e intercambiar información entre ellos de acuerdo a OGSI.

Además esta infraestructura explota un componente importante del framework de los servicios web: el uso de WSDL para describir servicios web a partir de enlaces con protocolos múltiples, codificar estilos, seleccionar el estilo de los mensajes (RPC versus orientado a documento), etc.

La definición que da el Global Grid Forum (GGF) de OGSI es la siguiente:

″ OGSI define mecanismos para crear, gestionar e intercambiar información entre entidades llamadas servicios grid utilizando tecnología Grid y de web services. Sucintamente, un servicio grid es un web service que se ajusta a una serie de convenciones (interfaces y comportamientos) que definen como los clientes interaccionan con el servicio grid. Estas convenciones y otros mecanismos OGSI asociados con la creación y localización de servicios grid permiten una gestión adecuada (control, fiabilidad , seguridad) de la información de estado distribuido y de larga duración que se requiere en la ejecución de aplicaciones distribuidas ″

OGSI define un modelo de componentes que extiende la definición del esquema WSDL y XML con el fin de incorporar los siguientes conceptos:

- Servicios web dinámicos o con estado.
- Extensión de las interfaces de los servicios web.
- Notificación asíncrona de cambio de estado.
- Referencias a instancias de servicios web.
- Colecciones de instancias de servicios.
- Datos de estado del servicio que aumenten las limitaciones de la definición del esquema XML.

4

WINGS

El término WINGS (Workflow In Next Generation Grids) define un sistema Workflow multigrid diseñado para hacer más sencilla la creación de aplicaciones Grid [3]. Está compuesto por dos elementos: el lenguaje de descripción (WINGS) y el motor de ejecución (WINGS Run Time). Ambos elementos se describen en las secciones siguientes, pero para tener una primera visión podemos decir que el lenguaje WINGS está basado en XML y utiliza conceptos muy similares a los usados en AGWL[1] [20], pero añade nueva funcionalidad y no permite la creación de definiciones para una aplicación o entorno concreto. De este modo, los principales objetivos del lenguaje WINGS se basan en proporcionar la capacidad de evolucionar fácilmente gracias a los desarrolladores, añadiendo nuevos elementos con nueva funcionalidad para adaptarse a los middlewares actuales y futuros. Por otro lado, el motor de ejecución de WINGS se encarga de proporcionar la funcionalidad definida a través del lenguaje XML, creando así un entorno concurrente para el lanzamiento de trabajos.

[1] Abstract Grid Workflow Language

4.1 Especificación del lenguaje

El lenguaje de modelado se llama WINGS y ha sido definido utilizando el lenguaje XML y todas las ventajas que éste proporciona (documentos estructurados, fácil extensibilidad, etc.).
Este lenguaje se basa principalmente en cuatro conceptos para modelar un workflow: fuentes de datos, operaciones, actividades y ejecuciones. Estos conceptos permiten representar las tareas y los datos del workflow, pero se necesitan otros elementos para completar la funcionalidad del sistema en un entorno grid: autorizaciones, estructuras de control y recursos. Todos estos componentes serán descritos en los siguientes puntos.

4.1.1 dataGroup

Las fuentes de datos son una abstracción que se utiliza para manejar cualquier información que será parámetro de entrada o salida de la ejecución. Además se consideran los puntos de comunicación entre las diferentes ejecuciones del workflow, ya que son los elementos donde las tareas guardan los datos que se utilizarán en las siguientes ejecuciones.
Los *dataGroup* representan, por tanto, los datos de entrada y salida de todas las ejecuciones, y los datos iniciales y finales del workflow completo.

Existen cinco tipos estándar para representar un *dataGroup*: Boolean para datos booleanos, Integer para enteros, Float para flotantes, String para cádenas y File para ficheros. Los data groups que representan datos de entrada pueden definirse utilizando un conjunto de valores, un rango numérico (valor inicial, valor final y salto) o un rango de ficheros usando metacarácteres/carácteres comodín. Un ejemplo de esto sería el siguiente:

Input dataGroup de tipo Integer

```
<dataGroup name="input1" type="Integer">
         <value value="1"/>
        <value value="2"/>
        <value value="3"/>
</dataGroup>
```

Input dataGroup de tipo File

```
<dataGroup name="input2" type="File">
        <filerange path="/home/Workflow" file="*.dat"/>
</dataGroup>
```

Con el objetivo de poder recoger los datos de salida de una tarea, los dataGroups permiten: filtrado de resultados, renombrado de ficheros e incluso la creación de réplicas de éstos en alguna localización física.

Output dataGroup de tipo File

```
<dataGroup name="datosfin" type="File">
        <filter in="*dinamreor*.img">
                <replica path="D:/results"/>
        </filter>
</dataGroup>
```

Este ejemplo muestra un dataGroup llamado *datosfin* utilizado para obtener los resultados de la tarea. En este caso los ficheros resultado se filtran utilizando la expresión "*dinamreor*.img" y los ficheros que cumplan este filtro serán replicados en el directorio especificado.
La réplica se realiza en un directorio local, pero las fuentes de datos pueden además utilizarse para considerar sistemas de almacenamiento de datos efectivos como puede ser FTP, servidores GSI-FTP, un sistema de ficheros fisíco o cualquier otro dataGroup que pueda ser expresado en forma de URI.

Input dataGroup de tipo File

```
<dataGroup name="basal" type="File">
        <filerange name="basal" source="gsiftp://server.es" path="/datos" file="*.tgz"/>
</dataGroup>
```

En este segundo ejemplo se muestra un parametro de entrada del workflow cuya información se obtiene a través de un servidor GridFTP utilizando un rango de ficheros (filerange).

4.1.2 Operation

Las operaciones pueden verse como tareas simples ejecutadas por el runtime, cuyo consumo de recursos es bajo para evitar sobrecargar el sistema. Típicamente representan tareas simples para el pre o post proceso de los datos utilizados por las tareas del workflow. Este concepto pretende impedir que los usuarios tengan que crear tareas artificiales para realizar este tipo de trabajos simples.
Se han desarrollado un conjunto de operaciones para extraer los campos de un fichero de texto, para encontrar un grupo de ficheros que satisfagan una cierta expresión regular, para realizar operaciones aritméticas y matemáticas de reducción, etc.

Un ejemplo de operación podría ser el siguiente:

Operation

```
<operation name="op1">
        <input level="1" source="inoper"/>
        <output name="out2" destination="outoper"/>
        <getDataFields>
```

```
                <fields separator=" ">
                        <field number="1"/>
                        <field number="4"/>
                </fields>
        </getDataFields>
</operation>
```

Este fragmento de código muestra la operación *getDataFields*, que permite al usuario extraer los campos de un fichero de texto, especificando un carácter de separación y el número de campos y/o líneas del mismo. Concretamente, en este caso se leen los ficheros de entrada y se obtienen los campos 1 y 4 de cada línea de los ficheros, utilizando el espacio en blanco como carácter de separación.

4.1.3 Activity

Las actividades son abstracciones de las tareas Grid encargadas de definir el interfaz y la funcionalidad a ser ejecutada. Pueden verse como un equivalente a las funciones en los lenguajes de programación procedural. Una actividad está definida por los parámetros de entrada y salida (el interfaz) y por los datos específicos para ejecutarla en los diferentes despliegues Grid disponibles.
El siguiente fragmento de código XML muestra un esquema de una actividad.

Activity

```
<activity name="ActivityName">
        <input name="in1" type="String"/>
        <input name="in2" type="File" granularity="2"/>
        <output name="result" type="File"/>
        <deployments>
                ...  ...  ...
        </deployments>
</activity>
```

Para la actividad del ejemplo llamada *ActivityName* se definen dos parámetros de entrada (in1, in2), uno de tipo String y otro de tipo File, y un parámetro de salida de tipo File. Este es un caso típico de tarea Grid. Si nos fijamos en el código del ejemplo, podemos ver que uno de los parámetros de entrada, en concreto in2, tiene asignado un atributo particular llamado *granularidad*, utilizado para definir el número de elementos que necesita dicho parámetro para completar una ejecución. La sección de *deployments* muestra una lista de los diferentes despliegues donde la actividad puede ser ejecutada.

Deployments

La etiqueta deployments representa abstracciones de los diferentes entornos de ejecución donde pueden lanzarse las tareas. Cada deployment describirá los detalles necesarios para lanzar una actividad utilizando el middleware especificado.
Este esquema permite definir las actividades independientemente del middleware Grid utilizado finalmente para lanzarlas. Además se debe especificar la clase que se encargará de lanzar el trabajo en el middleware correspondiente. El concepto de deployment o despliegue ha sido concebido para representar cualquier entorno de ejecución como puede ser un middleware Grid, un servicio web, un subworkflow, un sistema batch de clusters (PBS, SGE, LSF, etc), una conexión ssh a una máquina del estilo Unix, etc. Actualmente se han desarrollado, en el marco del proyecto donde se incluye esta tesis, tres plug-ins de ejecución: para middleware Fura, para middleware Globus y para Subworkflows. Solo vamos a centrarnos en el que concierne a Globus, porque los otros dos no son objeto de la tesis.

- **Deployment Globus:** Como ya hemos visto en el capítulo anterior, Globus es el estándar de facto de los middlewares Grid y una ejecución en Globus se define especificando el fichero ejecutable, y cualquier otro fichero necesario para la ejecución.

Globus Deployment

```
<globusDeployment class="wings.core.globus.GlobusActivity">
        <executable file="gsiftp://server.es:2811//home/data/reggrid.sh"/>
        <file file="gsiftp://server.es:2811//home/data/reggrid.ex"/>
</globusDeployment>
```

  En este ejemplo la tarea a ejecutar estaría representada por el script "reggrid.sh" el cual necesita el fichero "reggrid.ex" para completar su ejecución.

4.1.4 Execution

Las ejecuciones son los trabajos efectivos lanzados en el Grid; instancias de actividades finalmente lanzadas en los recursos Grid. Una ejecución tiene la misma interfaz que la actividad a la que referencia, con la diferencia de que en el caso de la ejecución se especifican las fuentes de datos para obtener/fijar los datos para cada parámetro. Adicionalmente un nodo de ejecución puede

especificar ciertos datos concretos del middleware para utilizarlos en la ejecución.

Execution

```
<execution name="ExecutionName" activity="ActivityName">
        <input name="param1" value="5"/>
        <input name="param2" source="dg1"/>
        <input name="param3" source="dg2"/>
        <output name="salida1" destination="dg3"/>
        <deploymentData type="Globus">
                ...  ...  ...
        </deploymentData>
</execution>
```

La ejecución del ejemplo es una instancia de la actividad definida en la sección anterior. Se fijan las fuentes para obtener los datos necesarios de los parámetros de entrada, y el destino de los datos que representan los parámetros de salida. Los parámetros de entrada pueden fijarse directamente con un valor constante, como en el caso del parámetro "param1" o puede utilizarse un dataGroup para definirlos (param2, param3).

4.1.5 Authorisation

Como ya hemos mencionado en varias ocasiones, en los entornos Grid la seguridad es un asunto importante a tener en cuenta. El sistema Workflow debe permitir la gestión de la identidad o identidades del usuario para que este pueda acceder a los diferentes recursos Grid disponibles. Para hacer posible el acceso a diferentes tipos de middleware, el sistema debe ser capaz de hacer frente a diferentes tipos de estrategias de autorización.
Actualmente han sido desarrolladas las estrategias de autorización más comunes: certificados X509, ficheros proxy de Globus y la técnica usuario-contraseña (User - Password).
Los datos de autorización pueden ser especificados a nivel de workflow, pero pueden ser particularizados en cada ejecución o en el acceso a las fuentes de datos. Permiten utilizar diferentes credenciales para acceder a diferentes recursos Grid. En un entorno multigrid el mismo usuario podría tener diferentes credenciales para autenticarse frente a los middlewares disponibles. Para proporcionar autorizaciones multigrid, el lenguaje WINGS permite fijar información de autorización especifíca para cada despliegue. En el siguiente ejemplo se usa la estrategia de usuario-contraseña para acceder a un middleware Fura y un fichero proxy para el caso de los recursos Globus.

Authorisation

```
<authorisation>
        <deployment type="Fura">
                <userpass user="micafer" password="fdsoiji"/>
        </deployment>
        <deployment type="Globus">
                <proxy proxyfile="/tmp/x509_1000"/>
        </deployment>
</authorisation>
```

4.1.6 Control Structures

A pesar de que el lenguaje WINGS está orientado a flujo de datos, permite ciertas funciones de control implícitas.
El filtrado de ficheros en los dataGroups permite separar los ficheros en diferentes dataGroups y crear ejecuciones bifurcadas o ramificadas. Además el uso de algunas operaciones (como "matches") hace posible la selección de un conjunto de ficheros que cumplan una cierta expresión regular creando así diferentes hilos de ejecución.

Para completar el lenguaje WINGS se añadieron estructuras de control más potentes:

- **If:** Permite crear bifurcaciones o ramificaciones en la ejecución del workflow. Utiliza una ejecución o una operación como condición para decidir qué bifurcación tiene que ejecutar. La estructura If actúa como un disparador (trigger); cuando la tarea que se usa como condición está lista para ser lanzada (tiene datos a ser procesados) ésta se ejecuta y las tareas en la parte "then" o "else" serán seleccionadas para ser ejecutadas en función de si se cumple la condición establecida.
- **Iterator:** El lenguaje WINGS realiza implícitamente la iteración de datos, todos los datos disponibles en un dataGroup utilizado como fuente de datos de una ejecución serán procesados. El iterador permite evaluar una condición para cancelar el procesamiento de todos los datos. Dentro del bloque iterativo se especifican ciertos dataGroups utilizados en las tareas y un paso de iteración que indica cuando será evaluada la condición.

Control Structures

```
<iterator name="iter1">
```

```
            <input name="paramiter1" source="dg4" iterationStep="2"/>
            <condition value="false">
                    <operation name="op3">
                            <input level="1" source="outiter"/>
                            <output name="out2" destination="outoperiter"/>
                            <matches regexp=".*OK.*"/>
                    </operation>
            </condition>
            <iteratedBlock>
                    <execution name="execution6" activity="activity5">
                            <input level="1" name="parametro1" source="dg4"/>
                            <output name="salida1" destination="outiter"/>
                    </execution>
            </iteratedBlock>
    </iterator>
```

En este ejemplo cada vez que 2 elementos del dataGroup ”dg4” son procesados por ”execution6” la condición ”op3” es evaluada para comprobar si la ejecución debe continuar procesando los datos disponibles o por el contrario, tendrá que ser cancelada.

4.1.7 Resources

En este punto nos referimos a la lista de recursos donde finalmente se ejecutarán las tareas. Al estar en un entorno workflow multigrid podemos encontrarnos con diferentes tipos de recursos que necesitan datos particulares para especificar un recurso Grid.
En WINGS un recurso no es un recurso Grid ”estándar”, considerado un computador conectado al grid, es un front-end para acceder a una cierta infraestructura Grid. Puede ser un servidor Fura[21], un nodo Globus, un servidor gLite WMS[22], un metaplanificador GridWay[23], un servicio GMarte[24], etc.

Globus Resource

```
<resources>
        <globusResource>
                <gram server="ramses.dsic.upv.es" port="2119" jobmanager="jobmanager-fork"
                        gsiftpport="2811"/>
        </globusResource>
</resources>
```

En este ejemplo definimos un recurso Globus donde se ejecutarán las tareas, en este caso es un servidor GRAM donde se lanzarán todas las peticiones generadas por el Workflow.

4.2 WINGS-RT

La especificación de WINGS considera sólo un lenguaje de flujo de datos puro donde los workflows se definen por medio de secuencias de dataSource - Ejecución o Operación - dataSource. Dos ejecuciones o operaciones no pueden estar conectadas directamente. Este esquema simplifica tanto la definición como el entendimiento del workflow e incrementa la expresividad.

Para complementar el lenguaje WINGS, existe un motor de ejecución workflow llamado WINGS-RT (WINGS Run Time) encargado de procesar el modelo de workflow definido con WINGS para posteriormente lanzar, monitorizar y coordinar la ejecución de las tareas en el Grid. El WINGS-RT aplica un modelo de flujo de datos donde las tareas pueden ser lanzadas cuando todos los parámetros de entrada tienen al menos un dato que procesar, consiguiendo así lanzar las diferentes intancias de la tarea en paralelo. Este modelo aplica paralelismo implícito; en definitiva todas las tareas que no tienen dependencias de datos son lanzadas al mismo tiempo.

Tanto la definición del lenguaje como el motor de ejecución tienen en cuenta los objetivos principales del sistema: capacidades multigrid y fácil extensibilidad.

Un componente dentro del motor de ejecución es el "dispatcher", cuya misión es seleccionar el mejor recurso para cada ejecución de entre los disponibles, deacuerdo con los despliegues proporcionados en la actividad que define la ejecución.

Y otro punto fuerte del motor de ejecución es el hecho de que realiza todas las combinaciones posibles entre los datos de entrada, creando lo que hemos llamado **Microtareas**. Es decir, dentro de una actividad se lanzan tantas microtareas como combinaciones sean posibles teniendo en cuenta la granularidad de cada parámetro (será 1 si no se especifica nada).

Si por ejemplo tenemos tres ficheros y una lista de tres valores enteros como parámetros de entrada, en el sistema se lanzarán nueve microtareas (fichero1 - valor1, fichero1 - valor2, fichero1 - valor3, fichero2 - valor1, fichero2 - valor2, fichero2 - valor3, fichero3 - valor1, fichero3 - valor2, fichero3 - valor3).

4.2.1 Dispatcher

Actualmente existen diferentes herramientas para la planificación de trabajos en una infraestructura Grid como por ejemplo: gLite WMS, GridWay, GMarte, Condor[25], Fura Server etc. El dispatcher de WINGS no intenta reemplazarlas; trabaja en una capa superior seleccionando la mejor infraestructura para lanzar las tareas del workflow.

Este componente se encarga de seleccionar el mejor recurso disponible de la lista de recursos especificada en la definición del workflow. Tal como se ha descrito en secciones anteriores, los recursos en WINGS no se entienden como recursos Grid estándares, son puntos de acceso a diferentes middlewares Grid. A pesar de que se pueden especificar recursos grid estándares en la lista de recursos, se concibe para contener una lista de diferentes nodos para acceder a un planificador Grid para una cierta infraestructura Grid.
El *Dispatcher* utiliza un simple algoritmo para seleccionar la mejor infraestructura Grid para cada tarea, eligiendo los recursos que son compatibles con la lista de despliegues especificada por la tarea. Para ello es importante obtener información sobre las infraestructuras disponibles, y esto se consigue utilizando el plugin de sistemas de información que se describirá en la siguiente sección.
Un desarrollador puede añadir al sistema una clase java que proporcione información sobre una infraestructura concreta. Si no existe información sobre una infraestructura, ésta se marca como última opción. Si no hay información disponible para ningún despliegue el despachador selecciona los recursos aleatoriamente.

4.3 Arquitectura de WINGS

El sistema WINGS ha sido diseñado teniendo en mente el principal objetivo del middleware: fácil extensibilidad. WINGS presenta una arquitectura modular donde los nuevos elementos pueden ser añadidos por el usuario sin necesidad de modificar otras partes del sistema. La estructura del motor de ejecución (execution engine) está diseñada para permitir que el sistema pueda ampliarse añadiendo plugins.

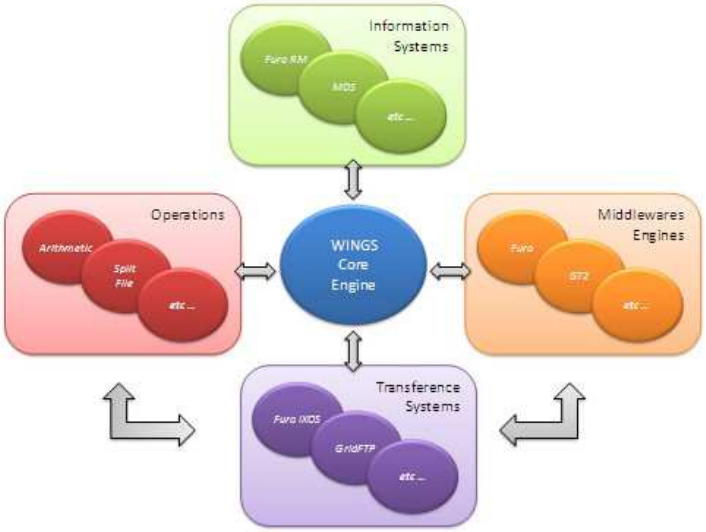

Figura 4.1: Arquitectura de WINGS

Como podemos ver en la figura anterior, el componente central de la arquitectura de WINGS es el *Core Engine*, encargado de la coordinación de todos los elementos y plugins del sistema. La funcionalidad de este elemento consiste en:

- Leer el fichero XML con la especificación del workflow.
- Lanzar y coordinar la ejecución de las diferentes tareas Grid utilizando los plugins del middleware Grid apropiado.
- Realizar las transferencias entre los recursos Grid utilizando los módulos de sistema de transferencia necesarios.
- Lanzar las operaciones requeridas.

La arquitectura de WINGS posee un subsistema modular formado por cuatro compontes que proporcionan la funcionalidad final para realizar toda la ejecución. Todos los componentes de este subsistema utilizan un diseño de plugin a través del cual los usuarios pueden añadir nuevos componentes y utilizarlos en el sistema completo.

1. **Middleware Engines:** Estos plugins hacen posible el acceso a los diferentes middlewares Grid disponibles en el sistema WINGS. Este motor se ocupa de los detalles específicos del middleware (envío de trabajos, obtener el estado de las tareas, etc.), de manera transparente al *Core Engine*.

2. **Transference System:** Cuya función es la de realizar las transferencias entre los recursos Grid. Cada elemento posibilita la transferencia utilizando un conjunto de protocolos. El motor de ejecución es el encargado de seleccionar los elementos necesarios para llevar a cabo todos los movimientos de datos entre los cómputos o las réplicas de ficheros.

3. **Operations:** El sistema WINGS proporciona un conjunto de operaciones para el pre o post proceso de la información utilizada en las tareas Grid.

4. **Information Systems:** Conjunto de plugins para obtener información sobre las infraestructuras de ejecución con el objetivo de seleccionar el mejor recurso donde lanzar las tareas.

5

Plug-in para Globus

En este punto queremos centrarnos en la definición del trabajo que se ha realizado como tesis de máster. Nuestro objetivo ha sido la creación de un conjunto de plug-ins que pudieran ser utilizados por el WINGS-RT para el lanzamiento de tareas en entornos Grid.

Para ello nos hemos basado en todos los conceptos descritos hasta el momento: workflow, WINGS, WINGS-RT y Globus, principalmente.
Basándonos en la arquitectura modular que se ha descrito en puntos anteriores, hemos implementado por un lado un plug-in de ejecución para incorporarlo en el componente *Middleware Engines*, cuyo misión será gestionar el lanzamiento y la ejecución de trabajos Grid utilizando Globus Toolkit. Por otro lado, se ha desarrollado un plug-in de transferencia que como su nombre indica se encargará de las transferencias de datos entre los diferentes recursos Grid y que se integrará en la parte de *Transference Systems*.

Este trabajo forma parte de un proyecto en el cual ya se ha había desarrollado un lenguaje de definición de Workflows, WINGS, que como hemos visto anteriormente permite la ampliación del sistema por parte del usuario gracias a la utilización de plugins. Nuestra finalidad, por tanto, es la creación de un plug-in que permita el lanzamiento de trabajos Grid a través de un sistema workflow. De esta manera se aprovechan las ventajas tanto de los sistemas Workflow como de los entornos Grid, y se consigue el lanzamiento automatizado de trabajos Grid. Y adicionalmente se necesita adaptar el sistema de transferencias utilizado en el workflow para que soporte los protocolos de transferencia con los que queremos trabajar.

Para poder integrar los plug-ins en el sistema, se han tenido que añadir nuevas reglas al lenguaje WINGS con el objetivo de permitir la definición

de todos los elementos necesarios para ejecutar una tarea con un middleware Grid, en este caso Globus, y para poder soportar ciertos protocolos de transferencia. Dado que el motor de ejecución de WINGS permite la ejecución de tareas utilizando diferentes middlewares, cada vez que se cree un plugin para uno de ellos hay que definir todo lo necesario para que el run time lo reconozca.

5.1 Plug-in de Transferencia

El plug-in de transfererencia, tal como hemos comentado en anteriores ocasiones, se encarga de las transferencias de datos entre los diferentes recursos Grid que participan en la ejecución. El motor de ejecución es el encargado de recopilar todos los datos que se tienen que transmitir y utiliza el sistema de transferencia para que haga efectiva dicha transmisión.
Con la creación de este plug-in se pretende dar soporte a los protocolos de transferencia más utilizados en entornos Grid, como por ejemplo GSIFTP, consiguiendo así que se puedan ejecutar tareas utilizando diferentes middlewares Grid.

Para llevar a cabo las trasnferencias de datos requeridas en la ejecución, nuestro plug-in utiliza dos clases proporcionadas por Java CoG Kit:

- **GridFTPClient:** Esta clase utiliza el protocolo GridFTP de Globus para realizar operaciones de transferencia de datos entre un cliente y un servidor. El protocolo GridFTP es una extensión del protocolo FTP estándar adaptada a entornos Grid. La creación de los directorios remotos y la operación de cambio de permisos sobre ficheros remotos, se lleva a cabo creando un cliente GridFTP que se conecta remotamente a la máquina destino (server) que le indiquemos, utilizando un proxy válido para autenticarse.

- **UrlCopy:** Esta clase implementa la función globus-url-copy de Globus la cual permite realizar movimientos de datos entre diferentes protocolos. Soporta los siguientes protocolos: gsiftp:// (GridFTP), ftp://, http://, https://, y file:///, especificados en las URLs que utiliza para definir el origen y el destino de la trasferencia de datos. Una de las características más interesantes de esta clase es la capacidad de realizar transferencias *third party*. El concepto de *third party transfers* consiste realmente en mover dos ficheros entre dos servidores GridFTP, ya que UrlCopy acaba utilizando el protocolo GridFTP a más bajo nivel, de

una manera transparente al usuario. En este tipo de transferencias hay tres entidades involucradas: el cliente que inicia la transferencia pero que no toma partido en ella, y dos servidores, uno de los cuales mandará datos al otro. Este escenario es muy común en las aplicaciones Grid y tiene lugar cuando el cliente GridFTP que inicia la transferencia no es ni la fuente ni el destino de la transferencia. El cliente realiza un movimiento de datos entre dos máquinas remotas actuando él como intermediario.

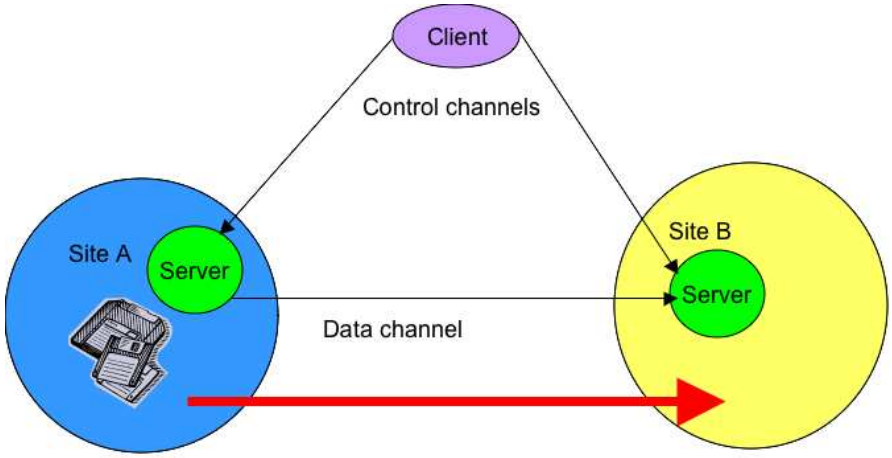

Figura 5.1: Transferencia third party

Cada vez que el motor de ejecución llama al sistema de transferencia para transferir datos entre recursos, el plug-in de transferencia se encarga de comprobar si tanto el protocolo de origen como el de destino se corresponden con protocolos soportados en el sistema. En caso de ser así, también gestiona los proxies necesarios para la comunicación entre los dos recursos. Debido a que se utiliza el modo de transferencia third party, se requiere un proxy para conectarnos a la máquina origen y otro para la máquina destino. Dichos proxies pueden ser el mismo, en el caso de que se utilicen las mismas credenciales para acceder a los recursos Grid, y serán distintos en el caso contrario.

Con toda esta información, y conociendo además la localización de los datos a transferir y el destino donde tenemos que copiarlos, el sistema de transferencia ya puede mover la información de la máquina origen al destino utilizando los protocolos que corresponda de entre los soportados.

5.2 Plug-in de Ejecución

El plug-in de ejecución por su parte se encarga del lanzamiento y ejecución de los trabajos Grid que componen el workflow.

Como ya sabemos, para poder lanzar tareas en el sistema tenemos que definir un archivo XML donde se especifique todas las características y elementos de la ejecución/ejecuciones. Por ello, cuando se trata de tareas Grid, tenemos que indicar en la sección de "deployments" que se trata de un middleware Grid:

Globus Deployment

```
<deployments>
        <globusDeployment class="wings.core.globus.GlobusActivity">
            <executable file="gsiftp://odin.itaca.upv.es:2811//Workflow/prueba1.sh"/>
        </globusDeployment>
</deployments>
```

Y le indicamos también, en la sección de "resources", el recurso donde se van a lanzar las peticiones:

Globus Resource

```
<resources>
        <globusResource>
                <gram server="ngiesce.itaca.upv.es" port="2119"
                                jobmanager="jobmanager-fork" gsiftpport="2811"/>
         </globusResource>
</resources>
```

Otro asunto a tener en cuenta es el de la autorización. Como Globus trabaja con proxies tenemos que incluir en el fichero XML la información requerida para la creación de un proxy válido que nos permita ejecutar en un entorno Grid.

En los siguientes ejemplos podemos ver las dos maneras posibles de especificar la información:

Ejemplo 1

```
<authorisation>
        <deployment type="Globus">
                <certkey certfile="/.globus/usercert.pem" keyfile="/.globus/userkey.pem"
```

```
                                keypasword="12345"/>
        </deployment>
</authorisation>
```

En este primer ejemplo especificamos la ubicación de dos ficheros junto con un password. Estos tres elementos se corresponden con el certificado de la clave pública, la clave privada y el password que se utilizarán para crear un proxy válido a través del cual lanzar en el Grid.

Ejemplo 2

```
<authorisation>
        <deployment type="Globus">
                <proxy proxyfile="/tmp/x509_1000"/>
        </deployment>
</authorisation>
```

En este segundo ejemplo se proporciona un fichero con toda la información del proxy a utilizar. La información de este fichero se utiliza para generar el proxy.

Para describir la implementación realizada, vamos a centrarnos en los pasos a seguir para lanzar una tarea en el Workflow:

1. Preparar la actividad.
2. Preparar los parámetros de entrada.
3. Ejecutar la tarea.
4. Recoger los parámetros de salida.

La implementación del plug-in ha consistido básicamente en la creación de un conjunto de funciones que realizan las operaciones necesarias para poder lanzar un trabajo en el Workflow utilizando Globus Toolkit. En las siguientes secciones describiremos detalladamente cada uno de los pasos enumerados más arriba y los problemas o dificultades que han supuesto. Hay que destacar que el motor de ejecución del workflow llama al sistema de transferencia, donde hemos incluido el plug-in descrito en el punto anterior, cada vez que tiene que realizar una transferencia de datos.

5.2.1 Función prepareActivity

La implementación de esta función implica un serie de pasos básicos a realizar con el objetivo de preparar la actividad para su futura ejecución. El primer paso consiste en la creación de un proxy válido a través del cual poder ejecutar en un entorno Grid. Dicho proxy se crea a partir de los datos de autorización especificados en el archivo XML (etiqueta <authorisation>) y almacenados en una clase Java definida para ello.

Para cada ejecución que se lanza, el sistema gestiona un "almacén de proxies". Dicho almacén no es más que una estructura de datos donde se depositan los proxies que se van generando para cada una de las actividades. Inicialmemte el almacén está vacío y cada vez que se lanza una tarea o actividad se comprueba si los datos del proxy ya están en esta tabla (el proxy ya ha sido generado anteriormente) o si por el contrario hay que generar un proxy nuevo y almacenar los datos para su uso futuro. La información que obtenemos como resultado de esta consulta es un proxy en forma de credencial representado con la clase GSSCredential de Java. Este objeto es de vital importancia a lo largo de toda la ejecución ya que se utiliza además para lanzar los trabajos en el Grid, para cualquier tipo de operación remota realizada a través del protocolo GridFTP. En nuestro caso se requiere la utilización de un proxy válido para la creación de directorios remotos (en la máquina de ejecución), la transferencia de ficheros y para ciertas operaciones sobre datos ubicados en una máquina remota (listar contenido de un directorio, cambio de nombre o de permisos de un fichero, etc).

Una vez hemos obtenido el proxy, el siguiente paso será definir el entorno de ejecución de la tarea, que en este caso es un entorno Globus. Para ello, tal como se ha mencionado en la definición del lenguaje WINGS, se utiliza el concepto de despliegue. El plug-in que hemos creado proporciona una clase específica para representar dicho concepto, la clase *GlobusDeployment*.
Los datos referentes a la ejecución de la actividad, que se definen en el archivo XML (etiqueta <globusDeployment>), se mapean a los campos correspondientes de la clase GlobusDeployment. Estos campos se corresponden básicamente con el fichero ejecutable y ficheros adicionales que puedan requerirse para complementar al anterior, cuando corresponda. Gracias a esta clase el motor de ejecución del workflow sabe que se está lanzando una actividad en un entorno Grid, y realiza las acciones oportunas.

Una vez preparados los datos referentes a la ejecución, lo que nos toca ahora es preparar la máquina donde se van a lanzar las tareas. Para ello crearemos los directorios necesarios para almacenar todos los datos de la ejecución. En la función *prepareActivity()* nos centramos en la creación de un

directorio de trabajo para cada tarea, dentro del cual se irá construyendo la estructura de directorios necesaria para almacenar todos los datos. Teniendo en cuenta que se pueden lanzar varias actividades a la vez, debido al paralelismo implícito del sistema, al nombre de todos los directorios que vayamos creando añadiremos una etiqueta con su hora de creación. Se pretende evitar un posible solapamiento de los datos.
Dentro del directorio de trabajo mencionado creamos ahora un directorio de ejecución donde copiaremos el fichero ejecutable y los ficheros adicionales indicados en el fichero XML. Hay que asegurarse de que los ficheros ejecutables que copiemos en la máquina destino tengan permiso de ejecución, así que una vez allí se les fija dicho permiso.
Tras el proceso de copia del ejecutable a la máquina destino es importante almacenar la ubicación de éste para poder referenciarlo posteriormente. Dicha copia se realiza a través del plug-in de transferencia.
Generalmente las ejecuciones se lanzan a una máquina remota, lo cual significa que hay que hacer todas las operaciones descritas en el último paso de manera remota. Es decir, se realiza una creación remota de directorios, una copia remota y una operación de cambio de permisos también de manera remota.

5.2.2 Función prepareInitialData

En la seccción anterior nos hemos centrado en los parámetros de la ejecución y en los datos directamente relacionados con ella. Aquí sin embargo nos ocupamos de la entrada de la ejecución. En referencia a la entrada, lo que se hace en esta función es crear un directorio para almacenar los datos de entrada de la ejecución, directorio de inputs, que se creará dentro del directorio de trabajo mencionado en el punto anterior.
En el directorio de inputs se copiarán todos datos de los dataGroups definidos en el fichero XML como parámetros de entrada de la actividad (etiqueta <input>) que sean ficheros.
En la definición de los parámetros de entrada se especifica la ubicación de dichos datos, por tanto, nuestra función realiza la copia de los datos de la máquina origen (donde se ubican los datos de entrada) a la máquina destino (donde se va a ejecutar la actividad). La función de copia definida en el plug-in de transferencia (copyFiles) se encarga de comprobar los protocolos utilizados en la transferencia permitiendo las siguientes combinaciones: de gsiftp a gsiftp, de gsiftp a file y de file a gsiftp. La opción file - file no se contempla porque se trataría de una transferencia local. En nuestro caso dado que se trata de un plug-in que opera bajo Globus, el protocolo de destino de las transferencias será siempre GSIFTP, sin embargo el protocolo de origen

puede ser cualquier otro protocolo soportado por el sistema.

Dicha función también gestiona el proxy o proxies necesarios para la operación; determina si el proxy origen será igual al destino, en caso afirmativo consulta el almacén de proxy para comprobar si ya lo tiene en la tabla y si no lo tiene genera un nuevo proxy. En caso de que el proxy no sea el mismo en origen y destino, consulta la tabla en busca de los dos proxies y en caso de no encontrarlos los genera.
Otros factores a tener en cuenta son la granularidad del parámetro de entrada y si hay que aplicar algún filtro a los datos que contiene dicho parámetro. Los dataGroups de tipo *filerange* o rango de ficheros, permiten especificar los ficheros uno a uno, indicando para cada caso la ubicación concreta y su nombre. Otra opción sería definir la ruta completa de acceso a un directorio que contiene los ficheros y fijar un filtro para los ficheros que se tienen que utilizar, y en caso de no indicar filtro se copiarían todos los ficheros del directorio como datos de entrada de la actividad.
Una vez copiados los parámetros correspondientes (ficheros) en la máquina donde vamos a ejecutar, ya somos conscientes del número total de datos que va a utilizar la actividad para la ejecución. Esto nos permite calcular el número de *microtareas* que generará la actividad. Como ya se ha mencionado en otras ocasiones, se realizan todas las posibles combinaciones de los datos de entrada (ficheros, rango de valores, lista de valores, etc.) y cada una de las combinaciones representa una microtarea, y por lo tanto una ejecución de la actividad en cuestión con la combinación de datos que corresponda.

En este punto es importante remarcar el tratamiento que se hace de los datos de entrada en la función que calcula el número de microtreas (getNumMicrotareas). Se recorren todos los dataGroups definidos en el XML como inputs, y se obtienen los datos de cada uno de ellos para almacenarlos en una estructura de datos que no es más que una lista de arrays (array de arrays), donde tenemos un array por cada dataGroup de entrada y el tamaño de dicho array depende del número de datos que contenga el dataGroup. Teniendo los datos agrupados en esta estructura, el cálculo que tenemos que realizar se limita a sumar los elementos que contiene cada dataGroup y realizar el producto.
Para operaciones posteriores es útil tener los datos agrupados para no tener que recorrer cada vez el fichero XML en busca de los datos de entrada.

Lo mejor para entender esta descripción es basarnos en un ejemplo. Imaginemos que en el fichero XML tenemos definidos los siguientes dataGroups como inputs:

```
<dataGroup name="input1" type="File">
        <filerange name="fr1" source="gsiftp://server.es" path="/Workflow" file="*.txt"/>
</dataGroup>

<dataGroup name="input2" type="Integer">
        <range name="fr2" initialValue="1" finalValue="2" step="1"/>
</dataGroup>
```

Supongamos que el primer parámetro de entrada ”input1” contiene tres ficheros que cumplen el filtro indicado (f1.txt, f2.txt, f3.txt) y el segundo parámetro ”input2”como vemos contiene un rango de valores enteros que se compone de dos elementos (1,2). La estructura de datos resultante al almacenar dicha información sería la siguiente: una lista de dos arrays (2 parámetros de entrada) donde el primer array tendría tres elementos correspondientes a los tres ficheros y el segundo array tendría dos elementos que definen el rango de valores enteros. Por lo tanto el número de microtareas en este caso sería seis, como resultado de combinar todos los datos de entrada (f1.txt - 1, f1.txt - 2, f2.txt - 1, f2.txt - 2, f3.txt - 1, f3.txt - 2).

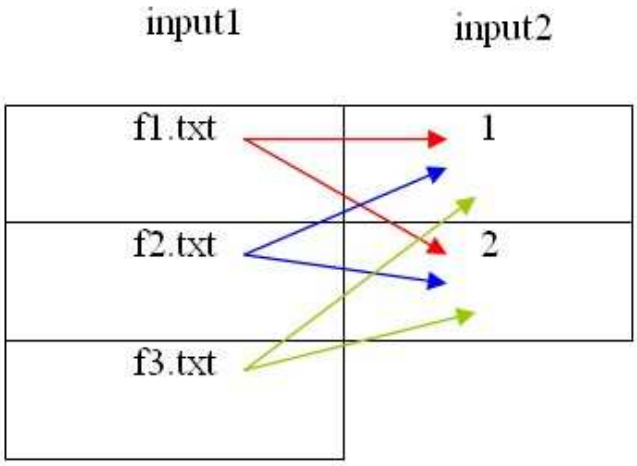

Figura 5.2: Estructura de datos de entrada

Para realizar las combinaciones hay que tener en cuenta la granularidad del parámetro, que es la que nos indica cuantos datos de los que contiene el parámetro de entrada hay que utilizar. En el caso del ejemplo anterior, si el parámetro ”input1” se definiera con granularidad igual a dos, significaría que para cada microtarea que lanzaramos dentro de la actividad tendríamos que coger dos datos del parámetro ”input1”. Como en este caso hay tres ficheros de entrada sólo podríamos hacer combinaciones utilizando dos de ellos, por

lo tanto tendríamos tres microtareas con los siguientes parámetros: f1.txt - f2.txt - 1, f1.txt - f2.txt - 2, f1.txt - f2.txt - 3.
La granularidad será uno mientras no se indique lo contrario en el fichero XML.

Obtener el número de microtareas para una actividad dada nos da también el número de directorios de salida que tenemos que crear, uno por cada microtarea a ejecutar. Los directorios de outputs al igual que los de inputs se crean dentro del directorio de trabajo de la actividad en la máquina de ejecución.
En el caso de que se trate de una actividad que no tiene parámetros de entrada, nos limitaríamos a crear un directorio de salida donde se guardaría el resultado de la ejecución de la actividad. En este caso concreto no habría microtareas. En definitiva el número de microtareas determina los trabajos Grid que se van a lanzar, y que se especificarán en el siguiente punto.

5.2.3 Ejecución de la tarea - execute()

En la parte de ejecución de las tareas nos centraremos en el procedimiento que sigue la clase GRAM de Globus Toolkit a la hora de lanzar trabajos. Como ya sabemos, GRAM es el módulo que permite la ejecución remota de trabajos y la gestión de su estado. Es responsable por tanto de:

- Verificar y procesar los requerimientos del trabajo expresados en el lenguaje de especificación de recursos (Resource Specification Language ó RSL).
- Lanzar los trabajos en los recursos remotos.
- Permitir la monitorización remota de los trabajos creados.

Por lo tanto la única tarea que tenemos que realizar en este punto es preparar los ficheros RSL con los datos de ejecución de cada tarea y lanzar un trabajo por cada RSL que tengamos. Después solo nos quedará monitorizar el estado de cada trabajo a través de un jobmanager para saber cuando ha finalizado y poder recoger los datos de salida.

Resumiendo, por cada trabajo que vayamos a lanzar, GRAM requiere como parámetro de entrada un fichero RSL con toda la información de la ejecución: nombre absoluto del ejecutable, argumentos de ejecución, directorio de trabajo, número de ejecuciones del ejecutable (por defecto 1), entre otros. Para la creación de los RSL hemos implementado una función recursiva que recorre la matriz de parámetros de entrada y va creando una cadena

en lenguaje RSL por cada combinación de datos de entrada. Dicha cadena contiene para cada caso: el ejecutable, los argumentos que correspondan y el directorio de trabajo.

Una vez creados los RSLs, creamos los trabajos GRAM que posteriormente lanzaremos en el GRID. Para la creación de los trabajos utilizamos la clase GramJob de JAVA pasándole como parámetros un proxy válido y el fichero RSL que contiene los datos del trabajo. Esta clase permite la definición de ″listeners″ para monitorizar el estado del trabajo. Por lo tanto, para cada GramJob creado definimos un listener.

Llegados a este punto ya tenemos el trabajo listo para su lanzamiento, utilizamos entonces la función *request* de la clase GramJob para someter cada trabajo al Grid, indicando en la llamada el host remoto donde se ejecutará el trabajo y una variable booleana que representa el modo de sometimiento del trabajo (batch o interactivo). Elegimos el modo *batch* que implica lanzar el trabajo y no permanecer a la espera de los resultados. Esta decisión se debe al hecho de que queremos paralelizar el proceso de lanzamiento de trabajos, eso significa que se lanzan simultáneamente los trabajos que no tienen dependencias de datos entre ellos y se van monitorizando para saber cuando termina cada uno. Si utilizaramos el modo interactivo tendríamos que lanzar un trabajo y esperar el resultado antes de poder lanzar el siguiente, con lo cual no se conseguiría el paralelismo buscado.

En el interfaz de la aplicación el usuario tiene un botón que le permite cancelar todo el proceso, en ese caso el plug-in cancelaría todos los trabajos lanzados hasta el momento y no lanzaría los siguientes, en caso de que quedara alguno por lanzar.

Si la ejecución finaliza con éxito, el plug-in se encarga de recoger los resultados y realizar las operaciones oportunas con ellos, como veremos en el siguiente punto.

5.2.4 Recogida de los resultados - getOutputData()

Como hemos mencionado en el punto anterior, a medida que van terminando los trabajos recogemos sus resultados. Para ello hay que recorrer los directorios de salida creados en la preparación de la actividad, porque es allí donde se almacenan los resultados de cada ejecución.
Al igual que ocurría con los parámetros de entrada de las tareas, las salidas también se almacenan en una estructura de datos para facilitar su posterior

tratamiento. Antes de almacenar los ficheros que representan la salida de la tarea, los renombramos para asegurarnos de que no se sobreescribe ningún dato.
El tratamiento que se hará de los datos de salida depende de lo que el usuario haya especificado en el fichero XML de entrada. Si no se requiere hacer una réplica de los datos en alguna ubicación específica, lo que hacemos es obtener los resultados y almacenar su ubicación en la estructura de datos. En caso de tener que realizar un réplica de la información el usuario especifica en el fichero XML el destino de la réplica, es decir, la máquina donde copiar los datos. Después el motor de ejecución utiliza el plug-in de transferencia para realizar dicha réplica indicándole los datos que tiene que copiar en cada caso.

Tras recoger los datos y hacer las operaciones oportunas, hay que limpiar todos los directorios intermedios y ficheros temporales creados y utilizados durante la ejecución. El proceso de limpieza se realiza al final del workflow, es decir, cuando termina la ejecución de todas las tareas.
Para este propósito, dado que la clase GridFTP no proporciona una función de borrado recursivo de directorios remotos, el plug-in implementa esta funcionalidad.

Una vez terminado el proceso de limpieza de directorios, hemos llegado al final de la ejecución.

6

Caso de Estudio

Para poner a prueba la funcionalidad y eficiencia del plug-in creado hemos elegido una aplicación de biomedicina que implementa la ejecución del proceso de co-registración de imágenes médicas.

La co-registración de imágenes consiste en alinear los pixels volumétricos (voxels) de dos o más imágenes en el mismo espacio geométrico utilizando las transformaciones necesarias para hacer que las imágenes flotantes sean lo más parecidas posible a la imagen de referencia.

Dicho proceso puede ser elástico o rígido. La registración rígida solo utiliza transformaciones afines (desplazamientos, rotación, escalado) en las imágenes flotantes. La registración elástica, por otro lado, permite al usuario aplicar deformaciones elásticas a las imágenes flotantes. La registración de imágenes puede aplicarse tanto en 2D (individual para cada sección) como en 3D.

El sistema workflow utilizado para hacer las pruebas está compuesto por tres etapas:

1. la primera se corresponde con la co-registración rígida,
2. la segunda con la co-registración elástica (el paso con más consumo de CPU),
3. y la última etapa es un proceso que consiste en transponer los N estudios (con K secciones) resultado de la co-registración en K estudios con N secciones.

Esto permite ver la evolución en el tiempo de una de las áreas (secciones) de los estudios.

Los datos de entrada utilizados en los tests son series dinámicas de imágenes 3D de resonancia magnética tras la inyección rápida de un medio de contraste en el área del abdomen para estudiar la perfusión del hígado. Se van analizando dinámicamente las imágenes para ver la difusión del medio de contraste a través del hígado, ya que dicho medio aparece en las imágenes como una superficie brillante perfectamente distinguible. El conjunto de datos está formado por 5 estudios con 12 secciones (104 KB por estudio).

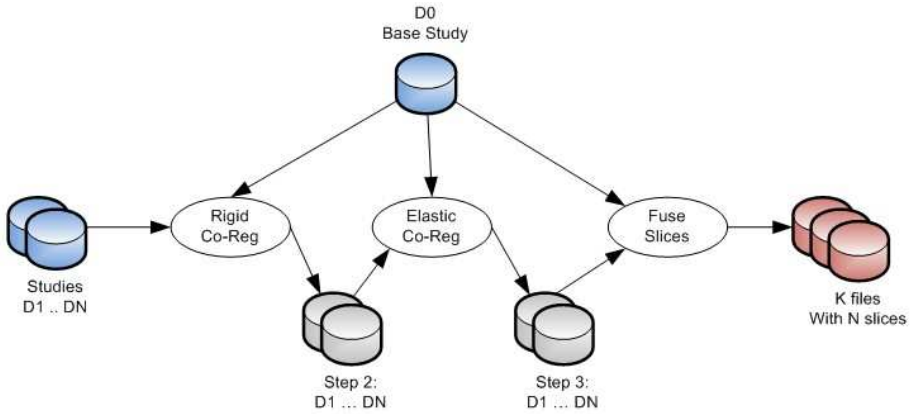

Figura 6.1: Ejemplo de workflow de co-registración

En el caso de uso hemos utilizado la combinación de dos infraestructuras: Globus y Fura.
Los recursos Fura están compuestos por un servidor y dos agentes Linux. El primer agente (F1) es un AMD Opteron 2.4 GHz con 1GB de memoria RAM y el segundo (F2) un AMD Opteron 2.2 GHz con 1GB de RAM.

Por lo que se refiere a Globus, hemos utilizado un cluster con dos nodos. Cada nodo (GN) es un Pentium Xeon 2 GHz con 512 MB de RAM.

Todos los elementos utilizados en los tests están conectados a través de una red Gigabit Ethernet.

Hemos realizado tres pruebas con este ejemplo: la primera utilizando Fura, la segunda con recursos Globus y la última es un test cruzado utilizando middleware Globus y Fura.

Los tiempos obtenidos en la ejecución de las tres pruebas se muestran en la tabla que aparece a continuación. La fase de ejecución (execution phase) muestra el tiempo de las tres etapas del workflow, considerando solamente el tiempo de uso de CPU. En la tabla aparece el tiempo de las ejecuciones junto con la máquina que se ha utilizado para lanzarlas. El tiempo total de la fase de ejecución es la suma de las ejecuciones no paralelas de las tres etapas.

La operación de limpieza o borrado (clean op.) se realiza al final de la ejecución del workflow para borrar los datos intermedios utilizados en las ejecuciones. En el caso de Globus, este paso es bantante largo debido a la operación de borrado recursivo de directorios remotos. Dicha operación no está soportada nativamente por el protocolo GridFTP, por lo tanto hay que realizarla manualmente accediento al directorio en cuestión, listando su contenido y borrando uno a uno los ficheros que contenga.

Finalmente el tiempo de sobrecarga (overhead) incluye todas las operaciones de transferencia de ficheros y la sobrecarga introducida por el middleware y por el sistema workflow. Tal como muestran los resultados el overhead en el caso de Globus es más elevado, debido a que los protocolos son más pesados.

La distribución de las tareas entre los diferentes recursos disponibles se muestra también en la tabla.
En el caso de Fura, es el despachador (Fura dispatcher) el que se encarga de enviar los trabajos a los mejores recursos disponibles, en este caso elegiría primero el nodo F1 y luego el F2.
Globus sin embargo lanza los trabajos a un sistema de colas PBS, y éste es responsable de seleccionar los nodos de ejecución.

En la solución mixta, las cuatro ejecuciones del proceso de co-registración recaen en los nodos Fura. El despachador del workflow elige los recursos Fura porque son más potentes que los de Globus. Como el tiempo de ejecución de la tarea es muy corto, no se necesitan más recursos. En la segunda etapa las dos primeras ejecuciones son lanzadas con el middleware Fura, pero debido a que esta etapa es bastante larga, los recursos Fura ya están llenos y los siguientes trabajos se lanzan con Globus. La última tarea (Fuse slices) se manda también al sistema Fura.

	Execution Phase						
	Rigid CoReg	Elastic CoReg	Fuse Slices	Total	Clean Op.	Overhead	Total
Fura	2"(F1) 2"(F2) 2"(F1) 2"(F2)	40'16"(F1) 43'43"(F2) 40'05"(F1) 41'23"(F2)	1"(F1)	84'6"	4"	55"	85'5"
Globus	4"(GN) 4"(GN) 4"(GN) 4"(GN)	50'23"(GN) 50'26"(GN) 51'39"(GN) 50'24"(GN)	5"(GN)	102'14"	1'28"	3'28"	107'10"
Mixed Fura/Globus	2"(F1) 2"(F2) 2"(F1) 2"(F2)	39'38"(F1) 41'32"(F2) 50'56"(GN) 51'09"(GN)	1"(F1)	51'12"	23"	2'25"	54"

Tal como vemos en la tabla, los mejores resultados corresponden a la prueba combinada utilizando Globus y Fura. El tiempo total de procesamiento del workflow es más bajo debido al uso de todos los recursos de ejecución, es decir, tenemos cuatro ejecuciones largas y cuatro recursos donde lanzarlas por lo tanto no hay esperas a la hora de lanzar los trabajos.
El tiempo de overhead en este caso es más elevado debido a las transferencias de ficheros entre Fura y Globus, y al hecho de trabajar con protocolos más pesados (Globus).

7

Conclusiones y Trabajos Futuros

En este documento hemos descrito un plug-in para el sistema workflow conocido como WINGS. Dicho plug-in permite el lanzamiento de ejecuciones utilizando Globus Toolkit.

Como hemos mencionado en la introducción al documento, WINGS es fruto de un proyecto de investigación realizado en nuestro grupo de investigación (GRYCAP). Este sistema ya posee un componente que hace posible el lanzamiento de trabajos a través de Fura, y con el plug-in Globus se pretende aumentar la funcionalidad global del sistema. Debido a la naturaleza de WINGS, el plug-in desarrollado se añade como un componente más sin tener que modificar nada.

Hemos analizado y tenido en cuenta otros trabajos que compartían nuestros objetivos, pero ninguno de ellos se ajustaba a nuestras necesidades. Es por ello que decidimos ampliar el sistema WINGS.

Actualmente se han desarrollado plug-ins para ejecuciones Fura, Globus y sub-workflow. El siguiente objetivo sería la creación de nuevos motores de ejecución para completar el motor workflow con middlewares que no estén basados en Grid, como por ejemplo PBS, que nos permitiría lanzar trabajos batch en el sistema. Del mismo modo, también se está trabajando en la posibilidad de añadir nuevos plug-ins basados en sistemas de planificación Grid como pueden ser GridWay o GMarte, y nuevos componentes destinados a llamadas a servicios web.

Otra área de trabajo importante sería la extensión del sistema con nuevas operaciones y el estudio de la posibilidad de distribuir la ejecución de las operaciones. Ahora se ejecutan en el entorno de ejecución del workflow, pero si la cantidad de datos a procesar es elevada se produce un overhead importante tanto en la transferencia de datos como en la ejecución. Si se sitúan las operaciones cerca de donde tenemos los datos, las transferencias de grandes cantidades de datos se evitan, y por tanto tenemos menos sobrecarga en el sistema.

Y por último queremos revisar y mejorar ciertos detalles de implementación del plug-in creado, sobretodo detalles relativos a la monitorización de trabajos lanzados. En particular, queremos centrarnos en el relanzamiento de trabajos fallidos o que tarden mucho en entrar a ejecución, y en el sistema de gestión de las credenciales del usuario. Por un lado, el sistema de monitorización se encargaría de cancelar aquellos trabajos que fallaran (estado FAILED) y volverlos a lanzar en el sistema. Lo mismo ocurriría con los trabajos que tardaran mucho en entrar a ejecución.
En cuanto a las credenciales se debe estudiar la mejor manera de gestionarlas, es decir, el modo más seguro de manejarlas.

Bibliografía

[1] ***The Workflow Portal***.
http://www.e-workflow.org/.

[2] *Jia Yu & Rajkumar Buyya*.
A Novel Architecture for Realizing Grid Workflow using Tuple Spaces.
The Proceedings of 5th IEE/ACM International Workshop on Grid Computing (GRID'04), November 2004.

[3] *Carlos de Alfonso, Miguel Caballer & Vicente Hernández*.
WINGS: A Multigrid Wokflow Engine.
Proceedings of CGW'08, Cracow, pages 129 – 136, October 2008.

[4] ***The State of Workflow***.
http://www.theserverside.com/tt/articles/content/Workflow/article.html.

[5] ***Workflow Management Coalition***.
http://www.wfmc.org.

[6] *David Hollingsworth*.
The Workflow Reference model. Workflow Management Coalition.
Technical Report WfMC-TC-1003, January 1995.

[7] *Workflow Management Coalition Members*.
Workflow Client API Specifications (WAPI).
Technical Report WfMC-TC-1002, July 1998.

[8] *Workflow Management Coalition Members*.
Workflow Standard-Interoperability, Abstract Specification.
Reporte técnico WfMC-TC-1012, December 1999.

[9] *M. Rusinidewicz & A. Sheth*.
Specification and Execution of Transactional Workflows in the Modern Database Systems: The Object Model, Interoperability, and Beyond, pages 592–620.
Addison-Wesley, 1995.

[10] *Geoffrey Fox & Dennis Gannon.*
Workflow in Grid Systems.
Concurrency and Computation: Practice & Experience, 18(10):1009–1019, August 2006.

[11] *J. Song, J. Singh, C.K. Koh, Y.S. Ong & C.W. See.*
Grid Workflow Management System.

[12] ***Página Oficial del Proyecto SETI@home***.
http://setiathome.berkeley.edu/.

[13] ***RFC 959 - File Transfer Protocol (FTP)***.
http://www.ietf.org/rfc/rfc959.txt.

[14] ***RFC 2228 - FTP Security Extensions***.
www.ietf.org/rfc/rfc2228.txt.

[15] ***Internet X.509 Public Key Infrastructure Program Version 2, Update 1 (RFC 2743)***.
http://www.ietf.org/rfc/rfc2743.txt.

[16] ***Public Key Infrastructure (PKI)***.
http://www.pki-page.org/.

[17] *W3C.*
Simple Object Acces Protocol (SOAP) 1.1.
Note 8, 2008.

[18] ***Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language***.
http://www.w3.org/TR/wsdl20/.

[19] ***UDDI Version 3.0.2, UDDI Spec Technical Committee Draft***.
http://www.uddi.org/pubs/uddi_v3.htm.

[20] ***AGWL Portal***.
http://www.dps.uibk.ac.at/projects/agwl/.

[21] ***GridSystems - Fura***.
http://www.gridsystems.com/.

[22] ***gLite Workload Management System (WMS)***.
http://www.dps.uibk.ac.at/projects/agwl/.

[23] ***GridWay Portal***.
http://www.gridway.org/.

[24] *J.M. Alonso, V. Hernández & G. Moltó.*
GMarte: Grid middleware to abstract remote task execution.

Concurrency and Computation: Practice & Experience, 18(15):2021–2036, May 2006.

[25] ***Condor - High Throughput Computing***.
http://www.cs.wisc.edu/condor/.

[26] *Workflow Management Coalition Members*.
XML Process Definition Language.
Technical Report WfMC-TC-1025, June 2005.

[27] *Workflow Management Coalition Members*.
Workflow Standard-Interoperability, Abstract Specification.
Technical Report WfMC-TC-1012, December 1999.

[28] *J.L. Vázquez-Poletti, Eduardo Huedo Cuesta, Rubén Santiago Montero & Ignacio Martín Llorente*.
Una Visión Global de la Tecnología Grid.

[29] *Fernando Martín Sánchez & Alberto Villafranca Ramos*.
Introducción a la Tecnología Grid.

[30] *Rob Allen (Open Image Systems Inc. - United Kingdom)*.
Workflow: An Introduction.