"Diseño e implementación de herramientas grid basadas en WSRF"

Sistemas Informáticos
Facultad de Informática
Universidad Complutense de Madrid

Germán P. Santos Benavides
Eduardo Paz Orduña
Eduardo Hermida García

Curso 2004/2005

Índice general

Índice de figuras

Índice de cuadros

Prólogo

Bienvenidos a la documentación del proyecto de la asignatura de Sistemas Informáticos de la Facultad de Informática de la Universidad Complutense de Madrid. El proyecto lo hemos realizado entre los alumnos Germán Pablo Santos Benavides, Eduardo Hermida García y Eduardo Paz Orduña y tiene como título "Diseño e implementación de herramientas grid basadas en WSRF". El objetivo final ha sido la realización de un *MAD* (Middleware Access Driver) para el GridWay Framework desarrollado por el *Distributed Architecture and Security Group* de la Universidad Complutense de Madrid, y por el *Advanced Computation, Simulation and Telematic Applications Laboratory* del Centro de Astrobiología.

Resumen

El objetivo de este proyecto ha sido el desarrollo de una aplicación que permitiese acceder al *middleware* de Glosbus. Este *software* debe servir de puente entre el *middleware* y GridWay, que es una aplicación diseñada para obtener mejor rendimiento al enviar trabajos a un conjunto de recursos grid dinámicos. El motivo para implementar esta aplicación es la nueva implementación en C del *toolkit* lanzada el 30 de Abril de 2005. Antes de comenzar a hablar de la aplicación hay que introducir al lector a toda una serie de conceptos sobre computación grid, Web Services y globus. Algunos ejemplos son XML, mensajes SOAP, OGSA, stateful Web Services, WSRF,...

Summary

The objetive of this project has been the development of an application to access Globus middleware. This application must be the bridge between

the middleware and GridWay, an application designed to obtain the best performance from a set of dynamic grid resources. The motivation to do this, it's the brand new C implementation of the toolkit that Globus released on April 30 2005. Before we start with this subject, we have to introduce the reader to all the new concepts around grid computing, Web Services and Globus. Some of them are XML, SOAP messages, OGSA, stateful Web Servics, WSRF,...

Agradecimientos

Nos gustaría agradecer, en conjunto, el apoyo que hemos recibido por parte de los integrantes del grupo de desarrollo de GridWay, particularmente a José Luis Vázquez Polleti, Rubén Santiago e Ignacio Martín Llorente. También a Eduardo Huedo, que cada consulta que le hemos hecho nos la ha resuelto rápida y eficientemente.

Por otra parte, también queríamos agradecer a nuestras familias el apoyo que nos han dado durante el desarrollo de este proyecto y, en general, a todas las personas que han formado parte de nuestra vida durante la carrera. Aunque, en la mayoría de los casos, no han entendido nuestro proyecto, por lo menos pusieron cara interesante cuando se lo intentábamos explicar. Para todos ellos, aquí presentamos esta documentación. Os queremos.

Capítulo 1

Computación Grid

1.1. Los fundamentos de la computación grid

La computación grid es llevar al siguiente nivel de evolución la computación distribuida. El objetivo es crear la ilusión de una gran computadora virtual, potentemente auto-gestionada, a partir de una gran colección de sistemas conectados y compartiendo una combinación variada de recursos.

La estandarización de las comunicaciones entre sistemas heterogéneos desencadenó la explosión de Internet. La emergente estandarización en la compartición de recursos, junto con la disponibilidad de grandes anchos de banda, están acercando la posibilidad de un salto similar en la evolución de la computación grid.

1.1.1. Qué puede hacer la computación grid

Al desarrollar un grid hay que tener en cuenta, como siempre, los requisitos del cliente. Para poder aprovechar las capacidades de la computación grid en dichos requisitos, es muy útil tener en cuenta las razones por las cuales es conveniente usar la computación grid.

1.1.1.1. Explotación de recursos ociosos

El uso más fácil de la computación grid es la ejecución de una aplicación existente en otra máquina. Si, por ejemplo, la máquina en la que normalmente se ejecuta dicha aplicación está ocupada debido a un pico inusual en la actividad, el trabajo puede ser ejecutado en cualquier máquina ociosa en

otra parte del grid. Existen al menos dos prerrequisitos para esta situación. Primero, la aplicación debe ser remotamente ejecutable sin producir una excesiva sobrecarga. Segundo, la máquina remota debe tener el *hardware*, el software y/o los recursos necesarios impuestos por la aplicación.

En muchas organizaciones, existen grandes cantidades de recursos de computación poco utilizados. Muchos de los ordenadores de sobremesa están ocupados solamente el 5 % del tiempo. En algunas organizaciones, incluso los servidores pueden estar relativamente ociosos. La computación grid proporciona el marco ideal para poder aprovechar dichos recursos y la posibilidad de poder incrementar la eficiencia en el uso de recursos.

Los recursos de procesamiento no son lo únicos que pueden caer en desuso. A menudo las máquinas poseen una gran cantidad de espacio libre de almacenamiento en disco. La computación grid, más concretamente en este caso un *data grid*, puede ser usado para agregar este espacio libre de disco a un almacenamiento virtual de mucho mayor tamaño, y posiblemente dar la opción de mejorar el funcionamiento y la fiabilidad de los datos con respecto a una sola máquina. Si un trabajo necesita leer grandes cantidades de datos, éstos pueden ser automáticamente replicados en varios puntos estratégicos del grid. Así, si el trabajo necesita ejecutarse en una máquina remota del grid, los datos ya están allí y no necesitan ser llevados a ese punto. Esto ofrece grandes ventajas en la ejecución. Además, dichas copias pueden ser usadas como copias de seguridad cuando las copias primarias están dañadas o no disponibles.

1.1.1.2. Capacidad de paralelismo de CPU

El potencial para una capacidad masiva de paralelismo de CPU es uno de los aspectos más atractivos de un grid. En general, muchas aplicaciones científicas podrán mejorar su rendimiento, porque el atributo común de dichas aplicaciones es que han sido escritas de forma que se pueden dividir en partes independientemente ejecutables. Una aplicación de grid intensiva puede ser pensada como una gran cantidad de pequeñas *subtareas*, cada una de ellas ejecutándose en una máquina distinta del grid. Cuanta menos comunicación necesiten las *subtareas* entre ellas, más escalable será la aplicación. Un aplicación será perfectamente escalable si con el doble de procesadores se ejecuta en la mitad de tiempo. Son campos como, por ejemplo, la biomedicina, los modelos financieros, la animación por ordenador, los que más se están beneficiando de esta potencia de computación, que les está haciendo

evolucionar.

Existen, sin embargo, barreras para la perfecta escalabilidad. Una de ellas es el propio algoritmo utilizado para dividir la aplicación. Si sólo puede ser dividido en un número limitado de partes ejecutables, eso supondrá un límite a la escalabilidad. La segunda barrera aparece cuando las distintas partes no son completamente independientes; esto causa contención en la ejecución, lo que limita también la escalabilidad. Por ejemplo, si todos los subtrabajos necesitan leer de un fichero común o de una base de datos, los límites en el acceso al fichero o a la base de datos serán el factor que determine la escalabilidad de la aplicación. Por último, otro problema de contención en aplicaciones paralelas en un grid puede ser la latencia en la comunicación mediante mensajes entre las *subtareas*, la capacidad de comunicación de la red, los protocolos de sincronización, el ancho de banda de entrada/salida con periféricos y la latencia con requisitos de tiempo real.

Hay que entender que no todas las aplicaciones pueden ser paralelizadas para poder mejorar su rendimiento en un grid. Tampoco existen herramientas que transformen aplicaciones arbitrarias para que sean capaces de aprovechar las ventajas del grid. Aunque existen algunas herramientas que paralelizan aplicaciones, la automatización de ese proceso es una ciencia todavía muy joven. Solamente las aplicaciones diseñadas específicamente para trabajar en paralelo, son en la actualidad las más firmes candidatas a poderse ejecutar en el grid.

1.1.1.3. Colaboración de recursos y organizaciones virtuales

Otra importante contribución de la computación grid es habilitar y simplificar la colaboración entre una amplia audiencia. La computación grid posibilita esta colaboración mediante el uso de estándares que hacen que sistemas heterogéneos puedan trabajar conjuntamente para dar la impresión de formar un sistema virtual de computación que ofrece una variedad mucho mayor de recursos. Los usuarios del grid pueden ser agrupados en *organizaciones virtuales*, cada una con su propia política, y que pueden compartir sus recursos para aumentar el tamaño del grid.

Los participantes y usuarios del grid pueden ser miembros de distintas organizaciones reales y virtuales. El grid puede facilitar el aumento de reglas de seguridad entre ellas además de implementar distintas políticas, que pueden resolver prioridades con respecto a recursos y usuarios.

1.1.1.4. Acceso a recursos adicionales

Una posibilidad muy interesante que un grid puede ofrecer es el acceso a una gran cantidad de recursos y de equipos especiales, software, licencias y otros servicios. La compartición comienza por los datos, ya sea en forma de ficheros o de bases de datos. Un *data grid* puede aumentar en gran manera las posibilidades de manejo de los datos. No obstante, el grid será tanto más útil cuantos más recursos se compartan, ya sean *software*, *hardware*, servicios, licencias u otros. Por ejemplo, si un usuario necesita ampliar su ancho de banda a Internet para poder implementar un servicio de búsqueda de datos, el trabajo puede ser dividido entre máquinas del grid que tengan su conexión independiente a Internet. Otro ejemplo es el caso de máquinas que tienen instalado software con licencias realmente caras y que el usuario necesita utilizar. Sus trabajos podrían ser enviados a dicha máquina remota para que ejecute los trabajos con dicho *software*. Otras máquinas podrían poseer periféricos especiales. Un caso particular sería el utilizar remotamente una impresora que tenga especiales características como la velocidad o impresión a color avanzada. Igualmente, el grid puede ser usado para utilizar otros equipos especiales como puede ser una máquina con mayor velocidad, una grabadora de DVDs de mayor velocidad, o lo que puede ser más interesante, máquinas con algún dispositivo caro y raro, ya sea por ejemplo un microscopio, en el que poder mandar con antelación las muestras y obtener en nuestra máquina los resultados.

Todo esto hace que el grid parezca una gran máquina virtual con una gran variedad de recursos y equipos virtuales que pueden ser accesibles mediante una máquina convencional.

1.1.1.5. Equilibrio de recursos

En un grid se pueden encontrar una gran cantidad de recursos aportados por cada una de las máquinas individuales que forman un gran recurso global y virtual. Para las aplicaciones que pueden ejecutarse en un grid, éste les ofrece un uso equilibrado de dichos recursos mediante una planificación de los trabajos grid en las máquinas con menor uso. Esta característica es de gran importancia cuando se producen subidas o picos en la actividad en determinadas zonas de una organización grande. Esto puede suceder de dos maneras:

- Un pico inesperado puede ser redirigido a máquinas relativamente ocio-

sas en el grid.

- Si el grid ya está suficientemente ocupado, el trabajo con menor prioridad que está siendo ejecutado en el grid puede ser temporalmente suspendido o incluso cancelado para una posterior ejecución con el fin de dejar su sitio a trabajos de mayor prioridad.

Para ver un ejemplo de la primera característica, supongamos que nuestra organización está dividida en departamentos. Si, temporalmente, un departamento tiene una actividad mucho mayor necesitará más recursos que los que utiliza normalmente. En esta situación, el grid nos permitiría distribuir la carga de trabajo por el resto de las máquinas de la empresa. Por ejemplo, si las aplicaciones están capacitadas para trabajar en un grid, pueden ser movidas a máquinas ociosas durante dichos picos.

En general, el grid proporciona una manera de consistente de equilibrar las cargas en un amplio conjunto de recursos. Además sin una infraestructura definida de grid, estas tareas de equilibrado de recursos resultan muy complejas de ejecutar y priorizar.

Otros beneficio que puede aportar el grid mediante un planificador avanzado es reducir en gran manera la comunicación y otros aspectos de contención en el grid. Es decir, cuando los trabajos necesitan comunicarse entre ellos, con Internet o con recursos de almacenamiento, el planificador puede organizar los trabajos de tal manera que minimice el tráfico o la distancia entre los trabajos implicados en la comunicación.

Por último, el grid proporciona una infraestructura excelente para transacciones de recursos. Los recursos pueden ser catalogados determinando su disponibilidad y su capacidad, y esto puede ser un factor importante en la planificación del grid. Las distintas organizaciones que pueden participar en el grid pueden crear unos "créditos" del grid y usarlos cuando necesiten el uso de dichos recursos. Esta solución puede llegar a ser la base para crear una contabilidad en el grid en lo referente a la utilización de recursos y así mejorar la planificación y distribuir los trabajos más equitativamente.

1.1.1.6. Fiabilidad

Los sistemas convencionales de computación de gama alta emplean *hardware* muy caro para incrementar su fiabilidad. Están construidos a partir de chips con circuitos redundantes y que contienen sistemas para poder recuperarse fácilmente de fallos de *hardware*. Además, suelen tener procesadores

duplicados con capacidad de ser instalados en caliente, para que al fallar, puedan ser reemplazados sin la necesidad de apagar el sistema. Las fuentes de alimentación y los sistemas de refrigeración también suelen estar por duplicado. Además suelen tener instalados en las fuentes de alimentación generadores que empiezan a funcionar cuando la corriente es interrumpida. Todo esto produce sistemas muy fiables, con grandes costes, debido a la duplicación de prácticamente todos sus elementos.

En el futuro veremos una propuesta de fiabilidad que tendrá en cuenta tanto *hardware* como *software*. El grid es sólo el comienzo de esta tecnología. Los sistemas del grid pueden ser relativamente menos costosos y geográficamente más dispersos. Por lo que, si hay un fallo de suministro eléctrico en una zona, otras partes del grid no resultarán afectadas. El software de administración del grid puede automáticamente reenviar trabajos a otras máquinas cuando un fallo sea detectado. Y en situaciones críticas, situaciones de tiempo real, múltiples copias de los trabajos importantes pueden ejecutarse en distintas máquinas a lo largo del grid. Además así sus resultados pueden ser comparados para detectar inconsistencias, como fallos de ordenador o corrupción de datos.

Este tipo de sistemas grid utilizan *computación autónoma*. Se trata de un tipo de software que automáticamente soluciona problemas en el grid, incluso antes de que un operador o un administrador pueda darse cuenta. En principio, la mayor parte de los atributos llevados a cabo en *hardware* para sistemas de alta disponibilidad podrán ser imitados mediante *software* en el grid.

1.1.1.7. Administración

Disponer de un grid le permite a la organización poder llevar a cabo tareas de registro en cuanto a uso de recursos. Esta información sobre qué recursos se usan más y cuáles están cayendo en desuso podrá ser tenida en cuenta a la hora de actualizar los sistemas, haciéndole la tarea más sencilla a los departamentos a la hora de controlar los gastos en los recursos de computación a lo largo de una gran organización. La agregación de los datos utilizados en los proyectos al grid puede dar a las organizaciones la facilidad de hacer más fácil las futuras actualizaciones. Así cuando se realicen tareas de mantenimiento, los trabajos pueden ser llevados a otras máquinas mientras se realizan las mismas sin involucrar al desarrollo de proyectos.

El grid ofrece también la administración de las prioridades entre los dis-

tintos proyectos. En el pasado, cada proyecto era responsable de los recursos de *hardware* y los gastos asociados a ellos. A menudo, este *hardware* estaba inutilizado, mientras otros proyectos se encontraban con grandes problemas, necesitando más recursos debido a eventos inoportunos. Para estos casos, el grid puede llegar a ofrecer grandes facilidades en el control y manejo de estas situaciones.

También la *computación autónoma* forma parte de este punto, ya que existen varias herramientas que pueden localizar importantes tendencias en el grid que puedan requerir la atención del administrador.

1.1.2. Conceptos y Componentes del grid

En esta sección, vamos a introducir los varios conceptos, términos y componentes del grid con más detalle.

1.1.2.1. Tipos de recursos

Como se ha ido viendo en puntos anteriores, existen muchos tipos de recursos que pueden ser compartidos por las máquinas conectadas a un grid. Vamos a definir los diferentes tipos de recursos y veremos que, mientras algunos de ellos pueden ser usados por todos los usuarios del grid, otros pueden tener restricciones específicas.

- Computación
  El recurso más común son ciclos de computación proporcionados por los procesadores de las maquinas del grid. Los procesadores pueden variar en velocidad, arquitectura, plataforma software y otros factores asociados, como memoria, almacenamiento y conectividad. Hay tres formas principales de explotar los recursos de computación de un grid. La primera y más simple es usarlos para ejecutar una aplicación en una maquina disponible en el grid en vez de localmente. La segunda es usar una aplicación diseñada para dividir su trabajo, de tal manera que diferentes partes de la aplicación puedan ejecutarse en paralelo en diferentes procesadores. La tercera es ejecutar una aplicación, que necesita ser ejecutada varias veces, en diferentes maquinas del grid.

- Almacenamiento
  El segundo recurso más común usado en un grid es el almacenamiento de datos. Cada maquina del grid suele proveer de cierta cantidad de

almacenamiento para el uso del grid. El almacenamiento puede estar ligado al procesador o puede ser almacenamiento secundario usando discos duros u otros dispositivos de almacenamiento permanente. La memoria ligada al procesador suele tener rápida velocidad de acceso pero es volátil. Es mejor usarla como caché de datos o como almacenamiento temporal para las aplicaciones en proceso de ejecución.
El almacenamiento secundario en un grid puede ser usado de maneras interesantes para incrementar la capacidad, la compartición y la fiabilidad de los datos. La capacidad puede ser incrementada usando el almacenamiento de varias máquinas con un sistema de ficheros unificado. Cada fichero individual o base de batos puede abarcar varios dispositivos de almacenamiento y máquinas, eliminando las restricciones de tamaño máximo que suelen imponer los sistemas de ficheros de los sistemas operativos. Un sistema unificado de ficheros puede proveer también de un espacio de nombres uniforme para el almacenamiento del grid. De una manera similar, un software especial de bases de datos puede unir varias bases de datos individuales para formar una mayor, y de más fácil comprensión, pudiendo acceder a ella usando funciones de *query* de la base de datos.
Sistemas de ficheros más avanzados permiten la duplicación de datos, para aumentar la fiabilidad por medio de la redundancia. Un planificador inteligente puede seleccionar el dispositivo de almacenamiento apropiado para los datos. Así las tareas pueden ser planificadas en las máquinas directamente conectadas a los dispositivos de almacenamiento que guardan los datos que dicha tarea va a utilizar.
El sistema de ficheros de un grid también puede implementar métodos para que los datos puedan ser recuperados con más fiabilidad ante ciertos tipos de fallos.

- Comunicaciones
  El rápido crecimiento de la capacidad de comunicación entre máquinas hoy en día hace de grid práctica para la computación, comparado con el limitado ancho de banda disponible cuando empezó a emerger el procesamiento distribuido. Por eso, no sorprende que otro importante recurso de un grid sea la capacidad de comunicación de datos. Esto incluye comunicación interna y externa al grid. Las comunicaciones dentro del grid son importantes para el envío de tareas y los datos requeridos a puntos en el interior del grid. Algunas tareas necesitan

gran cantidad de datos para ser procesados y puede que estos no residan en la máquina en la que se esta ejecutando dicha tarea. El ancho de banda disponible para esas comunicaciones es un recurso crucial que puede limitar la utilización del grid.
La comunicación externa puede ser importante, por ejemplo, si deseamos construir un motor de búsqueda. Las máquinas del grid pueden tener conexiones a Internet además de las conexiones internas del grid. En algunas ocasiones serán necesarias rutas de comunicación redundantes para un mejor tratamiento de posibles fallos en la red y excesivo tráfico de datos. Un sistema de gestión del grid puede mostrar la topología de esta y resaltar los posibles cuellos de botella en las comunicaciones. Esta información puede ser usada para planificar posibles modernizaciones del hardware.

- Software y licencias
  El grid puede tener instalado software demasiado caro como para ser instalado en cada máquina del grid. Usando un grid, las tareas que necesitan ese software son enviadas a máquinas específicas, pudiendo ser las que tienen dicho software instalado.
  Algunos convenios en licencias de software permiten que este sea instalado en todas las máquinas de un grid, pero limita el número de máquinas que pueden usar dicho software simultáneamente. El software de gestión de la licencia no permite que se estén ejecutando más copias de las acordadas.

- Equipamiento especial, capacidades, arquitecturas, políticas
  Las plataformas del grid tendrán habitualmente diferentes arquitecturas, sistemas operativos, dispositivos, capacidades y equipamiento. Cada uno de estos elementos representa un tipo diferente de recurso que el grid puede usar como criterio a la hora de asignar tareas a máquinas. Mientras que cierto software puede estar disponible en varias arquitecturas, otro estará diseñado para ser ejecutado en un tipo de hardware y sistema operativo en particular. Esta cuestión tiene que ser tomada en cuenta a la hora de asignar tareas a los recursos del grid.
  En algunos casos, el administrador del grid puede crear un tipo de recurso artificial que es usado por los planificadores para asignar trabajos de acuerdo a una determinada política o a otras restricciones.

1.1.2.2. Tareas y aplicaciones

A pesar de que varios tipos de recursos del grid pueden ser compartidos y usados, normalmente se accede a ellos mediante la ejecución de una aplicación o una tarea. Normalmente usamos el término aplicación como el más alto nivel de una parte de un trabajo del grid. No obstante, algunas veces el termino tarea se usa de igual manera. Las aplicaciones pueden ser divididas en cierto número de tareas individuales. Estas a su vez, pueden ser divididas en subtareas. La industria de grid utiliza otros términos como transacción o unidad de trabajo para referirse a lo mismo que una tarea.

Las tareas son programas ejecutados en un punto apropiado del grid. Pueden servir para calcular algo, ejecutar uno o más comandos del sistema, mover o recopilar datos? Una aplicación grid que está organizada en tareas suele diseñarse de tal manera que dichas tareas puedan ejecutarse en paralelo en diferentes máquinas del grid.

Las tareas pueden tener dependencias específicas que pueden impedir su ejecución paralela en todos los casos. Por ejemplo, pueden necesitar algún dato específico de entrada que debe ser copiado a la máquina en la que la tarea se esta ejecutando. Algunas tareas pueden necesitar la salida producida por otras tareas y no pueden ser ejecutadas hasta que estas hayan terminado de ejecutarse. Este flujo de trabajo puede crear una jerarquía de tareas y subtareas. Finalmente, los resultados de todas las tareas deben ser recogidos y ensamblados debidamente para producir la respuesta final de la aplicación.

1.1.2.3. Planificación, reserva y *scavenging*

El sistema del grid es responsable de enviar una tarea a una máquina determinada para ser ejecutada. En el sistema más simple, el usuario puede seleccionar una máquina adecuada para ejecutar su tarea y después ejecutar un comando grid que envía la tarea a la máquina seleccionada. En sistemas grid más avanzados suele haber un planificador de tareas de algún tipo, que automáticamente encuentra la máquina más apropiada para ejecutar cierta tarea dada que esté esperando a ser ejecutada. Los planificadores reaccionan ante la disponibilidad real de los recursos del grid. El término planificación no debe ser confundido con reserva de recursos para mejorar la calidad del servicio.

En un sistema de *scavenging* del grid, cualquier máquina que pase a un estado inactivo, informara de ese estado de inactividad al nodo de gestión

del grid. Este nodo de gestión asignará a esta máquina inactiva la siguiente tarea satisfecha con los recursos de dicha máquina. El *scavenging* suele implementarse de tal modo que no interfiera en el uso normal de la máquina. Si la máquina está ocupada con trabajo local no perteneciente al grid, las tareas pertenecientes al grid suelen ser suspendidas o retrasadas. Esta situación hace que sea difícil predecir el tiempo de finalización de las tareas grid.

Para crear un comportamiento más previsible, las máquinas del grid suelen estar dedicadas en exclusiva al grid. Esto permite a los planificadores calcular el tiempo aproximado de finalización de una serie de tareas, cuando sus características de ejecución son conocidas.

Los recursos del grid pueden ser reservados con antelación para una serie de tareas en concreto. Esto se hace para garantizar calidad en el servicio. La planificación y la reserva son sencillas cuando solo está involucrado un tipo de recurso, normalmente CPU. Sin embargo, se pueden conseguir optimizaciones para tener en cuenta más tipos de recursos a la hora de la planificación y la reserva. Por ejemplo, sería deseable asignar tareas a las máquinas más cercanas a los datos que estas tareas van a utilizar. Esto reduciría el tráfico en la red y posiblemente los límites de escalabilidad. La planificación óptima, considerando múltiples recursos es un problema matemático complicado. Por eso algunos planificadores utilizan heurísticas para mejorar la probabilidad de encontrar la mejor combinación de planificación y reserva.

1.1.2.4. De intragrid a intergrid

Ha habido intentos de formular una definición precisa de lo que es un grid. En realidad, el concepto de computación grid todavía está desarrollándose y la mayoría de los intentos de definirlo precisamente acaban excluyendo implementaciones que muchos considerarían que son grids.

Las grids pueden ser construidas de todos los tamaños, desde unas pocas máquinas en un departamento hasta grupos de máquinas organizadas jerárquicamente alrededor de todo el mundo. Aquí vamos a describir algunos ejemplo en este rango de topologías de sistemas grid.

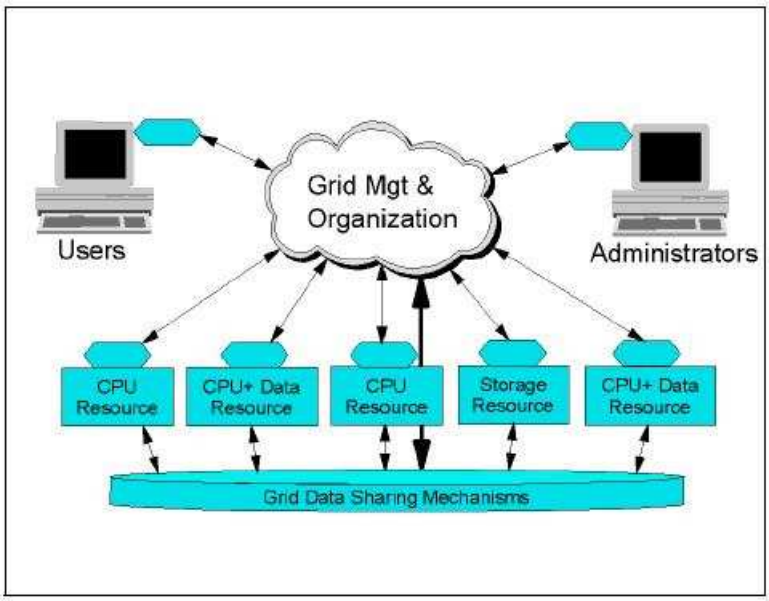

Figura 1.1: Grid sencillo consistente en unas pocas máquinas

Como se puede ver en la figura, el grid más simple consiste en unas pocas máquinas, todas con la misma arquitectura y el mismo sistema operativo, conectadas a una red local. Este tipo de grid usa sistemas homogéneos con lo cual hace falta tener en cuenta pocas consideraciones, y puede usarse para experimentar con software grid. Las máquinas suelen estar conectadas en un departamento de una organización y su uso como grid puede no requerir de políticas especiales. Como las máquinas tienen la misma arquitectura y sistema operativo, elegir software para estas máquinas suele ser simple. Algunas personas llamarían a esto clúster antes que grid.

El siguiente paso sería incluir máquinas heterogéneas. En esta configuración hay más tipos de recursos disponibles. El sistema grid probablemente incluya algunos componentes de planificación. También debería conseguirse la compartición de ficheros usando sistemas de ficheros distribuidos. Las máquinas que participan en este tipo de grid pueden ser de diferentes departamentos, pero dentro de una misma organización. Eso se conoce como intragrid.

Al expandirse el grid a varios departamentos, se deben usar políticas con respecto al uso del grid. Por ejemplo, debe haber políticas acerca de que

tipos de trabajos se permiten y en que momento. Se debe establecer un orden de prioridades entre departamentos o entre tipos de aplicaciones que deben acceder a los recursos del grid. También la seguridad aumenta su importancia al aumentar el número de organizaciones involucradas. Puede ser necesario proteger algunos datos de un departamento para que no puedan ser accedidos por las tareas que se ejecutan en otro departamento.

El grid puede crecer geográficamente en organizaciones que tenga infraestructuras en diferentes ciudades. En ocasiones se pueden usar túneles VPN u otras tecnologías para conectar las distintas partes de la organización. La importancia de la seguridad incrementa una vez que se traspasan los límites de una determinada infraestructura. El grid puede crecer para jerarquizarse y así reducir el control central, incrementando la escalabilidad.

Finalmente, el grid puede crecer, traspasando los límites de una organización, pudiendo ser usada para colaborar en proyectos de interés común. Esto se conoce como intergrid. Se necesitan niveles muy altos de seguridad para evitar ataques y espionaje.

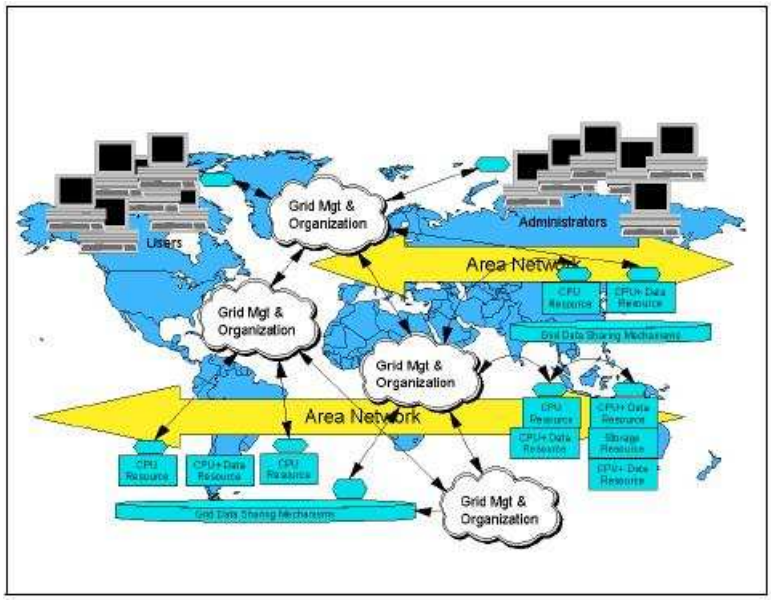

Figura 1.2: Grid a gran escala

1.1.3. Construcción del grid

Vamos a explicar en este punto cuáles serían los pasos a dar para dotar a una organización de un grid. Para ello, habrá observar las características de la organización y elegir las tecnologías grid acordes con sus requisitos. Y es que si bien unos pocos programadores pueden instalar un grid en su tiempo libre, a medida que crezca el grid será necesario una planificación.

1.1.3.1. *Planning* de desarrollo

Una de las primeras consideraciones a tener en cuenta es el hardware disponible y cómo éste está conectado vía LAN o WAN. Cómo hemos visto, el grid permitirá obtener mayor rendimiento del hardware que ya se dispone en la organización, lo que no impide que en un futuro se pueda querer añadir equipos adicionales para aumentar las prestaciones del grid. También es importante entender las aplicaciones que se van a ejecutar sobre el grid, ya que sus características pueden afectar a las decisiones de elección y configuración del hardware y la conectividad de los datos.

- Seguridad
  La seguridad es un factor mucho más importante en el *planning* y el mantenimiento de un grid que en el procesamiento distribuido convencional, donde la compartición de datos comprende la mayoría de la actividad. En un grid, las máquinas que lo forman están configuradas para ejecutar programas, más que para trasladar datos, lo que hace que uno inseguro sea tierra de cultivo de virus y troyanos. Por eso, es importante comprender qué componentes del grid deben ser asegurados rigurosamente.

- Organización
  Las consideraciones tecnológicas son importantes al desplegar un grid. No obstante, los temas organizativos y de negocios pueden ser igualmente importantes. Es importante saber como interactúan y operan los departamentos de una organización. A veces hay barreras entre departamentos, con el fin de proteger sus recursos para incrementar la probabilidad de éxito. Sin embargo, una mayor distribución de recursos puede beneficiar a toda la organización. Más aún, la compartición de recursos mediante un grid ofrece la capacidad de repartirlos dinámicamente, mediante la creación de políticas de arbitraje y la asignación

de prioridades. Así, será más sencillo reaccionar y recursos donde se encuentre el escenario con mayor prioridad.

1.1.3.2. Componentes de software del grid

Esta sección presenta algunas componentes claves que deben ser discutidas antes de diseñar la arquitectura del grid

- Componentes de gestión
  Cualquier sistema grid tiene algunos componentes de gestión. Hay un componente que controla los recursos disponibles del grid y los usuarios miembros del grid. Por otra parte, hay componentes de medición que determinan tanto la capacidad de los nodos del grid como su grado de utilización en un determinado instante. La información devuelta por los dos módulos anteriores se usa para planificar las tareas en el grid. También se usará dicha información para determinar la salud del grid, alertando de problemas como apagones y congestiones, y, además, para generar patrones de uso generales y estadísticas, así como para registrar y cuantificar el uso de los recursos del grid. Tercero, el software de gestión de grid más avanzado puede gestionar automáticamente muchos aspectos del grid. Este software puede recuperar de distintos tipos de fallos del grid.

- Software de envío
  Normalmente cualquier máquina del grid puede ser usada para enviar tareas al grid. Sin embargo en algunos sistemas grid, esta función es implementada como un componente separado, instalado en los nodos de envío, o clientes de envío.

- Gestión distribuida de grid
  En los grids más grandes el trabajo relacionado con la gestión del grid está distribuido. Dotar al grid de estructura jerárquica influirá positivamente en su escalabilidad. Vamos a comentar dos ejemplos para ejemplificar lo anteriormente expuesto. Un planificador central no planificará el envío de una tarea a la máquina que la va a ejecutar, si no que envía la tarea a un planificador de más bajo nivel, que gestiona un conjunto determinado de máquinas, para que se encargue del envío a una máquina específica. De igual manera, la información estadística

está distribuida. Los clusters de menor nivel recogen la información de las máquinas individuales y la envían a niveles mayores de la jerarquía

- Planificadores
  La mayoría de los sistemas grid incluyen algún tipo de software de planificación. Este software localiza una máquina en la que ejecutar una tarea que ha sido enviada por un usuario. En los grids más simples se puede asignar a ciegas la tarea, siguiendo un algoritmo de *round-robin,* enviándola a la siguiente máquina que cumpla todos los requisitos de recursos. Sin embargo, existen ventajas al usar un planificador más avanzado. Algunos planificadores implementan un sistema de prioridad de tareas. Esto se hace usando varias colas de tareas, cada una con una prioridad diferente. A medida que las máquinas del grid van quedando libres para ejecutar tareas, éstas se escogen primero de las colas de mayor prioridad. Hay implementadas políticas de varios tipos usando planificadores. Las políticas pueden incluir varios tipos de restricciones sobre las tareas, los usuarios y los recursos. Por ejemplo, puede haber una política que restrinja ejecutar tareas en ciertos momentos del día. Otra ventaja es que los planificadores suelen reaccionar inmediatamente a la carga de trabajo existente en el grid. Usan información sobre la utilización actual de las máquinas para determinar cuáles no están ocupadas antes de enviar una tarea.
  Los planificadores más avanzados controlarán el progreso de las tareas planificadas, gestionando el flujo general de trabajo. Si las tareas se pierden debido a apagones en el sistema o en la red, un buen planificador reenviará automáticamente las tareas. Sin embargo, si una tarea parece haber caído en un bucle infinito y supera su *timeout*, dicha tarea no debe ser replanificada.
  La obtención de recursos del grid se realiza mediante un sistema de reservas. Es más que un planificador, es un calendario para reservar recursos para determinados intervalos de tiempo, y evitar que otros reserven los mismos recursos para el mismo tiempo. También se encarga de eliminar o suspender tareas que estén ejecutándose en una máquina una vez que se ha alcanzado el tiempo de reserva.

- Comunicaciones
  Un sistema grid puede incluir software para ayudar a las tareas a comunicarse entre ellas. Como hemos visto anteriormente, una aplicación

puede dividirse en un amplio número de *subtareas*, siendo cada una de ellas planificada independientemente en el grid. Sin embargo, el algoritmo utilizado por la aplicación puede que necesite que las *subtareas* se comuniquen entre sí, es decir, necesitan localizar a otras *subtareas* específicas, establecer una comunicación con ellas y enviar los datos apropiados. Es por esto que el estándar MPI y otras variaciones suelen estar incluidos en el sistema grid.

- Observación, gestión y medidas
  Hemos mencionado antes que los planificadores reaccionan a las cargas actuales del grid. Normalmente el software incluye herramientas que miden la carga y actividad actuales de una determinada máquina usando utilidades del sistema operativo o por medida directa. Este software se suele denominar sensor de carga. Dicha información de medida es útil no sólo para planificación, sino para descubrir patrones generales de uso del grid. Las estadísticas pueden mostrar tendencias que pueden delatar la necesidad de hardware adicional. Las medidas pueden también guardarse para cuantificación, o para gestionar las prioridades más adecuadamente. Además la información puede ser mostrada de varias formas para visualizar mejor el uso y la actividad del grid

1.1.4. Uso del grid: perspectiva del usuario

1.1.4.1. Instalación y activación del software de grid

El usuario primero se da de alta como usuario de grid, y después instala el software que se le suministra en su propia máquina. También puede, opcionalmente, dar de alta a su máquina como contribuyente al grid.

Darse de alta en el grid puede requerir autentificación por razones de seguridad. El usuario se identifica mediante una Autoridad de Certificación. La responsabilidad de mantener sus credenciales grid seguras recae en el propio usuario.

Una vez que tanto usuario como máquina están identificados, se le proporciona al usuario el software de grid, para que lo instale en su máquina con el propósito de usar el grid, así como de contribuir a éste. Este software puede ser preconfigurado automáticamente por el sistema de gestión del grid para conocer las direcciones de los nodos de gestión del grid, y la información sobre la identificación de máquina y usuario.

1.1.4.2. Registro en el grid

Para usar el grid, la mayoría de los sistemas necesitan que el usuario se registre en un sistema usando el ID de usuario que esta dado de alta en el grid. Otros sistemas grid pueden tener su propio login, diferente del que ahí en el sistema operativo. El login de grid es más apropiado para usuarios.

Una vez registrado, el usuario puede usar el grid y enviar tareas. Algunas implementaciones de grid permiten algunas funciones de búsqueda si el usuario no está registrado e incluso si no está dado de alta en el grid.

1.1.4.3. Consultas y envío de trabajos

El usuario normalmente querrá hacer algunas consultas para comprobar como de ocupada esta el grid, para ver como progresan las tareas que ha enviado, y para buscar recursos en el grid. Los sistemas grid suelen traer una línea de comandos y hasta interfaces gráficas de usuario para consultas. Las herramientas de la línea de comandos son muy útiles en el caso en que el usuario quiera escribir una *script* para realizar una secuencia de acciones.

El envío de tareas suele constar de tres partes, aunque se realice con un único comando. Primero, algunos datos de entrada y posiblemente el programa ejecutable o la script de ejecución se envían a la máquina que va a ejecutar la tarea. Los datos y los ficheros de programa pueden ser preinstalados en las máquinas del grid o accedidos mediante sistemas de ficheros distribuidos. Algunas tecnologías grid necesitan que el programa y los datos de entrada sean procesados primero de algún modo por el sistema grid (wrapping). Esto se hace para aumentar la protección de la ejecución o simplemente para unir todos los ficheros de datos en uno.

Segundo, la tarea es ejecutada en la máquina grid. El software grid residente en la máquina ejecuta el programa en un proceso en nombre del usuario. Puede usar una ID de usuario de la máquina o su propia ID de usuario, dependiendo de la tecnología que se esté usando.

Tercero, los resultados de la tarea se devuelven a la máquina que la envió. En algunas implementaciones, los resultados intermedios pueden ser vistos por el usuario que envió el trabajo.

Las scripts son también muy útiles a la hora de enviar lotes de tareas. Algunos problemas de cálculo consisten en la búsqueda de los resultados deseados partiendo de ciertas entradas. El objetivo es encontrar que parámetros de entrada producen el mejor resultado deseado. Para cada parámetros de

entrada, se ejecuta una tarea distinta para obtener el resultado para ese valor. La aplicación consiste en varias tareas que exploran los resultados para muchos conjuntos de parámetros de entrada distintos. Las scripts se suelen usar para lanzar las diferentes subtareas, cada una de las cuales recibe su propio conjunto de valores para los parámetros.

Cuando hay un número amplio de subtareas, el trabajo requerido para recoger los resultados y producir el resultado final suele realizarlo un único programa, que suele ejecutarse en la máquina que envió la tarea. Si hay muchas subtareas necesarias para una aplicación, el trabajo de recoger los resultados debe ser también distribuido.

1.1.4.4. Configuración de datos

Los datos a los que acceden las tareas del grid pueden simplemente entrar y salir del sistema grid. Sin embargo, dependiendo del tamaño de estos y del número de tareas, esto puede aumentar considerablemente el tráfico de datos. Por eso, se intenta efectuar el menor movimiento de datos posible.

Si hubiera muchas subtareas ejecutándose en la mayoría de los sistemas grid, y dichas subtareas fueran de una aplicación que se ejecutará repetidamente, los datos que usan pueden ser copiados a cada máquina y residir en ella hasta la próxima ejecución. Esta opción es mejor que usar un sistema de ficheros distribuido para compartir esos datos, porque así evitamos el movimiento de datos.

1.1.4.5. Control del progreso y recuperación

El usuario puede consultar al sistema grid y ver como progresan su aplicación y sus subtareas. Cuando el número de subtareas es grande, resulta difícil ver la lista de todas en una ventana gráfica. En vez de eso, puede haber simplemente una barra de progreso. Así resulta muy difícil para el usuario ver si una subtarea en particular no se está ejecutando adecuadamente.

Un sistema grid, junto a su planificador de tareas, suele proporcionar cierta capacidad de recuperación de las tareas que fallas. Una tarea puede fallar debido a:

- Errores de programación: el trabajo para debido a un fallo en el programa.

- Fallo de hardware o de alimentación: las máquinas o dispositivas usados paran de trabajar de repente.
- Interrupción en las comunicaciones: la ruta de comunicación falla o está saturada por tráfico de otros datos.
- Una lentitud excesiva provocada por un bucle infinito o por una tarea con mayor prioridad.

No siempre es posible determinar automáticamente si la razón por la que una tarea falla se debe a un problema con el diseño de la aplicación o por fallos en la infraestructura del sistema grid. Los planificadores suelen estar diseñados para clasificar los fallos en las tareas de algún modo y reenviar automáticamente las tareas que parece que pueden completarse. En algunos sistemas, el control del reenvío queda en manos del usuario.

1.1.4.6. Reserva de recursos

Para mejorar la calidad del servicio, el usuario puede organizar la reserva de una serie de máquinas para el futuro, para su uso exclusivo y prioritario.

En un sistema grid con scavenging, puede no ser posible reservar máquinas específicas con antelación. En lugar de esto, los sistemas de gestión de grid pueden asignar gran parte de su capacidad a una reserva determinada, para aparentar que sus recursos no están disponibles. Esto debe hacerse junto con las herramientas que controlan la capacidad de carga del grid para tener estadísticas fiables acerca de la capacidad del grid para hacer frente a la reserva.

1.1.5. Uso del grid: perspectiva del administrador

1.1.5.1. Planning

El administrador debe conocer lo que la organización va a requerir del grid para elegir de la mejor manera posible las tecnologías grid que satisfagan dichos requisitos. Las siguientes secciones describen brevemente los pasos que debe seguir el administrador para la gestión del grid. Es aconsejable empezar desplegando un grid pequeño primero, para aprender acerca de su instalación y gestión.

1.1.5.2. Instalación

Primero, el sistema grid seleccionado debe ser instalado en un conjunto de máquinas configuradas adecuadamente. Estas máquinas deben estar conectadas usando redes con suficiente ancho de banda a otras máquinas del grid. Es de vital importancia conocer los escenarios de fallo para el sistema grid dado, de modo que el grid pueda continuar trabajando incluso si alguna de las máquinas de gestión falla de algún modo. Todos los datos esenciales, bases de datos importantes deben tener backups.

Después de la instalación, el software de grid puede necesitar ser configurado para las direcciones e ID's de la red local. El administrador normalmente pedirá acceso de root a las máquinas de gestión del grid. En algunos sistemas grid, también necesitará acceso de root a las máquinas que comparten sus recursos en el grid, necesario para instalar el software también en estas máquinas.

Una vez que el grid está operativo, puede haber software de aplicación y datos que deben ser instalados también en las máquinas que comparten sus recursos en el grid. Algunos sistemas grid incluyen herramientas de ayuda para la gestión de la licencia a nivel de todo el grid.

1.1.5.3. Dar de alta a máquinas y usuarios

Una tarea del administrador grid es gestionar los miembros del grid, tanto las máquinas que aportan recursos como los usuarios. Los usuarios pueden estar organizados en grupos de proyecto. El administrador es el responsable de controlar los derechos de los usuarios del grid. Las máquinas pueden tener también derechos de acceso que necesitan ser gestionados. Las tareas grid que se ejecutan en las máquinas pueden ser ejecutadas bajo un ID de usuario especial, en nombre de los usuarios que envían las tareas. Los derechos de esos IDs de usuario deben ser establecidos de tal manera que esas tareas no pueden acceder a las partes de la máquina a las que los usuarios no están autorizados.

Al entrar en el grid, la identidad de los usuarios debe ser confirmada e introducida en la Autoridad de Certificación. El usuario y sus credenciales deben añadirse a la lista de usuarios usando el software apropiado. En algunos casos, el administrador debe propagar la información del usuario a varias máquinas del grid.

Hay que seguir un proceso similar para dar de alta máquinas de aportación

de recursos al grid. La identidad de la máquina es establecida y registrada con la Autoridad de Certificación. El administrador del grid debe llegar a un acuerdo con el administrador de la máquina acerca de los IDs de usuario, el software, los derechos de acceso?El administrador debe introducir la identificación, las credenciales, las direcciones y las características de los recursos de la máquina usando el software apropiado. En algunos casos, el administrador puede necesitar propagar manualmente esta información a otras máquinas del grid.

Los correspondientes procedimientos de eliminación de usuario y máquinas son ejecutados por el administrador.

1.1.5.4. La Autoridad de Certificación

Es crucial garantizar los niveles más altos de seguridad en un grid, porque éste está diseñado para ejecutar código y no solo compartir datos. Por consiguiente puede ser caldo de cultivo de virus, troyanos y otros ataques. La Autoridad de Certificación es uno de los aspectos más importantes a la hora de conseguir una seguridad fuerte en el grid. Las responsabilidades principales de la Autoridad de Certificación son:

- Identificar a las entidades que solicitan certificados.
- Emitir, eliminar y archivar certificados.
- Proteger el servidor de la Autoridad de Certificación.
- Mantener un espacio con identificadores únicos para los propietarios de certificados.
- Ofrecer certificados firmados para los que los necesitan.
- Actividad de registro.

La Autoridad de Certificación está basada en un sistema de encriptación de claves públicas. En este sistema, las claves se generan a pares, una pública y una privada.

1.1.5.5. Gestión de recursos

Otra responsabilidad del administrador es gestionar los recursos del grid. Esto incluye establecer permisos para que los usuarios del grid puedan usar los recursos así como controlar el uso de recursos e implementar un sistema de cuentas. Las estadísticas de uso son útiles para identificar tendencias en una organización que puede necesitar la adquisición de hardware adicional.

Algunos componentes de grid, normalmente los planificadores de tareas, proporcionan métodos para imponer prioridades y políticas de distintos tipos. Es responsabilidad del administrador configurarlas para orientarlas de la mejor manera posible a la consecución de los objetivos de la organización.

1.1.5.6. Compartición de datos

En grids pequeños, la compartición de datos puede ser muy fácil, usando los sistemas de ficheros distribuidos existentes, bases de datos o protocolos de transferencia de datos estándar. A medida que el grid crece y los usuarios se vuelven más dependientes de algún repositorio de almacenamiento de datos, el administrador debe pensar procedimientos para mantener copias de backup y replicas para mejorar el rendimiento.

1.1.6. Uso del grid: perspectiva del desarrollador de aplicaciones

Las aplicaciones grid pueden clasificarse en tres tipos:

- Aplicaciones que no permiten su uso en múltiples procesadores pero pueden ser ejecutados en máquinas diferentes.
- Aplicaciones que están diseñadas para su uso en múltiples procesadores.
- Aplicaciones que necesitan ser modificadas para ser usadas en un grid.

El último tipo es muy interesante para los desarrolladores de aplicaciones grid. Estos necesitaran herramientas para depurar y medir el comportamiento de las aplicaciones grid. Estas herramientas basadas en grid todavía están en una primera etapa. Puede ser útil para los desarrolladores configurar su propio pequeño grid y así poder usar los depuradores en cada máquina para controlar y observar el funcionamiento detallado de las aplicaciones.

Globus es más un toolkit de desarrollo para construir aplicaciones grid que un exhaustivo sistema grid. Tiene los componentes básicos necesarios para construir nuevos métodos para gestionar las operaciones de grid, medir, reparar y depurar aplicaciones grid.

1.1.7. El presente y el futuro

El Globus Toolkit es un conjunto de herramientas útiles para la construcción de un grid. Su fortaleza es un buen modelo de seguridad, preparado para la recopilación jerárquica de daros, y con las estructuras básicas para la implementación de un grid sencillo. Globus crecerá con el tiempo a través del trabajo de muchas organizaciones que están aumentando sus posibilidades.

La mayoría de los sistemas grid incluyen algún planificador de tareas, pero a medida que los grid abarcan mayores áreas, harán falta más metaplanificadores que pueden gestionar conjuntos de clusters y grids más pequeños.

Hoy en día, los sistemas grid están todavía lejos de proporcionar un almacenamiento y compartición de datos fiable, de alto rendimiento y recuperable automáticamente.

OGSA es un estándar abierto en la base de todas estás mejoras futuras de grid. OGSA estandarizará los interfaces grid que serán usados por los nuevos planificadores, agentes de cálculo autónomo, y cualquier servicio que sea desarrollado para grid. Esto hará más fácil ensamblar los mejores productos, aumentando el valor global de la computación grid.

1.1.8. Lo que grid no puede hacer

Debemos ser cautos sobre grid, después del entusiasmo anterior. Grid no puede hacer que una aplicación se ejecute 1000 veces más rápido sin comprar más máquinas o software. No toda aplicación puede ejecutarse en un grid. Algunos tipos de aplicaciones simplemente no pueden ser paralelizadas. En otras, el trabajo que se necesita para modificarlas para conseguir mayor productividad es inabarcable. La configuración de un grid puede afectar mucho al rendimiento, fiabilidad y seguridad de la infraestructura de computación de una organización. Por estas razones, es importante que los usuarios comprendan hasta donde ha evolucionado grid a día de hoy y que características vendrán en un futuro próximo.

1.2. Arquitectura y consideraciones sobre seguridad

1.2.1. Consideraciones de aplicación

Mientras que un grid puede ofrecer múltiples ventajas, cualquier aplicación dada puede no beneficiarse necesariamente de un grid.

El uso más sencillo del grid es simplemente ejecutar la aplicación en algún lugar cuando tu máquina está demasiado ocupada. Casi cualquier tipo de aplicación se puede ejecutar en un entorno de grid de este modo. No observarás mejoras espectaculares en el rendimiento a no ser que la máquina en la que se ejecuta sea mucho más rápida que la tuya. Las aplicaciones que se pueden ejecutar por lotes son las más sencillas de manejar. Las aplicaciones que necesitan interacción por medio de interfaces gráficas de usuario son más difíciles de ejecutar en un grid.

Las aplicaciones específicamente diseñadas para usar múltiples procesadores u otros recursos del grid se beneficiarán más.

1.2.1.1. Consideraciones sobre la CPU

Si tenemos en cuenta la computación grid, debemos examinar las aplicaciones que consumen grandes cantidades de tiempo de CPU. Las siguientes cuestiones nos ayudarán a determinar si una aplicación puede ser ejecutada en un grid o si no se puede beneficiar.

El paso más importante para permitir a una aplicación usar grid es determinar si los cálculos pueden hacerse en paralelo o no. Mientras que los cluster HPC (High Performance Computing) suelen ser usados para tratar la ejecución de aplicaciones que pueden usar procesamiento paralelo, los grids proporcionan la capacidad de ejecutar esas aplicaciones a través de un conjunto de clusters heterogéneos y geográficamente dispersos. La aplicación puede aprovecharse del amplio conjunto de recursos del grid. No todos los problemas pueden transformarse en cálculos paralelos. Si en el algoritmo, cada resultado depende de un resultado anterior, es necesario encontrar otro algoritmo. Por ejemplo el proceso de sumar una larga lista de números. Observamos que por la propiedad asociativa, podemos dividir la lista en siete, y siete programas sumaran cada lista y un octavo sumara los resultados de cada uno de los siete.

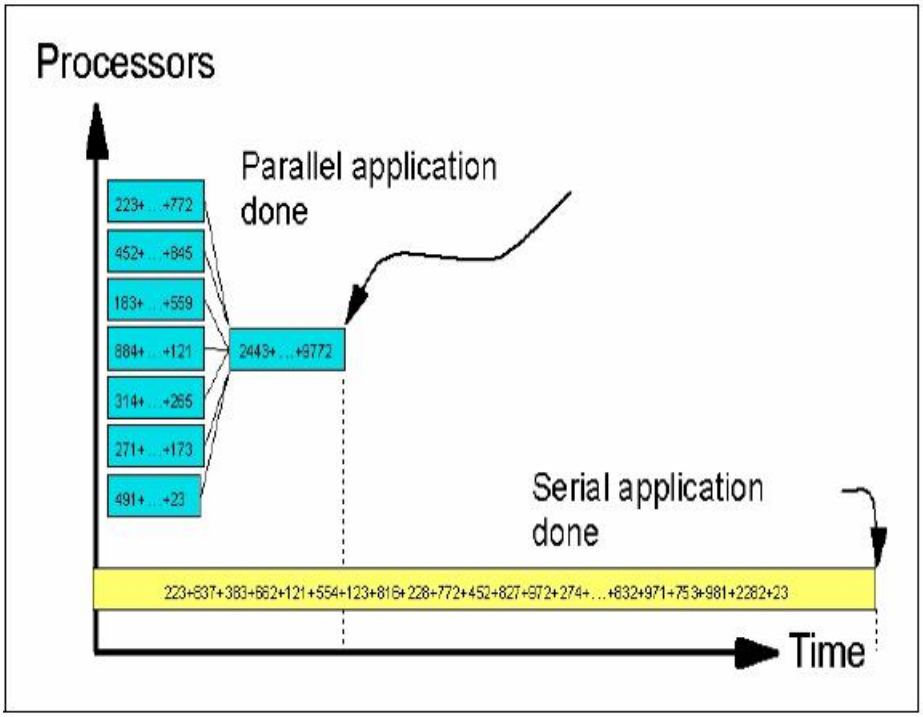

Figura 1.3: Aplicación paralelizable

Por otro lado, algunos cálculos no pueden ser reescritos para ejecutarse en paralelo. Por ejemplo, no hay formulas simples que muestren donde estarán situados tres o más cuerpos en movimiento después de un tiempo determinado, cuando sus gravedades les afecten. Este tipo de cálculos se hacen simulando el movimiento de los cuerpos, usando las leyes de Newton a pequeños intervalos de tiempo, y calculando como las fuerzas y los cuerpos afectarán a cada uno, dando la nueva posición de los objetos después de cada pequeño incremento de tiempo.

Esto se repite muchas veces hasta que se alcanza el tiempo deseado. Cada cálculo depende del anterior. Debido a que los incrementos de tiempo no son infinitamente pequeños, después de muchos incrementos, los pequeños errores se empiezan a sumar. La posición final de los objetos calculada puede ser errónea. Para mejorar la precisión en esos cálculos, hacemos los incrementos mucho más pequeños. Esto aumenta el número de cálculos y el tiempo global de cálculo.

A menudo, una aplicación puede ser una mezcla de cálculos independiente y cálculos dependientes. Es necesario analizar la aplicación para ver si hay algún modo de dividir el trabajo.

Otro enfoque para reducir las dependencias de datos es buscar modos de usar cálculos redundantes. Si la dependencia se encuentra en un subconjunto de cálculos anteriores, puede resultar beneficioso, recalcular los resultados en lugar de esperar que lleguen de otra tarea. Si la dependencia está en un cálculo que tiene una respuesta de tipo booleano, puede ser aconsejable calcular las dos ramas y luego descartar la rama errónea cuando la dependencia es conocida finalmente.

Esta técnica puede ser llevada al extremo de varias formas. Por ejemplo en dependencias de dos bits de datos, podemos hacer cuatro copias de los siguientes cálculos, una con cada posibilidad. Iterando podemos hacer n copias para n bits de dependencia. A medida que n crece, aumenta rápidamente el coste de calcular todas las combinaciones posibles. Por eso debemos utilizar reglas heurísticas para suponer cual de las combinaciones es la correcta.

La computación grid se adecua perfectamente a los problemas de espacio de parámetros, aunque no sea posible su paralelización. El paralelismo aparece al ejecutar muchas tareas diferentes que cubren el espacio de parámetros. Algunos productos de grid proporcionan herramientas para simplificar el envío de subtareas en una aplicación de este tipo.

Muchas veces, una aplicación que fue desarrollada para un único procesador puede no ser organizada para cálculo paralelo. Puede haber sido escrita de modo que su alcance su máxima eficiencia en una máquina con un solo procesador. Sin embargo, puede haber otros métodos o algoritmos que, si bien son menos eficientes que aquellos, sean mejores para dividirlos en subcálculos independientes. Este nuevo algoritmo puede usar más eficientemente un gran número de procesadores. Por eso, otra opción para hacer que una aplicación se adecue a grid es revisar las opciones descartadas cuando se trato el problema, porque quizá alguna de esas opciones sea mejor para usar en grid.

En una aplicación distribuida, los resultados parciales o las dependencias de datos pueden conocerse mediante la comunicación entre subtareas. Esto es, una tarea puede calcular un resultado intermedio y transmitirlo a otra tarea del grid. Si es posible, se puede pensar si es más eficiente recalcular el resultado intermedio en el punto donde es necesario en vez de esperar por él desde otra tarea.

1.2.1.2. Consideraciones sobre datos

Cuando se dividen las aplicaciones para su uso en un grid, es importante tener en cuenta la cantidad de datos que se necesita que sean enviados al nodo que realiza un cálculo y el tiempo necesario para el envío. Lo ideal sería una aplicación se puede dividir en pequeñas unidades de trabajo que necesiten pocos datos de entrada y produzcan pequeñas cantidades de salida. El envío de estos datos junto con el fichero ejecutable al nodo de grid que realiza el trabajo es parte de las funciones de la mayoría de los sistemas grid. Sin embargo, en la mayoría de los casos, están involucradas grandes cantidades de datos de entrada y de salida.

Cuando la aplicación grid se divide en subtareas, a menudo los datos de entrada es un gran conjunto de datos fijos. Esto ofrece la posibilidad de compartir estos datos en lugar de entregar el conjunto con cada subtarea. Sin embargo hay que tener en cuenta que incluso con un sistema de ficheros compartido, los datos se envían a través de la red. El objetivo es colocar los datos compartidos cerca de las tareas que los necesitan. Si los datos se van a usar más de una vez, se pueden replicar, en la medida que el espacio lo permita.

Si hay más de una copia de los datos almacenada en el grid, es importante hacer que las subtareas accedan a la copia que tengan más cerca. Esto resalta la importancia de un servicio que proporcione información sobre el grid. Más aun, se debe tener cuidado de que la red no se convierta en un cuello de botella para alguna aplicación. Si cada tarea procesa los datos muy rápido y siempre está esperando nuevos datos, entonces la compartición no es el mejor modelo si la velocidad de la red no es al menos como la de los discos.

Las aplicaciones que acceden a los datos necesitan que sean previsibles, por eso hay varias técnicas que pueden ser usadas para mejorar su rendimiento en el grid. Si cada subtarea necesita acceder a todos los datos, entonces las copias compartidas puede ser la mejor solución. Si cada parte de los datos es examinada solo una vez, no.

Uno de los problemas más complicados al duplicar bases de datos que cambian rápidamente es mantener la sincronización. Si sólo cambian partes de la base de datos se puede pensar en tener versiones de la base de datos en memoria.

Puede ser necesario ejecutar una aplicación redundantemente por razones de fiabilidad, por ejemplo. La aplicación puede ejecutarse simultáneamente en diferentes partes del grid para reducir las posibilidades de que un fallo haga

que la aplicación no pueda completar su trabajo. Si la aplicación modifica bases de datos o tiene otras comunicaciones de datos, necesita estar diseñada para tolerar la redundancia de datos causada por la ejecución de varias copias de la aplicación. De otro modo, los resultados calculados pueden ser erróneos

1.2.2. Seguridad

1.2.2.1. Introducción a la seguridad de grid

Los requisitos de seguridad son fundamentales en un diseño grid. Los componentes de seguridad básicos del Globus Toolkit proporcionan los mecanismos para la autentificación, autorización y confidencialidad en la comunicación entre ordenadores grid. Sin esta funcionalidad, la integridad y confidencialidad de los datos procesados en el grid estaría en peligro. Para asegurar adecuadamente el entorno grid, hay varias herramientas y tecnologías disponibles.

Fundamentos de seguridad La seguridad necesita tres servicios fundamentales: autentificación, autorización y encriptación. Un recurso grid debe autentificarse antes de que se pueda hacer cualquier comprobación. Una vez que los recursos grid han sido autentificados en el grid, se puede garantizar al usuario ciertos derechos al acceder a un recurso grid. Esto, sin embargo, no evita que los datos en camino entre recursos grid puedan ser capturados o modificados. El servicio de seguridad que asegura que esto no ocurra es la encriptación.

Conceptos importantes en seguridad de grid Encriptación simétrica: Usar la misma clave secreta para la encriptación y desencriptación de datos

Encriptación asimétrica: Usar dos claves diferentes para la encriptación y desencriptación. La técnica de encriptación mediante clave pública es el ejemplo más sencillo de uso de clave pública y clave privada

Secure Socket Layer/Transport Layer Security (SSL/TLS): En esencia son el mismo protocolo, pero se les referencia de manera diferente.

Infraestructura de clave pública (PKI): Los diferentes componentes, tecnologías y protocolos que crean el entorno PKI

Autentificación mutua: En vez de usar un repositorio LDAP para guardar la clave pública, dos elementos que se quieran comunicar el uno con el otro usan la clave pública guardada en el certificado digital para autentificarse.

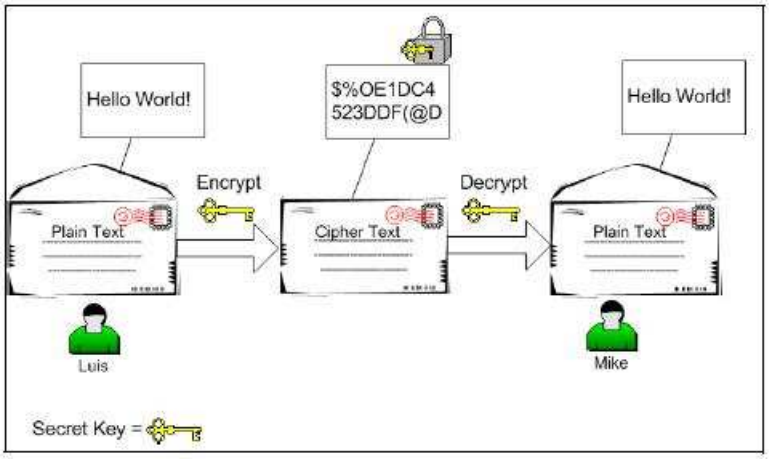

Figura 1.4: Encriptación de clave pública

Encriptación simétrica de claves La encriptación simétrica de claves esta basada en el uso de una clave secreta compartida para realizar tanto la encriptación como la desencriptación de datos. Para asegurar que los datos se leen solo por el emisor y el receptor, la clave tiene que ser distribuida de manera segura entre las dos partes y ninguna más. Si alguien tuviera acceso a la clave secreta que se usa para encriptar datos, podría desencriptar la información. Esta forma de encriptación es mucho más rápida que la encriptación asimétrica

Encriptación asimétrica de claves Otro método común es la criptografía de clave pública. Se usa un par asimétrico de claves. La clave para la encriptación es diferente de la usada para la desencriptación. La criptografía de clave pública necesita que los propietarios de las claves protejan sus claves privadas, mientras que sus claves públicas no son completamente secretas y pueden ser puestas a disponibilidad de todos. Normalmente, la clave pública se encuentra en el certificado digital.

El par de claves asimétrico se genera mediante un cálculo que comienza encontrando dos números primos de gran tamaño. Aunque la clave pública se distribuya, es prácticamente imposible que algún ordenador calcule la clave privada para esa clave pública. La seguridad deriva del hecho de que es muy difícil la factorización de números de cientos de cifras.

La Autoridad de Certificación Una autoridad de certificación implementada adecuadamente tiene varias responsabilidades, vistas en un punto anterior. Dentro de algunos entornos PKI, una Autoridad de Registro (RA) trabaja junto a la CA para ayudar en algunas de esas tareas. La RA es responsable de aceptar o rechazar las solicitudes de certificado de claves públicas y trasladar la información del usuario a la CA.

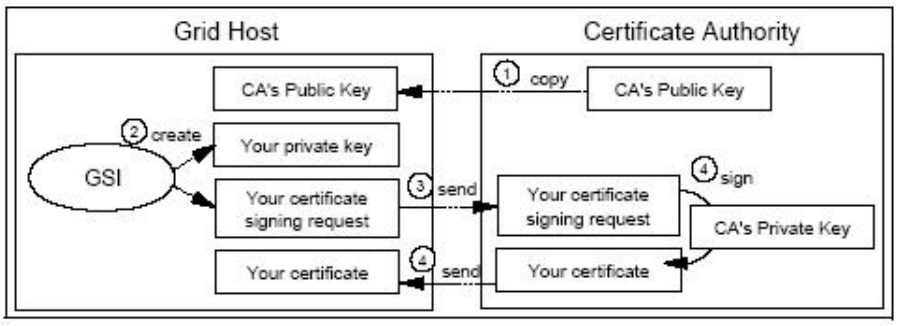

Figura 1.5: Autoridad de Certificación

Uno de los temas fundamentales dentro de un entorno PKI es garantizar la fiabilidad del sistema. Antes de que una CA pueda firmar y emitir certificados para otros, tiene que hacer lo mismo para ella misma, de modo que su identidad pueda estar representada por su propio certificado. Para ello:

1. La CA genera aleatoriamente su propio par de claves
2. La CA protege su clave privada
3. La CA crea su propio certificado
4. La CA firma su certificado con su clave privada

Certificados digitales Los certificados digitales son documentos digitales que asocian un recurso grid con su clave pública específica. Un certificado es una estructura de datos que contiene una clave pública y los detalles pertinentes sobre el propietario de dicha clave.

Los certificados digitales, también llamados certificados X.509, actúan a modo de pasaportes, proporcionan un medio para identificar los recursos

grid. Un certificado digital puede copiarse y distribuirse sin restricciones. No suelen contener información confidencial y su libre distribución no crea un riesgo de seguridad.

El factor a tener en cuenta acerca de los certificados digitales es que la CA certifica que la clave pública adjunta pertenece a la entidad señalada en el certificado. La implementación técnica hace que sea extremadamente difícil alterar cualquier parte del certificado sin ser detectado. La firma de la CA proporciona un control de integridad al certificado digital.

1.2.2.2. Infraestructura de seguridad de grid

Consiguiendo acceso al grid Para construir un entorno usando las componentes GSI (Grid Security Infraestructura), hay que crear conjuntos de claves para la criptografía de claves públicas y pedir un certificado a la Autoridad de Certificación y una copia de la clave pública de la CA. Los pasos para conseguir la comunicación GSI son:

1. Copiar la clave pública de la CA en tu host de grid con el que activas GSI
2. Crear tu clave privada y una solicitud de certificado
3. Enviar tu solicitud de certificado a la CA por *e-mail* u otro modo más seguro
4. CA firma la petición y la devuelve

Cuando este proceso ha sido completado y has recibido tu certificado digital firmado, habrá tres ficheros importantes en tu host de grid. Son:

- La clave pública de la CA
- La clave privada del host de grid
- El certificado digital del host de grid

Para proporcionar una autentificación segura y comunicación para tu computador grid, nadie más debe tener acceso a la clave privada.

Cuando es necesario comunicarse con una aplicación de otro ordenador de grid, y se quieren garantías de que los datos de ese host provienen realmente

de ese host, se pueden usar las funciones de autentificación de GSI. También tenemos la opción de que el recurso de grid nos de acceso a los recursos en nuestro nombre. En este caso usamos la función de autorización de grid.

En situaciones en las que se quiera distribuir tareas a máquinas remotas del grid y permitir que éstas distribuyan sus tareas hijas a otras máquinas bajo tu política de seguridad, se puede usar la función de delegación de GSI. Se pueden crear proxys para delegar la autoridad. Este proxy actúa como tu mismo, y envía solicitudes a otros hosts en tu nombre.

Comunicación de la seguridad de grid Es importante conocer las funciones de comunicación dentro de Globus Toolkit. Por defecto la comunicación se basa en la autentificación mutua de certificados digitales y SSL/TLS

El proceso de autentificación mutua comienza cuando dos recursos grid quieren compartir sus recursos. En vez de usar un repositorio de claves, cada recurso de grid se autentifica a los demás gracias a su certificado digital. Antes de que el servidor permita acceso a los clientes, necesita autentificarlo.

1.2.2.3. Seguridad de la infraestructura de grid

Seguridad física El entorno físico de un sistema forma parte de su infraestructura. Si los servidores se guardan en una habitación abierta, no importa como de seguras sean las aplicaciones o como de complicados sean los algoritmos de encriptación, el servidor puede ser apagado por cualquiera.

El servidor de la CA debe situarse en una habitación cerrada y robusta. Todos los accesos a ella deben estar restringidos y sólo el personal relacionado con la CA pueda entrar. Las fuentes de alimentación no se deben apagar nunca.

Seguridad del sistema operativo Se deben comprobar los ficheros de configuración de cada sistema operativo y cada componente middleware dentro del alcance que el proyecto permita. Entre otras tareas, se deben eliminar los procesos innecesarios de los servidores, así como los usuarios y grupos innecesarios, se debe restringir el acceso al directorio de globus, se deben realizar inspecciones periódicas al sistema operativo, y se debe activar la protección antivirus.

Firewalls Los firewalls se usan dentro de un entorno de red para separar lógicamente varios conjuntos de ordenadores que necesiten seguridad adicio-

nal. En un entorno grid ocurre lo mismo. El uso de firewalls dentro de un diseño grid ayuda a restringir el acceso por red a los ordenadores. El firewall es una parte importante de la infraestructura de seguridad, por eso debe ser previamente analizado antes de su implementación.

Detección de intrusos Una opción recomendada para mejorar la seguridad de los ordenadores del grid es proveerlo de un producto de detección de intrusión en el host (IDS). La detección de intrusos puede añadir una gran defensa ante alguien que esté manipulando ficheros y no debería estar haciéndolo. Si el IDS detecta algún fichero modificado en el servidor, puede enviar una señal de alerta al panel central de control.

1.2.2.4. Riesgos potenciales de seguridad

Construir un entorno PKI proporciona los servicios necesarios junto al GSI para diseñar un grid seguro. Sin embargo, esto no garantiza que no podamos encontrar riesgos de seguridad.

Vulnerabilidades en el PKI Solo por haber construido un entorno PKI no significa que la red sea completamente segura. Hay varias vulnerabilidades que hay que tener en cuenta.

Dentro de un entorno PKI, siempre hay que preocuparse de la ubicación de las claves privadas y de posibles robos de certificados digitales. Hay que tener en cuenta:

- Obtención de certificados de manera fraudulenta
- Uso no autorizado de la clave privada asociada a un certificado válido
- Uso de un clave de una CA para firmar certificados fraudulentos o destruir una clave privada

1.2.2.5. Vulnerabilidades en el servidor de grid

Cualquier servidor o estación de trabajo que participa en el grid constituye una vulnerabilidad potencial ante un hacker externo o interno. Es muy importante proteger cualquier ordenador del grid de cualquier red que no necesite acceder al grid. Deben ser protegidos los siguientes aspectos del servidor de grid:

- Seguridad física que limite la probabilidad de que un alguien pueda acceder físicamente al servidor
- Proteger algunos directorios del directorio globus
- Cualquier modificación del fichero gridmap
- Vulnerabilidades en las aplicaciones que se ejecutan en el grid.

1.2.3. Diseño

1.2.3.1. Construcción de una arquitectura grid

La idea de diseñar una solución grid suele provenir de una inversión en infraestructura. Sin embargo, una solución grid no es simplemente instalar software para asignar los recursos demandados. Hay distintos tipos de grids diseñados para cubrir distintas necesidades y requisitos. También hay varias topologías diseñadas para satisfacer diferentes restricciones geográficas y requisitos de las redes.

Una vez que se conocen los requisitos funcionales y no funcionales, el arquitecto debe ser capaz de seleccionar el tipo de grid y la mejor topología necesaria para satisfacer la mayoría de los requisitos de negocio.

Es importante empezar construyendo la estructura básica del diseño. En vez de intentar diseñar la solución final de una vez, es mejor construir el grid en varias fases. El objetivo para la fase inicial es conseguir una solución intragrid, que es en esencia una caja de arena grid que proporciona un conjunto básico de servicios grid. Esta solución proporciona un lugar simple construido sobre las componentes del núcleo de grid, como el modelo de seguridad, servicios de información, gestión de la carga de trabajo y los dispositivos del host.

Un modo sencillo de comenzar el diseño es empezar por el modelo de seguridad de grid. El modelo de seguridad de grid está normalmente construido sobre una estructura PKI y es la base para la autentificación de usuarios en grid. Conocer el tipo de grid, la topología y el modelo de seguridad deseado es fundamental para realizar un diseño de alto nivel de grid a medida. Dado que la característica principal de grid es que la red y la infraestructura hardware son compartidas por muchos usuarios de muchos lugares, es lógico que las primeras decisiones sobre la arquitectura se basen en conceptos de seguridad.

El primer paso para el diseño es construir una representación gráfica de las componentes del grid. Las siguientes fases se centraran en el siguiente nivel de arquitectura.

Básicamente, el diseño de la arquitectura grid debe ofrecer lo siguiente:

- El "plano" del diseño conceptual detallado.
- El uso de estándares abiertos indicados por la estructura del grid.
- Una vista multidimensional escalonada y por capas de la infraestructura del grid.
- Los componentes de middleware y subsistemas para la integración de las infraestructuras de grid.
- Un diseño para la comunicación tanto de personal de la empresa como personal técnico.
- La distribución de aplicaciones y subsistemas.
- Un método de identificación para componentes técnicos y de infraestructura.

1.2.3.2. Modelos de arquitecturas grid

Existen diferentes tipos de arquitecturas grid para solucionar diferentes tipos de problemas en las empresas. Conocer los objetivos de las empresas ayudará a elegir el tipo adecuado de estructura grid. La elección de un tipo de grid determinado, tendrá un impacto directo en el diseño de la solución grid.

Grid computacional Un grid computacional junta la capacidad de procesamiento de un conjunto distribuido de sistemas. El ejemplo más conocido es la grid del programa SETI. Los ciclos sin utilizar de los ordenadores personales del grid de SETI se combinan para crear un grid computacional usado para analizar transmisiones de radio del espacio exterior.

Las características principales de los grid computacionales son:

Están formados por clusters de clusters

Permite scavenging para un mejor uso de los recursos

Proporciona el poder de cálculo necesario para procesar grandes tareas

Satisface el requisito de acceso instantáneo a los recursos solicitados

Grid de datos Los grids de datos se centran en proporcionar acceso seguro a pools de datos distribuidos y heterogéneos. A través de la colaboración, los grids de datos pueden incluir un nuevo concepto, las bases de datos federadas.

Los grids de datos aprovechan los recursos de almacenamiento y de red situadas en distintos dominios administrativos, respetan las políticas locales y globales acerca de cómo pueden ser usados los datos, planifican los recursos eficientemente y proporcionan un acceso rápido y fiable a los datos.

1.2.3.3. Topologías grid

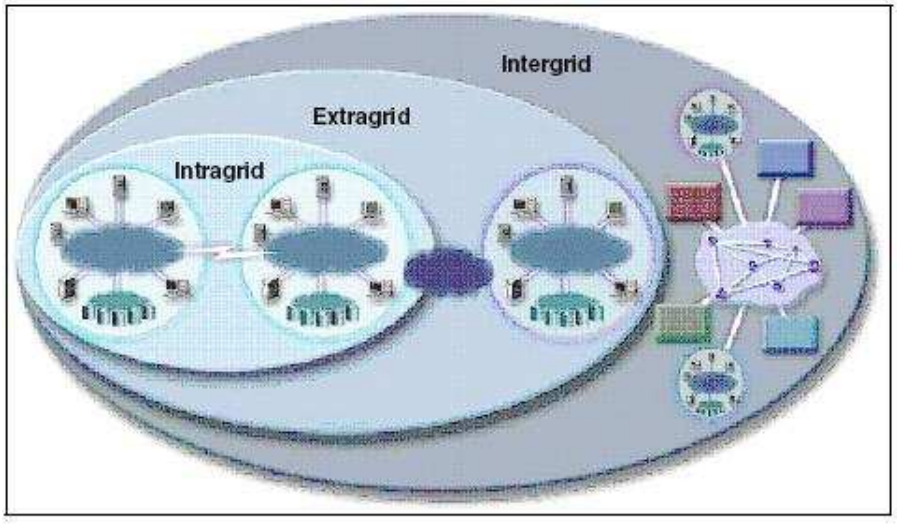

Figura 1.6: Intragrid, Extragrid e Intergrid

Desde el punto de visto topológico, existen los siguientes tipos de grid.

- Intragrids
  - Organizaciones sencillas
  - No hay integración
  - Un solo cluster
- Extragrids

    - Organizaciones múltiples
    - Integración
    - Múltiples clusters

- Intergrids

    - Muchas organizaciones
    - Múltiples socios
    - Varios clusters múltiples

Capítulo 2

Web Services

2.1. Introducción

Para facilitar las tareas de comunicación, las aplicaciones de las empresas deben comunicarse unas con otras y compartir datos. El desarrollo de XML ha incrementado la posibilidad de comunicación de un sistema a otro. Los web services son programas que usan XML para intercambiar información con otro software por medio de protocolos de Internet comunes. Básicamente, un web service se comunica con una red para proporcionar un determinado conjunto de operaciones, que otras aplicaciones pueden invocar. Esto significa que una aplicación que esté en un ordenador pueda enviar peticiones y, posiblemente, recibir respuestas de aplicaciones de otros ordenadores. Los web services pueden intercambiar información a través de varios protocolos de Internet, pero normalmente utilizan el protocolo de transferencia de hipertexto (http), el protocolo más importante de la Web.

Los web services poseen ciertas características que los diferencian de otros modelos de computación. Primero, los web services son programables, encapsulan una tarea, cuando una aplicación les pasa unos datos, el servicio los procesa y si es necesario, envía una respuesta a la aplicación. Segundo, los web services están basados en XML, que al ser un estándar abierto, basado en texto, permite a los web services comunicarse con otras aplicaciones incluso si estas están escritas en distintos lenguajes y se ejecutan en distintas plataformas. Los web services, además, contienen información que explica que hacen y como otras aplicaciones pueden acceder a ellos y usarlos. Esta información suele estar escrita en WSDL (Web Services Description Language),

un estándar basado en XML.

2.1.1. Tecnologías

2.1.1.1. XML

Desarrollado a partir del Standard Generalized Markup Language (SGML), XML es un estándar ampliamente acertado para descripción de datos y creación de lenguajes de marcado. A diferencia de otras tecnologías, que comienzan siendo marcas registradas y se convierten en estándar, el XML fue concebido por el W3C como un tecnología estándar abierta.

La independencia de datos, o la separación del contenido de su presentación, es la característica principal de XML. Debido a que los documentos XML describen sólo datos, cualquier aplicación que comprenda XML (sin tener en cuenta el lenguaje o la plataforma de la aplicación), tiene la capacidad de dar formato a XML de varias formas. Teniendo en cuenta esto, los desarrolladores de software están integrando XML en sus aplicaciones para mejorar la funcionalidad Web y la interoperabilidad. Los documentos XML contienen datos, pero no instrucciones de formato, por lo tanto las aplicaciones que procesan documentos XML deben decidir como mostrar los datos del documento. Por ejemplo, una PDA puede traducir un documento XML de modo diferente a como lo haría un inalámbrico o un ordenador personal.

Un parser XML es un programa que comprueba la sintaxis del documento XML y hace que los datos de dicho documento sean útiles para las aplicaciones. Si la sintaxis es correcta, entonces el documento cumple la especificación XML 1.0 y por lo tanto se le considera "bien formado". Si la sintaxis no es correcta el parser genera uno o más errores y las aplicaciones no pueden usar ese documento. Un documento XML puede referenciar a otro documento que define la estructura de dicho documento. Este otro documento puede ser un DTD (Document Type Definition). Cuando un documento XML referencia a un DTD, algunos parsers pueden leer el DTD y comprobar que el XML sigue la estructura que el DTD define. Si el XML se ajusta a la estructura, entonces es valido.

A medida que las aplicaciones se orientan hacia la Web, parece que XML se convertirá en la tecnología universal para representar datos intercambiados entre aplicaciones web. Este nivel de interoperabilidad hace que XML sea la tecnología ideal para permitir web services.

El siguiente fragmento de código es un ejemplo de XML.

```
<memo id= "51213">
<message>Hola Mundo</message>
</memo>
```

En XML, los datos se marcan usando tags, que son nombres encerrados entre <>. Los tags se usan a pares para delimitar el comienzo y el final de la marca. Un tag que comienza una marca se llama start tag y uno que la termina end tag. Un ejemplo de start tag es <message>. Los end tags se diferencian de los start tags en que contienen una barra (/) como por ejemplo </message>.

Las unidades elementales de marcado se llaman elementos. Los elementos son los bloques más importantes en la construcción de un documento XML. Los documentos XML contienen un elemento (llamado elemento raíz, en el ejemplo memo) que contiene a todos los demás elementos del documento. Los elementos se pueden anidar formando jerarquías (el elemento raíz está en el nivel más alto de la jerarquía). Por ejemplo, message es el elemento anidado. Organizar de esta manera a los elementos permite a los autores crear relaciones jerárquicas entre datos. Los documentos XML pueden contener cualquier número de elementos. Además de ser puestos entre tags, los datos pueden ponerse en atributos, que se sitúan en los start tags y proporcionan información adicional sobre los elementos. Los elementos pueden tener cuantos atributos quieran. En el ejemplo,

```
<memo id= "51213">
```

se le asigna al atributo id el valor 51213. Los elementos XML y los nombres de los atributos pueden ser de cualquier longitud y pueden contener letras, dígitos, subrayados y guiones, pero deben comenzar con una letra o un subrayado.

En los lenguajes de programación orientados a objetos como Java o C++, se agrupan clases en paquetes. Estos paquetes previenen la colisión de nombres entre identificadores definidos por el usuario. Por ejemplo, una clase llamada Transaction puede representar una transacción monetaria entre dos empresas; sin embargo un banco puede usar la clase Transaction para representar una transacción monetaria con el Banco de España.

XML admite namespaces, que proporcionan un modo de identificar de manera única los elementos XML. Debido a que los creadores de documentos hacen su propio lenguaje de marcado y vocabulario, los namespaces son necesarios para agrupar los elementos de un vocabulario.

Los creadores de documentos crean sus propios prefijos del namespace y URIs (Uniform Resource Identifiers), que representan la ubicación o dirección de un objeto o un recurso en una red. Por ejemplo, una URL es un tipo de URI. El uso de URLs asegura que los namespaces son únicos.

2.1.1.2. SOAP

SOAP (Single Object Acess Point) es uno de los estándars más comunes usados para desarrollar web services.

El objetivo de SOAP es permitir transferencia de datos entre sistemas distribuidos sobre una red. Cuando una aplicación se comunica con un web service, los mensajes SOAP son la forma más común para el intercambio de datos entre los dos sistemas. Un mensaje SOAP enviado a un web service invoca a un método proporcionado por el servicio, haciendo que el servicio ejecute una tarea en particular. El servicio usa la información contenida en el mensaje SOAP para realizar su función. Si es necesario, el web service devuelve el resultado por medio de otro mensaje SOAP.

Al ser un protocolo de comunicación basado en XML, SOAP consiste básicamente en un conjunto de esquemas XML estandarizados. Los esquemas definen un formato para la transmisión de mensajes XML en una red, incluyendo los tipos de mensajes que el mensaje puede incluir y el modo en que el mensaje debe estar estructurado para que el servidor del otro extremo de la comunicación pueda interpretarlo correctamente. SOAP se asienta sobre un protocolo de Internet, como http, y puede ser usado para transferir datos a través de la Web y otras redes. El uso de http permite a los web services comunicarse atravesando los firewalls, ya que la mayoría de los firewalls están diseñados para aceptar peticiones http.

En el modelo de intercambio de mensajes SOAP, los nodos SOAP procesan los mensajes SOAP. Un nodo que envía un mensaje SOAP se denomina emisor SOAP, mientras que el que recibe el mensaje se llama receptor SOAP. Cuando un nodo procesa un mensaje, ese nodo se llama actor SOAP. Los actores SOAP se identifican por un nombre de actor, que es una URI. Un mensaje SOAP usa el nombre de actor SOAP para identificar a los receptores y los intermediarios.

Este modelo representa el modo más simple de transmisión SOAP. Usando como base el modelo, los desarrolladores pueden crear arquitecturas de comunicación más complejas. Por ejemplo, pueden crear un modelo request/response.

La mayor parte de la especificación SOAP define la estructura de un

mensaje SOAP. SOAP encapsula los datos en mensajes que se transfieren desde y hacia los web services. Cada mensaje SOAP contiene un elemento de recubrimiento inicial, envelope, que está compuesto por una cabecera opcional y un cuerpo obligatorio. El envelope necesita la información relacionada con el namespace y los esquemas del mensaje. El envelope no define los contenidos de la cabecera y el cuerpo. La cabecera SOAP puede contener información acerca del mensaje, instrucciones de parking para los nodos que reciben el mensaje e información de seguridad. Un mensaje SOAP no tiene porque viajar siempre directamente desde su emisor hasta su receptor. Por eso la cabecera puede contener cierta información de enrutamiento. El cuerpo de un mensaje SOAP describe el propósito del mensaje SOAP y contiene el *payload* (los datos e instrucciones destinados a la aplicación receptora). Por ejemplo, el cuerpo puede incluir instrucciones para las tareas que el receptor debe realizar, como llamadas a algún método, o puede incluir información que deba ser procesada por una aplicación. Los datos están marcados como en XML y se ajustan a un determinado formato, que está definido en los esquemas mencionados anteriormente.

La especificación SOAP proporciona reglas que describen como pueden estar representados tipos específicos de datos en un mensaje SOAP. Estas reglas, conocidas como codificación SOAP, permiten a las aplicaciones que reciben mensajes SOAP reconocer el formato de los datos del mensaje, y por consiguiente procesarlos. A pesar de que la especificación define un conjunto de reglas de codificación, los desarrolladores pueden usar cualquier método de codificación, siempre que indiquen que reglas están usando. El estilo de codificación SOAP especifica las reglas para definir los tipos de datos de los datos individuales del mensaje SOAP. La cabecera o el cuerpo SOAP pueden tener el atributo encodingStyle que es el atributo que contiene una URI que apunta a las reglas de codificación. La codificación SOAP admite tipos simples, como strings y enteros, así como tipos complejos.

Hay varios protocolos que se pueden usar en vez de SOAP para permitir web services. Por ejemplo, XML-RPC es una tecnología más antigua que proporciona una funcionalidad similar. No obstante, la mayoría de los productores de software eligen SOAP por encima de otras posibles tecnologías. Hay varias razones técnicas para que la industria utilice SOAP. Sin embargo, es importante resaltar las principales ventajas de SOAP (simplicidad, extensibilidad, interoperabilidad. Los mensajes SOAP básicos no necesitan grandes cantidades de código, y se necesita muy poco software adicional para enviar o recibir mensajes SOAP. SOAP también proporciona mecanismos que

permite a los desarrolladores extender el estándar para adecuarse a sus necesidades específicas. Más aun, debido a que SOAP usa XML para comunicarse con http, SOAP puede ser usado en teoría para transferir datos entre dos sistemas que estén conectados a Internet, sin tener en cuenta los lenguajes de programación, los sistemas operativos y las plataformas hardware.

2.1.1.3. WSDL

Otro estándar que juega un papel crucial en los web services es WSDL (Web Service Definition Language). Hemos mencionado anteriormente que una característica principal de los web servicies es que se describen a si mismos, lo que significa que cada web service va acompañado de información que permite a los desarrolladores usar el web service. Estas descripciones suelen estar escritas en WSDL, un lenguaje basado en XML a través del cual un web service puede transmitir a otras aplicaciones los métodos que el servicio proporciona y como pueden acceder a dichos métodos.

Cuando SOAP y otras tecnologías de web services fueron desarrolladas, los fabricantes de software se dieron cuenta de que las aplicaciones que llamaban a servicios a través de la red necesitarían información sobre un servicio específico antes de interactuar con él. Sin embargo, cada fabricante empezó a crear su propio método de descripción, dando lugar a descripciones incompatibles unas con otras. WSDL surge cuando Microsoft e IBM deciden combinar sus tecnologías de descripción en un estándar universal.

Casi todo web service publicado en Internet está acompañado por un documento WSDL asociado, que enumera las prestaciones del servicio, sitúa su lugar en la Web y proporciona instrucciones sobre su uso. Un documento WSDL define los tipos de mensajes que un web service puede enviar y recibir, así como la especificación de los datos que una aplicación debe proporcionar al web service para que éste realice el trabajo. Los documentos WSDL también proporcionan información técnica específica sobre cómo conectar y comunicar con los web services a través de http u otros protocolos de comunicación.

Es importante comprender que WSDL es un lenguaje pensado para ser leído por aplicaciones más que por humanos. A pesar de que la estructura de los documentos WSDL puede parecer compleja, los computadores que comprenden WSDL pueden procesar los documentos y extraer la información que necesitan. Más aún, la mayoría de las herramientas de desarrollo de web services generan documentos WSDL automáticamente. Esto significa que, si un programador desarrolla un web service, el software usado para construir

el servicio crea el documento WSDL apropiado para ese servicio. Por consiguiente, no es necesario para los desarrolladores comprender la sintaxis de WSDL por completo cuando construyen y despliegan web services.

2.1.1.4. UDDI

El tercer estándar principal de los web services, UDDI (Universal Description, Discovery and Integration) permite a los desarrolladores publicar y ubicar web services en una red. Fue diseñado por Microsoft, IBM y Ariba, y empezó siendo un modo de intercambios B2B para que los usuarios intercambiaran información sobre sus negocios. UDDI define un formato basado en XML en el cual las compañías pueden describir sus prestaciones electrónicas y sus procesos de negocio; la especificación también proporciona un método estandarizado para el registro y ubicación de las descripciones en una red, como por ejemplo Internet. Parte de la información que las compañías pueden suministrar son datos respecto a los web services disponibles. Las compañías pueden almacenar su información en registros privados UDDI, que son accesibles solo por socios, o en registros públicos UDDI, que cualquiera puede usar. El registro público UDDI más extenso es el UDDI Business Registry (UBR) que fue desarrollado para facilitar la formación de nuevas relaciones empresariales.

La estructura de un registro UDDI, como por ejemplo UBR es, conceptualmente hablando, similar a un listín telefónico. Los registros contienen "páginas blancas" que contienen información de la compañía como su nombre, dirección, información de contacto e identificadores; "páginas amarillas" que dividen las compañías en varias categorías de acuerdo a sus productos o servicios y permiten a los usuarios buscar compañías o servicios que pertenezcan a una categoría en particular; y "páginas verdes", que contienen información técnica sobre los productos, servicios y web services de la compañía. Esto permite a un cliente comunicarse con un web service, porque la información define como invocar al servicio.

El modelo de información UDDI es la información referida a un web service que un cliente puede consultar. Esta información incluye información de negocio, información de servicio, información de comunicación, información de especificación e información de publicación.

Cada componente del modelo de información UDDI se encuentra dentro de una estructura de datos que consiste en elementos y atributos XML. Estos elementos y atributos XML describen los componentes del modelo de

información. La representación XML del modelo de información UDDI se usa cuando se interactúa con un registro UDDI. Debido a que hay cinco componentes de información, existen cinco estructuras de datos interrelacionadas.

2.2. WSA

La arquitectura de los web services tiene cuatro modelos. El nombre de cada modelo señala el concepto más importante de cada uno.

2.2.1. El modelo orientado a mensaje

El modelo orientado a mensaje se centra en los aspectos de la arquitectura relacionados con los mensajes y su procesamiento. Específicamente, en este modelo, no nos preocupamos con cualquier significado semántico del contenido de un mensaje o su relación con otros mensajes. Sin embargo, el MOM se centra en la estructura de los mensajes, en la relación entre los emisores y receptores de mensajes y como se transmiten los mensajes.

El MOM está ilustrado en esta figura:

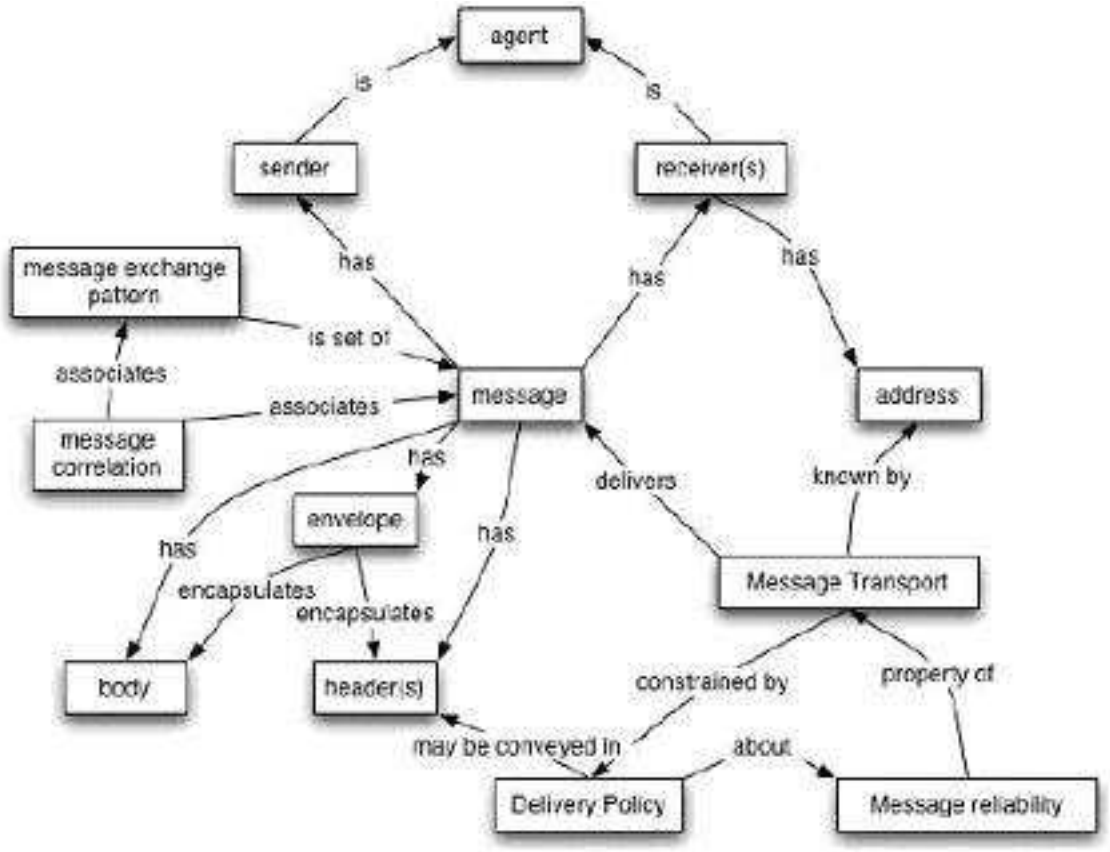

Figura 2.1: Modelo orientado a mensaje

Conceptos importantes:

Mensaje: Un mensaje representa la estructura de datos que se pasa del emisor a sus receptores. La estructura del mensaje está definida en una descripción del servicio. Las partes principales del mensaje son su recubrimiento, sus cabeceras y el cuerpo. Un mensaje puede ser tan simple como una petición HTTP GET, en el cual los parámetros codificados en la URL son el contenido. También puede ser un documento XML o un SOAP XML.

Recubrimiento de un mensaje: Es la estructura que encapsula el resto de las partes del mensaje, el cuerpo y las cabeceras.

Cabecera del mensaje: La cabecera representa información sobre los mensajes que es independiente del cuerpo. Puede haber formas estándar de cabecera que describan la autenticidad de los mensajes. La función principal de las cabeceras es facilitar el procesamiento modular del mensaje, aunque pueden ser usadas para ayudar al enrutamiento. La cabecera puede incluir información sobre seguridad, o gestión de las funcionalidades del web service. Pueden ser procesadas independientemente del cuerpo del mensaje.

Cuerpo del mensaje: El cuerpo del mensaje proporciona un mecanismo para la transmisión de información al receptor del mensaje. La forma del cuerpo del mensaje así como otras características pueden formar parte de la descripción del servicio. En muchos casos, la interpretación precisa del cuerpo del mensaje dependerá de las cabeceras del mensaje.

Emisor del mensaje: Es un agente que transmite un mensaje a otro agente. A pesar de que cada mensaje tiene un emisor, la identidad del emisor puede no estar disponible, en el caso de interacciones anónimas. El mensaje puede pasar por intermediarios, pero el agente emisor no es consciente de dichos intermediarios.

Receptor del mensaje: El receptor del mensaje es un agente que tiene la intención de recibir un mensaje del emisor. Los mensajes pueden pasar a través de intermediarios que procesan algunos aspectos del mensaje, normalmente examinando la cabecera. A menudo se identifica como receptor final a un receptor específico que será el responsable de completar el procesamiento del mensaje.

Dirección: Para que los mecanismos de transporte de mensajes funciones, suele ser necesario pasarles información que permita que los mensajes sean entregados. Esto se llama la dirección del receptor de mensajes. La forma de la dirección dependerá del transporte del mensaje. En el caso de transporte http, la dirección será una URL. El método que un emisor de mensajes usa para transmitir la dirección dependerá también del mecanismo de transporte. En ocasiones, la dirección puede pasarse como argumento del procedimiento de invocación o puede situarse dentro del mensaje, normalmente en el recubrimiento.

Política de entrega: Las políticas de entrega son las políticas relacionas con la entrega de mensajes. Normalmente, una política de entrega utiliza la combinación de un mensaje particular y un mecanismo de transporte determinado. Las políticas incluyen asegurar la calidad del servicio, la seguridad, etc.

Correlación de mensajes: La correlación permite a un mensaje estar asociado con un propósito o contexto particular. La técnica a utilizar es asociar un identificador a los mensajes. El identificador permite a un mensaje recibido estar correlacionado con la petición que lo originó. El emisor también puede añadir un identificador para servicio que será el receptor del mensaje.

Patrón de intercambio de mensajes: Las aplicaciones distribuidas de una arquitectura web service se comunican a través de intercambio de mensajes. Este intercambio de mensajes está regulado por patrones que se pueden componer a diferentes niveles para formar patrones mayores. Un patrón de intercambio de mensajes es un template que describe un patrón genérico para el intercambio de mensajes entre agentes. Los patrones se pueden definir mediante máquinas de estado que definen el flujo de mensajes, incluyendo la gestión de faltas de mensajes y la correlación.

Fiabilidad: El objetivo de tener mensajes fiables es reducir la frecuencia de errores y proporcionar suficiente información sobre el estado del envío de un mensaje. Dicha información permite a un agente tomar una decisión cuando ocurren errores.

Transporte de mensajes: El transporte de mensajes es el mecanismo usado para enviar mensajes, como por ejemplo, HTTP sobre TCP, SMTP. La responsabilidad del transporte es enviar el mensaje del emisor a uno o varios receptores. Para que el transporte funcione, el agente emisor debe proporcionar la dirección del receptor.

2.2.2. El modelo orientado a servicio

El modelo orientado a servicio (SOM) se centra en los aspectos de la arquitectura relacionados con el servicio y la acción.

El objetivo principal del SOM es explicar las relaciones entre un agente y los servicios que éste proporciona y requiere.

El SOM está ilustrado en ésta figura:

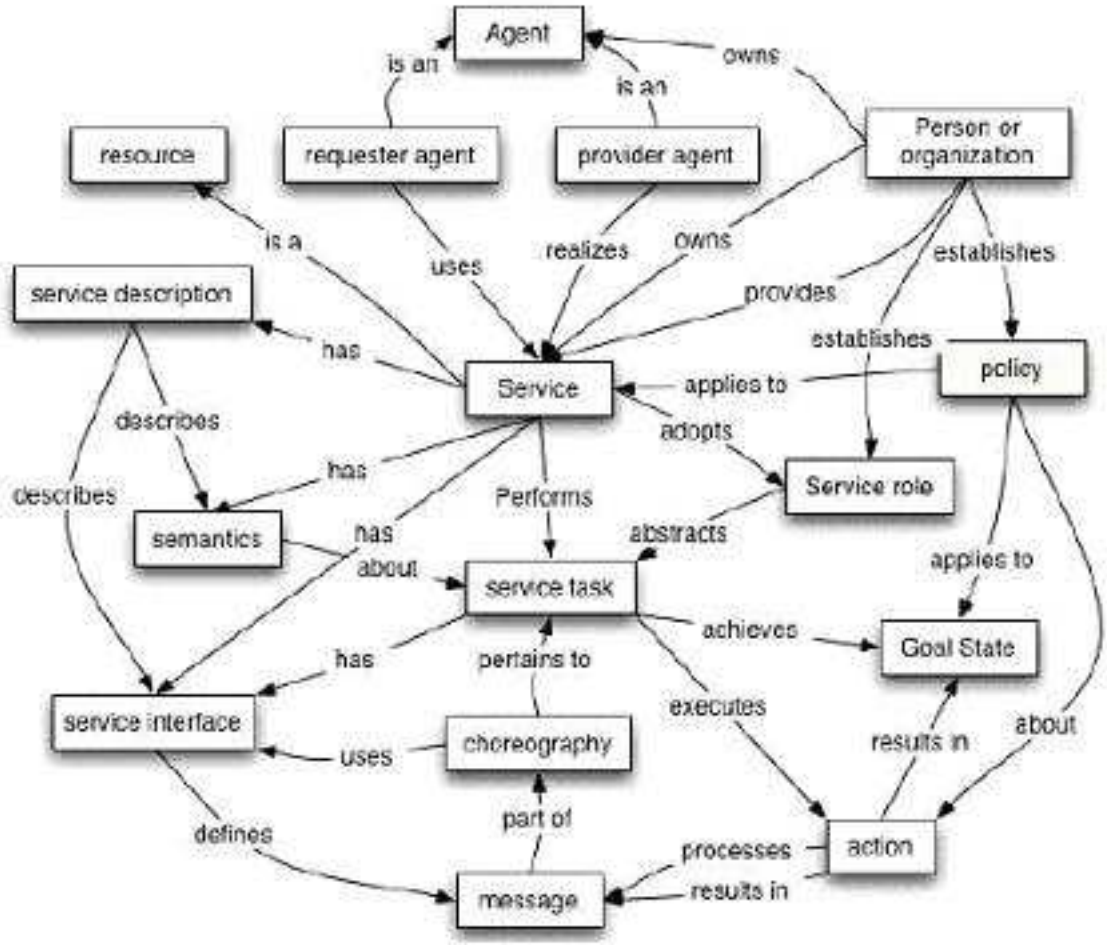

Figura 2.2: Modelo orientado a servicio

Conceptos importantes:

Servicio: Es un recurso abstracto que representa a una persona u organización en cierto conjunto de tareas relacionadas. El servicio puede ser realizado por uno o más agentes proveedores que actúan en nombre de la persona u organización. La diferencia de un web service con otros recursos web es que los web services no tienen necesariamente una

representación; sin embargo, están asociados con acciones.

Descripción de servicio: Contiene los detalles del interfaz y, potencialmente, el comportamiento esperado del servicio. Esto incluye sus tipos de datos, operaciones, información sobre el protocolo de transporte, y dirección. Puede incluir también, categorización y otros metadatos para facilitar su utilización. La descripción completa puede realizarse mediante un conjunto de documentos XML.

Interfaz de servicio: Es la frontera abstracta que un servicio expone. Define los tipos de mensajes y los patrones de intercambio de mensajes que están involucrados en la interacción con el servicio.

Intermediario de servicio: Es un tipo específico de servicio que actúa normalmente como una especie de filtro de los mensajes que gestiona. Normalmente, los intermediarios no consumen mensajes sino que más bien los envían a otros servicios. Suelen modificar los mensajes, pero no modifican el significado del mensaje.

Rol de servicio: Es una abstracción intermedia entre servicio y tarea. Un mensaje dado que es recibido por un servicio puede implicar el procesamiento asociado con varios roles de servicio.

Semántica del servicio: Es el comportamiento esperado cuando se interactúa con el servicio. La semántica expresa un contrato entre la entidad proveedora y la entidad solicitante. Expresa el efecto real que produce la invocación de un servicio.

Tarea del servicio: Es una acción o combinación de acciones asociadas con un estado objetivo deseado. La realización de la tarea implica la ejecución de las acciones y está orientado a la consecución de un estado objetivo en particular.

Agente: Son programas que toman parte en acciones en nombre de alguien o algo. En nuestro caso, los agentes realizan y requieren web services. Los agentes software son proxies para las entidades que los poseen.

Agente proveedor: Es el agente software que realiza un web service, ejecutando tareas en nombre de su dueño. Un servicio puede ser ofrecido por más de un agente, especialmente en el caso de servicios compuestos, y un proveedor puede realizar más de un web service.

Entidad proveedora: Es la persona u organización que ofrece un web service. El agente proveedor actúa en nombre de la entidad que lo posee.

Agente solicitante: Es el agente software que requiere que se realice una cierta función en nombre de su dueño. Desde un punto de vista arquitectónico, éste es el agente que busca e inicia la interacción con el agente proveedor.

Entidad solicitante: Es la persona u organización que quiere hacer uso de un web service.

Acción: Una acción, en términos de ésta arquitectura, es cualquier acción que pueda ser realizada por un agente, posiblemente como resultado de la recepción de un mensaje, o que desencadena el envío de un mensaje. Desde la perspectiva de los agentes, una acción suele realizarse ejecutando un fragmento de programa. Las acciones realizadas por los agentes emisor y receptor están fuera del ámbito de la WSA, excepto si son resultado de los mensajes emitidos o enviados.

Coreografía: Es un modelo de la secuencia de operaciones, estados y condiciones que controlan las interacciones involucradas en los servicios participantes.

Habilidad: Los agentes que participan en un intercambio pueden implementar una amplia gama de características. Por ejemplo, puede haber varios modos de alcanzar el envío fiable de un mensaje, o varios mecanismos de mantener la seguridad. Un web service debe anunciar que tiene una determinada habilidad, y el agente que pide esa habilidad puede seleccionar el servicio basándose en eso.

Estado objetivo: Están asociados a las tareas. Las tareas son las unidades de acción asociadas a los servicios. Un estado objetivo se caracteriza por un predicado que es cierto en ese estado. Es simplemente un modo de ser capaz de señalar "éxito" cuando una tarea se ha completado satisfactoriamente.

2.2.3. Modelo orientado a recurso

El modelo orientado a recurso (ROM) se centra en los aspectos de la arquitectura relacionados con los recursos. Los recursos son un concepto fundamental que sustenta gran parte de la web y de los web services; por ejemplo,

un web service es un tipo particular de recurso que es importante para su arquitectura.

El ROM se centra en las características clave de los recursos que son relevantes para el concepto de recurso, independientemente del papel que éste tenga en el contexto de los web services. Por consiguiente nos centramos en temas como la posesión de recursos, políticas asociadas con los recursos, etc?Por tanto, basándose en el hecho de que los web services son recursos, estás propiedades son heredadas por los web services.

El modelo orientado a recurso está ilustrado en la siguiente figura:

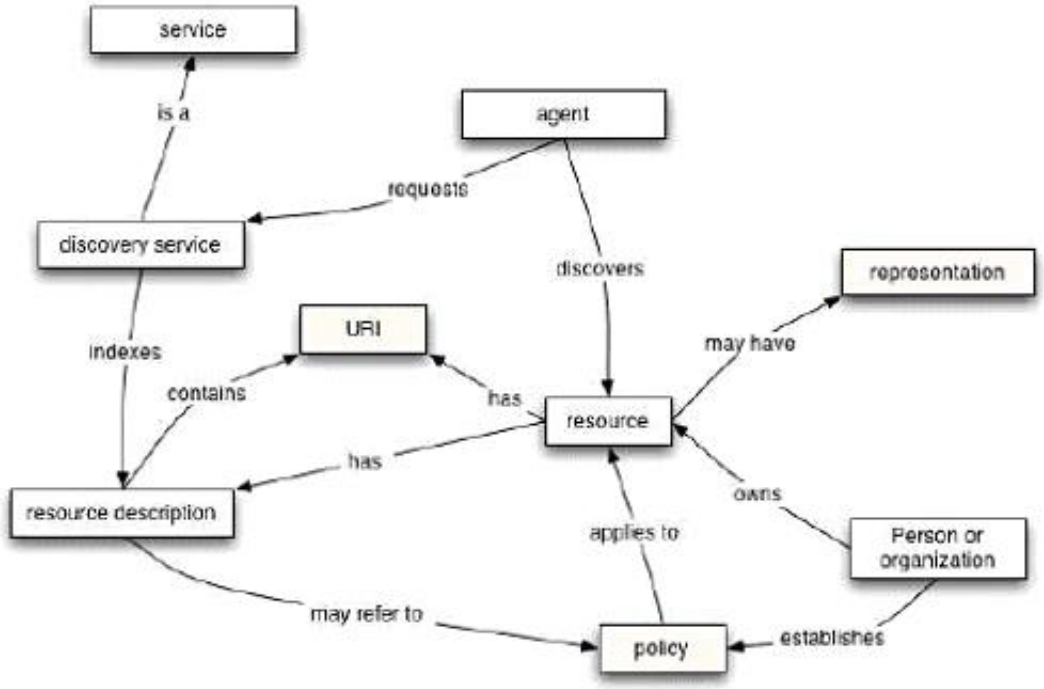

Figura 2.3: Modelo orientado a recursos

Conceptos importantes:

Recurso: Un recurso se define como "cualquier cosa que pueda tener un identificador". A pesar de que los recursos en general pueden ser cualquier cosa, está arquitectura sólo tiene en cuenta los recursos

relevantes para los web services y por tanto tienen algunas características adicionales. En particular, incorporan los conceptos de propiedad y control: un recurso que aparece en esta arquitectura es una "cosa" que tiene un nombre, que puede tener representaciones y que se puede decir que tiene un dueño. La propiedad de un recurso está muy relacionado con el derecho de emplear una política en el recurso.

Descripción de recurso: Una descripción de recurso es cualquier conjunto de datos procesables por una máquina que permite que los recursos sean descubiertos. Las descripciones de recursos pueden ser de diferentes formas, adaptadas para propósitos específicos, pero todas las descripciones de recursos deben contener el identificador de recurso.

Descubrimiento: Es la acción de localizar una descripción procesable de un recurso relacionado con un web service que puede haber sido desconocido hasta entonces y que se ajusta a ciertos criterios funcionales. El objetivo es encontrar un recurso relacionado con un web service apropiado.

Servicio de descubrimiento: Un servicio de descubrimiento se usa para publicar y buscar descripciones que se ajusten con ciertos criterios funcionales o semánticos. Está pensado principalmente para ser usado por las entidades solicitantes, para facilitar el proceso de encontrar una entidad proveedora adecuada para una tarea en particular. Sin embargo, dependiendo de la implementación y la política del servicio de descubrimiento, puede ser usado por las entidades proveedoras para publicar activamente sus descripciones de servicio.

Identificador: Un identificador es un nombre no ambiguo para un recurso.

Representación: Las representaciones son objetos que reflejan el estado de un recurso. Un recurso tiene un identificador único. La representación de un recurso no necesita ser la misma que el recurso en si mismo; por ejemplo, el recurso asociado al estado de las cuentas de un restaurante tendrá diferentes representaciones dependiendo de cuando se muestra la representación. La representación suele mostrarse realizando un HTTP "GET" en una URI.

2.2.4. Modelo de política

El modelo de política se centra en los aspectos de la arquitectura relacionados con políticas y, por extensión, seguridad y calidad del servicio.

La seguridad se fundamenta en restricciones acerca del comportamiento de las acciones y en el acceso a recursos. Igualmente, la calidad del servicio se basa en restricciones en el servicio. En el modelo de política, estas restricciones se modelan alrededor del concepto principal de política y las relaciones con otros elementos de la arquitectura. Por consiguiente, el modelo de política es un marco en el cual la seguridad se puede cimentar.

El modelo orientado a recurso está ilustrado en la siguiente figura:

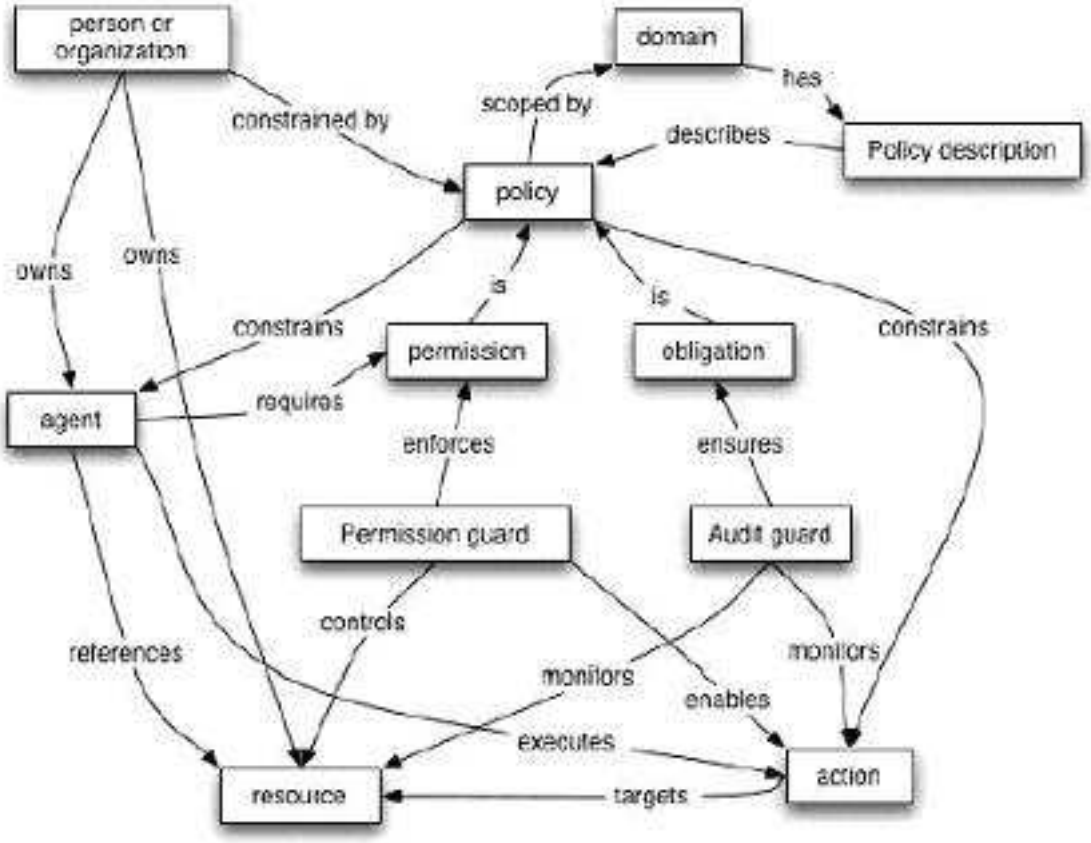

Figura 2.4: Modelo de política

Conceptos importantes:

Política: Una política es una restricción en el comportamiento de agentes a la hora de realizar acciones o acceder a recursos. Hay varias clases de políticas, algunas relacionadas con el acceso a recursos de modos particulares, otras relacionados con las acciones que se permite que realice un agente. Identificamos dos tipos de política: permisos y obligaciones.

Permiso: Es uno de los dos tipos fundamentales de políticas. Cuando un agente tiene permiso para realizar una acción, para acceder a algún recurso, o para alcanzar cierto estado, entonces se espera que cualquier intento de realizar cualquiera de lo anterior tenga éxito. Por el contrario, sin un agente no tiene el permiso requerido, entonces la acción debe fallar incluso si por lo demás debería haber funcionado.

Obligación: Es el otro tipo fundamental de política. Cuando un agente tiene la obligación de realizar una acción, entonces es necesario que la haga. Cuando la acción se realiza, se dice que el agente ha cumplido con su obligación. No todas las obligaciones se refieren a acciones. Por ejemplo, un agente que proporciona un servicio puede tener la obligación de mantener cierto estado de inmediatez.

Protectores de permisos: Un protector de permisos es un mecanismo que se usa para aplicar las políticas de permisos. Su papel es asegurar que cualquier uso de un servicio o recurso sea consistente con las políticas establecidas por el gestor o dueño del servicio.

Descripción de política: Una descripción de política es una descripción procesable por una máquina de algunas restricciones sobre el comportamiento de los agentes a la hora de realizar accione y acceder a los recursos. No es la política en sí, pero define la política y puede ser usada para determinar si la política se aplica en una situación determinada.

Protector de política: Es un mecanismo que aplica una o varias políticas. Se despliega en nombre de un propietario.

Dominio: Un dominio define el ámbito de aplicación de políticas. Un dominio puede ser definido explícitamente o implícitamente. Los miembros de un dominio definido explícitamente son enumerados por una autoridad central y los definidos implícitamente no. Por ejemplo, la pertenencia en un dominio definido implícitamente puede depender del estado del agente y por tanto ser dinámico

Persona u organización: El concepto de persona u organización para la arquitectura de web services se refiere a las personas que son representadas por agentes para que éstos realicen acciones en su nombre. Todas las acciones consideradas en esta arquitectura radican en último término en acciones humanas.

2.3. SOA

2.3.1. Sistemas distribuidos

Un sistema distribuido consta de diversos agentes software que deben trabajar juntos para realizar algunas tareas. Además, los agentes de un sistema distribuido no operan en el mismo entorno de procesamiento, por lo que deben comunicarse por protocolos hardware/software sobre una red. Esto significa que las comunicaciones en un sistema distribuido son intrínsecamente menos rápidas y fiables que los que usan invocación directa de código y memoria compartida. Esto tiene unas implicaciones arquitectónicas porque los sistemas distribuidos necesitan que los desarrolladores (de infraestructura y aplicaciones) consideren la impredecible latencia de acceso remoto, y tengan en cuenta la cuestión de la concurrencia y la posibilidad de un fallo parcial.

Los sistemas de objetos distribuidos son sistemas distribuidos cuya semántica de inicialización de objetos e invocación de métodos está expuesta a sistemas remotos por medio de mecanismos registrados o estandarizados para enviar las peticiones más allá de los límites del sistema. Los sistemas de objetos distribuidos suelen caracterizarse porque los objetos tienen un complejo estado interno necesario para sostener sus métodos, una interacción de grano fino entre un objeto y el programa que lo usa, y que se centra en una jerarquía de interfaces entre el objeto y el programa que lo usa.

Una arquitectura orientada a servicio (SOA) es una forma de arquitectura de sistemas distribuidos que suele caracterizarse por las siguientes propiedades:

Visión lógica: El servicio es una visión lógica abstraída de los programas, bases de datos, procesos de negocio, definida en términos de lo que hace.

Orientación a mensaje: El servicio está definido normalmente en términos del intercambio de mensajes entre los agentes proveedores y los

agentes solicitantes, y no de las propiedades de los agentes en sí mismos. La estructura interna de un agente, incluyendo las características como su lenguaje de implementación, estructura de proceso e incluso estructura de base de datos, son deliberadamente abstraídos en la SOA: usando la disciplina de SOA uno no necesita conocer como está construido un agente que implementa un servicio.

Orientación a descripción: Un servicio se describe mediante metadatos procesables por una máquina. La descripción sostiene la naturaleza pública de la SOA: sólo deben ser incluidos en la descripción aquellos detalles expuestos al público y que sean importantes para el uso del servicio. La semántica del servicio debe estar documentada, bien directa o bien indirectamente, por su descripción.

Granularidad: Los servicios tienden a usar un pequeño número de operaciones con mensajes relativamente largos y complejos.

Orientación a red: Los servicios tienen a estar orientados hacia el uso en una red, a pesar de que esto no es un requisito absolutamente necesario.

Plataforma neutra: Los mensaje se envían en un formato estandarizado de plataforma neutra entregado a través de los interfaces. XML es el formato más obvio que se ajusta a esta restricción.

2.3.2. Web services y estilos arquitectónicos

Los sistemas de objetos distribuidos tienen varios desafíos arquitectónicos:

- Problemas ocasionados por la latencia y baja fiabilidad del transporte subyacente.
- La falta de memoria compartida entre el invocador y el objeto.
- Los numerosos problemas ocasionados por los escenarios de fallo parcial.
- Los desafíos del acceso concurrente a recursos remotos.
- La fragilidad de los sistemas distribuidos si se introducen mejoras incompatibles en alguno de los participantes.

Estos desafíos existen independientemente de si el sistema de objetos distribuido está implementado usando CORBA o tecnologías de web services. Los web services no son menos apropiados que otras alternativas siempre y cuando se cumplan los criterios fundamentales de las arquitecturas de objetos distribuidas. Si se cumplen dichos criterios las tecnologías de web services pueden ser apropiadas.

En cambio, el uso de tecnologías de web services para implementar un sistema distribuido no convierte por arte de magia una arquitectura de objetos distribuida en una SOA. Tampoco las tecnologías de web services son necesariamente la mejor elección para implementar SOAs (por ejemplo en el caso de que la infraestructura está preparada para usar CORBA y no se necesita neutralidad de las plataformas).

En general la SOA y los web services son más apropiadas para:

- Aplicaciones que deben operar en Internet donde la fiabilidad y la velocidad no se pueden garantizar.
- Donde no hay capacidad para gestionar el despliegue, y por lo tanto todos los proveedores y solicitantes se mejoran a la vez
- Donde los componentes del sistema distribuido se ejecutan en plataformas diferentes.
- Donde una aplicación existente necesita ser expuesta para su uso en una red, y puede ser envuelta en un web service.

Capítulo 3

Servicios y Arquitecturas Grid

Este capítulo muestra algunos aspectos sobre los que está basado el GT4. El Globus Toolkit v4 se basa principalmente en la arquitectura estándar OGSA (Open Grid Services Architecture) que a su vez implementa el WSRF (Web Service Resource Framework) que es una especificación de Web Services persistentes. La anterior implementación en la que estaba basada OGSA, usada para versiones anteriores del Globus Toolkit, era OGSI, que no permitía la persistencia de los Web Services.

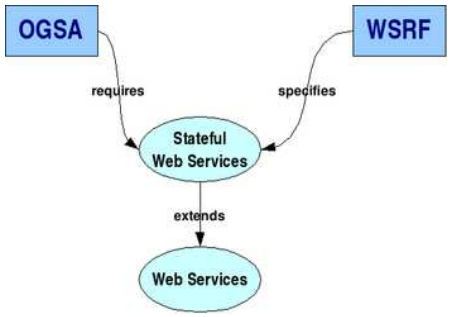

Figura 3.1: Relación OGSA,WSRF y Web Services

3.1. Arquitectura OGSA

3.1.1. Introducción

Una aplicación grid constará normalmente de varios componentes. Por ejemplo, una aplicación típica puede constar de:

- **Servicio de administración de organizaciones virtuales**: para el manejo de nodos y usuarios forman parte de cada organización virtual.
- **Servicio de administración y descubrimiento de recursos:** para que las aplicaciones puedan descubrir recursos que se ajusten a sus necesidades y los manejen.
- **Servicio de manejo de trabajos:** para que los usuarios puedan enviar tareas (en forma de trabajos o "*jobs*") al grid.

Todos estos servicios, y mucho más no mencionados, están siempre en constante interacción. Con tantos servicios, y con tanta interacción, puede llegar a existir un caos en potencia. ¿Qué pasaría si cada fabricante decidiera implementar a su manera el Servicio de manejo de trabajos? Esto acabaría con distintas interfaces y funcionalidades que serían difícilmente, o imposible, de unificar.

La solución a todo esto es la estandarización, es decir, definir una interfaz común para cada tipo de servicio, y como se puede deducir, es donde entra en juego OGSA.

La Open Grid Services Architecture (OGSA) fue presentada por I. Foster, C. Kesselman, J. Nick y S. Tuecke en el documento "Physiology of the Grid".

La OGSA se presenta como una arquitectura capaz de enfrentarse al reto de la estandarización. Ha sido desarrollada por The Global Grid Forum y su objetivo es definir una arquitectura abierta, estándar y común a todas las aplicaciones grid. La meta de OGSA es estandarizar prácticamente la totalidad de los servicios, que se pueden encontrar en una aplicación grid, definiendo para ellos un conjunto de interfaces estándar. Está construida a partir de conceptos y tecnologías de grid y Web Services, y define una semántica uniforme de servicios: los Grid Services o Servicios Grid. Además, define mecanismos estándar para la creación, nombramiento, y descubrimiento de instancias de Grid Services; proporciona transparencia en la localización y múltiples protocolos para instancias de servicios; y facilita la integración con las distintas plataformas.

3.1.1.1. Grid Service

La habilidad para poder virtualizar y crear servicios depende en mucho de la definición de una interfaz estándar. Además se requiere una semántica estándar para las interacciones entre servicios para que, por ejemplo, distintos servicios sigan los mismos pasos para la notificación de errores. Con este fin, OGSA define Grid Service como:

> Un Web Service que proporciona un conjunto de interfaces bien definidas y que sigue una serie de convenciones específicas. Las interfaces definen el descubrimiento, la creación dinámica de servicios, la gestión del tiempo de vida, la notificación, y el manejo. Las convenciones definen el nombrado y la actualización.

Las interfaces y las convenciones que definen un Grid Service están relacionadas, en particular, con el comportamiento relacionado a instancias transitorias de servicios. Los participantes de las organizaciones virtuales suelen mantener estáticamente una serie de servicios persistentes para manejar las peticiones de las actividades complejas de sus clientes. Además, normalmente, se necesitan crear dinámicamente nuevas instancias transitorias de servicios, que manejarán la utilización y las interacciones asociadas al estado de la actividad requerida. Cuando el estado de la actividad no es necesario, el servicio puede ser destruido. Un ejemplo de instancias transitorias de servicios puede ser una consulta en una base de datos, una reserva de ancho de banda de red. Las instancias de los servicios llegan a ser entidades muy ligeras, creadas para manejar actividades de muy poca duración. La transitoriedad tiene importantes implicaciones en como los servicios son manejados, nombrados, descubiertos y usados.

Los Grid Services son el núcleo de la arquitectura OGSA, ver 3.1.1.3. Suelen ser recursos computacionales, recursos de almacenamiento, programas, o bases de datos. Tomando el modelo de los Web Services como ejemplo, los Grid Services encajan muy bien en los conceptos de registro, descubrimiento, y uso. Los dos aspectos críticos para los usuarios en una Arquitectura Orientada a Servicios (SOA, ver 2.3) como es ésta, son las definiciones de las interfaces de los servicios y de la identificación de los protocolos usados para invocar un servicio.

No existe ninguna restricción en la especificación de los Grid Services acerca de cómo deben ser escritos, en que sistemas operativos deben funcionar,

que lenguajes deben usar, o a que modelo de programación se refieren.

La interacción entre los servicios tiene lugar mediante mensajes, y la existencia de un estado interno hace que sea importante el hecho de que un mensaje sea enviado una vez o no del todo. Esto llega incluso más importante en el contexto de un sistema distribuido poco fiable.

La premisa básica de OGSA es que cualquier recurso computacional como ciclos del procesador, almacenamiento, memoria, bases de datos, etc, puede ser entendido como un servicio. Además estos servicios tienen una serie de propiedades y necesidades que deben ser descritas como un conjunto de interfaces (u operaciones). La especificación de Grid Services llevaba a cabo por OGSI, ver 3.2, especifica este conjunto de comportamientos como operaciones dentro de varios Port Types, que pueden ser agrupados con otros Port Types para crear un servicio.

Como se ha dicho antes, los Grid Services extienden el concepto de Web Services diseñando un conjunto de interfaces bien definidas que están encaminadas hacia el descubrimiento, la creación dinámica, el manejo del tiempo de vida, la notificación, el manejo, y un conjunto de convenciones para el nombrado y la actualización. Estas interfaces y convenciones son vitales para permitir una interoperabilidad fiable entre los servicios y las aplicaciones que los invocan. WSDL, ver 2.1.1.3, se refiere a esas interfaces como los portTypes.

3.1.1.2. Características de OGSA

El mayor objetivo de la tecnología grid es promocionar un entorno virtual de computación, que significa el uso de un conjunto de servicios, a través del uso transparente y coordinado de recursos heterogéneos y distribuidos. Los servicios son la abstracción de los recursos, y pueden ser ciclos de computación, software, documentos, datos, almacenamiento, etc.

Para poder conseguir que esto sea real y útil, se deben alcanzar los siguientes objetivos:

Transparencia significa que el usuario utilizará el entorno con la misma calidad que si estuviera usando un sistema local. Por lo tanto, el grid debe proporcionar herramientas fáciles para ayudar a los usuarios a especificar sus necesidades y calidades y una buena calidad de servicio (QoS), que en este caso significa un rápido acceso a los servicios usando autenticación inteligente y comunicación rápida.

Coordinación significa que es necesario tener un sistema de administración que proporcione el emparejamiento entre las necesidades de los usuarios y la disponibilidad de recursos, monitorizando el uso de los servicios y proporcionando facilidades adicionales , como el control local de recursos, representación de recursos y control de estados, identificación, seguridad y otros.

Distribuido y heterogéneo significa que es necesaria la interoperabilidad, y que se trata de una interfaz común y estandarizada que traduce las necesidades de los usuarios y la disponibilidad de los recursos en un sólo lenguaje, sin tener en cuenta el hardware, software y el sistema en el que se encuentra cada recurso.

OGSA une a las comunidades grid y de Web Services para que se enfrenten juntas a los problemas de los servicios a través organizaciones virtuales, dinámicas, heterogéneas y distribuidas. OGSA puede considerarse por llevar a los Web Services a un entorno donde el estado es importante, ver 3.2.1. El núcleo de la arquitectura OGSA son los Grid Services, ver 3.1.1.1.

Con la idea de crear este entorno, la Globus Alliance creó, junto con aportaciones de la Global Grid Forum's OGSA Working Group (GGF-OGSA-WG), la Open Grid Services Architecture (OGSA), que define el marco, la arquitectura y las funcionalidades de los sistemas grid, y que acabará con el desarrollo de la Open Grid Services Infrastructure (OGSI) por el GGF-OGSI-WG.

3.1.1.3. Detalles técnicos

En este punto se explican algunos de los asuntos y direcciones tomadas por Globus y el GGF para poder llevar a cabo el reto de crear organizaciones virtuales.

Modelo de Servicios de OGSA

La premisa básica de OGSA es que todo está representado por un servicio: una entidad con conexión a red que proporciona alguna capacidad a través del intercambio de mensajes. Los recursos computacionales, los recursos de almacenamiento, las redes, los programas, las bases de datos, etc, son todos servicios. La adopción de este modelo uniforme orientado a servicios significa que todos los componentes del entorno son virtuales.

Más específicamente, OGSA representa todo como un Grid Service: un Web Service creado a partir de un conjunto de convenciones y mantiene un interfaces estándar para esos propósitos, como la gestión del tiempo de vida. Este núcleo de interfaces consistentes, desde las cuales todos los Grid Services son implementados, facilita la construcción de una jerarquía de servicios de mayor orden que pueden ser tratados de manera uniforme mediante distintas capas de abstracción.

Los Grid Services están caracterizados (escritos) por las capacidades que ofrecen. Un Grid Service implementa una o más interfaces, donde cada interfaz define un conjunto de operaciones que son invocadas mediante el intercambio de una secuencia definida de mensajes. Las interfaces de los Grid Services se corresponden a los *portTypes* de WSDL, ver 2.1.1.3. El conjunto de *portTypes* mantenidos por un Grid Service, además de cierta información correspondiente a la versión, están especificados en el *serviceType* del Grid Service, una extensión de WSDL definido por OGSA.

Los Grid Services pueden mantener un estado interno durante el tiempo de vida del servicio. La existencia de estado distingue una instancia de servicio de otra que proporciona el mismo servicio. Se usa el término instancia de un Grid Service para referirse a una instanciación particular de un Grid Service.

El protocolo de unión asociado a la interfaz de un servicio puede definir una semántica de envío encaminada a, por ejemplo, la fiabilidad. Los servicios interactúan unos con otros mediante el intercambio de mensajes. En un sistema distribuido propenso a fallos de componentes, nadie puede garantizar que un mensaje enviado llegue a su destino. La existencia de un estado interno llega a ser muy importante para garantizar el hecho de los mensajes enviado puedan o no llegar a su destino, incluso existiendo mecanismos de recuperación de fallos como el reenvío. En estas situaciones, querremos emplear un protocolo que garantice un sólo envío o similar. Otro comportamiento muy deseable asociado a un protocolo será la autenticación durante la comunicación.

Los servicios OGSA pueden ser creados y destruidos dinámicamente. Los servicios debe ser destruidos explícitamente, o deben ser destruidos o ser inacesibles como resultado de algún fallo del sistema como una caída del sistema operativo o de la red. Las interfaces están definidas para manejar el tiempo de vida de los servicios.

Ya que los Grid services son dinámicos y con estado, necesitamos una manera para poder distinguir una instancia creada dinámicamente de un

servicio con respecto a otra. Por ello, a toda instancia de Grid Service se le asigna un único nombre global, el *Grid Service Handle* (GSH), que distingue cualquier instancia específica de un Grid Service de cualquier otra instancia Grid Service que haya existido, exista, o pueda existir en un futuro (Si un Grid Service falla y es reiniciado de forma que preserve su estado, entonces es esencialmente la misma instancia, y el mismo GSH puede ser usado).

Los Grid Services necesitan, como casi todo, ser actualizados durante su tiempo de vida, por ejemplo para poder soportar nuevas versiones de un protocolo o añadir protocolos alternativos. Por ello, el GSH no lleva información específica de los protocolos que la instancia puede soportar como puede ser la dirección de la red o los protocolos de unión. En cambio, esta información es encapsulada, junto con información específica de la instancia sobre como interactuar con instancias de servicios específicos, en una abstracción llamada *Grid Service Reference* (GSR). Al contrario que el GSH, que es invariante, el GSR para una instancia Grid Service puede cambiar a lo largo del tiempo de vida del servicio. Cada GSR tiempo un tiempo de caducidad, y OGSA define mecanismos de mapeado, descritos más abajo, para obtener un GSR actualizado.

El resultado de emplear un GSR cuyo tiempo de vida haya acabado está indefinido. Hay que notar también que el empleo de un GSR válido tampoco garantiza el acceso a una instancia Grid Service: políticas locales o restricciones de control de acceso (por ejemplo el número máximo de peticiones actuales) pueden prohibir la realización de una petición.

Como todo en OGSA en un Grid Service, debe haber Grid Services que manipulen el servicio, el manejo y las referencias definidas en el modelo de OGSA. Una definición de un conjunto de servicios será una interpretación específica del modelo de servicios de OGSA. Aun así sólo se definirá un conjunto básico de interfaces OGSA (e.d. WSDL portTypes) que manipulen el modelo de servicios para crear una proposición flexible, de manera que se pueda producir una gran cantidad adicional de Grid Services mediante la combinación de aquellos.

Creación de servicios transitorios : Factory

OGSA define una clase de Grid Services que implementan una interfaz que crea nuevas instancias Grid Service. Llamamos a esta interfaz *Factory* y al servicio que implementa dicha interfaz *factory*. La interfaz *Factory* define una operación que crea un Grid service requerido y devuelve el GSH y el

GSR inicial para esa nueva instancia.

La interfaz *Factory* no especifica como se crea la instancia del servicio. Un escenario típico donde puede ser implementado la interfaz factory puede ser un entorno, como .NET o J2EE, que proporcione mecanismos estándar para la creación (y posterior manejo) de nuevas instancias de servicios. El entorno debe definir como están implementados los servicios (e.d. el lenguaje), pero eso es transparente para los solicitantes de servicios en OGSA, que sólo ven la interfaz factory. Alternativamente, se pueden construir interfaces factory de mayor nivel que creen servicios mediante la delegación del trabajo en otras interfaces factory.

Gestión del tiempo de vida

La introducción a instancias de servicios transitorias nos conduce a determinar el tiempo de vida de un servicio: esto es, determinar cuando un servicio puede o debe terminar de forma que sus recursos asociados puedan ser recuperados. En condiciones normales de operación, una instancia de servicio transitoria es creada para realizar una determinada tarea y termina o al realizar su tarea o via petición explícita del solicitante o desde otro servicio designado por el solicitante. En sistemas distribuidos, sin embargo, hay componentes que pueden fallar o los mensajes se pueden perder. Un resultado podrá ser que un servicio no verá una solicitud de terminación explícita y puede estar consumiendo recursos indefinidamente.

OGSA aborda este problema mediante estados, por lo que las instancias Grid Service son creadas con un determinado tiempo de vida. El tiempo inicial de vida puede ser ampliado mediante una petición explícita del cliente o mediante un Grid service que actúe en la parte del cliente (todo claro sujeto a la política del servicio). Si ese período de tiempo finaliza sin haberse recibido una re-confirmación del interés por parte del cliente, tanto el entorno de aplicación como la instancia del servicio por sí misma tienen la libertad de acabar con la instancia del servicio y de liberar los recursos asociados.

La propuesta para la gestión del tiempo de vida tiene dos propiedades deseables:

- El cliente sabe, o puede determinar, cuando la instancia Grid Service terminará. Este conocimiento permite al cliente determinar fiablemente cuando la instancia del servicio ha terminado y sus recursos recuperados, incluso a pesar de fallos del sistema. El cliente sabe exactamente

cuanto tiempo tiene a la hora de pedir un estado final de la instancia del servicio o a la hora de pedir una ampliación en el tiempo de vida. Además, sabe que si ocurre un fallo del sistema, no tendrá la necesidad de seguir contactando con el servicio después del tiempo de vida especificado, y que los recursos asociados con dicho servicio serán liberados al pasar dicho tiempo - a no ser claro que otro cliente tenga éxito al ampliar el tiempo de vida. En resumen, la gestión del tiempo de vida permite una robusta terminación y detección de fallos, gracias a la definición de la semántica del tiempo de vida de un servicio.

- El entorno garantiza que el consumo de los recursos esté limitado, incluso si los fallos del sistema no están a su alcance. Si se llega a la terminación de un servicio, el entorno puede reclamar todos los recursos asociados.

El estado que maneja el tiempo de vida está implementado vía la interfaz *SoftStateDestruction*, que define operaciones para el negocio del tiempo de vida inicial para las nuevas instancias, para la petición de ampliación del tiempo de vida, y para la recolección de una instancia cuando su tiempo ha terminado. Ahora describiremos cada mecanismo:

Negociación del tiempo de vida inicial. Cuando se pide la creación de una nueva instancia Grid Service mediante una factory, el cliente indica los tiempos de vida mínimos y máximos aceptables. La factory selecciona el tiempo de vida inicial y se lo devuelve al cliente.

Petición de ampliación del tiempo de vida. El cliente pide una ampliación mediante un mensaje *SetTerminationTime* a la instancia Grid Service, y especificará unos tiempos mínimos y máximos aceptables para el nuevo tiempo de vida. La instancia selecciona el nuevo tiempo y lo reenvía al cliente. Hay que señalar que un mensaje de *keep alive* es igual de efectivo.

La periodicidad de los mensajes *keep alive* puede ser determinado por el cliente basándose en el tiempo inicial de vida negociado con la instancia y del conocimiento acerca de la fiabilidad de la red. El tamaño de los intervalos otorga cierta información que permite discenir sobre el *overhead* de la comunicación.

Esta característica de la gestión del tiempo de vida de las instancias proporciona a los servicios bastante autonomía. Las peticiones de ampliación desde los clientes no son necesarias, ya que el servicio puede aplicar sus propias políticas para garantizarse dichas peticiones. Un servicio puede decidir siempre que quiera el ampliar su tiempo de vida, ya sea por petición del cliente o por otro motivo. También una instancia de servicio puede cancelarse a sí mismo cuando quiera, por ejemplo si las restricciones de los recursos y las prioridades dictan que debe abandonar éstos. Las posteriores peticiones al servicio fallarán.

El uso de un tiempo absoluto definido en la interfaz *SoftStateDestruction* implica la existencia de un reloj global que está lo suficientemente bien sincronizado. El *Network Time Protocol* (NTP) proporciona mecanismos estándar para la sincronización de relojes y puede hacerlo en rangos de diez milisegundos, que es muy adecuado para propósitos de la gestión del tiempo de vida.

Gestión de manejadores y referencias

Como hemos dicho antes, el resultado a una petición a un *factory* es un GSH y un GSR. Mientras el GSH sirve como garantía para poder referenciar a la instancia creada de por vida, el GSR se crea con un tiempo de vida finito y puede cambiar durante el tiempo de vida del servicio. Mientras esta estrategia ofrece ventajas en el aumento de la flexibilidad desde la perspectiva del proveedor del servicio, también introduce el problema de la obtención de un GSR válido cuando el GSR devuelto en la creación del servicio finaliza su tiempo. El problema se trata pues, de cómo contactar con un servicio sólo por su nombre.

La solución tomada en OGSA es definir una interfaz de correspondencias entre manejadores y referencias (*HandleMapper*). La operación proporcionada por dicha interfaz coge un GSH y devuelve un GSR válido. Las operaciones de correspondencia pueden tener accesos controlados y por lo tanto algunas peticiones pueden ser denegadas. Una implementación de la interfaz *HandleMapper* llevará la cuenta de que instancias Grid Services están actualmente existiendo y no devolverá, por tanto, referencias a instancias ya terminadas. Sin embargo, el tener un GSR válido no asegura que se llegue a contactar con la instancia: el servicio puede fallar o haber terminado durante el tiempo en que el GSR fue devuelto y usado.

Mediante la presentación de la interfaz *HandleMapper*, se divide el pro-

blema de la obtención de un GSR de un servicio cualquiera en varios subproblemas:

1. Identificar un servicio handleMapper que contenga la correspondencia para el GSH especificado, y

2. Contactar con este handleMapper para obtener el GSR deseado.

Hablaremos ahora de estos dos problemas por turno. Para asegurarnos que siempre podremos realizar la correspondencia de un GSH a un GSR, se requiere que cada instancia Grid Service sea al menos registrada por al menos un handleMapper, que llamaremos el handleMapper *base*. Estructurando el GSH para que incorpore la identidad de este handleMapper base, podemos fácilmente y escalablemente determinar el handleMapper que debemos contactar para obtener el GSR para un GSH dado. Por lo tanto, se pueden determinar nombres únicos localmente, evitando así problemas asociados con servicios centralizados de reserva de nombres. Además las correpondencias de un GSH pueden existir en varios handleMappers. Sin embargo, cada GSH debe tener un sólo handleMapper base.

¿Cómo podemos identificar el handleMapper base de un GSH? Cualquier servicio que implemente la interfaz HandleMapper es un servicio Grid Service, y como tal tiene también un GSH. En la construcción del GSH, estamos de vuelta con el mismo problema de obtener un GSR desde el GSH del servicio handleMapper. Para resolver este problema, necesitamos la manera de poder obtener un GSR para el handleMapper sin la necesidad de un handleMapper. Esto se consigue mediante el requisito de que todos los servicios handleMapper base estén identificados por una URL y que soporten una operación de inicialización limitada un único y conocido protocolo, como HTTP (o HTTPS). Por lo tanto, en vez de usar un GSR para describir que protocolos deben ser usados para contactar con el servicio handleMapper, se usa una operación GET de HTTP sobre la URL que apunta al handleMapper base, y el GSR para el handleMapper, en forma WSDL, es devuelto.

Notar que existe una relación entre los servicios que implementan las interfaces *HandleMapper* y *Factory*. Específicamente, el GSH devuelto por una petición factory debe contener la URL del handleMapper base, y la correspondencia GSH/GSR debe ser introducida y actualizada en el servicio handleMapper. La implementación de la factory debe decidir que servicio hay que usar como el handleMapper base. También un sólo servicio puede implementar a ambas interfaces.

Información de servicios y Descubrimiento de servicios

Asociada a cada instancia Grid Service existe un conjunto de *servicios de información*, que son una colección de fragmentos XML encapsulados como Elementos de Información Grid Service (GSIE, de Grid Service Information Elements). La agrupación de GSIE incluye además un nombre que es único para la instancia Grid Service, un tipo, y el tiempo-de-existencia que puede usar para la gestión de su tiempo de vida.

La interfaz *Discovery* define operaciones estándar WSDL para recabar la información de un servicio y para la petición de conjuntos de información. Estas operaciones permiten la especificación del lenguaje usado, que puede ser por ejemplo Xquery.

La especificación Grid Service define para cada interfaz Grid Service un conjunto de cero o más GSIEs que deben ser soportados por cada instancia Grid Service que soporte a dicha interfaz. Asociada a la interfaz *Discovery*, y obligatoria para cada instancia Grid Service, hay un conjunto de GSIEs que contienen información básica acerca de la instancia Grid Service: GSH, GSR, clave primaria, y el handleMapper base.

Una aplicación de la interfaz *Discovery* es el descubrimiento de servicios. Como hemos dicho antes se posee un GSH para representar un servicio deseado. Pero ¿cómo se obtiene ese GSH por primera vez? Esta es la esencia del descubrimiento de servicios, que definiremos como el proceso de identificar un conjunto de GSHs a partir de un conjunto de atributos GSH como el proveedor de interfaces, el número de peticiones servidas, la carga en el servicio, o políticas como de peticiones permitidas.

Un Grid Service que permite el descubrimiento de servicios se llama *registry*. Un servicio *registry* está definido por dos cosas: una interfaz *Registry*, que proporciona mediante las cuales los GSHs se pueden registrar con el servicio registry, y un GSIE asociado que contiene información acerca de los GSHs registrados. Por lo tanto, la interfaz *Registry* es usada para registrar un GSH y la interfaz *Discovery* se usa para recabar información sobre los GSHs registrados.

La interfaz *Registry* permite a los GSH registrarse con un servicio registry determinado con el fin de aumentar el conjunto de GSHs considerados. Como ocurre en MDS-2, un servicio puede usar esta operación para notificar a los interesados dentro de una organización virtual (VO) de la existencia y del servicio que proporciona. Las partes interesadas normalmente incluyen varias normas de descubrimiento de formas que recolectan y estructuran la

información de los servicios de forma que puedan responder eficientemente a peticiones de descubrimiento de servicios. Al igual que con otras interfaces con estado en OGSA, el registro GSH es una operación relacionada con estados y debe ser periódicamente actualizada, para permitir a los servicios de descubrimiento el tratar con la disponibilidad dinámica de los servicios.

Hay que notar que la especificación de los atributos asociados a un GSH no está ligada al registro del GSH a un servicio que implemente la interfaz *Discovery*. Esta característica es importante porque lo valores de los atributos deben ser dinámicos y existen distintas formas en las que dichos valores son obtenidos, incluido la consulta a otro servicio que implemente la interfaz *Discovery*.

Cuando se usa una *factory* que implemente el descubrimiento de instancias Grid Services creadas por dicha factory, el cliente puede especificar la clave primaria del Grid Service a la factory como parte de la petición de creación. El tipo de la clave es especificado a la factory y la clave debe ser única con respecto a todas las instancias existentes en ese momento creadas por esa factory. Un cliente puede usar esa clave para pedir a la factory una determinada instancia de servicio.

Notificación

Las operaciones estándar WSDL están definidas para (1) permitir a los clientes registrarse para que sean informados de determinados mensajes, y (2) el reparto unidireccional y asíncrono de dichas notificaciones. El entorno de notificación permite tanto el reparto de mensajes de notificación entre servicio-servicio, y la integración con terceras partes, como servicios de mensajes comúnmente usados en el mundo comercial, o servicios que filtran, transforman y reparten mensajes especiales en la parte de la fuente de notificación. Por ejemplo, un protocolo SOAP/HTTP o una unión UDP proporcionará notificaciones punto-a-punto sin mucho esfuerzo, mientras otros protocolos de unión (como servicio propietario de mensajes) proporcionarán un mejor servicio que los anteriores. Un protocolo de unión de difusión podrá soportar varios destinatarios.

En la definición e implementación de mecanismos de notificación, podremos crear en otro servicio Grid Service los mecanismos para la creación fiable, la gestión del tiempo de vida, etc. Por ejemplo, una fuente de notificación contiene un estado acerca de que información está siendo propagada y a quién está siendo distribuida. Debe ser por tanto creada de forma fiable, y

debe acabar si los clientes fallan. Éstas son dos funcionalidades que ganamos "gratuitamente" gracias al uso de mecanismos Grid Service. Igualmente, el receptor de los mensajes también contiene un estado acerca que hacer con los mensajes recibidos. Debe ser creado también fiablemente y debe acabar en el caso en que la fuente falle. Los intermediarios en la comunicación son simplemente almacenes de fuentes y receptores, con varios estados definidos por sus semánticas (cacheo/registro de notificaciones, fiabilidad y comportamientos QoS).

Estás características están soportadas mediante las interfaces *NotificationSource* y *NotificationSink*. Si un servicio desea soportar la subscripción de mensajes de notificación, debe implementar la interfaz *NotificationSource* para poder manejar las subscripciones. La interfaz *NotificationSink* se usa para repartir mensajes de notificación a los servicios que la implementen. Para empezar la notificación desde un servicio en particular, se debe invocar la operación de subscripción en la interfaz de la fuente de notificación, dándole el GSH del servicio que quiere ser el receptor de la notificación. La acciones normales del servicio pueden ser ahora usadas para manejar el envío y recepción de notificaciones.

Un flujo de mensajes de notificación fluyen de la fuente al destino, mientras que el receptor envío periódicamente mensajes de *keepalive* para notificar a la fuente que sigue interesado en recibir dichas notificaciones. Si se desea un recibo fiable de notificaciones, este comportamiento puede ser implementado definiendo un protocolo de unión adecuado para el servicio.

Manejo de cambios

Para poder soportar el *descubrimiento* y el *manejo de cambios* de Grid Services, las interfaces Grid Service deben ser nombradas de manera global y única. En WSDL, una interfaz es definida definiendo un portType y es nombrada única y globalmente mediante el qname del portType. Cualquier cambio a estas operaciones, deben reflejarse en nuevos nombres de interfaz (e.d. nuevos portTypes). Esta característica permite a los clientes que requieren Grid Services con propiedades particulares (tanto interfaces en particular como semánticas de implementación) el descubrir servicios compatibles.

Gestión

Cada Grid Service debe soportar un conjunto estándar de operaciones WSDL de gestión. Estas operaciones permiten que grandes conjuntos de instancias de Grid Service sean monitorizadas y administradas desde consolas de administración, herramientas automáticas, y similares.

3.2. OGSI, la infraestructura antigua

La Open Grid Services Infrastructure (OGSI) versión 1.0, lanzada en Julio del 2003, define una serie de convenciones y extensiones en el uso de WSDL y de esquemas XML para el uso de Web Services con estado. Define mecanismos para la creación, el manejo, y el intercambio de información entre las entidades definidas anteriormente como Grid Services, ver 3.1.1.1.

La OGSA, ver 3.1, integra tecnologías Grid con mecanismos Web Service para crear el marco de un sistema distribuido basado en la *Open Grid Services Infrastructure (OGSI)*. Una *instancia Grid Service* es un servicio (no permanente) conforme a una serie de convenciones, expresado como una interfaces, extensiones y comportamientos *Web Service Definition Language* (WSDL), para propósitos el manejo del tiempo de vida, el descubrimiento de características, y la notificación. Los Grid Services proporcionan un manejo controlado del estado distribuido y, normalmente, duradero que requieren las aplicaciones distribuidas sofisticadas. OGSI además introduce una interfaz *factory* (ver 3.1.1.3) estándar y una de registro de servicios útiles para la creación y el descubrimiento de Grid Services.

Como ya sabemos OGSA está basado fundamentalmente en Grid Service, pero los Grid Services deben su especificación a OGSI. Todos los servicios en OGSA, como la gestión de trabajos, la seguridad, etc, están basados, e implementados, sobre Grid Services.

Y, ¿por qué OGSA usa Grid Services? Los Web Services son una muy buena opción, quizá la mejor, para OGSA, pero cuenta, como todo, con ciertas limitaciones y carencias que tratan de suplir los Grid Services:

- No tienen estado.
- No son persistentes (son transitorios).
- No tienen servicios de "apoyo" (notificaciones, gestión ciclo de vida, etc).

Por ello, la especificación OGSI *amplía* los Web Services para superar estas limitaciones, y así define los Grid Services que son Web Services ampliados y totalmente compatibles con éstos.

3.2.1. Mejora de los Web Services

Entre las mejoras llevadas a cabo por OGSI, sobre los Web Services y convertirlos en Grid Services, señalamos:

3.2.1.1. Soporte para la extensión y herencia de nuevas interfaces

OGSI se basa en Grid Services; en particular, usa WSDL como mecanismo para describir las interfaces públicas de los Grid Services. Sin embargo, la versión 1.1 de WSDL era deficiente en los aspectos de la falta de extensión de las interfaces (los portTypes) y la no posibilidad de poder describir elementos adicionales en un portType. Mientras el W3C Web Services Description Working Group se hacía cargo de esas carencias para la siguiente versión de WSDL, lo que hizo OGSI fue crear el GWSDL, que es una versión ampliada de WSDL 1.1, para solucionar momentáneamente esas carencias y seguir trabajando.

GWSDL es un *namespace* separado y temporal que añade una nueva construcción del elemento WSDL::portType según la especificación de la WSDL v1.2, para soportar la herencia y extensión del portType y el añadido de información adicional.

A continuación mostramos la definición de gwsdl::portType:

El atributo *extends* del elemento gwsdl::portType es una lista de QNames, donde cada QName debe referirse o a un wsdl::portType o a un gwsdl::portType.

En referencia a la extensión del portType, hay que hacer una aclaración con respecto a los elementos de operación. Aunque WSDL v1.1 permite la sobrecarga de nombres de operaciones, el W3C Web Services Decription Working Group ha decidido eliminar esta característica para la WSDL v1.2:

> Un componente de operación de un portType describe una operación que un portType dado soporta. Una operación es un conjunto de referencias a mensajes. Las referencias a mensajes deben ser a mensajes que la operación acepta, que son los mensajes de entrada, o a mensajes que la operación envía, que son los mensajes de salida o de error.

Los componentes de operación de un portType son locales a los componentes del portType, no pueden ser referenciados por un QName, a pesar de tener las propiedades de *{name}* y de *{target namespace}*.

Las propiedades de un Port Type Operation Component son:

- {name} un NCName definidos como los namespaces XML.
- {target namespace}
- {variety} Sólo salida, Sólo entrada, Entrada-Salida o Salida-Entrada.
- {messages} Un conjunto de componentes de referencias a mensajes.

Acerca de la equivalencia de componentes:

> Dos componentes del mismo tipo son consideradas equivalentes si los valores de las propiedades de un componente son iguales a los valores de las propiedades del segundo componente.
>
> Con respecto a componentes de mayor nivel, como mensajes, portTypes, servicios y bindings) la equivalencia se traslada a la equiparación del {name} y del {target namespace}.

3.2.1.2. Servicios con estado y potencialmente transitorios

Otra característica de los Web Services que no se desea en OGSI es que no poseen un estado (*stateless*), es decir, no mantienen su estado de una llamada a otra. En OGSI, si se quiere conservar ese estado de una llamada a otra, por lo que es otro factor que distingue los Grid Services de los Web Services.

Los Web Services son siempre persistentes, ya que se activan en el momento en el que se inicia el contenedor de Web Services y no son destruidos hasta que se para el contenedor. En este aspecto podría decirse que son creados estáticamente. Por el contrario, los Grid Services son transitorios, pueden ser creados y destruidos cuando queramos, ya que no están ligados al contenedor de Grid Services. Son creados y destruidos dinámicamente. Este dinamismo, como ya hemos visto anteriormente, se crean gracias a las instancias Factory, ver 3.1.1.3. Con ésta, tenemos las ventajas de configurar una *factory* que cree

instancias de un servicio en particular, tener varias instancias de un servicio por cliente, tener varios clientes por instancias, y la destrucción puede correr parte del cliente o de la factory.

Para la *creación* de una instancia Grid Service, el cliente debe realizar la petición mediante la invocación de la operación createService de una instancia de un servicio que implemente un portType que herede del portType Factory o que contenga métodos especializados para la instanciación de nuevos servicios, como el portType NotificationSource.

Para la *destrucción* de una instancia Grid Service, el cliente puede o realizar una invocación explícita a una operación de destrucción a la instancia del servicio o dejar que se acabe el tiempo de vida de la instancia, la cual se destruirá automáticamente.

3.2.1.3. Gestión del ciclo de vida

El ciclo de vida de cualquier instancia Grid Service está delimitado por la creación y la destrucción de dicha instancia. Los mecanismos de creación y destrucción son una propiedad típica del entorno contenedor de los Grid Services.

Una instancia Grid Service debe soportar una gestión adecuada de su tiempo de vida, en el cual un cliente negocia un tiempo de vida inicial durante la creación de la instancia en la factory, y clientes autorizados enviarán mensajes de requestTerminationBefore/After para ampliar ese plazo de tiempo. Si se alcanza el plazo, el contenedor de la instancia la destruirá, reclamará los recursos asociados, y eliminará todo rastro de la instancia de sus manejadores.

3.2.1.4. Service Data ("Datos del servicio")

Se trata de una de las aportaciones más interesantes de OGSI a los Web Services. Surgió de la necesidad de un mecanismo de exponer los datos del estado de las instancias a los clientes en las peticiones, actualizaciones y notificaciones.

Para poder proporcionar una completa descripción de una interfaz de un Web Service con estado (e.d. un Grid Service), es necesario describir los elementos del estado que pueden ser observables externamente. Por externamente observable, se refiere a que el estado de la instancia del servicio sea expuesta a los clientes haciendo uso de la declaración de la interfaz. La idea

de declarar service data como parte de una declaración de interfaz externa equivale a la declaración de atributos en un lenguaje de programación orientado a objetos. El service data puede ser accedido para lectura, actualización, o motivos de suscripción.

Facilita el descubrimiento, la inspección, la monitorización y la gestión de los Grid Services.

Puede estar formado por dos tipos de datos:

- Información de estado: como resultados de operaciones, información de ejecución, resultados intermedios, etc.
- Metadatos: como datos sobre el propio servicio (configuración, coste, carga actual).

El Service data del servicio se encuentra definido en el portType del namespace GWSDL, por ello WSDL no lo soporta. El tipo de datos del valor se encuentra especificado en XML Schema (xsd:int, xsd:float, xsd:string).

3.2.1.5. Notificaciones

El propósito de la notificación es el reparto de mensajes de interés desde una fuente de notificación a un sumidero de notificación de la siguiente manera:

- Una *fuente de notificación* es una instancia Grid Service que implementa el portType NotificationSource, y es el que envía los mensajes de notificación. Puede ser capaz de enviar mensajes a un número variado de sumideros.
- Un *sumidero de notificación* es una instancia Grid Service que recibe mensajes de notificación desde cualquier número de fuentes. Debe implementar el portType NotificationSink.
- Un *mensaje de notificación* es un elemento XML enviado desde una fuente a un sumidero. El tipo XML del elemento está determinado por la expresión de subscripción.
- Una *expresión de subscripción* es un elemento XML que describe que mensajes deben ser enviados desde la fuente al sumidero.

Este entorno de notificación permite el reparto de mensajes de notificación de servicio-a-servicio y la integración de servicios intermediarios de reparto. Estos servicios intermediarios incluyen: servicios de mensajería, servicios de filtrado de mensajes, y archivo de mensajes y repetición de envío.

3.2.1.6. Agrupación de servicios

Un *ServiceGroup* es una instancia Grid Service que guarda información acerca de un grupo de Grid Services. Estos Grid Services formarán parte de un determinado grupo por algún motivo, como ser parte de un servicio federado, o no puede haber relativa relación entre ellos, como servicios contenidos en un índice o registro por motivos de búsqueda.

Para que una instancia Grid Service puede ejercer esta funcionalidad deberá implementar el portType ServiceGroup. Existen tres portTypes que proporcionan una interfaz para los grupos de servicios: ServiceGroup, ServiceGroupEntry, y ServiceGroupRegistration.

3.3. De OGSI A WSRF

Como se ha apuntado en el anterior punto, la Open Grid Service Infrastructure v1.0 (OGSI), que fue presentada en Julio del 2003, define un conjunto de convenciones y extensiones en el uso de WSDL y del XML Schema para conseguir unos Web Services con estado. Introduce la idea de dichos Web Services con estado y define patrones para la creación, el nombramiento y el manejo del ciclo de vida de las instancias de los servicios; para la declaración y la inspección de los datos de los servicios; para la notificación asíncrona de cambios del estado del servicio; para la representación y el manejo de colecciones de instancias de servicios; y para el manejo habitual de fallos en las instancias.

En Enero del 2004, se presentó el WS-Resource Framework (WSRF) como una nueva evolución de OGSI dirigida hacia la explotación de los nuevos estándares en Web Services, más concretamente WS-Addressing, y hacia una sustitución de OGSI con la ayuda de la experiencia obtenida.

3.4. Web Services Resource Framework (WSRF) la infraestructura actual

WSRF es una especificación, desarrollada por OASIS, cuyo propósito es definir un marco genérico para el modelado y acceso a los recursos persistentes usados por los Web Services. Incluye mecanismos para describir el estado, para soportar la administración del estado a través de las propiedades del Web Service, y de cómo se extienden estos mecanismos al resto de Web Services.

El WSRF es un conjunto de especificaciones para los Web Services que son una acercamiento a la propuesta WS-Resource en términos de un intercambio específico de mensajes y definiciones XML relacionadas. El WS-Resource es un compendio que ha sido propuesto como el medio de expresar la relación entre los recursos persistentes y lo Web Services. Las especificaciones encontradas en el WSRF permiten al programador declarar e implementar la asociación entre un Web Service y uno o varios recursos persistentes. Además describen como se hace, el estado de un Web Service, accesible a través de una interfaz Web Service y definen mecanismos relacionados con el direccionamiento y el agrupamiento definidos en el WS-Resource.

3.4.1. Introducción

Las interfaces Web Service proporcionan a menudo a un usuario la habilidad de acceder y manipular el estado, es decir, valores de datos que persisten a través de, y que evolucionan como resultado de, interacciones Web Service. En otras palabras, el mensaje de intercambiar que los Web Services implementan se entiende que habilita el uso de recursos persistentes. Sin embargo, la noción de recursos persistentes que se presupone de la implementación de los Web Services en realidad no está explícita en dicha definición del interfaz. Los mensajes que los servicios envían y reciben, *"implican"* la existencia de algún tipo de recurso asociado persistente.

Es deseable pues, la definición de un serie de convenciones Web Service que permitan la interacción con recursos persistentes de manera estándar. Esto desembocó en la propuesta WS-Resource que modela un estado en el contexto de los Web Services. Un WS-Resource se define como una composición entre un Web Service y un recurso con estado que es (1) expresado como una asociación entre un documento XML, con un tipo definido, y un portTy-

pe Web Service, y es (2) direccionado y accedido de acuerdo al patrón del recurso, es decir, mediante el uso de referencias a WS-Addressing endpoints. En el patrón del recurso implicado, un identificador de un recurso con estado es encapsulado en una referencia endpoint y se usa para identificar el recurso con estado que se empleará en la ejecución el intercambio de mensajes del Web Service.

El WS-Resource Framework (WSRF) permite la declaración, la creación, el acceso, la monitorización de cambios y la destrucción de WS-Resources mediante los mecanismos convencionales empleados por los Web Services, pero no requiere que el componente Web Service del WS-Resource, que proporciona el acceso a los recursos con estado, sea implementado como un procesador de mensajes con estado.

El WSRF es un conjunto de cinco especificaciones técnicas que definen la descripción de la propuesta WS-Resource en términos de intercambios de mensajes específicos de Web Services y de definiciones XML relacionadas. Estas especificaciones, resumidas en la tabla 3.1 en la página siguiente, definen los medios por los cuales:

- se destruye un WS-Resource, y a su vez sincronizado con una petición de destrucción o a través de mecanismos de destrucción basados en el tiempo, y cuáles son las propiedades del recurso que nos sirven para inspeccionar y monitorizar el el tiempo de vida de un WS-Resource.

- el tipo de la definición de un WS-Resource puede ser compuesta por una descripción de interfaz de un Web Service y por un documento XML sobre las propiedades de un recurso, y como el estado del WS-Resource puede ser pedido y modificado a través del intercambio de mensajes Web Service.

- una referencia endpoint de un Web Service puede ser renovada cuando su direccionamiento o la información sobre la política que tiene se vuelve fallida.

- pueden ser definidas colecciones heterogéneas de Web Services, teniendo o no WS-Resources como servicios.

- la información sobre posibles fallos puede ser devuelta de una manera más estandarizada usando tipos XML Schema para los fallos base y reglas sobre cómo esos fallos se usan y se extienden a los Web Services.

Nombre	Describe	Sección
WS-ResourceLifetime	Mecanismos para la destrucción de WS-Resources, incluyendo el intercambio de mensajes que permite al usuario destruir el WS-Resource de forma inmediata o mediante un tiempo planificado.	4.4.2
WS-ResourceProperties	Definición de un WS-Resource, y mecanismos para recabar, cambiar y borrar las propiedades del WS-Resource.	4.4.3
WS-RenewableReferences	Funcionalidad adicional para las referencias WS-Addressing endpoint, con información sobre las políticas empleadas, necesaria para recabar información de nuevas versiones cuando las referencias empleadas pasan a ser inválidas.	4.4.4
WS-ServiceGroup	Una interfaz para colecciones de Web-Services.	4.4.5
WS-BaseFaults	Un tipo XML para los fallos base usado para devolver fallos durante el intercambio de mensajes en los Web Services.	4.4.6

Cuadro 3.1: WS Resource Framework

3.4.2. WS-ResourceLifetime

La especificación WS-ResourceLifetime está encaminada a definir tres importantes aspectos acerca del ciclo de vida del WS-Resource, de su creación (**WS-Resource factory**), de la identificación (WS-Resource identity), y de la destrucción (WS-Resource destruction).

3.4.2.1. WS-Resource Factory

El WS-Resource Framework no trata de definir los intercambios de mensajes usados en la creación de WS-Resources, simplemente puntualiza que los WS-Resources deben ser creados por algún tipo de mecanismo, o mediante el uso de un patrón de uso para los Web Services que la especificación WS-ResourceLifetime denomina WS-Resource factory. Una WS-Resource factory es un Web Service con la capacidad de crear nuevos WS-Resources. El mensaje típico de petición de la operación de la WS-Resource factory contiene al menos una referencia endpoint que se refiere al nuevo WS-Resource, luego es la factory la responsable de relacionar la referencia con el nuevo WS-Resource y de realizar algún tipo de almacenamiento para su posterior uso.

3.4.2.2. WS-Resource Identity

El componente Web Service del WS-Resource puede construir una dirección para el WS-Resource incluyendo un identificador del recurso con estado en la componente propiedades de una referencia WS-Addressing endpoint. Esa referencia endpoint entonces es puesta a disposición de las entidades de un sistema distribuido, que pueden usar dicha referencia para enviar peticiones a dicho WS-Resource. Es por tanto la referencia WS-Addressing endpoint la que identifica el servicio que ofrece Web Service, que forma parte del WS-Resource, y es éste el que usa el identificador, incluido en las propiedades de la referencia. para saber que recurso con estado se menciona en la ejecución del mensaje.

3.4.2.3. WS-Resource Destruction

Un usuario que pide a una WS-Resource factory crear un nuevo WS-Resource estará interesado en usar el nuevo WS-Resource durante un tiempo limitado. Después de ese tiempo, será posible destruir el WS-Resource de forma que su sistema asociado o lo recursos empleados por ése pueden ser

liberados. El WS-Resource framework estandariza dos tipos de propuestas para la administración del tiempo de vida de los WS-Resources: la destrucción inmediata o la destrucción planificada, es decir, con un tiempo límite.

3.4.3. WS-ResourceProperties

La especificación WS-ResourceProperties define el tipo y los valores de aquellos componentes del estado de un WS-Resource que pueden ser accedidos y modificados por usuarios del servicio a través de las interfaces Web Service. Las ideas principales son:

- El WS-Resource tiene un *documento XML de propiedades del recurso* definido mediante XML Schema.
- Los usuarios del servicio determinarán el tipo del WS-Resource mediante la obtención, usando los medios estándar, de la definición WSDL del portType.
- Los usuarios del servicio usarán el intercambio de mensajes entre Web Services para leer, modificar, y consultar el documento XML que representa el estado WS-Resource.

Se usa el término *propiedad del recurso* para referirse a una componente individual de un estado WS-Resource. Y se le llama al documento XML que describe el tipo de un recurso con estado, que pertenece a un WS-Resource, con el nombre de *documento de las propiedades del WS-Resource*.

Por último, la especificación WS-ResourceProperties también define los medios por los que los usuarios de los servicios pueden suscribirse para recibir notificaciones de cambios en los valores de las propiedades de un recurso de un WS-Resource.

3.4.4. WS-RenewableReferences

La especificación WS-RenewableReferences define mecanismos que serán usados para renovar una referencia endpoint que ha pasado a ser inválida. Estos mecanismos pueden ser aplicados a cualquier referencia, pero son especialmente útiles en el caso de una referencia endpoint que se refiere a un WS-Resource, ya que puede proporcionar una referencia estable y persistente al WS-Resource que permitirá repetidos accesos al mismo estado.

Una referencia endpoint WS-Addressing debe contener, además de la información acerca del direccionamiento, información acerca de la política concerniente a la interacción con el servicio. Típicamente, las referencias endpoint se construyen mediante información, del direccionamiento y de la política empleada, proveniente de una fuente autorizada. Una referencia disponible para un cliente, es en realidad una copia de aquella información y que, en algún momento, pasará a ser incoherente debido a cambios, realizados por la fuente autorizada, que afectan a la localización del endpoint y/o a la política empleada. En estas situaciones, se hace realmente importante el renovar la referencia endpoint.

3.4.5. WS-ServiceGroup

La especificación WS-ServiceGroup define los medios para representar y manejar colecciones heterogéneas de Web Services. Esta especificación puede ser usada para organizar colecciones de WS-Resources, por ejemplo para construir registros, o para construir servicios que puedan realizar operaciones colectivas sobre colecciones de WS-Resources.

Esta especificación puede expresar reglas, obligaciones, y clasificaciones, para los miembros del ServiceGroup, usando el modelo de la propiedad del recurso de la WS-ResourceProperties. Los grupos pueden ser definidos como colecciones de miembros que cumplen con las obligaciones y restricciones expresadas en las propiedades del recurso. La especificación además debe definir interfaces para la administración de los miembros del ServiceGroup.

Las interfaces definidas por el WS-ServiceGroup se suponen que estarán compuestas por interfaces Web Services que definen interacciones más especializadas con el grupo o con los servicios miembros del grupo.

3.4.6. WS-BaseFaults

La especificación WS-BaseFaults define un tipo base para fallos que debe usarse al devolver fallos en el intercambios de mensajes Web Services. Como no existe nada concreto dentro de esta especificación, no quiere decir que no sea utilizada por las distintas implementaciones WS-Resource framework para dar consistencia a los fallos devueltos por las operaciones en las especificaciones, incluyendo la notificación consistente de fallos relacionados con la definición de WS-Resource y su uso.

Capítulo 4

Globus Toolkit 4

4.1. Introducción

A medida que avanzaba la tecnología en interconexión de computadoras y el número de éstas se multiplicaba, se empezó a pensar que podía ser posible crear *entornos distribuidos de alto rendimiento a gran escala*. La idea era poder acceder a recursos de gama alta, tanto por capacidad de cálculo como de almacenaje, de forma extendida. Así, a finales de 1994, Rick Stevens, director de matemáticas y de la división de informática en el Laboratorio Nacional Argonne, y Tom DeFanti, director del Laboratorio de Visualización Electrónica en la Universidad de Illinois, propusieron el experimento de crear un *grid* a nivel nacional (el "Y-WAY") conectando temporalmente 11 redes de alta velocidad durante dos semanas. Haciéndose cargo del reto, un grupo liderado por Ian Foster en Argonne creó los nuevos protocolos necesarios que permitieron a los usuarios ejecutar aplicaciones en ordenadores por todo el país. Gracias a este exitoso experimento se consiguió financiación por parte del DARPA (Defense Avanced Research Projects Agency), que tendrían como fruto inmediato la aparición de la primera versión del Globus Toolkit en 1997.

A medida que avanzaba la tecnología en interconexión de computadoras y el número de éstas se multiplicaba, se empezó a pensar que podía ser posible crear *entornos distribuidos de alto rendimiento a gran escala*. La idea era poder acceder a recursos de gama alta, tanto por capacidad de cálculo como de almacenaje, de forma extendida. Así, a finales de 1994, Rick Stevens, director de matemáticas y de la división de informática en el Laboratorio Nacional Argonne, y Tom DeFanti, director del Laboratorio de Visualización

Electrónica en la Universidad de Illinois, propusieron el experimento de crear un *grid* a nivel nacional (el "Y-WAY") conectando temporalmente 11 redes de alta velocidad durante dos semanas. Haciéndose cargo del reto, un grupo liderado por Ian Foster en Argonne creó los nuevos protocolos necesarios que permitieron a los usuarios ejecutar aplicaciones en ordenadores por todo el país. Gracias a este exitoso experimento se consiguió financiación por parte del DARPA (Defense Avanced Research Projects Agency), que tendrían como fruto inmediato la aparición de la primera versión del Globus Toolkit en 1997. Actualmente, el Globus Toolkit es la base de varios productos Grid comerciales y existen, además, proyectos científicos y de ingeniería con presupuestos millonarios, repartidos por todo el mundo, de los cuales es una parte central.

Entonces, ¿qué es el Globus Toolkit? El Globus Toolkit es un *software* de código libre usado para crear grids. Esto permite a las personas y a las compañías compartir sus recursos, ya sea poder de cálculo, bases de datos, o cualquier otro recurso, de forma segura y sin importar su situación geográfica. Las relaciones con los conceptos anteriormente estudiados pueden observarse en la figura 4.1. Globus implementa los servicios necesarios impuestos por el estándar OGSA, apoyándose en la infraestructura de WSRF, de la que tiene una implementación completa. Después de esta breve introducción veamos cómo es por dentro el globus toolkit.

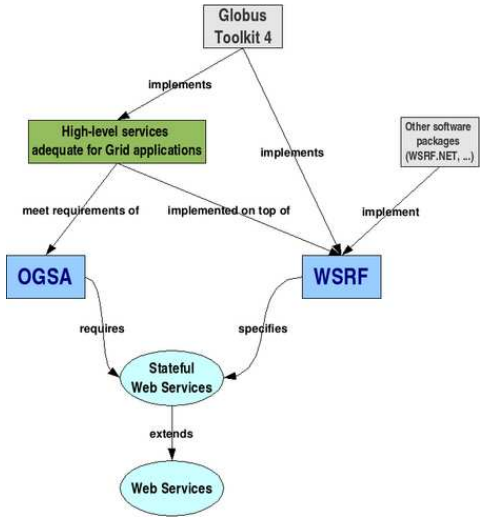

Figura 4.1: Relación entre OGSA, WSRF y Globus

4.2. Arquitectura de Globus Toolkit 4

La filosofía con la que se desarrolla el *toolkit* es servir de *middleware* a las aplicaciones que se ejecuten en el grid. Es decir, ofrecer una serie de servicios que sirvan para acceder de forma transparente y segura al grid. Actualmente, los servicios que implementa son gran parte de los marcados en el estándar OGSA, expuesto en el apartado 3.1, y esto ha convertido al Globus Toolkit en un estándar *de facto* en la computación grid.

En cuanto a la arquitectura, el Globus Toolkit 4 está compuesto de varios componentes *software*. Como podemos ver en la figura 4.2, los componentes pertenecen a una de las siguientes cinco categorías: Seguridad, Manejo de Datos, Control de Ejecución, Servicios de Información, y *Common Runtime*. La mayoría de estos servicios son Web Services, apoyados en la completa implementación de WSRF que tiene el *toolkit,* pero existen también elementos llamados no-WS o pre-WS. Estos elementos se han convertido en un estándar del *GGF* (Global Grid Forum), debido a que las implementaciones actuales de los Web Services son demasiado lentas para utilizarlas en partes de la ejecución donde el rendimiento es crítico.

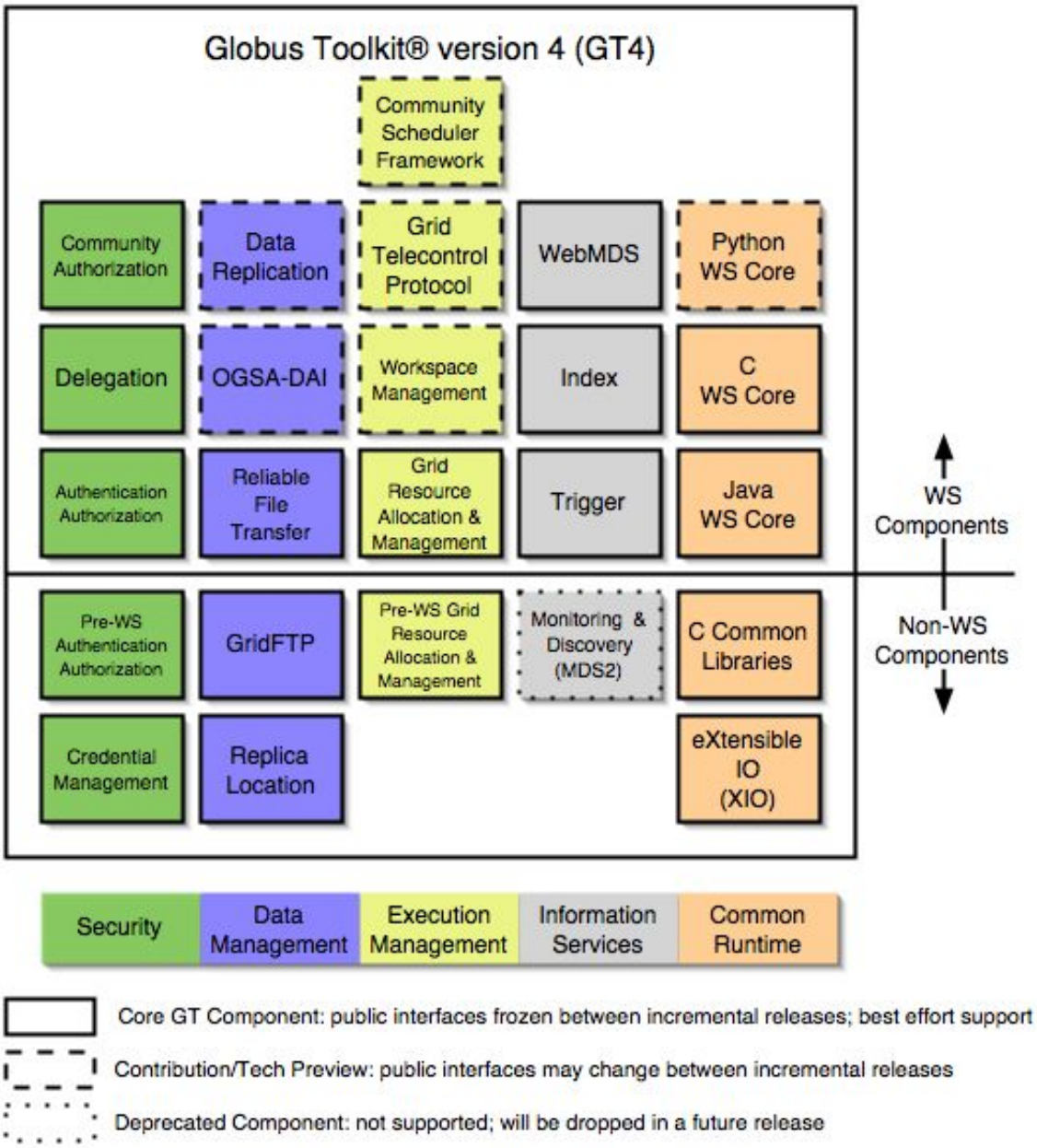

Figura 4.2: Componentes de GT4

4.2.1. Common Runtime

Los componentes de *Common Runtime* proporcionan un conjunto de librerías fundamentales y herramientas que son necesarias para construir el resto de los servicios, ya sean Web Services o no-WS. Estas herramientas y librerías permiten a los servicios ser independientes de la plataforma donde se ejecuten. Los componentes de esta sección son bastante heterogéneos, así que vamos a verlos por separado:

WS Core proporciona APIs (en C, Java y Python) para crear Web Services WSRF y clientes según los estándares *WSRF* (Web Services Resource Framework) y *WSN* (Web Service Notification).

C Common Libraries es la librería donde se encuentran definidos los tipos y estructuras de datos utilizadas a lo largo del *toolkit*. Ofrece, además, una serie de funciones que sirven como envoltorio a las de la *libc*. La librería sirve, por tanto, para ofrecer un nivel de abstracción más a los desarrolladores tanto de Globus Toolkit como de herramientas relacionadas. Algunos ejemplos son la llamada *globus_free* o el tipo de datos *globus_boolean_t*.

XIO es una librería escrita en C que proporciona un interfaz común de entrada salida tanto para el manejo de archivos como para los protocolos TCP, UDP, HTTP, GSI, GSSAPI_FTP y TELNET. En la figura 4.3 está representada esquemáticamente la arquitectura XIO, donde destacan los componentes *framework* y la pila de *drivers*. El Globus XIO framework recibe las peticiones de entrada/salida realizadas a través de la API y las direcciona al interfaz de los *drivers*. Los datos del usuario serán manipulados en cada uno de los *drivers* de transformación existentes en la pila. El resultado de las transformaciones se le pasará al *driver* de transporte, que es el encargado de realizar las operaciones de enviar y recibir. Los datos recibidos tendrán que pasar la pila de *drivers* en sentido inverso.

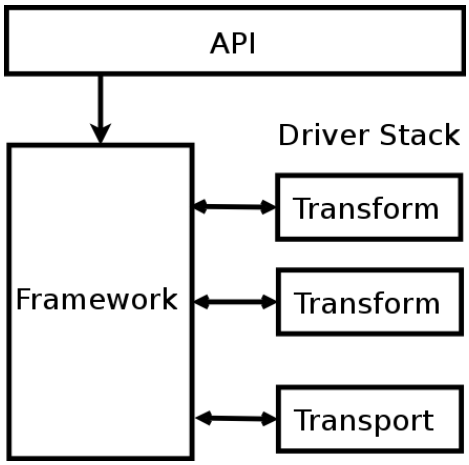

Figura 4.3: Arquitectura XIO

4.2.2. Seguridad

Globus Toolkit utiliza los mecanismos de seguridad de Grid explicados en el punto 1.2.2., ver 1.2.2 en la página 37 Su base es la criptografía asimétrica.

Las principales motivaciones que se esconden detrás de GSI (Grid Security Infraestructure) son:

- La necesidad de una comunicación segura (autentificada y quizás confidencial entre elementos de un Grid
- La necesidad de mantener la seguridad más allá de los límites de la organización.

Construido dentro de GSI se encuentra CAS (Community Authorization Server), CAS permite a los proveedores de recursos especificar las políticas de control de acceso en términos de comunidades, delegando políticas de control de acceso de grano fino a la comunidad.

CAS funciona de la siguiente manera:

1. Se inicia un servidor CAS para una comunidad: un representante de la comunidad adquiere una credencial GSI para representar a la comunidad como un todo, y entonces ejecuta un servidor CAS usando esa identidad.

2. Los proveedores de recursos proporcionan privilegios a la comunidad. Cada proveedor de recurso verifica que el poseedor de la credencial y las políticas de la comunidad son compatibles con las políticas del proveedor de recursos. Una vez que se establece una relación de confianza, el proveedor de recursos proporciona derechos a la comunidad.

3. Los representantes de la comunidad usan CAS para gestionar las relaciones de confianza de la comunidad y proporcionar control de acceso de grano fino a los recursos. El servidor CAS también se usa para gestionar sus propias políticas de control de acceso; por ejemplo, los miembros de la comunidad que tienen los privilegios adecuados pueden autorizar a otros miembros de la comunidad para gestionar grupos, dar permisos para el uso de los recursos de la comunidad, etc.

4. Cuando un usuario quiere acceder a los recursos ofrecidos por el CAS, ese usuario hace una petición al servidor CAS. Si la base de datos del servidor CAS indica que el usuario tiene los privilegios adecuados, el CAS emite al usuario una credencial, dando al usuario el derecho de realizar las acciones requeridas.

5. El usuario entonces, utiliza las credenciales del CAS para conectar con el recurso con cualquier herramienta normal de Globus (por ejemplo GridFTP). El recurso entonces aplica su política local para determinar los accesos proporcionados a la comunidad, y restringe este acceso basándose en la política de las credenciales CAS.

Para la comunicación por mensajes, el GT 4.0, WS Authentication & Authorization Message-Level Security implementa el estándar WS-Security y la especificación WS-SecureConversation para proporcionar protección a los mensajes SOAP. Las características incluyen autentificación del emisor, encriptación del mensaje, protección de la integridad del mensaje y protección de repetición.

La implementación WS-Security trabaja en el procesamiento de mensajes. Busca el mensaje de cualquiera de las cabeceras WS-Security. De estas cabeceras extrae cualquier material relacionado con las claves, que puede ser un certificado X.509 o una referencia a una sesión de conversación segura establecida previamente. También comprueba cualquier firma o elementos de desencriptación en el cuerpo SOAP.

El SecureConversation trabaja en la seguridad en el lado del servidor. En el SecureConversation el cliente establece un contexto con el servidor antes de enviar cualquier dato. Este contexto sirve para autentificar la identidad del cliente al servidor y para establecer una seña compartida usando GSI Secure Conversation Service. Una vez que el establecimiento del contexto de completa, el cliente puede invocar de manera segura una operación del servicio registrándose, o encriptar los mensajes salientes usando la seña secreta capturada en el contexto.

WS Authentication and Authorization Transport-Level Security proporciona un canal seguro para usar HTTP sobre SSL/TLS (HTTPS) para el trasporte de mensajes. Este mecanismo de seguridad soporta todas las herramientas proporcionadas por SSL/TLS además de soportar los certificados Proxy X.509

4.2.3. Gestión de datos

El Globus Toolkit proporciona una serie de componentes que permiten gestionar conjuntos grandes de datos dentro de nuestra organización virtual. Estos componentes se dividen en dos categorías principales: de *movimiento de datos* y de *replicación de datos*.

4.2.3.1. Movimiento de datos

GridFTP Es un protocolo que proporciona una transferencia de datos segura, robusta, rápida y eficiente. Está basado en el popular protocolo de transporte de archivos de Internet *FTP* (File Transfer Protocol), añadiendo algunas características y eliminando otras, para cumplir con los requisitos de los proyectos actuales de data grid.

Reliable File Transfer RFT es un web service WSRF que proporciona la funcionalidad de envío y borrado de archivos y directorios. El usuario crea un recurso RFT lanzando una petición (que es un conjunto de transferencias gridFTP) a un RFT Factory service. El recurso, que será creado una vez el usuario es correctamente autorizado y autenticado, tiene operaciones para controlar y gestionar las transferencias. Además, muestra el estado de la transferencia como una característica del recurso, o *resource property*, luego el usuario puede suscribirse para ser informado de sus cambios o hacer consultas periódicas.

4.2.3.2. Replicación de datos

Replica Location Service RLS es un componente de gestión de datos para entornos Grid. Proporciona la capacidad de mantener una o más copias de ficheros en un entorno grid, manteniendo un registro de en qué sistemas de almacenamiento físico existen replicas de un mismo archivo. Los usuarios registran los ficheros en un servidor RLS cuando son creados. Así, consultando posteriormente los servidores, podrán encontrarse esas réplicas. El trabajo de RLS es, por tanto, mantener asociaciones entre los nombres de ficheros lógicos y uno o más nombres de ficheros físicos de las réplicas. Un usuario puede proporcionar un nombre lógico de fichero a un servidor RLS y pedir todos los nombres físicos registrados de las réplicas. El usuario también puede consultar un servidor para averiguar el nombre lógico asociado a la ubicación física de un fichero determinado.

El componente RLS está pensado como registro distribuido, es decir, consiste en múltiples servidores en diferentes sitios. Así es más escalable y se evita la creación de un único punto de fallo en el sistema de gestión de datos del grid. Sin embargo, si se desea, RSL también puede ser desplegado como un servidor centralizado.

Data Replication Service DRS de Globus Toolkit 4 proporciona un servicio de gestión de datos a alto nivel que combina los dos componentes de gestión de datos existentes, RFT y RLS. DRS (Data Replication Service) es un servicio de gestión de datos a alto nivel cuya función es asegurar que un conjunto de ficheros específico resida en un lugar de almacenamiento.

El servicio DRS empieza consultando los servidores RLS para descubrir donde se localizan los ficheros deseados dentro del grid. Una vez que se localizan los ficheros, DRS crea una petición de transferencia que es ejecutada por RFT. Cuando las transferencias se han completado, DRS registra las nuevas réplicas en RLS.

DRS está implementado como un web service y sigue las especificaciones de WSRF. Cuando se recibe una petición de DRS, crea un WS-Resource que es usado para mantener el estado de cada fichero que está siendo replicado, incluyendo información sobre qué operaciones con el fichero han fallado o tenido éxito.

4.2.4. Servicios de información

Los servicios de información de Globus Toolkit 4, llamados *MDS* (Monitoring and Discovery Services), son una serie de componentes que realizan funciones como descubrir y monitorizar recursos dentro de una organización virtual. Aunque en la figura 4.2 se muestra que existe un elemento *MDS* no-WS, éste será eliminado con toda seguridad en futuras versiones del *toolkit.*

Los servicios *MDS* proporcionan interfaces de consulta y subscripción a datos detallados sobre recursos arbitrarios. Más concretamente, *MDS4* incluye los siguientes servicios basados en WSRF:

Index Service es un registro similar a UDDI, pero mucho más flexible. Los índices recogen información y la publican como *propiedades del recurso.* Los clientes usarán los interfaces de consulta y de suscripción/notificación para recuperar información de un índice. Los índices pueden registrarse unos a otros de forma jerárquica para agregar datos a diferentes niveles. Como nota adicional, hay que destacar que cada contenedor de Globus que tenga instalado *MDS4,* automáticamente tiene una instancia *Index Service* por defecto. Cualquier servicio GRAM, RFT, o CAS que se ejecute se registrará por defecto.

Trigger Service es un servicio que ejecuta una serie de acciones si se cumpla cierta condición asociada a un recurso. Para ello, recoge información sobre los recursos y la compara con el conjunto de condiciones definidas en un fichero de configuración. Cuando se cumple una condición, tiene lugar su acción asociada, como por ejemplo, enviar un correo al administrador del sistema cuando queda poco espacio en disco.

Web MDS que proporciona a los usuarios la capacidad de monitorizar la información a través de un navegador estándar, sin necesidad de instalar ningún software adicional. WebMDS ha sido implementado como un *servlet*[1] que reúne toda la información monitorizada y la transforma mediante *XSLT* (Extensible Stylesheet Language Transformation) hasta convertirla en HTML.

[1] un *servlet* es un módulo de código Java que se ejecuta en la parte del servidor para responder a las peticiones de los clientes.

4.2.5. Gestión de ejecución

Los componentes de la gestión de la ejecución son los que tratan con la inicialización, monitorización, manejo, planificación y coordinación de los programas ejecutados en un grid, normalmente llamados tareas o *jobs*. El Globus Toolkit 4 proporciona tanto un sistema de Web Services como un servidor Unix no-WS para enviar, monitorizar y cancelar las tareas de los recursos de computación del grid. Ambos sistemas se conocen con el nombre *GRAM* (Grid Resource Allocation and Management), mientras que WS GRAM se refiere solo a la implementación como servicio.

Las tareas son trabajos computacionales que pueden realizar operaciones de entrada/salida mientras se ejecutan, lo que afecta al estado de los recursos de cálculo y sus sistemas de ficheros asociados. En la práctica, dichas tareas pueden necesitar un *staging* coordinado de datos dentro del recurso antes de la ejecución de la tarea y fuera del recurso después de la ejecución. Es decir, la aplicación puede necesitar algunos ficheros como entrada, que habrá que copiarlos en el host donde se ejecutará, y puede generar una serie de archivos como resultado, que habrá que transportar a la máquina que hizo la petición.

En la siguiente sección se estudia más detenidamente el protocolo seguido por WS.

4.2.5.1. WS GRAM

El Globus Toolkit 4 proporciona tanto un sistema de web services como un servidor Unix pre-ws para enviar, monitorizar y cancelar las tareas de los recursos de calculo de Grid. Ambos sistemas se conocen con el nombre GRAM, mientras que WS GRAM se refiere solo a la implementación web service. Las tareas son trabajos computacionales que pueden realizar operaciones de entrada/salida mientras se ejecutan, lo que afecta al estado de los recursos de cálculo y sus sistemas de ficheros asociados. En la práctica, dichas tareas pueden necesitar un staging coordinado de datos dentro del recurso antes de la ejecución de la tarea y fuera del recurso después de la ejecución. Algunos usuarios de benefician del acceso a los ficheros de datos de salida mientras la tarea está ejecutándose. La monitorización consiste en consultar y añadir información del estado como por ejemplo cambios en el estado de la tarea.

Los recursos de cálculo de Grid suelen operar bajo el control de un planificador que implementa las políticas de asignación y prioridad a la vez que

optimiza la eficiencia y la productividad de la ejecución de todas las tareas enviadas. GRAM no es un planificador de recursos, sino un protocolo de comunicación con una gama de diferentes planificadores de recursos locales mediante el uso de un formato de mensajes estándar.

WS GRAM combina servicios de gestión de tareas y adaptadores al sistema local con otros componentes de GT 4.0 para soportar la ejecución de ficheros con staging de ficheros coordinado.

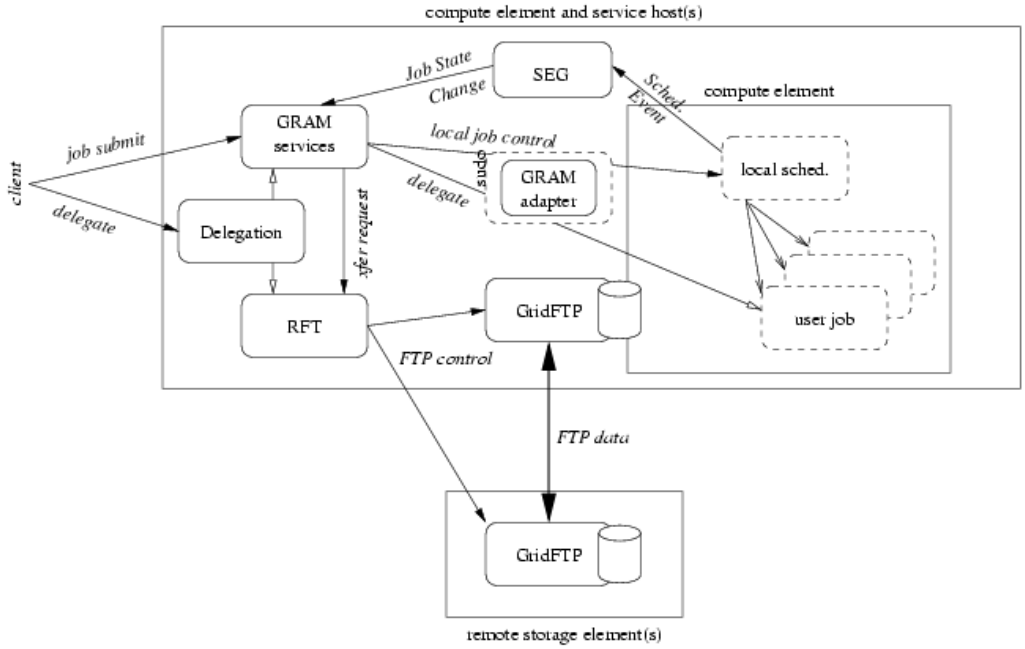

Figura 4.4: Componentes de WS GRAM

El núcleo de la arquitectura del servicio WS GRAM es un conjunto de web services diseñado para residir en el entorno de hosting del núcleo de WSRF.

GRAM está diseñado para tratar un conjunto de tareas donde la fiabilidad de la operación, la gestión de credenciales, la monitorización del estado y el staging de los ficheros son importantes. No está pensado para ser usado como un interfaz RPC para aplicaciones que no requieren muchas de las anteriores propiedades. Es más, su modelo de interfaz y su implementación pueden ser demasiado costoso para dichos usos. El servicio WS GRAM será más

adecuado a medida que las tecnologías web service subyacentes mejores, pero al igual que con el antiguo pre-WS GRAM, sus protocolos siempre implicarán muchos rodeos para soportar las propiedades avanzadas que no son necesarias para aplicaciones RPC simples.

Capítulo 5

Gridway

Gridway es una entorno de envío basado en Globus que permite una ejecución más sencilla y más eficiente de tareas en entornos dinámicos grid. Gridway realiza automáticamente todos los pasos de planificación de tareas, proporciona mecanismos de recuperación en caso de fallo, y adapta la planificación y ejecución de tareas a las condiciones cambiantes del grid.

5.1. Arquitectura

El núcleo de la estructura es un agente de envío que realiza todas las etapas del envío y que vigila la ejecución eficiente de la tarea. La adaptación a las condiciones cambiantes se consigue mediante replanificación dinámica. Una vez que la tarea se ha asignado, se puede replanificar cuando el rendimiento decrece o se detecta un fallo remoto, o periódicamente mediante el intervalo de *discovering*. El rendimiento de la aplicación se evalúa periódicamente en cada intervalo de *monitoring*, observando el tiempo en suspensión acumulado. Un modulo selector de recursos actúa como un distribuidor personal de recursos para construir una lista de prioridad de los recursos candidatos.

El agente de envío consta de los siguientes componentes:

Request Manager (RM): Para gestionar las peticiones del cliente.

Dispatch Manager (DM): Para realizar la planificación de tareas.

Submission Manager (SM): Para ejecutar y migrar tareas.

Performance Monitor (PM): Para evaluar el rendimiento.

La flexibilidad de la estructura es garantizada por un API bien definido para cada componente del agente de envío. Además, la estructura ha sido diseñada para ser modular para permitir la adaptabilidad, extensibilidad y mejora de sus funcionalidades. Los siguientes módulos pueden ser activados en un entorno por tareas:

Resource Selector (RS): Usado por el dispatch manager para seleccionar el host más adecuado para ejecutar la tarea de acuerdo con la categoría y arquitectura del host.

Middleware Access Driver (MAD): Usado por el submission manager para proporcionar un interfaz con el middleware de gestión de recursos subyacente.

Performance Evaluator (PE): El monitor de rendimiento lo usa para comprobar el progreso de la tarea.

Prolog: Usado por el submission manager para crear la jerarquía de directorios de la tarea en la máquina remota y transferir los ficheros de ejecución, de entrada y de reanudación.

Wrapper: Usado por el submission manager para lanzar a ejecución el programa y capturar el código de salida

Epilog: Usado por el submission manager para transferir los ficheros de salida y reanudación, limpiar la caché GASS y eliminar el directorio de la tarea de la máquina remota.

Ver 5.1 en la página siguiente.

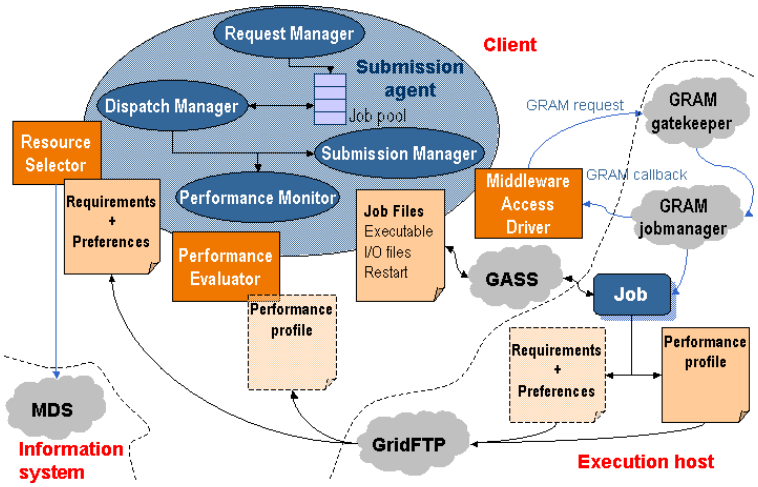

Figura 5.1: Arquitectura de GridWay

El agente de envíos realiza las siguientes acciones:

- La aplicación cliente usa un API de cliente para comunicarse con el request manager para enviar la tarea junto con su fichero de configuración, que contiene todos los parámetros necesarios para su ejecución. Una vez enviado, el cliente puede pedir operaciones de control al request manager, como tareas de *parada/reanudación*, *eliminación* o *replanificación.*

- El dispatch manager periódicamente, en cada intervalo *schedulling*, intenta enviar tareas *pendientes* y *replanificadas* a los recursos del Grid. Invoca la ejecución del módulo selector de recursos correspondiente a cada tarea, que devuelve una lista de prioridad de los hosts candidatos. El dispatch manager envía las tareas pendientes mediante la invocación de un submission manager, y también decide si la migración de las tareas replanificadas vale la pena o no.

- El submission manager es responsable de la ejecución de la tarea hasta que esta acaba o se detiene. Es invocado inicialmente por el dispatch manager junto con el primer host seleccionado, y es responsable también de realizar la migración de tareas a un nuevo recurso. Los componentes de gestión y protocolos de Globus son usados para dar soporte a estas acciones. El submission manager realiza las siguientes tareas:

- *Prólogo*: Prepara el RSL y envía el prólogo ejecutable. El prólogo activa el sistema remoto y transfiere los ficheros ejecutables y de entrada.
- *Envío*: Prepara el RSL, envía el wrapper ejecutable, monitoriza su correcta ejecución, actualiza los estados de envío mediante callbacks de Globus y espera eventos de migración, parada o eliminación desde el dispatch manager.
- *Cancelación*: Cancela la tarea enviada si el submission manager recibe un evento de parada, eliminación o migración.
- *Epílogo*: Prepara el RSL y envía el epílogo ejecutable. El epílogo transfiere de vuelta los ficheros de salida y limpia el sistema remoto.

- El performance monitor se activa periódicamente a cada intervalo de *discovering*. Solicita acciones de replanificación para detectar los mejores recursos cuando el rendimiento decrece.

5.2. Descubrimiento y selección de recursos

Debido a la naturaleza heterogénea y dinámica del grid, el usuario debe establecer los requisitos que deben ser cumplidos por los recursos y los criterios para ordenar los recursos encontrados. Los atributos necesarios para el descubrimiento y selección de recursos deben ser recogidos de los servicios de información, normalmente del Servicio de Monitorización y Descubrimiento de Globus (MDS). A menudo, el descubrimiento de recursos solo está basado en atributos estáticos recogidos del GIIS (Grid Information Index Service) mientras que la selección de recursos se basa en atributos dinámicos que pueden ser obtenidos del GRIS (Grid Resource Information Service).

El selector de recursos es ejecutado por el dispatch manager para conseguir un ranking de los hosts candidatos cuando la tarea está pendiente de ser enviada o si se ha pedido una replanificación. El selector de recurso es una script o un ejecutable binario, que recibe la plantilla de la tarea parseada como argumento y otros parámetros necesarios. Su salida estándar suele mostrar un recursos candidato por línea con el siguiente formato:

HOST[/SJM] HOST[/EJM] RANK SLOTS QUEUE ARCH MAD

Donde:

- **HOST**: Nombre del recurso candidato
- **SJM**: Gestor de tareas para el staging
- **EJM**: Gestor de tareas para la ejecución
- **RANK**: Valor numérico que muestra la idoneidad del recurso candidato para la ejecución de la tarea
- **SLOTS**: Número de slots libres en el recurso candidato
- **QUEUE**: Nombre de la cola a la que enviar la tarea
- **ARCH**: Arquitectura del procesador del recurso candidato
- **MAD**: Tag que identifica el MAD que se va a usar

Esta aproximación modular garantiza la extensibilidad de la selección de recursos. Se pueden implementar diferentes estrategias para la planificación a nivel de aplicación, desde la más simple basada en una lista predefinida de hosts hasta estrategias más avanzadas basadas en filtros de requisitos, filtros de autorización y expresiones de rango en términos de modelos de rendimiento.

5.3. El módulo MAD (Middleware Access Driver)

Gridway usa un módulo MAD para enviar, controlar y monitorizar la ejecución del prólogo, el wrapper y el epílogo. Este módulo proporciona las operaciones básicas para actuar con el gestor de recursos middleware.

El formato de envío de una petición al MAD, a través de su entrada estándar es:

OPERATION JID HOST[/JM] RSL

Donde:

- **OPERATION**: Puede ser uno de los siguientes:

  - **INIT**: Inicializa el MAD
  - **SUBMIT**: Envía una tarea
  - **POLL**: Consulta a una tarea para obtener una tarea
  - **CANCEL**: Cancela una tarea
  - **FINALIZE**: Finaliza el MAD

- **JID**: Es el identificador de tarea, elegido por Gridway
- **HOST**: Si la operación es SUBMIT, especifica el recurso al que se envía la tarea
- **JM**: Si la operación es SUBMIT, especifica el gestor de tareas para enviar la tarea
- **RSL**: Si la operación es SUBMIT, señala la especificación de recurso al que se envía la tarea.

Por otro lado, el formato de las respuestas recibidas del MAD, a través de su salida estándar es:

OPERATION JID RESULT INFO

Donde:

- **OPERATION**: Es la operación especificada en la petición que origina la respuesta.
- **JID**: Es el identificador de la tarea, igual que en la petición de envío.
- **RESULT**: Es el resultado de la operación. SUCESS o FAILURE.
- **INFO**: Si el resultado es FAILURE, contiene la causa del fallo. Si la operación es POLL, contiene el estado de la tarea.

5.4. Migración de tareas

Un entorno Grid presenta condiciones de cambio impredecibles, como carga dinámica, alta tasa de fallos, o aparición y eliminación continuada de recursos. La migración es la clave para la ejecución de tareas en entornos Grid dinámicos. Gridway tiene en cuenta las siguientes circunstancias, a partir de las cuales se puede iniciar una migración:

- Migración iniciada por Grid.
- Se ha encontrado un recurso mejor.
- El recurso remoto o su conexión de red han fallado.
- El trabajo enviado es cancelado o suspendido por el administrador de recursos.
- Migración iniciada por usuario.
- El usuario solicita explícitamente una migración.

La ejecución adaptativa se implementa a través de la replanificación de tareas, que puede conducir a su migración a recursos más adecuados. La estructura tiene en cuenta las siguientes razones para replanificar, es decir, situaciones en las cuales se puede activar la migración:

- El request manager recibe una petición de replanificación de un usuario.
- El performance monitor solicita una replanificación para detectar un recurso mejor.
- El performance monitor solicita una replanificación si el tiempo de suspensión supera el máximo permitido.
- El submission manager detecta un fallo.

La razón para la replanificación es evaluada para decidir si merece la pena la migración. Algunas razones como la cancelación de trabajos o un fallo, hacen que el dispatch manager desencadene un evento de migración al submission manager con un nuevo host, incluso si el nuevo host presenta un rango menor que el actual. Otras razones como el descubrimiento de un nuevo recurso,

hacen que el dispatch manager provoque un evento de migración solo si el nuevo host seleccionado presenta un rango suficientemente alto. También pueden ser evaluadas otras condiciones: tiempo para la finalización, coste de transferencia de los ficheros de entrada y salida?

Cuando la orden de migración se concede finalmente, el submission manager realiza las siguientes acciones:

1. La tarea se cancela.
2. El módulo prólogo se envía al nuevo recurso.
3. El módulo epílogo se envía al recurso antiguo.
4. El módulo wrapper se envía al nuevo recurso.

5.5. Tolerancia a fallos

Gridway proporciona a las aplicaciones la habilidad de detección de fallos necesaria en un entorno de fallos:

El gestor de tareas GRAM notifica los fallos en el envío como callbacks GRAM. Este tipo de fallos incluyen errores en la conexión, autenticación, autorización, parsing RSL, ejecución, etc.

El gestor de tareas GRAM es sondeado periódicamente en cada período de consulta.

La salida estándar del prólogo, wrapper y epílogo es parseada para encontrar fallos. En el caso del wrapper es útil capturar el código de salida, que es usado para determinar si la tarea se ejecutó correctamente o no. Si el código de salida de la tarea no está preparado, la tarea terminó prematuramente, por tanto la tarea falló o fue cancelada intencionadamente.

Cuando se detecta un fallo irrecuperable, se reintenta el envío del prólogo, wrapper y epílogo el número de veces especificado por el usuario y, cuando no quedan más reintentos, o bien detiene la tarea para reanudarla manualmente más tarde, o bien genera automáticamente un evento de replanificación.

Bibliografía

[1] "Web Services: a technical introduction", Prentice Hall, H. M. Deitel.

[2] "The anatomy of the Grid" Ian Foster, Carl Kesselman, Steven Tuecke.

[3] "The physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration" Ian Foster, Carl Kesselman, Jeffrey M.Nick, Steven Tuecke.

[4] "Introduction to Grid Computing with Globus" Luis Ferreira, Viktor Berstis, Jonathan Armstrong y otros.

[5] "Fundamentals of Grid Computing" Viktors Berstis

[6] "Web Services Architecture" W3C Working Group. 2004

[7] "The Globus Toolkit 4 Programmer's Tutorial" Borja Sotomayor. 2004/2005.

[8] "The WS Resource Framework" Ian Foster, Steve Tuecke y otros.

[9] "From Open Grid Services Infrastructure to WS-Resource Framework & Evolution" Ian Foster, Steve Tuecke y otros.

[10] http://www.gridway.org

[11] http://www.globus.org