ARQUITECTURAS GRID

orientadas a la gestión de recursos

Trabajo de investigación

Álvaro Fernández Casaní

Diciembre de 2004

IFIC - INSTITUTO DE FÍSICA CORPUSCULAR

Pablo Galdámez
Tutor del trabajo de investigación
DSIC, UPV

Miquel A. Senar
Tutor en el proyecto Crossgrid
UAB

Resumen

Las tecnologías grid están teniendo un gran auge en los últimos años, debido a que permiten resolver problemas con gran demanda computacional y de datos, y que tienen en su propia definición una característica muy importante dentro de los sistemas distribuidos, como es la compartición de recursos.
En la primera parte de este trabajo se presentan las características más importantes de los sistemas grid, así como su arquitectura. Se hace especial mención a las cuestiones orientadas a la gestión de los recursos distribuidos de estos sistemas.
En la segunda parte se describe el trabajo que he tenido la oportunidad de realizar colaborando en el proyecto Crossgrid, durante mi estancia en el Instituto de Física Corpuscular de Valencia (CSIC-UV). En él se buscan soluciones para la gestión eficiente de recursos en sistemas grid, orientado principalmente a la ejecución de aplicaciones paralelas, tanto en *batch* como de forma interactiva..
Quisiera agradecer a todos mis compañeros en este proyecto la ayuda aportada durante el desarrollo del mismo, pero especialmente a mis compañeros del propio IFIC y de la UAB junto con los que se ha desarrollado el trabajo aquí presentado. También quisiera agradecer a Pablo Galdámez la ayuda proporcionada para poder presentar este trabajo de investigación como parte del programa de doctorado de la UPV.

A Carmen, por su ayuda y apoyo incondicional.

ARQUITECTURAS GRID

ORIENTADAS A LA GESTIÓN DE RECURSOS

INDICE GENERAL

1. INTRODUCCIÓN

La computación necesaria para abordar los requerimientos de los proyectos científicos está siendo cada vez más elevada, debido a que los problemas son cada vez más complejos . Las aplicaciones en las que se plasman estos problemas son cada vez más complicadas y demandantes de potencia de cálculo, así como grandes consumidoras de datos.

Muchos de estos proyectos, además de requerir una gran capacidad de potencial computacional propiamente dicho, y del almacenamiento de inmensas cantidades de datos, requieren la colaboración de numerosos grupos de científicos. Éstos, así como los recursos de los que disponen, pueden pertenecer a una misma área en la que realizan sus investigaciones, pero encontrarse distribuidos geográficamente.

La utilización eficiente de estos recursos es todo un reto, ya que éstos recursos dispersos deben ser operados conjuntamente como sistemas. Además se necesita que estén disponibles la mayor parte del tiempo, y que además den un rendimiento elevado debido a los requerimientos de las aplicaciones.
Como ejemplo de una aplicación clave, para los trabajos HEP (High Energy Physics, o de Física de altas energías) tanto la comunidad de físicos, como los recursos que usan para procesar y analizar los datos, están distribuidos a lo largo del mundo. El tamaño del LHC (Large Hadron Collider, el nuevo acelerador de partículas listo para el 2007) Computing, va creciendo escalonadamente.
Casi 10 PB de datos serán producidos cada año y reprocesados, algunos de ellos varias veces. Una potencia de cálculo del orden de 10^8 SI2000 (SpecInt 2000, un PC común suele computar alrededor de 1000 SI2000) será requerido, distribuido entre el CERN (1/3 del total), ~10 Grandes Centros (Tier 1), posiblemente 3 veces más centros más pequeños (Tier 2), y cientos de departamentos de Universidades y centros de investigación distribuidos por el mundo.

El concepto de *Grid* ([1]), se adapta perfectamente a los requerimientos de estos proyectos. Propuesto por Ian Foster y Carl Kesselman, ha surgido en los últimos años para denominar un conjunto de recursos computacionales heterogéneos distribuidos, pertenecientes a distintas organizaciones. Las ideas principales son dotar de una infraestructuras de computación distribuidas a las *Organizaciones Virtuales* de usuarios (descritas en el artículo [16]), para que puedan llevar a cabo sus computaciones. El concepto surge de la analogía con la red eléctrica, ya que ésta es *persistente* (siempre está disponible, desde cualquier punto)*, estable* (es una infraestructura confiable) *y uniforme* (basado en protocolos abiertos).

1.1. Fundamentos de la computación Grid

Los fundamentos de la computación grid son básicamente tres:

1. **Compartición de recursos a gran escala**: la idea fundamental es poder compartir una serie de recursos entre los posibles usuarios, de manera que sea igual de sencillo poder acceder a una infraestructura local que a una localizada en cualquier parte del mundo. Es más, el uso debe poder llevarse a cabo utilizando muchos recursos distintos, localizándose estos distribuidos geográficamente. Entre los recursos que podemos compartir se pueden distinguir:
    - *Computadores*: proporcionando potencia de cálculo para realizar las computaciones necesarias, es el recurso básico al que queremos acceder.
    - *Redes:* de comunicaciones que permitirán interconectar el resto de recursos para que la ejecución de las aplicaciones sea posible
    - *Instrumentos:* generalmente instrumentos de carácter científico, que son necesarios en algunas aplicaciones. Ejemplos de estos instrumentos pueden ser cabinas de visualización, microscopios electrónicos, radio-telescopios, etcétera.
    - *Datos:* que deban ser compartidos por una comunidad para lograr sus objetivos, como datos para simulaciones nucleares, variables meteorológicas para la obtención de las predicciones, o cadenas de adn para ser procesadas. Además estos datos pueden tener requerimientos de privacidad, como pueden ser los expedientes médicos y datos asociados (mamografías, escáneres, etc.) a pacientes que pueden ser accedidos por una comunidad médica para analizarlos.

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

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

2. ARQUITECTURAS GRID

El propósito de definir una arquitectura para sistemas *grid* es poder proporcionar un conjunto de entidades y de nomenclatura para las mismas que nos sirva para poder describirlos correctamente. De esta manera podemos situar a cada elemento del sistema en su lugar correspondiente, para clarificar cuál es la función de cada uno y como se interrelacionan entre sí.

Para poder conseguir una imagen comprensible y coherente de la arquitectura es necesario primeramente identificar aquellos servicios que son necesarios en todo sistema grid, y que precisamente son los que nos dan las propiedades y características que anteriormente hemos enumerado para considerar un sistema como *grid*.

También deberemos considerar los protocolos necesarios para que la comunicación entre los diferentes elementos sea posible, contando con la máxima estandarización de los mismos para que la interoperabilidad entre posibles componentes de diferentes proveedores sea correcta.

2.1. Servicios requeridos

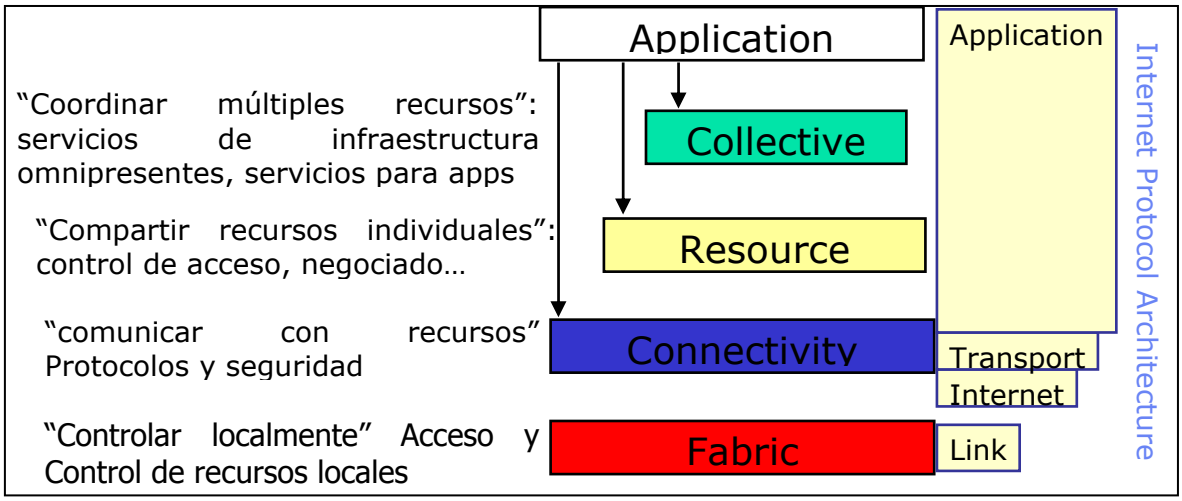
Para hacer posible que la ejecución de un trabajo en un testbed distribuido sea satisfactoria se requieren unos servicios que provean la funcionalidad que este trabajo requiere:

- Por ejemplo, un usuario debe poder identificarse, esto es, un servicio de ***autenticación***. Con este servicio el usuario puede certificar que es realmente quien dice ser, y asimismo el recurso que se quiere utilizar deberá autenticarse para que el usuario tenga la seguridad de que se ejecuta donde quiere, por lo que hablamos de autenticación mutua.
- Un servicio de ***autorización*** es necesario, también para permitir la ejecución de un usuario de un recurso, autorizándose como un usuario local, con sus permisos y restricciones dependiendo del contexto, la hora de petición o ejecución, etcétera.
- Un servicio de planificación o ***scheduling*** de los recursos, para que en un entorno distribuido como este sea eficiente la utilización de los mismos y haya un reparto equitativo. Dentro de este servicio se pueden aplicar muchas políticas diferentes dependiendo de las necesidades de los usuarios, las aplicaciones, y del estado dinámico de los recursos.
- Asimismo es necesario un servicio para ***descubrir recursos***, ya que en un entorno de estas características se pueden añadir y quitar los mismos por lo que su selección debe ser dinámica. Este será utilizado por el anterior servicio de planificación para asignar correctamente las distintas ejecuciones.
- Un servicio de ***caracterización de recursos*** que den información de los mismos, y será utilizado preferentemente por los dos anteriores.
- También es recomendable un servicio de ***reserva*** anticipada, para poder ejecutar en un grupo de recursos en los que normalmente no es posible hacerlo, y que debe compenetrarse con el servicio de planificación.
- Un servicio de ***acceso a datos remotos***, necesario para obtener los datos requeridos por un programa en ejecución ya que éstos pueden ser muy numerosos. Puede ser necesario un servicio de ***réplica*** para hacer copias de datos que sean muy "caros" de transportar y que pueda ser conveniente tener en una localización más cercana.
- Asimismo se deben contar con recursos para hacer ***transferencias rápidas*** de estos datos.
- Los servicios de ***monitorización*** son necesarios para controlar la correcta ejecución de los distintos trabajos, así como para controlar que los diferentes servicios que hemos comentado se encuentran disponibles y corriendo correctamente para su utilización.

2.2. Arquitectura Global

La arquitectura global del sistema puede dividirse en diferentes piezas, dependiendo de los diferentes niveles en los que actúe cada componente. Esto nos dará un típico modelo de arquitectura en capas, que puede ser comparado con las capas de modelo OSI [1].

[1] OSI (Open System Interconection, Interconexión de Sistemas Abiertos) El cual es usado para describir el uso de datos entre la conexión física de la red y la aplicación del usuario final



- En el nivel más bajo encontramos los servicios que son aplicables al control de los recursos locales, es lo que se denomina la capa ***Fabric*** y podría compararse a la capa *Link* del modelo OSI. En esta capa se modelan los recursos accesibles, aquellos como:
  - Recursos computacionales, como por ejemplo un *cluster* o un simple computador personal.
  - Sensores, instrumentos de laboratorio.
  - Sistemas de Almacenamiento de datos
  - Sistemas de archivos distribuidos

  La funcionalidad básica que deben proveer, y de la que dependen capas superiores es la que nos da información sobre los recursos que está modelando y de qué manera están disponibles para su utilización. Esta información puede ser muy dependiente y de hecho lo es según de qué recurso estamos hablando, pero por ejemplo para **recursos computacionales** debe:
  - Proveer información sobre el Hardware y el Software disponible, información sobre el estado actual y tal vez pasado del recurso como carga, utilización, disponibilidad etcétera.
  - Debe ser posible el monitorizar procesos que se estén ejecutando.
  - Un valor añadido puede ser la posibilidad de la reserva del recurso
  - Debe ser posible el control de recursos asociados a procesos
  - Debe dar información sobre el estado de una posible cola de ejecución en la que residan los procesos (PBS, SunGridEngine, NQE)

  También se pueden modelar **recursos de almacenamiento**, con lo cual en este caso se debería dar información diferente. Como en los recursos anteriores se da información sobre el HW/SW disponible, así como de información más sensible como el espacio disponible, etcétera.
  En estos recursos debe ser posible el almacenamiento tanto de ficheros simples, como el almacenamiento masivo de datos, por lo que básicamente debe dar una interfaz de servicios para poder transferir ficheros. En algunos casos las transferencias de datos pueden cortarse, por lo que es necesario que se sea posible reestablecer la transferencia, desde el último dato correcto que se obtuvo.
  El control sobre el espacio disponible y el ancho de banda de las transferencias es también un servicio que se puede dar, ya que de esta manera se puede restringir los mismos a un usuario o grupos de mismos dependiendo de las políticas locales de administración de cada sitio.

  Para los **recursos de red** además de la información sensible del dispositivo, se puede proveer información sobre la carga actual o incluso sobre predicciones a futuro de la misma. La utilización de servicios de prioritización para mejorar las comunicaciones en algunos casos es de nuevo una posibilidad, consiguiendo de esta manera una Calidad de Servicio (Qos) determinada.

Además de los comentados, pueden existir otra serie de recursos menos comunes pero que pueden formar parte del sistema, tales como dispositivos de laboratorio o incluso repositorios de código, o Catálogos como bases de datos.

- En el siguiente nivel, nos encontramos en la capa de **C*onectividad***, que tiene la función básica de proveer los métodos y protocolos de comunicación entre los recursos modelados en la capa anterior. En ésta se tienen muy en cuenta los protocolos y la seguridad, ya que es un requerimiento básico para el correcto funcionamiento del sistema.

  En esta capa habría una analogía con servicios de las capas *Transport* del modelo OSI, y tal vez también de la capa *Internet* de la pila de protocolos TCP/IP. En esta capa podemos encontrar los protocolos y estándares de comunicación tales como IP en la capa de internet, ICMP, y TCP y UDP en la capa de transporte.

  En arquitecturas grid esperamos encontrar en esta capa servicios que provean los medios adecuados para hacer posible la comunicación a cualquier middleware o aplicación que se encuentre por encima de estos.
  Servicios importantes son aquellos que confieren seguridad y autenticación para las comunicaciones. Entre éstos podemos encontrar servicios de *verificación de identidad de recursos y usuarios*, para hacer posible una autenticación mutua entre los mismos.

  También es necesario el *inicio de sesión único*. En un sistema distribuido de estas características como es el grid, involucra muchos recursos distribuidos geográficamente y que pueden necesitar de los servicios anteriores de *autenticación mutua.* Un mismo trabajo puede viajar por multitud de elementos hasta que alcance aquel en el que se va a ejecutar, o aquellos que necesita si es que requiere de multitud de elementos simultáneamente. Por lo tanto no se requiere que el usuario deba escribir una clave de acceso cada vez que el trabajo necesita autorizarse en otro recurso, sino que debe existir un sistema automático de *inicio de sesión único*, así como de *delegación de privilegios* para que elementos del sistema puedan ejecutar funciones de nombre del usuario original.

  Así mismo es habitual que los trabajos sean de larga duración, y que tengan un tiempo de vida limitado para su ejecución dentro del sistema, dado en primera instancia por el tiempo durante el que es válida la autorización. Por ello un sistema de renovación de credenciales también se encontraría en esta capa.

  Los servicios *de Integración con servicios de seguridad locales*, deben implementarse de nuevo en esta capa. En estos servicios se reúnen aquellas interfaces que dan la funcionalidad de los sistemas de seguridad locales, que han de ser utilizados por capas superiores.
  Las *Relaciones de confiabilidad* nos permiten autorizar a una serie de usuarios a partir de la autorización de uno sólo que hace las veces de punto de contacto. Esta característica también es importante en este tipo de sistemas, ya que el uso de *Organizaciones Virtuales* que sean capaces de compartir recursos entre sus miembros así lo requieren.

- La siguiente capa es la denominada de **Recursos**, y pretende compartir los recursos individuales por la utilización de protocolos estándar para el control de los mismos, sobre la capa anterior de conectividad.
  En esta capa se realizará la *negociación de recursos*, mediante los servicios subyacentes. Igualmente se llevarán a cabo la *iniciación de las transacciones* que sean necesarias para la realización del trabajo, tales como pueden ser la comunicación del ejecutable en el recurso en el que se vaya a ejecutar, como la localización y recuperación de aquellos datos que sean necesarios para la ejecución.

  Dentro de esta capa también tienen cabida los protocolos y servicios de *monitorización*, tanto del resto de servicios para determinar que están disponibles, como de aquellos trabajos o aplicaciones que ya han sido enviados y de los que se quiere conocer el estado.

  El *control de los trabajos* también debe ser posible realizarse en esta capa, y proporcionar los métodos para reiniciar, mover de localización, o cancelar el trabajo, así como de características más complejas que puedan requerirse. Dentro de este control también podemos incluir en parte

los servicios de *Accounting*, mediante los cuales podemos tener un registro estadístico de las aplicaciones que se han corrido en un determinado recurso, por ejemplo qué usuario lo ha ejecutado y durante cuánto tiempo. De esta manera podremos tener un registro de todo lo que ha ocurrido y hacer efectivo un posible *pago* de los recursos consumidos, a través también de acuerdos plasmados por ejemplo con una *SLA*[2].

La utilización de protocolos de *transmisión de la información* es necesaria para obtener de manera fiable información por ejemplo sobre aquellos recursos que están disponibles, o información más o menos estática o dinámica de los mismos. Esta información puede ser desde el tipo de recurso, o por ejemplo en el caso más típico de recursos de computación datos como el sistema operativo instalado, la memoria disponible, los resultados de benchmarks corridos sobre la máquina, etcétera.

Los *Protocolos de transmisión de datos* deben estar localizados así mismo en esta capa, y como su nombre indica proveen acceso a los datos y comunicación de alto rendimiento. Un protocolo comparable sería el FTP, que da los medios para transmitir información de forma confiable, aunque con algunas restricciones. Elementos como la implicación de los elementos de autenticación y seguridad de las capas anteriores son recomendables para que estos protocolos sean efectivos.

- La última capa es la denominada capa **Colaborativa**, y tiene como finalidad la coordinación de múltiples recursos accesibles por la capa anterior. En esta capa podemos encontrar por ejemplo aquellos *servicios de Información y Directorio* que nos dan una idea global del sistema. La idea es utilizar los protocolos de comunicación de información de la capa anterior para proveer una vista de los recursos, de manera global.

  Dependiendo de la organización de los elementos de estos sistemas de información, podremos obtener la información misma de una manera fiable y fiel a la realidad. Por ejemplo, un sistema de información centralizado puede no satisfacer todos los requerimientos de información en tiempo real para un gran número de recursos y de peticiones simultáneas. La organización dependerá de qué manera se plantea la arquitectura detallada de estos sistemas de información, y de la implementación de los mismos.

  La vista de los recursos es comúnmente realizada por comunidades virtuales, o *Virtual Organizations.* También en esta capa podemos encontrar servicios que hagan la monitorización de recursos completos para una *Organización Virtual***,** de manera que sea posible tener una visión completa de los recursos a nuestro alcance como miembros de una comunidad determinada.

  Aquí se sitúan servicios como los de *Scheduling, Co-Allocation, y Brokering*. Estos servicios proveen los métodos para buscar aquellos recursos que se adaptan a las necesidades de nuestros trabajos, conociendo algunos datos sobre los mismos e intentando optimizar la asignación global mediante métodos de búsqueda o *brokering*. Permiten así mismo hace una planificación de la utilización de los recursos, de manera similar a como lo haría un *scheduler* o planificador de un sistema operativo. Con los servicios de *co-allocation* podemos hacer reservas simultáneas de aquellos recursos que son necesarios para lo consecución correcta del trabajo.

  Podemos encontrarnos con una serie de tipos de trabajos que no se corresponden con el típico trabajo secuencial, sino que está formado por una serie de trabajos independientes que, dependiendo de su complejidad y de las relaciones entre estos subtrabajos pueden ejecutarse simultáneamente. Estos trabajos que pueden modelarse con grafos (DAGS, o *Directed Acyclyc Graphs*) pueden ser planificados de manera eficiente si se modelan correctamente y si existen los servicios adecuados para que se ejecuten.

  También pueden existir servicios de *replicación de datos* que podríamos encontrar en esta capa, de manera que se realizaran la copia de aquellos datos necesarios para la ejecución de un programa si estos datos estuvieran por ejemplo en una localización lejana. De esta manera estos datos podrían ser accedidos de manera más local, optimizando su acceso y aún más si se van a

[2] Service Level Agreement

necesitar múltiples veces. Estos servicios podrían ser utilizados por los anteriores de *planificación***,** que de forma correcta conocerían cuándo, como y por qué realizar estas replicaciones.
Las capas más altas como las aplicaciones o servicios más complejos de middleware deben utilizar los servicios que se provean en esta capa para la programación del *grid.*

3. GESTIÓN DE RECURSOS

3.1.Fundamentos de Gestión de Recursos

Dentro de la computación Grid existen varias áreas de investigación dependiendo de qué elementos entren en juego. Una de las más importantes y que asimismo también ha recibido mayor atención por parte de los investigadores es la *gestión de recursos***.** La gestión de recursos grid (*Grid Resource Management)* se define como el proceso de identificar requerimientos, hacer *matching* de recursos con las aplicaciones, asignar estos recursos, y planificar y monitorizar estos recursos grid a través del tiempo para correr las aplicaciones sobre el grid lo más eficientemente posible.

Las aplicaciones Grid compiten por recursos que son muy distintos en su naturaleza como hemos visto, incluyendo procesadores, datos, instrumentos científicos, redes y otros servicios. Los recursos son como vemos heterogéneos en sus distintas clases, pero además incluso en la misma clase podemos encontrar heterogeneidad en la misma clase y tipo de recurso. Por ejemplo, no hay dos clusters de computadoras que tengan el mismo software y configuración exactamente.

Mucho del trabajo inicial en la gestión de recursos grid ha estado orientado a hacer frente a esta heterogeneidad, con una serie de protocolos estándar de gestión de recursos [2], [3], y mecanismos estándar para expresar recursos y requerimientos de los trabajos [4].
Además el hecho de carecer de datos disponibles sobre el sistema actual y las necesidades de los usuarios, así como las de los dueños de los recursos y los administradores del sistema, hacen más complicada esta situación.

En la actualidad los *Grids* se están convirtiendo en una tecnología muy conocida, en la que se ha invertido una considerable cantidad de tiempo y de dinero, y que se está aprovechando en entornos de producción. Los desafíos iniciales de la computación grid (cómo correr un trabajo, cómo transferir grandes ficheros, cómo gestionar múltiples cuentas de usuario en sistemas diferentes) se han ido resolviendo en un primer orden, de manera que usuarios e investigadores pueden hacer ahora frente a las cuestiones que permiten hacer más eficiente el uso de los recursos.

Mientras que el uso de *grids* se ha convertido en casi ordinario en muchos ambientes, el uso de buenas herramientas está lejos de ser ubicuo a causa de las muchas cuestiones abiertas en el tema:

- **Múltiples capas de schedulers (planificadores):** La gestión de recursos grid tiene que contar con la participación de varias capas de planificadores. En el nivel más alto se encuentran los planificadores en el nivel grid, que pueden tener una visión general de los recursos pero que están muy lejos de los recursos donde la aplicación correrá finalmente. Estos primeros planificadores tienen por objeto selección a grandes rasgos donde se ejecutará la aplicación, por ejemplo en qué cluster, y la reserva de los mismos en el caso de que sea disponible. En el nivel más bajo encontramos los gestores de recursos locales que gestionan un recursos o grupo de recursos, como los gestores de colas (Pbs, LSF, …) que gestionan los clusters locales. Normalmente la interacción entre estas dos capas no es fácil ni directa, por lo que surgen muchas cuestiones que tienen que ser resueltas. Entre estas dos capas además podemos encontrar algunas más intermedias, por ejemplo aquellos planificadores que gestionan una serie de recursos específicos a un proyecto. En cada nivel adicional, más gente y software tiene que ser tenido en cuenta

- **Falta de control sobre los recursos:** Los planificadores grid no son sistemas de gestión locales, por lo que al más alto nivel un planificador grid puede no tener, y usualmente así ocurre, acceso directo o control sobre los recursos. La mayoría de las veces los trabajos son enviados desde estos gestores de alto nivel a una serie de recursos con los mismos permisos que un usuario de ese sistema tendría. Esta falta de control es uno de los desafíos que se tienen que hacer frente.

- **Recursos compartidos y varianza**: Relacionado con la falta de control está la falta de accesos dedicados a los recursos. Muchos recursos son compartidos entre varios usuarios y proyectos. Esta compartición resulta en una alto grado de varianza y no es predecible en la capacidad de los recursos disponibles para su uso. La naturaleza heterogénea de los recursos también juega un papel en la variada capacidad. La situación es más complicada por el hecho de que aplicaciones

grid a veces requieren la asignación de múltiples recursos, necesitando una estructura en la cual los recursos deben ser coordinados a través de varios dominios administrativos [5], [6].

- **Conflictos en los objetivos de rendimiento:** Cuando se utilizan los recursos grid para correr las aplicaciones de un usuario, pueden surgir conflictos entre los intereses de éstos y los de los dueños de los recursos. Desde optimizar el rendimiento de una simple aplicación para un coste específico a obtener el mejor rendimiento del sistema (*throughput*) o minimizar el tiempo de respuesta. Además muchos recursos tienen políticas locales que deben ser tenidas en cuenta. Cuestiones como la de quién debe hacer la planificación exactamente, siguen abiertas: ¿Cuánto de este proceso debe ser hecho por el sistema y cuánto por los usuarios? ¿Cuáles son las reglas para cada uno? Muchas de las investigaciones están orientadas al entendimiento y la gestión de estas diversas políticas desde la perspectiva de ambos, el proveedor de recursos y el consumidor [7], [8],[9], [10].

El surgimiento de arquitecturas orientadas a servicios, el creciente interés en soportar un amplio espectro de aplicaciones comerciales, y la natural evolución de funcionalidad, están conjuntamente permitiendo avances significativos en las capacidades de la gestión de recursos. Mientras los entornos Grid están primariamente orientados hacia los servicios *best-effort*, se espera que esta situación se torne significativamente diferente en los próximos años, con capacidades de aprovisionamiento de recursos extremo-a-extremo (*end-to-end*).

3.2.Planificación de trabajos

La planificación (*scheduling*) dentro de la gestión de recursos, es definido como el proceso de la toma de decisiones que tienen que ver con recursos sobre múltiples dominios administrativos. Este proceso puede incluir la búsqueda de estos múltiples dominios administrativos para usar una única máquina o planificar un único trabajo para usar múltiples recursos en un único sitio, o en varios de ellos.

Dentro de las especificaciones necesarias podemos definir un *trabajo (job)* como la instancia de una aplicación que va a ejecutarse en el grid y que necesita un recurso (de la clase que sea, de computación, de ancho de banda de red, de otra aplicación, etcétera). Ya hemos definido anteriormente el concepto de *recurso* como cualquier cosa que puede ser planificada en el tiempo, un máquina, espacio de disco, una red con capacidades de calidad de servicio (*Qos)*, y más.

Las diferencias entre lo que definíamos como un planificador grid de alto nivel, y un gestor local de recursos, es que éste ultimo hace la gestión de recursos en un único sitio, y representan la capa más baja en lo que se refiere a instancias de planificadores.

La principal característica de los planificadores grid es que no poseen y por lo tanto no pueden gestionar directamente los recursos de un sitio, por lo que no tienen control sobre estos. Deben tomar decisiones bastadas en un sistema base *best-effort,* y entonces mandar el trabajo a los recursos seleccionados, generalmente como el usuario. Además el planificador grid no tiene el control sobre el conjunto completo de trabajos enviados al recurso, o incluso conocer sobre los trabajos que están siendo enviado a los recursos que está usando, por lo que las decisiones que compensa los acceso de un trabajo por los de otro pueden no ser realizables en un sentido global. Esta falta de propiedad y control es la causa de muchos de los problemas a resolver en este área.

Una idea seria utilizar la ejecución especulativa de múltiples instancias, enviando el trabajo a múltiples recursos y cuando uno empieza a ejecutarse, entonces cancelar el resto. Sin embargo no es la aproximación común de los sistemas que existen en la actualidad y por lo tanto no haremos mucho hincapié en este tema. Sin embargo si que son temas importantes a tratar y sobre los que trabajaremos aquellos que se encargan de la selección de recursos (a veces denominada como descubrimiento de recursos [11]), asignación de los trabajos a estos recursos (resource *mapping [11]*, o *matching*), envío de datos necesarios y distribución, y otras tareas directamente relacionadas como la monitorización del trabajo y la obtención de sus resultados.

Al principio el primer *planificador grid* ha sido históricamente el usuario mismo, pero mucho esfuerzo ha sido puesto en este punto como veremos después en la descripción del estado de arte de las tecnologías

grid. En los siguiente puntos veremos los elementos y pasos que tiene que llevar a cabo un planificador grid para llevar a cabo su función correctamente.

3.3.Interacción con los sistemas de Información

Los planificadores grid tienen que tomar decisiones basándose en la información de la que disponen y la que se les provee. Por una parte pueden tener información del trabajo a ejecutar, sus características y sus requerimientos, así como información que pueda ayudar a una mejor planificación basándose en las preferencias del usuario. Esta información está normalmente y en su mayor parte provista por el usuario mismo.

Por otra parte se necesita información del grid mismo, sobre su estado actual y sus capacidades, para hacer una planificación eficiente. En general estos planificadores obtienen información de un sistema general de información del grid (*Grid Information System o GIS)*, que de hecho obtiene la información de los recursos individuales. Muchos planificadores grid asumen que una tienen disponible el 100 por ciento de la información necesaria, y con un nivel de detalle extremadamente fino y que es siempre correcta. En realidad la experiencia con estos sistemas dicta que esta situación ideal está lejos de ser la realidad, y que por lo general sólo podemos disponer de la información de más alto nivel de estos sistemas. En los siguientes capítulos describiremos cuáles son estos sistemas utilizados y los problemas que hemos detectado, así como algunas soluciones a ellos.

3.4.Etapas de la planificación sobre Grid

Podemos distinguir tres fases dentro del proceso que seguiría un planificador grid: *descubrimiento de recursos (resource discovery)*, que genera una lista de potenciales recursos, *recolección de la información (information gathering)* sobre estos recursos y selección del mejor o los mejores; y la *ejecución del trabajo*(*job execution)* que incluye el envío de los archivos necesarios y el posterior borrado.

3.4.1. Fase 1: Descubrimiento de recursos

En esta primera fase determina qué recursos están disponibles a un determinado trabajo de un usuario. La fase de descubrimiento de recursos incluye la selección de un conjunto recursos que serán investigados en más detalle en la fase 2, de recolección de información sobre los mismos.
Al final de esta primera fase dispondremos de un conjunto de recursos que habrán pasado unos mínimos requerimientos del trabajo. Esta fase se puede subdividir en tres pasos: *filtrado de autorizaciones, definición de los requerimientos del trabajo, y filtrado para satisfacer los requerimientos mínimos del trabajo.*

> **Paso 1: Filtrado de autorizaciones:** El primer paso consiste en determinar el grupo de recursos en los cuáles el usuario está autorizado. Esto es lógico ya que en un sistema amplio como el grid es posible que nosotros como usuarios, y por lo tanto nuestros trabajos, estemos autorizados únicamente en un subconjunto de recursos con cuyos dueños tenemos algún tipo de relación o acuerdo, y no a todos ellos. En este sentido, la computación sobre el grid no es diferente de enviar un trabajo a un sitio remoto: si no tenemos autorización en ese sitio el trabajo no correrá.
> Al final de esta fase dispondremos de una lista de recursos que el usuario está autorizado a utilizar. Una de las diferencias fundamentales es el número de recursos que estamos analizando, ya que debido a la extensión del sistema y del número de recursos, los sistemas de información GIS pueden darnos información sobre muchos recursos, incluso aquellos en los que en principio no estamos autorizados, por lo que una primera fase de filtrado es necesaria.
> Los esfuerzos recientes han ayudado a los usuarios con la seguridad una vez que disponen de las cuentas, pero muy poco se ha hecho para hacer frente a cuestiones como *accounting* y gestión de las cuentas *(account management)[12]*.
>
> **Paso 2: Definición de los requerimientos de las aplicaciones**: Para proceder en la búsqueda de recursos, el usuario debe ser capaz de especificar un mínimo set de requerimientos para filtrar más convenientemente el conjunto de recursos a los que tiene acceso en el siguiente Paso 3.
> El conjunto de posibles requerimientos del trabajo puede ser muy amplio y puede variar significativamente entre trabajos. Puede incluir campos estáticos, en el sentido de que los recursos van a mantenerlos sobre el tiempo, como el sistema operativo o el hardware para el cual ha sido diseñado la aplicación. También puede incluir detalles más dinámicos, como por ejemplo

los requerimientos de memoria, la conectividad en cuanto a ancho de banda, o el espacio temporal disponible. Cuantos más detalles son incluidos más eficiente será la planificación y el *matching* de los recursos, pero también debe tenerse en cuenta que cuantas más restricciones se impongan menor será el conjunto de posibles recursos donde ejecutar, y en un caso extremo este conjunto puede ser igual a vacío.
Cómo se definen los requerimientos depende del sistema que se use, con diferentes aproximaciones como veremos en los siguiente capítulos.
La información que se usa para corresponder los requerimientos del trabajo con aquellos ofrecidos por los recursos puede hacerse disponible por varios métodos, aunque se sigue trabajando en métodos que automáticamente recojan estos datos. Además puede darse la situación de que muchos de aquellos requerimientos como el tiempo de ejecución mínimo de la tarea para una salida satisfactoria sea manufacturado por el usuario para compensar las decisiones que pudiera tomar cualquier planificador.

Paso 3 Filtrado de requerimientos mínimos. Una vez ya disponemos de un conjunto de recursos, y de los requerimientos de los trabajos, el siguiente paso lógico es hacer un filtrado de aquellos recursos que no satisfacen los requerimientos de las aplicaciones. Normalmente en esta fase se deberían filtrar aquellos recursos que no satisfacen los requerimientos estáticos de las aplicaciones, es decir, aquellos valores que no suelen cambiar en un corto espacio de tiempo como el sistema operativo instalado o la configuración hardware. De esta manera se puede hacer un filtrado rápido cuando el número de recursos y trabajos es elevado, para facilitar las siguientes fases.

3.4.2. Fase 2 Selección del sistema

Dado un número determinado de recursos que se han obtenido después de la fase 1, esta siguiente fase decidirá donde será finalmente planificado el trabajo dependiendo de las políticas asociadas en cada fase. Para ello se requiere, por una parte información más detallada de los recursos y del estado del sistema, y por otra realizar la decisión final de ejecución del trabajo. Aunque son dos pasos independientes, dependen el uno del otro de modo que la decisión está guiada por la información disponible.

Paso 4 Recolecta de Información dinámica. Para hacer más efectiva la selección que se llevará a cabo en el siguiente paso, puede ser necesario la recolecta de información dinámica sobre los recursos. Esta información puede variar con respecto a la aplicación que se está planificando. Sobre recursos computacionales, la información disponible variará de sitio a sitio. Normalmente la información básica disponible proviene del planificador de recursos local. Además, información proveniente de elementos de monitorización en estos recursos, puede ser disponible para efectuar decisiones más adecuadas. Las políticas de autorización local de cada recurso a través de múltiples dominios de ejecución deben tenerse en cuenta también. Cada vez es más común el que los administradores de los recursos especifiquen el porcentaje de estos recursos, en términos de capacidad, tiempo o cualquier otra métrica, para ser considerados.
Un punto importante es la escalabilidad de los sistemas usados para la recolección dinámica de la información, ya que el número de recursos puede crecer considerablemente en sistemas grid. El número de recursos no sólo influye en que se producen más consultas, sino que si algún recurso no está disponible el sistema debe decidir qué hacer con los datos dinámicos que no están disponibles. La aproximación más fácil es desestimar estos recursos, aunque en sistemas más grandes alguna otra aproximación puede ser analizada.
También se puede utilizar sistemas de monitorización y predicción, donde se ha estado realizando mucho trabajo, para utilizar información que sea de utilizad en la toma de decisiones, aunque sin embargo en la actualidad no se están aplicando a sistemas en producción.

Paso 5 Selección y Planificación. Con la información obtenida en el paso anterior, el siguiente paso es decidir en qué recursos o grupos de recursos se va a seleccionar para la ejecución del trabajo. En esta fase se deben aplicar las políticas de planificación disponibles para seleccionar el que se considere el mejor recurso, dada la información del paso anterior y también un posible conjunto de preferencias establecido por el usuario. La inclusión de este tipo de preferencias por parte del usuario puede influir en el proceso de selección de recursos, dando información extra sobre lo que se considera en cada momento el mejor recurso para el usuario y aquel que él prefiere.

Sin embargo la decisión final estará en manos del planificador, que deberá hacer un balance adecuado de todas las posibilidades y aplicar sus políticas de planificación para ello

3.4.3. Fase 3 Ejecución del trabajo

La última fase tiene que ver con todos aquellos pasos que hacen posible la ejecución de un trabajo finalmente en los recursos seleccionados en la fase anterior. Para ello se deben realizan toda una serie de pasos intermedios, muchos de los cuales son casi estándares y describen una via uniforme entre los recursos.

Paso 6 Reserva (opcional). Para llevar a cabo el mejor uso posible de los recursos, a veces es conveniente realizar una reserva de los recursos, principalmente de aquellos que son más difíciles de obtener o que tienen una mayor demanda por parte de los trabajos. Dependiendo del tipo de recurso, si es computacional, de red, o incluso algún instrumento, puede ser más o menos fácil de realizar. Además el uso de reservas puede estar relacionado con algún tipo de *accounting* de los recursos, para poder hacer cumplir otras cuestiones como son las SLAs entre aquellos que ofrecen recursos y los clientes que los usan.
Para que la reserva sea efectiva, los recursos deben dar el soporte básico a ésta, pero sucede que en la actualidad es bastante difícil encontrar estos servicios de reserva de manera nativa, aunque como decimos la necesidad se va haciendo más determinante debido a que los *service level agreements* son cada vez más comunes.

Paso 7 Envío del trabajo. El envío efectivo del trabajo a los recursos se realiza en este paso a través de protocolos en los que se está trabajando y que pretenden convertirse en estándar de protocolos abiertos, entre los que se encuentra [13], [14].
En esta fase se debe contactar el gestor local de cada recursos para enviar el trabajo o sub-trabajo que le correspondan, coordinando todos aquellos que estén involucrados. Una parte importante es la gestión y la tolerancia a fallos que se den en los recursos en esta fase, que puede requerir el reenvio del trabajo a los mismos u otros recursos.
Además durante esta fase se pueden requerir el llevar a cabo más fases que mencionamos en el siguiente punto

Paso 8 Tareas preparatorias. Este paso tiene mucho que ver con el anterior, ya que el envío del trabajo puede requerir de otros pasos para que se realice correctamente. Ejemplos de estos pasos pueden ser el transferir aquellos ficheros necesarios para la computación, hacer algún tipo de setup necesario, requerir la reserva realizada anteriormente, etcétera.
Esta fase también puede requerir la obtención de credenciales de usuario específicos para correr un trabajo determinado, ya que aunque estos servicios son ofrecidos a más bajo nivel, puede que no satisfagan automáticamente las necesidades del trabajo. Por ejemplo si un trabajo necesita acceder a otro recurso con nuevas o diferentes credenciales también se deberían obtener durante esta fase.

Paso 9. Ejecución y monitorización. Una vez que se han realizado todas las acciones necesarias y el envío del trabajo se ha realizado correctamente, el siguiente paso lógico es comenzar la ejecución del mismo. La ejecución misma está fuera del control de los planificadores de alto nivel, y más relacionada con los planificadores locales, así como la monitorización en primera instancia de lo que está acaeciendo durante la ejecución del trabajo. Sin embargo pueden darse otras posibilidades como que el usuario pueda monitorizar el proceso de ejecución de su aplicación, y posiblemente cambiar su opinión sobre dónde o cómo se está ejecutando.
Además puede darse la posibilidad de detectar automáticamente que el trabajo no se está ejecutando conforme a lo esperado, o que se está produciendo algún tipo de interbloqueo, por lo que el planificador podría decidir re-enviar el trabajo o cancelarlo en última instancia. Esto último es considerablemente más difícil en un sistema grid que en cualquier máquina paralela, ya que la falta de control sobre los recursos puede que haga fallar los pasos anteriormente descritos para una nueva ejecución. Esto es debido a que nuevos trabajos pueden haberse enviado desde el anterior, ocupando los recursos.

Paso 10. Finalización del trabajo. Cuando el trabajo ha finalizado el usuario puede requerir el obtener información sobre la ejecución del mismo, y también la obtención de la información y

los ficheros que el trabajo haya producido. La notificación misma puede ser síncrona o asíncrona, así como la obtención de estos ficheros de salida.

Paso 11 Tareas de Limpieza. Asociado con el paso anterior, las tareas de limpieza se encargarían de obtener los ficheros útiles y de limpiar todos aquellos restos de la ejecución del trabajo cuando ya no fueran necesarios.

4. GLOBUS

En 1995 se celebró el congreso SuperComputing'95, donde se demostró que era posible el ejecutar aplicaciones distribuidas de varias áreas científicas entre 17 centros de Estados Unidos conectados por una red de alta velocidad de 155 Mbps. Este experimento se denominó I-Way, y fue el punto de partida de varios proyectos en diferentes áreas, con un denominador común que era la compartición de recursos distribuídos de computación. A partir de este momento el libro "The Grid: Blueprint for a New Computing Instraestructure" [16], editado por Ian Foster y Carl Kesselman supuso el primer paso para establecer unas primeras ideas claras sobre cómo debía llevarse a cabo esta nueva tecnología.

A partir de estas ideas se desarrolló surgió el *Globus toolkit* [17], un proyecto *open-source* desarrollado en el *Argonne Nacional Laboratory* dirigido por Ian Foster en colaboración con el grupo de Carl Kesselman de la Universidad de Southern California. Globus da los medios básicos de la tecnología para construir un grid computacional, y se ha convertido gracias a su evolución y adopción por la comunidad científica como el estándar de facto en la tecnología grid.

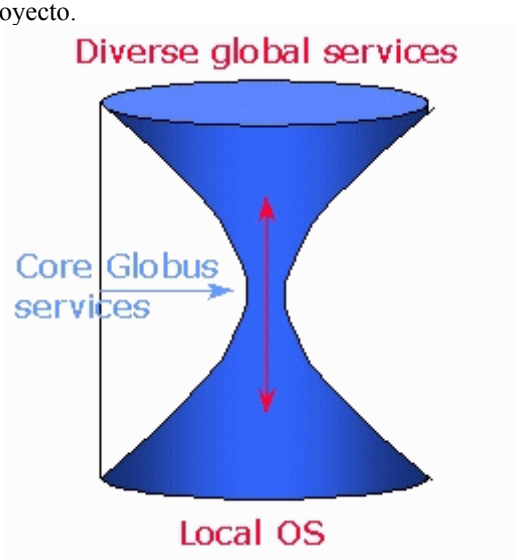
La arquitectura abierta de globus se estructura en capas, siguiendo los estandares propuestos por Foster y Kesselman y que se han presentado en la sección 2.2 e incluye servicios software para la monitorización de recursos, descubrimiento y gestión, además de servicios de seguridad y de gestión de ficheros.Se incluye software organizado en áreas como seguridad, infraestructura de la información, detección de fallos, portabilidad, etcétera. Está empaquetado como una serie de componentes que pueden usurase bien independientemente o conjuntamente para desarrollar aplicaciones.

El toolkit de globus fue concebido para quitar los obstáculos que impide la colaboración entre diferentes organizaciones o instituciones. Sus servicios centrales (*core services)*, interfaces y protocolos permiten a los usuarios acceder a los recursos remotos como si estuvieran presentes dentro de su propia sala de máquinas, a la vez que preservan el control local sobre quién y cuándo puede usar los recursos.

4.1. The Globus Hourglass

Los elementos del globus toolkit no asumen que los entornos locales estan adaptados para soportarlo. En principio fue diseñado e implementado para adaptar a los muchos y varios entornos locales bajo el cual pueda ejecutarse.

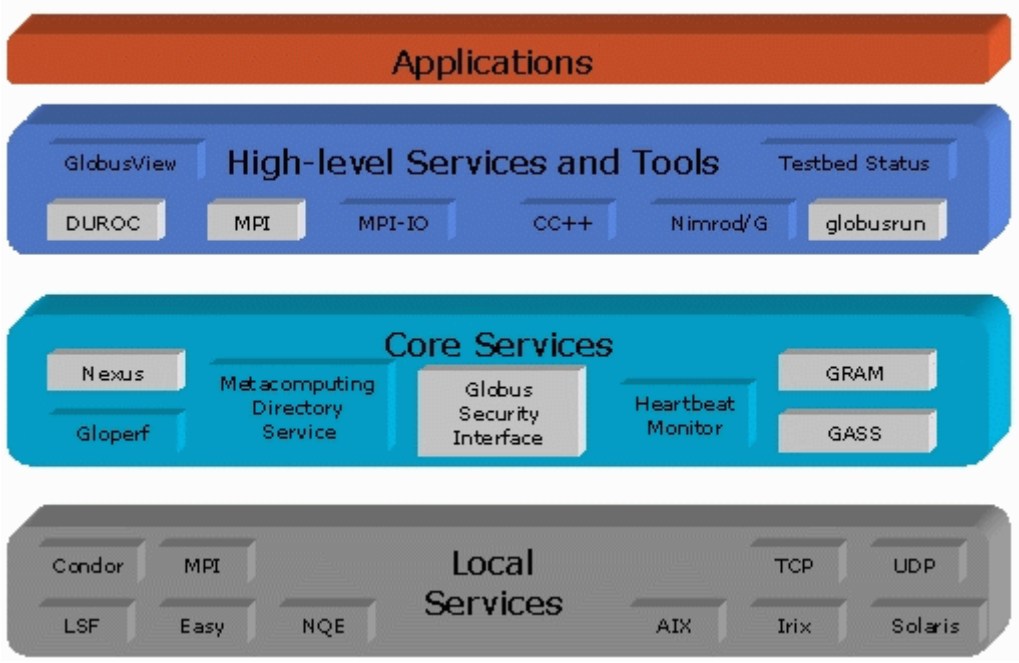
Globus ofrece una serie de servicios básicos para establecer una infraestructura básico. Éstos son luego usaros para construir soluciones específicas de cada dominio, de alto nivel. Para ello tres principios de diseño clave que se siguen son mantener los costes de participación bajos, mantener el control local cuando quiera que es posible, y proveer soporte para la adaptación del *toolkit* a las necesidades específicas de cada sitio y proyecto.



La base del *globus hourglass* representa los miles de recursos sobre los que los servicios de globus son construidos. Por ejemplo los sistemas operativos locales de las varios máquinas sobre las que corre

globus, además de los muchos tipos de redes, los sistemas de planificación, los sistemas de ficheros, etcétera. El medio es comprimido por los servicios básicos (*core services*) que globus ofrece, y la parte de arriba representa los servicios de alto nivel que globus ofrece, así como las aplicaciones escritas en globus.

Las implementaciones locales de los servicios de globus para un sistema operativo en particular liberan tanto a los servicios básicos como a los servicios de alto nivel de conocer cuestiones específicas del sistema operativo. Sólo los servicios locales necesitan conocer cuál es el sistema operativo sobre el que funciona, liberando a los programadores de aplicaciones de conocer estos detalles.
Una visión más detallada de los servicios ofrecidos por globus puede ser representado en la siguiente figura:



Los core services incluyen cuatro protocolos básicos que se ajustan al diseño de la arquitectura propuesto y proveen la funcionalidad básica necesaria. En la capa de conectividad se ofrece uno de los servicios más necesarios, el de seguridad con ***GSI (Grid Security Infraestructure)***.
Por encima de este y en la capa de recursos se ofrecen los servicios de control de recursos: ***GRAM (Grid Resource Allocation Management)***, los servicios de información: ***GRIP (Grid Resource Information Protocol)***, y de transferencia de datos: ***GridFTP (Grid File Transfer Protocol)***.

4.2.Seguridad en Globus: GSI

La seguridad es uno de los pilares fundamentales sobre los que se tiene que establecer un grid y todos los servicios superiores, y tenerse muy en cuenta ya que la filosofía de compartición de recursos tiene muchos problemas asociados.

Los recursos pueden ser valiosos y por lo tanto se debe permitir el acceso sólo cuando se desee y a las entidades que se desee. También los problemas a resolver pueden ser sensibles a la privacidad, así como los datos que estos problemas requieren o generan.

Por lo general, los recursos están situados en distintos dominios administrativos, por lo cual cada uno tiene sus propias políticas de acceso, procedimientos, mecanismos de seguridad, etcétera.
Para tener en cuenta estos puntos, la implementación de los servicios de seguridad tiene que estar públicamente disponible; lo que está relacionado con que los protocolos sean estándar, bien probados, y comprendidos por la comunidad.

El conjunto de recursos puede ser elevado, dinámico e impredecible, por lo que no estamos hablando de simple autorización y autenticación en un entorno cliente/servidor, sino que se necesita un método de

delegación de credenciales de servicio a servicio, por el cual sea posible que se pueda autorizar y autenticar en nuestro nombre bajo unas circunstancias determinadas y un entorno controlado.

Los requerimientos en cuanto a seguridad son varios, dependiendo desde el punto de vista que lo tomemos.

Para un usuario las características que se deben satisfacer son las siguientes:

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

Desde el punto de vista del dueño de los recursos, los puntos más importante a satisfacer serían:

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

Finalmente desde el punto de vista del desarrollador que necesita la utilización de estos servicios:

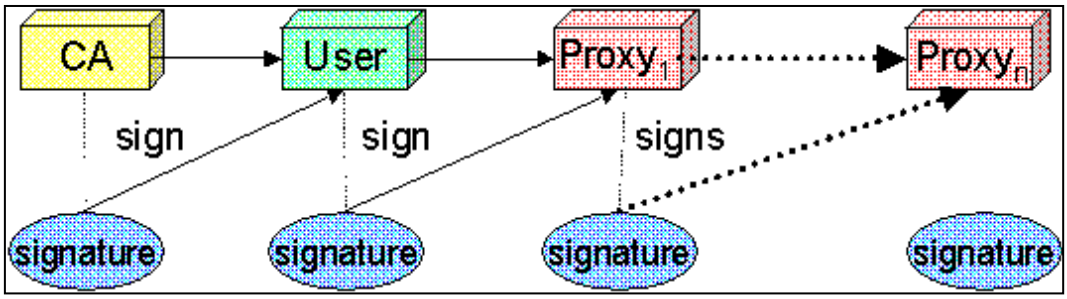
- Disponibilidad de un **API/SDK** con métodos para autenticar, hacer flexible la protección de mensajes, proveer métodos para la delegación, etcétera.
    - A través de llamadas directas (GSS-api)
    - O integrada en los servicios de nivel superior: GlobusIO, Condor-G, mpich-g2

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

Para la identificación se usan certificados X.509 [19], que se basa en una infraestructura de clave pública. Explicar en profundidad esta infraestructura está fuera de los propósitos de este trabajo, pero básicamente la infraestructura de clave pública (PKI) consiste es que permite conocer que una determinada clave pertenece a un determinado usuario (o entidad). Está basada en los principios de la encriptación asimétrica, en la que cada entidad posee un par de claves, una pública y otra privada, de manera que los datos encriptados con una clave sólo pueden ser desencriptados con la otra. La clave pública es conocida, mientras que la privada es secreta y sólo debe ser conocida por la entidad.

En la PKI, la clave pública es encapsulada en un certificado X.509 que identifica inequívocamente a la entidad para la que se emitió. Además del nombre único del dueño del certificado (*Distinguished Names o DN)*, éstos contienen otra información como la fecha de caducidad, el emisor del certificado, y una firma (hash) emitida por este último. El emisor del certificado será una Autoridad de Certificación (CA), que a través de su clave privada firmará este certificado dando fé de su autenticidad, y que siempre se puede comprobar a través de la clave pública de esta Autoridad.
Como hemos visto uno de los requerimientos principales para que sea posible la utilización en sistemas grids, es que sea posible hacer un único log-in y también la delegación de credenciales. Para ello es necesario la utilización de credenciales proxy (*proxy credentials) [21]*, que permitan que una entidad A otorgue a otra entidad B, el derecho de ser autorizada con otros como si fuera la primera entidad A De esta manera un proxy podrá ser creado a partir de un certificado normal X.509 o a partir de otro proxy, con el propósito de aplicar proxys restringidos según las necesidades. Además se podrá aplicar delegación , que consiste en la posibilidad de creación remota de un *proxy* credencial de segundo nivel.

Los proxies pueden ser restringidos, en el sentido de que se pueden delegar un subconjunto de derechos, lo que es deseable para tener restricciones de más fino grano. Después cada servicio en el que se quiere autenticar puede decidir si permite o no, por ejemplo como luego veremos el Job Manager requiere un *full* proxy sin limitaciones, mientras que GridFTP permite tanto un proxy con o sin limitaciones.



Uno de los logros recientes (Junio 2004) es que se ha conseguido integrar el sistema de delegacion de credenciales dentro de la infraestructura X.509 de manera oficial como un *Internet Standard* del IETF**,** cuyo RFC puede encontrarse en [22].
GSI además soporta el Standard GSSAPI [20] , implementando interfaces que soportan autenticación, delegación, integridad y confidencialidad de mensajes, lo que permite soportar aplicaciones como SSH, o GridFTP.

Una cuestión aparte es la **autorización**, que no es llevada a cabo directamente por GSI. Es decir una vez que una entidad se ha identificado como tal y sabemos inequívocamente quien es, el problema de autorización se basa en permitir o no el uso de estos recursos a esta entidad autenticada.
En la idea más simple, se mantiene en cada recurso que se quiere autenticar un fichero *gridmapfile* que contiene los *Distinguished Names* de aquellos certificados que se aceptan, por lo que cada vez que se accede a un servicio de este recurso se comprueba si el proxy con el que se está accediendo está incluido, permitiendo el acceso si así es o rechazándolo en caso contrario.

En la práctica un usuario de un grid, deberá disponer de un certificado de usuario X.509 expedido por una autoridad de certificación aceptada por éste grid. Con éste certificado le será posible contactar con los servicios de recepción de trabajos, pero realmente lo que se realiza en el cliente es obtener un proxy a partir del certificado de usuario que será el que realmente se envíe en la petición del servicio. Básicamente en el servidor se obtiene información del usuario a partir de este proxy, su DN que es el que se utiliza para ver si el usuario está finalmente autorizado o no a utilizar el servicio. Si este servicio necesita la comunicación con otros se utilizará la delegación de este proxy como método para autenticarse en éstos nuevos servicios.

4.3.Servicios de información

Los servicios de información son una parte esencial dentro de la arquitectura grid, ya que en estos sistemas el descubrimiento, la caracterización, y la monitorización de los recursos, servicios y computaciones es una tarea difícil debido al comportamiento dinámico y a la distribución geográfica de los recursos. Los servicios de información proveen los mecanismos básicos para el descubrimiento de recursos y su monitorización, y por lo tanto para planificar y adaptar el comportamiento de las aplicaciones. La arquitectura que utiliza Globus para los sitemas de información es la denominada MDS-2 [31], que forma parte del globus toolkit, y que ha sido utilizada hasta la fecha.

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

- Una gran colección distribuída de genéricos *proveedores de información* dan acceso a la información sobre entidades individuales, a través de operaciones o *gateways* a otras fuentes de información (ej. SNMP queries). La información es estructurada en terminos de un modelo estándar de datos, tomado de LDAP: una entidad es descrita por un conjunto de "objetos" compuestos de pares atributo-valor.
- *Servicios de alto nivel*, que recolectan, gestionan, indexan y responden a la información suministrada por los anteriores *proveedores de información*. Se distinguen en particular servicios agregados de directorio, que facilitan el descubrimiento y monitorización de Vos implementando vistas y búsquedas tanto genéricas como especializadas, para una colección de recursos. Otros servicios de alto nivel pueden usar esta información, e información directamente obtenida por a proveedores.

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

En la implementación de MDS-2, se definen 2 protocolos base, GRIP y GGRP, con el primero definido para ser LDAP y GGRP definido en [32] , pero sin especificar el protocolo de transporte. En MDS-2.1 se adopta LDAP como el protocolo GGRP, con sus mensajes mapeados en operaciones *add* de LDAP, y entonces llevados por el mismo protocolo LDAP. Esta elección fue hecha por razones pragmáticas ya que simplificaba el desarrollo. Protocolos alternativos para GRRP son ciertamente posibles y en el futuro se puede usar SOAP para este propósito, cuando éste sea el estándar para otros servicios grid.

4.3.1. Proveedores de Información (GRIS)

MDS-2 incluye un proveedor de información estándar y configurable denominado *Grid Resource Information Service* (GRIS). Este entorno está implementado como un servidor OpenLDAP como *backend* que puede ser personalizado incluyendo fuentes específicas de información. Hasta la fecha, incluye información sobre datos estáticos del host (versión del sistema operativo, tipo de CPI, número de procesadores, etcétera), información dinámica del mismo (carga del sistema, entradas en la cola del planificador), información sobre el almacenamiento secundario (espacio total y disponible), etcétera .Este GRIS se comunica con los proveedores de su información a través de una API bien definida, implementada de dos maneras. Una versión más sencilla que agrupa *scripts* que son llamados por el *backend.* Otra versión más compleja implementada a través de módulos cargables dinámicamente, de manera que pueden ejecutarse directamente en el *backend* , con baja latencia.

Para controlar la intrusividad de las operaciones del GRID, mejorar el tiempo de respuesta, y maximizar la flexibilidad, los resultados de cada proveedor de información pueden guardarse en cache durante cierto tiempo, configurable. Esta cache *Time-To-Live (TTL)* es especificada como parte de la configuración de cada proveedor.

4.3.2. Directorios Agregados (GIIS)

La implementación de MDS-2 sirve como *framework* para construir directorios agregados de información llamados *Grid Index Información Services (GIIS)*, y que provee la estructura jerárquica necesaria pare estos sistemas de información. El directorio acepta mensajes GRRP de sus "hijos" GRIS o GIIS, y hace el *merge* de estas fuentes de información dentro de un espacio de información unificado. Las búsquedas de los clientes también pueden obtener información de cualquiera de los GIIS que se encuentran debajo de la estructura.

El GIIS engloba a tres elementos principales: el manejador de GRRP, el constructor de índices cargable, y el manejador de búsquedas asimismo cargable. La funcionalidad está asimismo implementada con el servidor de OpenLDAP a través de un *backend* específico, y de hecho los dos módulos cargable de construcción de índices y manejo de búsquedas usan la misma API que el GIIS usa para acceder a los proveedores de información. En nuestro servicio de directorio agregado simple, se implementa el encadenamiento o *Chiang*: peticiones GRIP dirigidas a los GIIS son simplemente redirigidas al apropiado productor de la información para obtener la respuesta correcta.

El rendimiento tiene que ver principalmente con hacer cache de los datos dentro de los GIIS, provista dentro del mismo *framework*.
La implementacIón del GRIS y del GIIS tienen mucho en común, ya que los dos dependen de LDAP como *front-end* para el procesado del protocolo, autenticación y filtrado de resultados. Ambos usan un API común para la personalización y de hecho pueden coexistir en el mismo servidor físicamente.

4.4.Arquitectura de Gestión de Recursos

Una de las partes más importantes es la gestión de recursos de un sistema grid, que como ya hemos comentado se encarga de asignar los recursos necesarios a los trabajos que son enviados.
Según [23], la gestión de recursos realizada en globus está relacionada con los problemas de localizar y asignar recursos computacionales, y con la autenticación, creación de procesos, y otras actividades requeridas para preparar los recursos para su uso. No se ocupa de cuestiones como la planificación

(scheduling) , descomposición, asignación y ordenación de ejecución de tareas, o la gestión de otros recursos como memoria, discos y redes.
Los sistemas de meta-computación como son los Grids, introducen cinco desafíos de gestión de recursos que son tenidos en cuenta por globus: autonomía de los recursos, heterogeneidad, extensión de políticas, co-asignación, y control *on-line.*

- El problema de la *autonomía de los sites*, hace referencia al hecho de que los recursos son típicamente gestionados por diferentes organizaciones, en diferentes dominios administrativos. Así no podemos esperar encontrar cosas comunes en términos de políticas de uso, de planificación, mecanismos de seguridad, y demás.
- Relacionado con el punto anterior está la *heterogeneidad* de los sistemas, y que está representado por el hecho de que los diferentes *sites* pueden usar gestores locales diferentes como Condor[26], NQW, CODINE[24], EASY[30], LSF[28], PBS[27].
- La *política de extensibilidad* surge debido a que las aplicaciones sobre grid pueden venir de muchos recursos administrativos, cada uno con sus propios requerimientos. Una solución correcta debe soportar el desarrollo de nuevas estructuras de gestión específicas de cada dominio, sin requerir cambios en el código instalado en los sitios participantes.
- La *co-allocation* (o co-asignación) hace referencia al requerimiento de aplicaciones que necesitan recursos simultáneamente en varios sitios. La autonomía de los sitios y la posibilidad de fallo durante la asignación introduce la necesidad de mecanismos especializados para la asignación de múltiples recursos, iniciar la computación en estos recursos, y monitorizar y gestionar estas computaciones.
- El *control online* hace referencia a la negociación requerida para adaptar los requerimientos de la aplicación a la disponibilidad de recursos, particularmente cuando los requerimientos y las características de los recursos cambian durante la ejecución. Por ejemplo, una aplicación que necesita simular una entidad puede preferir renderizar a baja resolución, si la alternativa es que no se pueda simular en absoluto. Los mecanismos gestores de recursos deben soportar esta negociación.

No existe ningún gestor de recursos que haga frente a los cinco problemas recién planteados. Algunos sistemas de colas batch soportan co-asignación, pero no autonomía, políticas de extensibilidad, o control online. *Condor* soporta la autonomía de los sitios, pero no co-asignación ni control online. *Legion* y *Gallop* tienen control online y extensibilidad de políticas, pero no tienen en cuenta el substrato heterogéneo o los problemas de co-asignación.

En la arquitectura de gestión de recursos de globus, se tiene en cuenta el problema de la autonomía de los sites y la heterogeneidad de los diferentes recursos introduciendo las entidades que se denomina ***resource managers*** *(gestores de recursos)*, para proveer una interfaz bien definida de las diferentes herramientas de gestión de recursos, políticas, y mecanismos de seguridad que cada site utilice.

Para el control *on-line* y la extensibilidad de políticas se define un **lenguaje de especificación de recursos**, que soporta negociación entre diferentes componentes de la arquitectura de gestión de recursos. Se hace frente al problema de la co-asignacion definiendo varias estrategias de, que encapsulamos en los gestores.

4.4.1. Lenguaje de especificación de recursos

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

4.4.2. Asignación de recursos: GRAM (Grid Resource Allocation Management)

GRAM (Grid Resource Allocation Management representa el nivel más bajo de la arquitectura globus de gestión de recursos, que implementan los *resource manager* locales. Representa un conjunto de protocolos para enviar, monitorizar y terminar un job y es responsable de varias acciones:

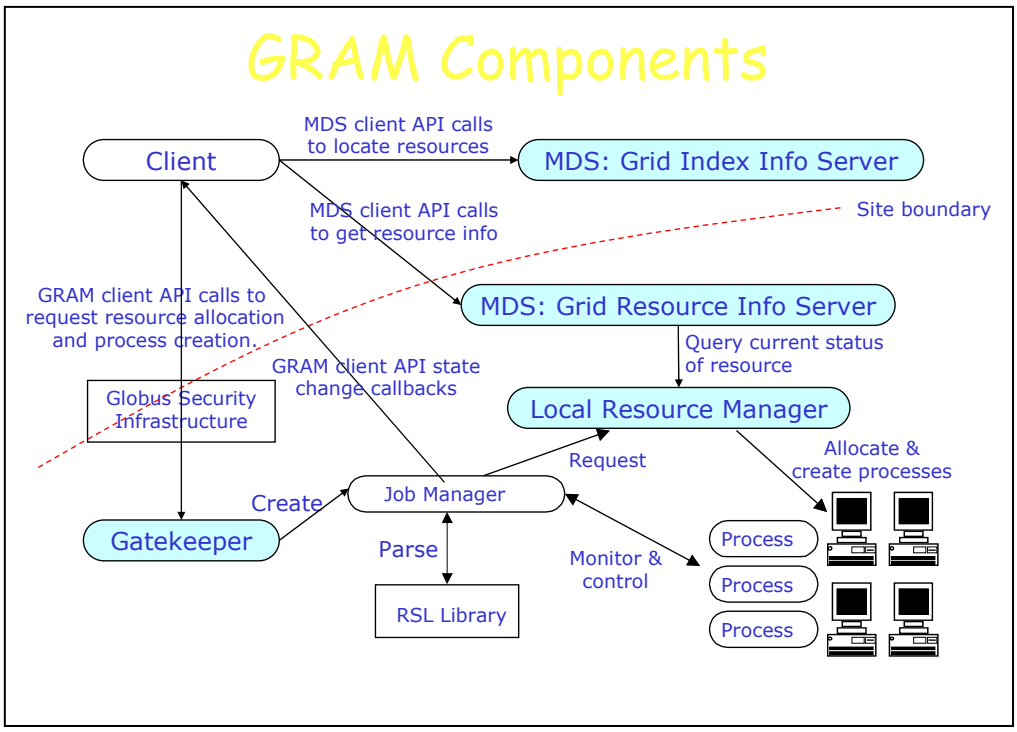
- Procesar las especificaciones RSL que representan peticiones de recursos, bien denegando la petición o creando uno o más procesos que satisfagan la petición.
- Habilitar la monitorización y gestión remota de trabajos creados en respuesta a estas peticiones.
- Periódicamente actualizar la información disponible sobre los recursos y su disponibilidad, en los sistemas habilitados para ello (Sistemas de Información).

Gram está pensado para hacer de interfaz entre los sistemas de planificación global, y los gestores locales de recursos, de manera que una petición que llega al resource manager local deberá ser lo suficientemente concreta para que se pueda satisfacer. Los gestores locales no tienen que representar un único host, sino que más bien representan un servicio que da acceso a varios recursos computacionales. Por lo tanto, Gram necesita tener interfaces con estos sistemas locales, de manera que se pueda mapear entre las especificaciones RSL y los recursos que están disponibles. En la actualidad se disponen de interfaces con sistemas gestores locales como son Condor [26], PBS [27], Easy, Fork, LoadLeveler, LSF [28] y NQE.

Estas interfaces implementan una serie de métodos y funciones que constituyen la API de GRAM, y que juega un papel para la gestión de recursos como hace IP para la comunicación: puede co-existir con mecanismos locales, como IP está sobre ethernet, FDDI o la tecnología ATM.

Esta API provee funciones para lanzar y cancelar trabajos, y para preguntar sobre el estado de los mismo: si está ejecutándose o si está aceptado y en cola para ello. Cuando un trabajo es enviado, un manejador (*job handle)* único es devuelto, y éste puede ser utilizado para monitorizar y controlar el progreso del trabajo. Adicionalmente en la petición de envío del trabajo, puede requerirse que el progreso del mismo sea comunicado asíncronamente a una dirección URL de *callback*. Estos identificadores pueden pasarse a otros procesos, y los *callbacks* no tienen que ser dirigidos obligatoriamente al proceso que envió el trabajo. Esto hace posible el envío de trabajos por terceras partes en nombre de la aplicación, como un gestor de más alto nivel o un *co-allocator*.

La implementación de esta arquitectura es la siguiente:



Los principales componentes son la librería GRAM cliente, el gatekeeer, la librería de *parking* de RSL, el job manager, y el GRAM *reporter*.

La librería **GRAM cliente,** es usada por el usuario, aplicación o quien actúe en su nombre para enviar el trabajo. Interactúa con el gatekeeper de un sitio remoto realizando una autenticación mútua a través de los métodos GSI que se han comentado en la sección 4.2. Envía una especificación de recursos, una posible petición de callback, y algunos otros componentes no directamente relacionados con la gestión de recursos.

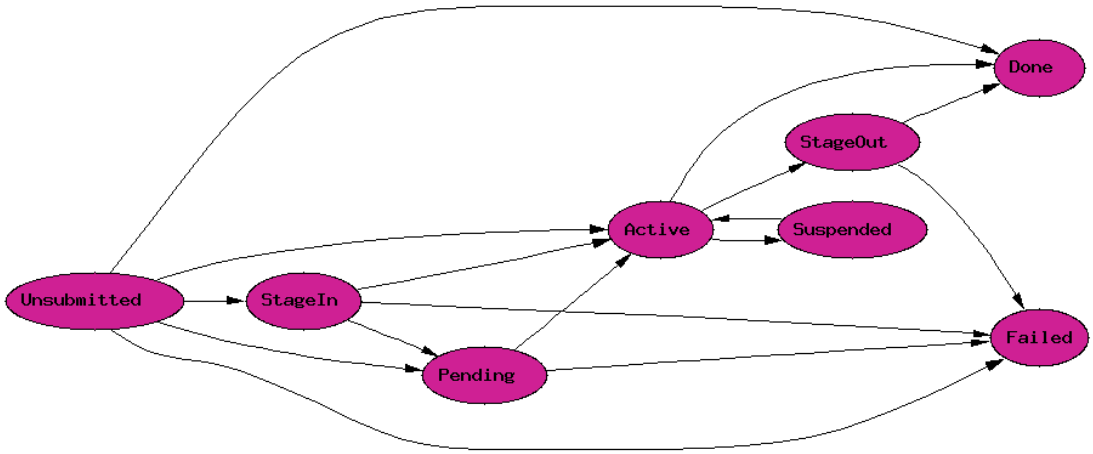
El ***gatekeeper*** gram es un componente bastante simple que debe estar ejecutándose como servicio remoto, y que hace básicamente tres cosas: realizar la autenticación mutua del usuario y los recursos a través de los servicios más básicos de GSI, determinar el usuario local bajo el cual se ejecutará el programa (también realizado en conjunción con estos servicios GSI), y ejecutar un *jobmanager* que se encargará de ejecutarse bajo ese usuario efectivo y hacerse cargo de cada petición. Los 2 primeros tareas son realizadas como hemos comentado por ls infraestructura de seguridad de Globus , que maneja la autonomía y el substrato heterogéneo en el dominio de la seguridad. Para empejar el job-manager, el gatekeeper debe correr como un usuario privilegiado. Aún así, debido a la efectividad de la interfaz con GSI el gatekeeper ha pasado numerosas revisiones por parte de organizaciones que han validado su código para su ejecución por parte de grandes centros de recursos. El mapeor de usuarios remotos a usuarios locales minimiza la cantidad de código que se debe ejecutar de mantera privilegiado. Además nos permite delegar en el sistema local.

El ***jobmanager*** es responsable de crear los procesos necesarios para satisfacer la petición del usuario. Esta tarea básicamente tiene que ver con realizar la petición correspondiente de recursos al sistema de gestión de recursos local (PBS, LSF, …), aunque también si no existe un simple *fork* puede realizarse. Una vez que los procesos son creados, el job manager también es responsable de monitorizar y verificar el estado de los procesos creados, notificando mediante *callbacks* los cambios de estados por los que pasa la petición, y implementando operaciones de control como las de terminación del trabajo. Realmente está formado por dos componentes:

- Componentes Comuner (*Common Components*): traslada los mensajes recibidos, del gatekeeper y el cliente en si a una API interna que es implementada por el componente específico de la máquina. También translada las peticiones de *callback* del componente especifico a través de la API interna en mensajes al gestor de la aplicación
- Componente específico de la Máquina (*Machine-Specific Component*): implementa la API interna en el entorno local. Esto incluye llamadas al sistema local, mensaje al monitor del recurso, y preguntas al Sistema de Información.

Este *jobmanager* terminará una vez que el trabajo del cual es responsable ha terminado.

Los estados por los que pasa el trabajo se pueden modelar por el siguiente diagrama:



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

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

La implementación del GRAM se ha llevado a cabo para seis planificadores hasta la fecha (Condor, LSF, NQE, Fork, EASY y Load Leveller). Gran parte del código es independiente del gestor local, y sólo una parte relativamente pequeña es dependiente directamente de éste. De hecho estas partes (principalmente del *jobmanager*) suelen ser implementadas a través de shell scrips que hacen uso de las APIs externas de estos planificadores. Las transiciones de estado son mayoritariamente manejadas por *polling* , porque se ha probado que es más fiable que monitorizar los procesos del trabajo a través del propio planificador local.

5. BENEFICIOS Y CARENCIAS DE LAS APROXIMACIONES ACTUALES

Como hemos visto, Globus proporciona una serie de componentes agrupados en el *globus toolkit*, que permiten utilizar recursos distribuidos pertenecientes a organizaciones distintas, y que son gestionados por éstas de manera independiente.
Por el desarrollo de este *globus toolkit* y su aceptación, se considera a globus como el estándar de facto en middleware y tecnología grid. Pero debido a su concepción, en la que se intenta proporcionar los servicios básicos necesarios para cubrir los objetivos primordiales del grid, puede que no satisfaga las necesidades de usuarios y aplicaciones determinadas.

Este es el caso que originó el proyecto Datagrid [33] en el que un grupo de aplicaciones necesitan de servicios comunes de igual o más alto nivel y que no son proporcionados directamente por ningún middleware, en particular globus.

Las aplicaciones que forman parte de este proyecto son de las áreas de Física de altas energías, Observación de la Tierra y Bio-Informática. Es estas aplicaciones la comunidad científica tiene y tendrá en los siguientes años la necesidad de acceder y procesar grandes cantidades de datos que exceden las cantidades presentes por, al menos, un orden de magnitud. Además estas comunidades están geográficamente distribuidas y pertenecen a diferentes organizaciones e institutos de investigación, cada uno gestionado independientemente, y para los que las ideas de la tecnología grid se ajustan perfectamente.

Casi Paralelamente, surgió el proyecto Crossgrid [34], en el que este trabajo se encuadra, y que se caracteriza por la necesidad de ejecutar aplicaciones paralelas e interactivas. En este proyecto las aplicaciones son de las áreas de Bio-medicina, Análisis de riesgos de inundaciones, Análisis distribuido en HEP, y finalmente Polución del aire y predicción del Tiempo. Son también aplicaciones intensivas en datos por lo que tiene este punto en común con el anterior, pero van un paso más allá añadiendo el paralelismo y la interacción directa de una persona durante la ejecución de la misma.

5.1.Nuevos Requerimientos

El principal objetivo del proyecto Crossgrid es crear un testbed de recursos distribuidos a lo largo de Europa, y proveer de soporte específico para la ejecución de aplicaciones paralelas e interactivas, especialmente aquellas de ejecución intensiva sobre datos.

En el diseño de nuestro gestor de recursos se tuvieron en cuenta los requerimientos planteados por las aplicaciones del proyecto *Crossgrid*, pertenencientes al *Work Package 1 (WP1)* y plasmadas en una serie de documentos que agrupan las necesidades de estas aplicaciones ([35],[36],[37],[38],[39]).

Por lo tanto se plantea la primera necesidad de soportar aplicaciones paralelas escritas usando MPI ya que constituyen las aplicaciones más mayoritarias. El soporte para otras librerías de programación en paralelo como PVM, o de entornos como CACTUS no es prioritario, aunque las diferencias con respecto a un soporte para MPI son pocas. Se barajaron implementaciones específicas de MPI, siendo las más comúnmente usadas las librerías MPICH y MPICH-G2., que constituye un desarrollo de Globus para la comunicación de máquinas en área extensa o WAN mediante los propios métodos de comunicación y seguridad GSI de Globus.

El término de aplicación o trabajo (job) tiene diferentes acepciones, de acuerdo con las descripciones de los documentos del WP1. Muchas de estas aplicaciones descritas pueden ser decompuestas en varios componentes, interoperables y dependientes entre sí. Estos componentes pueden correr en paralelo ([35]), pero también demuestran interdependencia entre ellos como en el caso de la aplicación sobre prevención de inundaciones [38]. Algunos de estos componentes tienen funciones especializadas como es el caso de aquellos que se encargan de la visualización, de la toma de datos, de la interacción con el usuario, y ello les fuerza a ejecutarse en recursos determinados, en principio sin ninguna intervención del *CrossBroker* ya que no deben ejecutarse en recursos grid, sino más bien en recursos locales. El resto de componentes son candidatos para ser ejecutados en recursos grid, y consisten en computaciones de mucho tiempo de ejecución como simulaciones (aplicación biomédica, de prevención de inundaciones y de polución del

aire, predicción del tiempo, y modelado de ondas marinas) y en otros casos de técnicas de *data mining* (en HEP y también predicción del tiempo).

Además, las aplicaciones una vez han sido enviadas al Grid y empezado su ejecución en recursos remotos, también pueden convertirse en interactivas de manera que el usuario pueda interactuar con ellas. Esto puede ser útil para analizar los resultados intermedios antes de que la aplicación finalice completamente, y poder actuar respecto a ellos. El usuario podría modificar algunos parámetros de sus simulaciones para cambiar el resultado final si merece la pena, o incluso cancelar el trabajo en el caso de que su simulación no vaya convergiendo, y el resto de computaciones hasta la finalización sean innecesarias.

En este último caso en el que el usuario mata su trabajo, podría enviar un nuevo trabajo con parámetros de ejecución diferentes, de manera que se podrían usar métodos para reenviar este trabajo a los mismos recursos o a otros diferentes.

Las aplicaciones HEP interactivas requieren también la existencia eventual de un componente responsable de reservar nodos e instalar servidores de datos en ellos, lo cual implica transferencias de datos antes de que la aplicación misma empiece a ejecutarse.

Los Requerimientos funcionales de las aplicaciones presentadas se pueden resumir en los siguientes puntos.

- Acceso transparente: los usuarios no tienen por qué conocer o darse cuenta dónde los datos o recursos de computación están disponibles
- Descubrimiento de recursos: determinar automáticamente los recursos disponibles para un trabajo, como recursos de computación, ancho de banda, y posibles múltiples copias de ficheros conteniendo los datos pedidos por los trabajos
- selección de recursos: hacer elecciones inteligentes entre recursos alternativos disponibles para un trabajo, teniendo en cuenta factores como:
    - potencia de cálculo disponible, vs. Necesidades de computación estimadas
    - proximidad de los datos a varios recursos de computación, vs. La cantidad estimada de datos que se necesitan acceder.
    - Espacio disponible en los recursos de almacenamiento en el grid, vs. La proximidad de los recursos de computación elegidos y la cantidad estimada de datos de salida.
- Gestión de los recursos: gestionar los recursos disponibles de la mejor manera, para optimizar su uso, y en el caso de que sea necesario, la creación de nuevos recursos como réplicas de datos en un elementos de almacenamiento distinto en particular, si se observa que esto será beneficioso para la ejecución de nuevos trabajos en el futuro.
- Único Log-in: un usuario debe ser capaz de dejar ejecutando un trabajo en los recursos disponibles a su Organización Virtual, una vez que se ha "logado" en los servicios grid.

En términos de tipos de trabajos podemos distinguir los siguientes requerimientos:

- Trabajos simples secuenciales con grandes requerimientos de datos
- Partición de trabajos: los trabajos HEP son por lo general trabajos caracterizados por diferentes eventos independientes, encuadrándose dentro de los trabajos denominados de High Throughput Computing. Esto debe poder aprovecharse potencialmente descomponiendo el análisis en muchos nodos, así como recombinándolos transparentemente para obtener un único output.
- Trabajos paralelos programados con *mpi*
- Requerimientos de interactividad en los trabajos.

5.2.Carencias Del Middleware Globus

Una de las carencias más importantes de las que adolece el *toolkit* es de un servicio de *brokering* automático de recursos dentro de los que sería una entidad como un súper-planificador (*superscheduler*) que se encargue de tomar las preferencias de los trabajos de los usuarios y buscar los recursos necesarios para satisfacerlas, de manera automática.

Esta entidad está supuestamente prevista en la arquitectura de Globus, de manera que traslada especificaciones generales de recursos en especificaciones más concretas, pero realmente sólo se plantea como una entidad abstracta y no existe implementación real de la misma con lo que la gestión de recursos

es incompleta. El usuario debe localizar los recursos y seleccionarlos manualmente, para después lanzar los comandos adecuados sobre éstos. Globus se encarga de enviar el trabajo a estos recursos y ejecutarlos, pero además el usuario es encargado de comprobar que la ejecución es correcta y progresa. En realidad aunque existen los métodos básicos para hacer casi todos lo pasos definidos en el punto 3.4 en cuanto a la planificación, las tareas llevadas a cabo automáticamente se reducen al Paso 7 que se refiere al envío del trabajo propiamente dicho.

Si la ejecución no es correcta, el usuario debe realizar las acciones adecuadas para cancelar el comando y volver a reenviar el trabajo, repitiendo los Pasos 1 al 6 de localización de recursos compatibles. Así mismo no se realizan las Tareas de limpieza necesarias, por lo que los datos del trabajo quedan en el lugar de ejecución remoto.

Además aunque existen los métodos básicos para la transmisión de ficheros de grandes volúmenes de datos, globus no contaba al principio con métodos para el tratamiento de réplicas de ficheros. Esto fue uno de los objetivos del proyecto Datagrid y como consecuencia ahora existen estos servicios concentrados en lo que se denomina el *Replica Location Service*.

Sin embargo no existe una integración a día de hoy de este componente con otros servicios de globus, que hagan más fácil la ejecución de aplicaciones intensivas en datos. Esto significa que el usuario debe actuar como planificador también en los datos dentro de lo que sería el Paso 8 de tareas preparatorias que se ha definido anteriormente. Si un trabajo necesita unos datos específicos se deben poder acceder desde el recurso donde finalmente se ejecute el mismo, lo que puede requerir su copia local. De esta manera es lógicamente más eficiente el acceder a datos que están más cercanos ya que será más eficiente. Esta planificación teniendo en cuenta los datos tampoco existe y el usuario debe realizarla manualmente, cuando realmente debe ser una tareas del *superscheduler* o broker.

Esta funcionalidad está soportada por el broker de DataGrid [41] para trabajos simples batch. Los trabajos paralelos no están soportados por este broker, aunque globus permite enviados en línea de comandos haciendo pasos manuales En el caso de los paralelos las tareas de planificación pueden ser más complicadas cuando podemos ejecutar entre varios recursos distribuidos, y no existe un soporte específico para ello. Los trabajos interactivos también requieren una planificación específica para su ejecución lo más rápidamente posible en los recursos adecuados.

Nuestro objetivo es satisfacer los requerimientos y dar un soporte automático para la planificación de estas aplicaciones, tanto para trabajos secuenciales, como paralelos de distintos tipos, teniendo en cuenta que además pueden ser interactivos. Para ello se tienen que satisfacer todos los pasos descritos en el punto 3.4 para estos tipos de trabajos . En el siguiente apartado se muestra la arquitectura de de nuestro super-planificador y cómo se soportan las funcionalidades requeridas, así como su implementación llevada a cabo dentro del proyecto Crossgrid [34].

6. GESTIÓN DE RECURSOS EN CROSSGRID: CROSS-BROKER

Según los requerimientos podemos determinar los tipos de trabajos que pueden ser enviados a nuestro CrossBroker. Por ejemplo, dependiendo del modelado de las aplicaciones, los trabajos pueden ser:

- *Trabajos simples*: que consisten en una aplicación que puede ser identificada con un ejecutable. Dependiendo de qué tipo de aplicación sea también podemos distinguir entre:
  - Aplicaciones secuenciales: son ejecutadas en un procesador, por lo cual pueden ser mapeadas directamente a un recurso computacional.
  - Aplicaciones paralelas: son ejecutadas en más de un procesador, determinado por el usuario por lo cual puede requerir más de un recurso computacional

- *Trabajos compuestos*: consisten en una colección de trabajos simples, cada uno de los cuales puede ser secuencial o paralelo, y que se relacionan entre si por una serie de interdependencias. Estos trabajos pueden ser representados mediante un Grafo Acíclico Dirigido (*DAG, Directed Acyclic Graph*) donde cada nodo representa un trabajo simple y los vértices representan cuales son las dependencias. Estas dependencias pueden ser las entradas, las salidas, o la ejecución de un trabajo es dependiente de otro.

Dependiendo de la interacción con el usuario, podemos distinguir:

- *Trabajos batch* los cuales una vez enviados no es necesaria ninguna interacción con el usuario. Una vez acabe el trabajo el usuario podrá obtener los resultados del mismo
- *Trabajos interactivos*: en los cuales en usuario interacciona con el trabajo, accediendo a su ejecución antes de que acabe y siendo posible modificar su ejecución.

Respecto a los trabajos interactivos se han detectado situaciones diferentes en las que el usuario puede interactuar con el trabajo:

1. El usuario puede cancelar un trabajo como resultado de la inspección de la salida de un trabajo durante su ejecución. Esta situación requiere un mecanismo de comunicación entre la aplicación y el usuario para transferir la salida, por lo que puede depender de los mecanismos de los gestores locales donde se ejecute el trabajo.

2. El usuario puede modificar la ejecución de su trabajo casi en tiempo real, conforme se va ejecutando. Asimismo se requiere un mecanismo de comunicación entre el usuario y la aplicación que puede estar ejecutándose en múltiples recursos remotos.

3. Puede ser necesario que el CrossBroker empiece inmediatamente la ejecución de una aplicación interactiva, logrando un tiempo de respuesta lo menor posible. De nuevo se requiere que los gestores locales soporten estas técnicas de prioritización de manera nativa para que la gestión sea eficiente. Características como la reserva avanzada pueden ser de apreciable valor para soportar estas situaciones. Otros métodos pueden ser la prioritización entre trabajos de un mismo usuario por ejemplo, en el caso en el que se lancen varios trabajos. Este esquema podrá ser usado conjuntamente con mecanismos de preemción (pre-emption) de trabajos, suspendiendo un trabajo que ya está corriendo y mandando el trabajo interactivo en estos recursos que se ejecutará hasta que acabe, finalmente dejando el recurso al trabajo original.

En la práctica los trabajos simples son cualquier tipo de aplicación batch que se pueda planificar en una cpu, durante una duración variables. Estas aplicaciones necesitan unos datos de entrada y producen unos datos de salida, que pueden ser localizados de forma distribuida en el testbed.

Los trabajos paralelos que vamos a ejecutar son aplicaciones que necesitan más de una cpus, con los mismos requerimientos de datos. Pueden ser compilados mediante varias implementaciones de librerías mpich, pero las más comunes serán mpich-p4 y mpich-g2. Con las aplicaciones mpich-p4 deben correr intra-cluster, es decir en un único cluster computacional que disponga de las cpu necesarias.

La librería mpich-g2 está compilada con un *device* especial de globus, lo que permite la comunicación usando estos métodos. Esto permite ejecutar estas aplicaciones de manera inter-cluster, aunque también

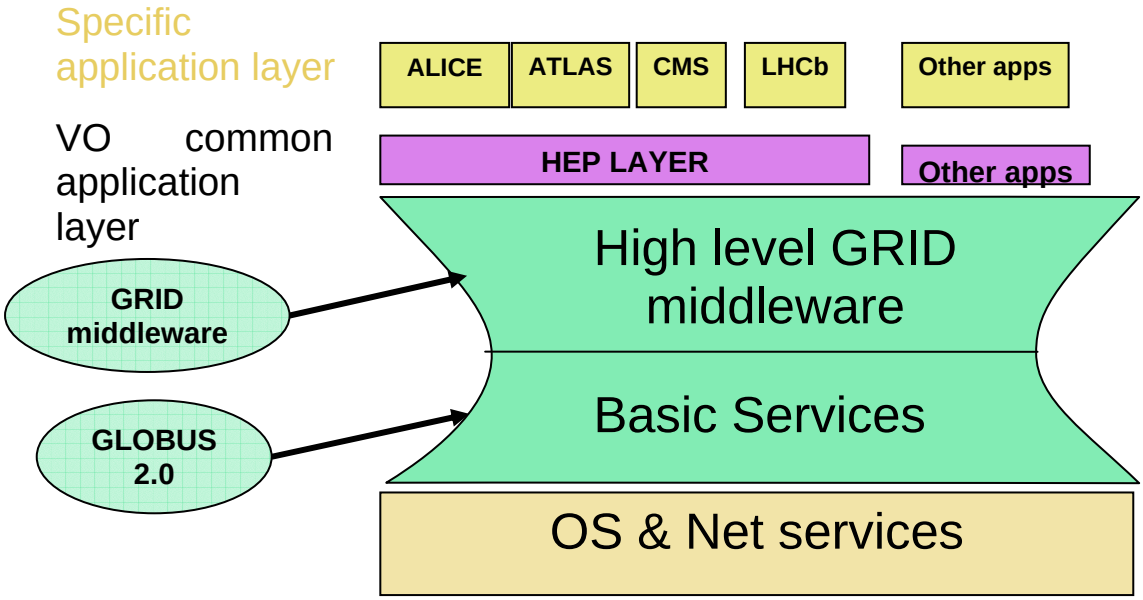
en un único cluster como p4. La planificación será más difícil para correr Inter-cluster ya que se necesitan más de 1 recurso.

6.1.Arquitectura del Testbed

En este punto se describe la arquitectura general del grid en forma de testbed, describiendo las entidades que entran a formar parte de él. En él se encuadra nuestro broker de gestión de recursos ya que es importante enmarcarlo dentro la arquitectura en la que se ha desarrollado.

Obviamente en el nivel básico tenemos la funcionalidad que aporta el *globus toolkit* [17] y que se usa como nivel básico de acceso a los recursos distribuidos. Como hemos visto este toolkit aporta métodos para la autentificación y autorización de usuarios, la ejecución de trabajos simples a través de GRAM, el envío fiable de ficheros, así como los sistemas de información donde obtener un estado del testbed.

Sin embargo para que se puedan utilizar estos servicios por parte de las aplicaciones y los usuarios finales se establecen capas superiores en el middleware que proveen la funcionalidad requerida.



Esta capa denominada Grid Middleware engloba los servicios que son necesarios para la gestión de recursos y de datos, así como las extensiones a globus que hacen de las aplicaciones más fácilmente utilizables.

Las entidades que forman parte de esta arquitectura distribuida tienen su base también en el proyecto globus y en el modelado posterior que se hace en el proyecto Datagrid. Adicionamente se incluyen las entidades propias del proyecto Crossgrid donde se encuada nuestro CrossBroker. Las entidades que forman parte de nuestra arquitectura se pueden dividir en diferentes áreas:

Intefaz de usuario:

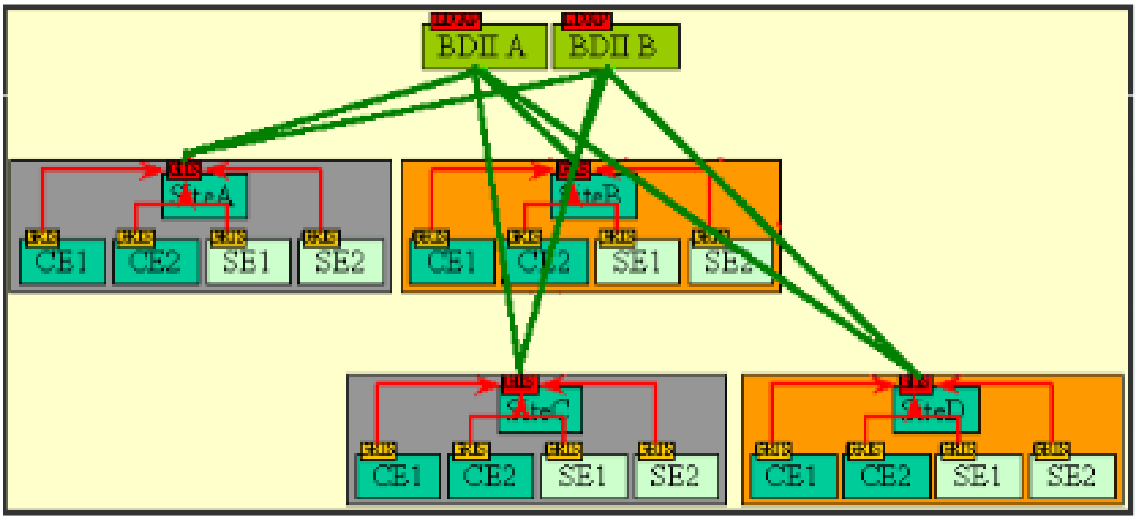
- *USER INTERFACE (UI):* Es el punto de acceso al grid, una máquina donde un usuario tiene una cuenta y puede acceder a su certificado privado. Desde esta interfaz puede enviar trabajos al grid y ejecutar un número de acciones determinada sobre datos, mediante linea de comandos o accediendo a una interfaz web.
- *MIGRATING DESKTOP (MD):* En un cliente java de acceso a los mismos. El usuario puede descargarlo a su navegador y acceder a través del Roaming Access Server
- *ROAMING ACCESS SERVER (RAS):* Es una entidad que mantiene los profiles de varios usuario, permitiendo el acceso remoto desde cualquier navegador, ya que se puede descargar el MD y acceder al RAS.

Recursos:

- COMPUTING ELEMENT (CE): Un elemento de computación es definido una cola grid de trabajo, y representa el acceso a unos recursos de computación determinado. Es identificado por una cadena única <hostname>:<port>/<batch_queue_name>. Podemos definir varias colas diferentes para un mismo hostname, pero se consideran CEs diferentes aunque residan en la misma máquina físicamente. En la práctica el CE está formado por un gatekeeper de globus, la parte servidor de GRAM, el servidor del LRMS que da acceso a los recursos locales, junto con el servidor local de Logging and Bookeping. El *Computing Element* es el responsable de aceptar trabajos y distribuirlos para su ejecución en los WN asociados
- *WORKER NODES (WN):* Cada uno de los nodos de computación asociados a un Computing Element, que ejecutan los trabajos que reciben de los mismos. Disponen de de los comandos y las APIs necesarias para que las aplicaciones realicen las acciones adecuadas sobre los recursos y datos.
- *STORAGE ELEMENT(SE):* Provee acceso uniforme y servicios para acceder a las datos, normalmente grandes volúmenes. Puede dar acceso a *arrays* de disco, sistemas de almacenamiento masivos (Mass Storage systems o MSS) como d-cache o Castor. Se da acceso a los datos a a través de varios protocolos, entre los que encontramos GSIFTP como método por defecto para el acceso seguro y eficiente a los datos. Para acceso remoto y no sólo copia se puede utilizar el protocolo RFIO (Remote File Input/Output procotol [54]). Se está desarrollando y testeado una interfaz común para acceso y gestión de los ficheros con todol los protocolos, denominada *Storage Resource Manager (SRM)*.

Servicios de Información:

- INFORMATION INDEX(II) / BDII: Representa el punto de entrada para conocer la información disponible sobre los recursos del Grid y su estado. Se adopta el MDS de globus y su implementación a través de LDAP. En la figura se muestras los principales elementos y como la información se propaga.

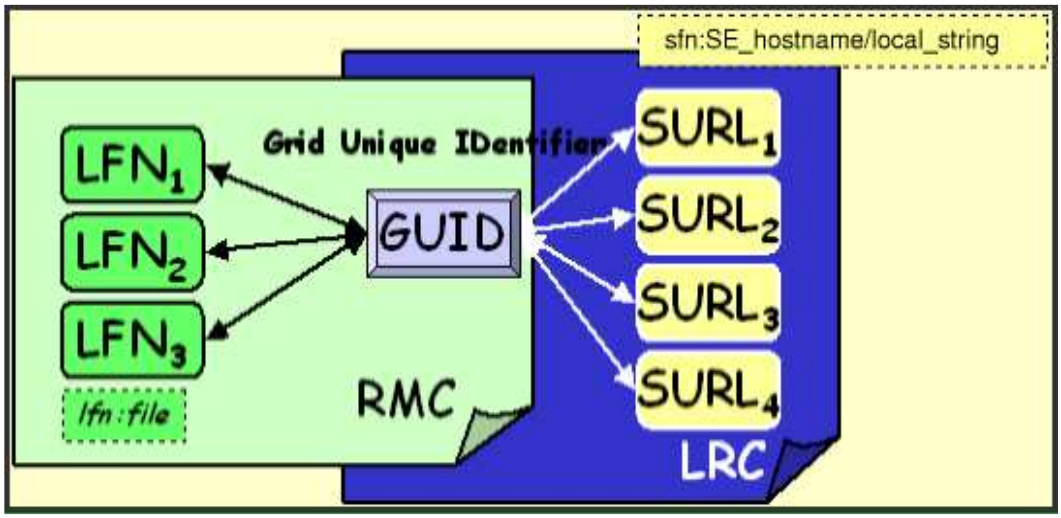


En la actualidad se utiliza un servidor BDII en lugar de un top GIIS global de MDS2.X debido a que este sufre problemas cuando un GIIS regional o un GRIS no responde correctamente. La interfaz externa sigue siendo la de LDAP propuesta por MDS2.X pero es mucho más estable
El BDII pregunta a cada GIIS que tiene configurado y actúa como cache de almacenamiento de los datos.

Gestión de Datos:

En un entorno grid los ficheros de datos están replicados, posiblemente de forma temporal, a muchos sitios diferentes dependiendo de donde los datos son necesitados. Los usuarios o aplicaciones no necesitan conocer dónde están estos datos, sino que se usan nombre lógicos y los servicios de gestión de datos son los encargados de localizar y acceder a estos datos.
Identificadores de ficheros: *Grid Unique Identifier (GUID), Logical File Name (LFN), Storage URL (SURL), Transport URL (TURL)*. GUIDs o LFNs refieren a ficheros y no réplicas, con lo

que no dicen nada sobre localizaciónes, mientras que SURLs y TURLs dan información sobre donde una réplica física está localizada.
Un fichero siempre puede ser localizado por su GUID, asignado cuando se registran los datos y basado en el UUID estandar para garantizar ids únicas. Tiene la forma *guid:<unique_string>*.
Todas las réplicas de un mismo fichero compartes el mismo GUID.
Sin embargo para localizar ficheros se suelen usar los LFNs ya que son cadenas legibles y más intuitivas, de la forma *lfn:<any_alias>*.
El SURL es usado por los sistemas de gestión de datos para encontrar donde una réplica está físicamente almacenada, y por el SE para localizarla. Son de la forma *sfn:<SE_hostname>/<local_string>,* where *<local_string>* es utilizada internamente por el SE para localizar el fichero.
Finalmente el TURL da la información necesaria para acceder una réplica física, incluyendo hostname, path, protocolo y puerto para que la aplicación final pueda acceder a el. La siguiente figura de [55] muestra las relaciones entre los distintos ficheros:



- REPLICA LOCATION SERVICE (RLS): mantiene información sobre la localización física de las réplicas (el mapeo con los GUIDs). Está compuesto de varios *Local Replica Catalogs* (LRCs) que mantienen la información de las réplicas para una única VO.
- REPLICA METADATA CATALOG (RMC): almacena el mapeo entre los GUIDs y sus alias (LFNs) asociados, y mantiene otros metadatos como el tamaño, fechas, propietarios, etcétera)
- REPLICA MANAGER (RM): presenta una interfaz única para el uso por otros servicios, e interactúa con el RLS y el RMS.

Gestión de Recursos:

- *RESOURCE-BROKER (RB):* Recibe los trabajos desde cualquier interfaz de usuario a través de un API definido, y gestiona los recursos grid realizando los pasos necesarios para la ejecución de los mismos. Interacciona con los servicios de Información y los de Gestión de datos para realizar una planificación eficiente. También con el LB para mantener información accesible sobre los trabajos Esta entidad es la que provee nuestro **CROSSBROKER** (descrito en el siguiente apartado 6.2) a través de distintos módulos, dando soporte para trabajos paralelos e interactivos además de trabajos simples *batch*.
- *LOGGING AND BOOKEPING (LB):* mantiene información histórica sobre los trabajos de cada usuario, disponible de manera persistente y con acceso restringido y seguro. El componente Servidor mantiene estos datos y existe una interfaz cliente para actualizarlos y también preguntar por ellos. Parte cliente que actualiza el estado de los trabajos residirá en el RB, CE, y WN. El cliente del UI adicionalmente es usuado para mostrar información al usuario sobre los trabajos.

6.2.Arquitectura del CrossBroker

El Gestor de Trabajos y recursos CrossBroker será encargado de realizar todas las tareas asociadas a la planificación de trabajos descritos en el punto anterior, de manera que se intentará planificar los trabajos lo más eficientemente posible, de manera que se llegue a un correcto balance entre la ejecución de la aplicación y del uso de los recursos.

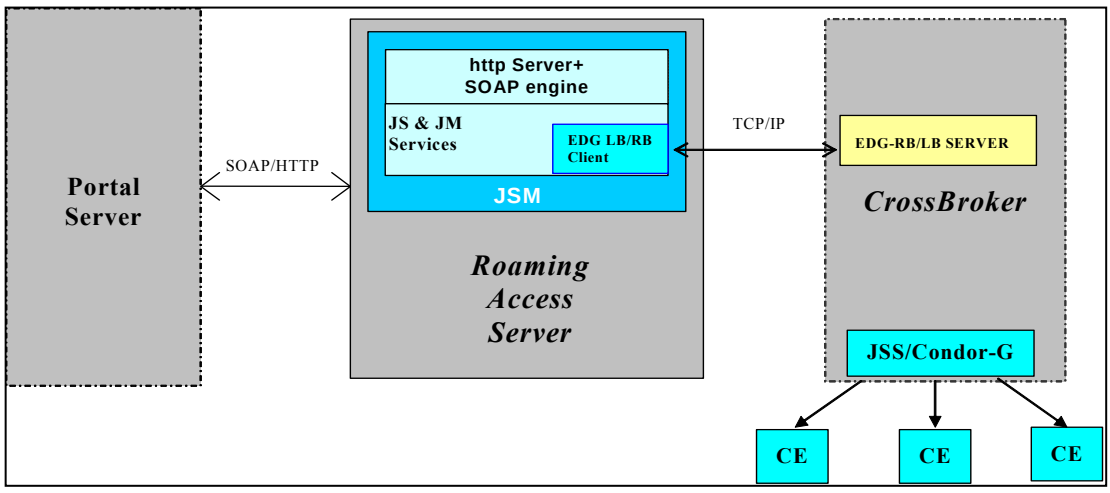
La arquitectura del CrossBroker tiene mucho que ver con cómo las Organizaciones Virtuales están definidas. Normalmente las VOs representan a comunidades que comparten recursos entre sus usuarios y por lo tanto lo más normal es que las decisiones de planificación puedan tomar ventaja de esta situación, por lo que nuestro broker esta orientado hacia un entorno multiusuario.

Realmente la arquitectura es suficientemente versátil para adoptar diferentes aproximaciones. La primera aproximación es una arquitectura totalmente centralizada en la que exista un único CrossBroker para todas las Vos y los usuarios de estas. De esta manera la información que podemos obtener es la mayor posible para los trabajos enviados por los diferentes usuarios de las Organizaciones Virtuales, y potencialmente la planificación se puede hacer más eficientemente. Esta aproximación tiene la desventaja de los sistemas centralizados ya que constituye un único punto de fallo, donde si falla esta entidad centralizada todo el sistema de gestión de trabajos falla. Además puede constituir un cuello de botella para el rendimiento, si se hacen multitud de peticiones simultáneamente.

La aproximación con la cual hemos trabajado hasta el momento es la de que existan varios CrossBrokers, para los usuarios de las distintas VOs. De esta manera seguimos teniendo mucha información disponible para la planificación, y mejoramos la distribución del sistema y el rendimiento general.

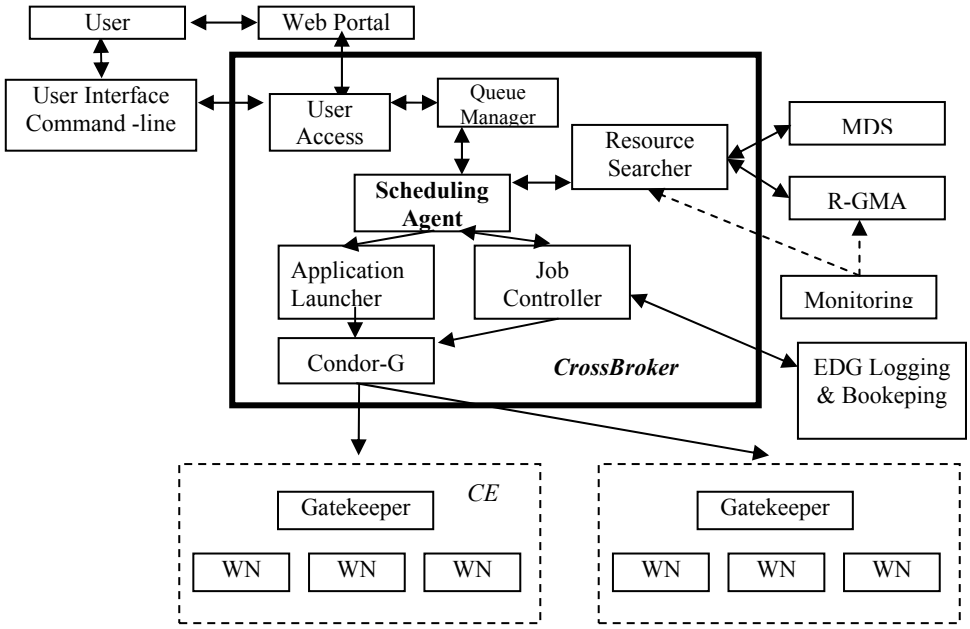
La última aproximación válida es la de que cada usuario disponga de su propio CrossBroker, de manera que el sistema está totalmente distribuido aunque de esta manera el sistema puede degenerar en la falta de "justicia" en la asignación de recursos.

El escenario típico es que el usuario mande su aplicación a través de la línea de comandos de su escritorio o de algún servicio como el Portal Web o el Migrating Desktop/Roaming Access Server [40], que presentan una interfaz de usuario más intuitiva y que también han sido desarrollados también en el proyecto Crossgrid. Todas estas interfaces de usuario se comunican a través de una interfaz común con nuestro sistema de gestión de trabajos.



La arquitectura interna se descompone en varios módulos que realizan distintas funciones, y entre las cuales podemos distinguir:

- User Access Module
- Queue Manager
- Scheduling Agent
- Resource Searcher
- Application Launcher
- Job Controller



6.2.1. User Access Module

El módulo de acceso de usuario es el punto de comunicación con el usuario o alguna aplicación que funciona en su nombre. A través de este módulo se comunica con las otras partes del CrossBroker, especialmente el Queue Manager y el Resource Selector.
Las principales tareas de este módulo son:

- Establecer y manejar la comunicación con el usuario a partir de una interfaz definida.
- Recibir los comandos de usuario para realizar las tareas específicas.
- Procesar las peticiones recibidas, reconociendo los comandos y comprobando su corrección.
- Pasar las peticiones que necesiten por procesado a los diferentes componentes
- Enviar las respuestas a la interfaz de comandos

Los comandos que se aceptan son los siguientes:

Job-submit: para enviar un trabajo
Job-list-match: recibir una lista de recursos disponibles de acuerdo con las descripciones provistas
Job-cancel: parar la ejecución de un trabajo
Job-get-output: recibe la salida de los ficheros de un trabajo ejecutado. Para los trabajos interactivos no hace falta ya que la salida es obtenida directamente
Job-get-status: comprueba el estado de un trabajo enviado, con información asociada.

El componente establece un thread para cada petición, realizando las acciones adecuadas. El primer paso común es establecer el mecanismo de seguridad a través de los mecanismos de *globus* y los protocolos GSIs. La interfaz es multiusuario, con lo que tiene que tener el cuenta el procesado y almacenado de los *proxies* con los cuales el usuario es identificado. Estos proxies se utilizarán luego por otros módulos en el caso de ser necesitados, como el envío de trabajos a los Computing Elements adecuados.

Un usuario sólo debe poder actuar sobre sus trabajos, una vez que han sido enviados. Obtener información sobre los mismos, sus ficheros de salida, o la cancelación deben seguir los protocolos de autenticación y seguridad adecuados.

En algunos comandos el mismo thread realiza el trabajo adecuado. Por ejemplo, para el *Job-list-match* se contacta directamente con el Resource Selector para comprobar qué recursos están disponibles y devolver una lista con los mismos. El usuario recibe la respuesta al poco tiempo después de lanzar el comando, obteniendo en el caso de utilizar la linea de comandos la salida por pantalla.

Los trabajos se describen utilizando un lenguaje específico, el Job Description Language JDL que se describe en el punto siguiente. Con ello se describen los requerimientos del mismo, que son utilizados internamente por los diferentes módulos.
Para el comando *Job-submit*, se envía esta descripción del trabajo y el *User Access Module* pasa esta petición al módulo de gestión de la cola, el Queue Manager que se encarga de mantener las peticiones pendientes. Una vez que el thread encola el trabajo, el usuario recibe un identificador del mismo, el cual será válido para realizar posteriores acciones como comprobar el estado, obtener la salida del mismo o cancelarlo. De esta manera el comando finaliza.

El comando *Job-get-status* acepta como entrada el identificador del trabajo, y devuelve el estado actual del mismo. El trabajo puede pasar por varios estados que serán descritos posteriormente, y este comando devuelve el estado actual del mismo.
Una variante es el comando *Job-get-logging-info*, que permite obtener la historia del trabajo, obteniendo todos los estados y el momento en el cual se ha producido el cambio. Este comando internamente contacta con el servicio de *Logging & Bookeping* desarrollado por Datagrid, que mantiene esta historia de los trabajos que son guardados para cada usuario en una base de datos interna.

El comando *Job-cancel* utiliza también el identificador del trabajo y realizará las acciones adecuadas para cancelar un trabajo que ya ha sido enviado.

Con *Job-get-output* se puede obtener la salida del trabajo, una vez que este está en el estado de trabajo acabado. El usuario podrá obtener los ficheros que ha definido como salida del trabajo, y que han sido guardados en el CrossBroker tras la finalización del mismo.

La implementación de este módulo se realiza sobre el *Network Server* de Datagrid [41] ya que provee de la funcionalidad necesaria para el envío de trabajos simples en entorno multiusuario. Sin embargo se han realizado las mejoras necesarias para el envío de trabajos compuestos, paralelos e interactivos, con las modificaciones de la interfaz de usuario para el envío y la recepción de la información adecuada. Se ha extendido el protocolo utilizado sobre TCP/IP para la comunicación Usuario/User Access Module.
De acuerdo con los desarrollos futuros de globus esta interfaz puede cambiar debido a que los servicios globus que son usados para la seguridad estarán basados en la especificación OGSA, de manera que el intercambio de mensajes utilice SOAP utilizando el protocolo HTTPS.
El servicio de transferencia de ficheros es realizado por los servicios GridFTP que permiten la transferencia segura y autenticada por los mismos servicios globus.

6.2.2. Descripción de trabajos

Los trabajos necesitan ser descritos por parte del usuario para conocer la naturaleza y tipo de los mismos, así como los requerimientos que pueden tener para su ejecución, y las preferencias para la ejecución en los recursos del *Grid.*
Por ello se necesita un lenguaje de descripción de los trabajos que permita describir las características de los trabajos, que cumpla con los requerimientos necesarios como son:

- Un lenguaje concreto y fácil de utilizar por el usuario final, que será el que lo escriba aunque se pueden dar herramientas gráficas que lo generen.
- Que permita expresar las necesidades concretas de las aplicaciones múltiples, paralelas e interactivas que son en las que nos focalizamos.
- Con un modelo de datos semi-estructurado, en el que no sea requerido un esquema específico
- Simétrico, con el que se puedan representar en el mismo lenguaje las distintas entidades del *Grid*, en particular los trabajos y los recursos (Computing Element, Storage Element). Esto hará posible realizar comparaciones entre trabajos y recursos para hacer emparejamientos fácilmente.

Estos requerimientos son satisfechos por el *Job Description Language (JDL) [44]* , que de hecho está basado en los ClassAds del proyecto Condor [46] .
Un classad es un conjunto de expresiones con nombre llamadas atributos. Una expresión contiene literales y referencias a otros atributos, compuestas con operadores. Además los classads pueden anidarse, por lo que por ejemplo un mismo classad puede ser una expresión, permitiendo la representación de conjuntos de recursos o trabajos.

Este JDL define algunos de los nombres de atributos que son predefinidos y tienen un significado especial, por ejemplo:

Executable define el nombre del ejecutable que será enviado y ejecutado en el *Worker Node*
Arguments una cadena conteniendo los argumentos en linea de comandos pasados al ejecutable (opcional)
StdInput fichero de entrada estándar (opcional)
StdErr fichero de error estándar (opcional)
InputSandbox lista de ficheros disponibles de forma local, que serán transferidos antes de la ejecución del ejecutable. Debe ser una lista de ficheros pequeños, ya que son enviados al broker y luego al worker node
OutputSandbox lo mismo para los ficheros de salida que son producidos por el ejecutable. Serán transferidos al broker al finalizar el trabajo, para después ser recogidos por el usuario.
InputData lista de ficheros de entrada normalmente expresados de forma lógica. Serán transferidos desde un Storage Element directamente al Worker Node donde se ejecute.
ReplicaCatalog Replica Catalog (proveniente de Datagrid [43]) que se utilizará para transformar los nombres lógicos de los ficheros en nombres físicos.
DataAccessProtocol lista de protocolos que el trabajo es capaz de hablar para acceder a los ficheros remotos
OutputSE Storage Element donde el trabajo almacenará los ficheros de salida
Requirements Expresión representando un conjunto de requerimientos de la aplicación. Será Utilizado por el Resource Searcher para la búsqueda de recursos
Rank Expresión representando las preferencias del usuario para la selección de recursos.

Específicamente en el CrossBroker se utilizan otros campos para definir el tipo de trabajo, que puede ser a parte de secuencial; paralelo de tipo intra-cluster(“mpich”) o Inter.-cluster (“mpich-g2”), e interactivo (“interactive”).

JobType especifica el tipo de trabajo, que puede ser “normal” (por defecto), “mpich”, “mpich-g2”. Adicionalmente se añadirá otro tipo para definir que un trabajo es “interactive”.
NodeNumber numeral que define el número de procesadores necesarios

A continuación se muestra un ejemplo de la definición de un trabajo usando JDL:

```
[
Executable = “simula_mpi”;
JobType = {“mpich-g2”};
NodeNumber = 70;
Arguments = “-d –n 100”;
StdInput = “simula_mpi.config”;
StdOutput = “simula_mpi.out”;
StdError = “simula_mpi.err”;
InputSandbox = {“/home/alferca/simula_mpi.config”,
“/usr/local/bin/simula_mpi”};
OutputSandbox = {“simula_mpi.out”, “simula_mpi.err”, “core”};
InputData = “LF:test367-2”;
ReplicaCatalog = “ldap://pcrc.cern.ch:2010/rc=ReplicaCatalog,
dc=pcrc, dc=cern, dc=ch”
DataAccessProtocol = {“file”, “gridftp”};
OutputSE = “loki03.ific.uv.es”;
Requirements = other.GlueHostArchitecture == “INTEL” &&
other. GlueHostOperatingSystemName == “linux”;
Rank = other.GlueHostBenchmarkSI00;
]
```

Este trabajo tendría las siguientes características:

- Es un trabajo paralelo "mpich-g2", que puede correr intra-cluster o inter-cluster. Se necesitan 70 cpus
- El nombre del ejecutable es "simula_mpi" y al ejecutarlo se le pasarán los argumentos "-d –n 100"
- Los ficheros `/usr/local/bin/simula_mpi` (el ejecutable mismo) y el fichero de configuración `/home/alferca/simula_mpi.config` , residen en la máquina desde donde se envía el trabajo y necesitan ser transferidos al Computing Element (finalmente al Worker Node) como parte del *input sandbox.*
- La salida Standard y de error son simula.out y simula.err. Son producidos en el Worker Node donde se ejecutará el trabajo y necesitan trasmitirse a la máquina desde donde se envía al acabar el trabajo, por lo que son incluidas en el *output sandbox*.
- En el *output sandbox* también se incluye un "core" por si acaso fuera producido
- Se necesita un fichero de datos cuyo nombre lógico es `test367-2.`
- Este nombre lógico puede resolverse usando el Replica catalog que se incluye, en una máquina llamada pcrc.cern.ch
- Es trabajo es capar de acceder ficheros a través de los protocolos gridftp o file.
- El trabajo pondrá la salida que produzca en el Storage Element loki03.ific.uv.es
- El trabajo requiere correr en máquinas con arquitectura INTEL corriendo sistema operativo linux.
- Se prefiere correr en el Computing Element que tenga mayor valor en el test SpecInt2000, que será publicado por cada CE a través del Sistema de información.

Como se puede observar, las referencias en los campos *Requirements* y *Rank* a los otros recursos (Computing Elements) se hace a través del campo **other**. Los atributos a los que se pueden referir son publicados por cada Computing Element a través de su *information provider (ver 4.3.1).* Se ha de notar que estos atributos siguen un esquema especifico denominado Glue Schema desarrollado por el consorcio DataTag([51]) y aceptado por globus como esquema de información ([52]) y que el infoprovider ha sido modificado para publicar en este esquema.

Esta especificación es enviada al Crossbroker para comunicar la información y que realice los pasos necesarios hasta la correcta ejecución del trabajo.

6.2.3. Queue Manager

Este módulo almacena los trabajos que le llegan al CrossBroker a través del *User Access Module*, de manera que son accesibles por el resto de componentes, en particular el *Scheduling Agent* que se encargará de gestionar los pasos adecuados hasta la ejecución del trabajo o su cancelación si finalmente no se ha podido realizar.

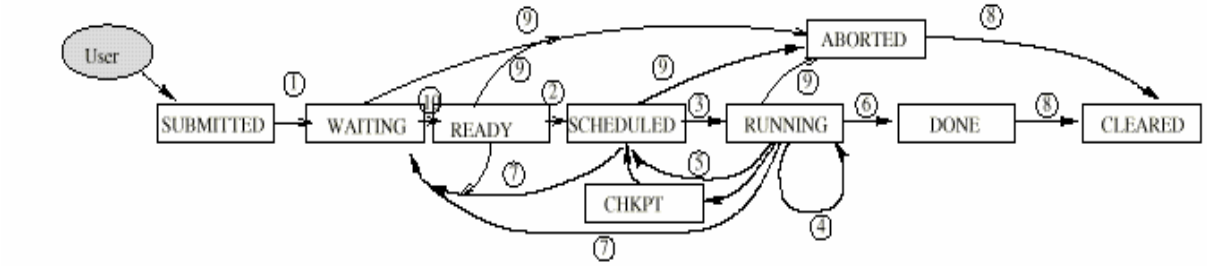
Este módulo utiliza una base de datos permanente de trabajos, que los almacena mientras estén en el CrossBroker. De esta manera se mantienen los trabajos aunque falle el CrossBroker o alguno de sus servicios, de manera que es una cola permanente en disco.

Se tiene una serie de funciones para encolar trabajos, sacar los trabajos de la cola por parte de los otros módulos, ordenar los trabajos con respecto a su prioridad, etcétera.

Durante la vida de un trabajo, éste puede estar en uno de los siguiente estados:

SUBMITTED:	Un usuario ha enviado el trabajo
WAITING:	Se ha recibido el trabajo pero no tiene aún recursos asignados donde correr
READY:	Se he realizado el proceso de matching y seleccionado los recursos adecuados
SCHEDULED:	El Computing Element ha recibido el trabajo
RUNNING:	El trabajo está corriendo en el Computing Element
DONE:	La ejecución del trabajo se ha completado
CLEARED:	El Usuario ha recibido todos los *output files* al finalizar el trabajo
ABORTED:	El job ha sido abortado por el sistema

CANCELLED: El trabajo ha sido cancelado por el usuario



Para los trabajos paralelos e interactivos los estados son similares a los de un trabajo batch de Datagrid. El cambio de los estados se produce de la siguiente manera:

1. Un trabajo es enviado desde el UI al broker
2. Una petición de envío del trabajo al CE seleccionado
3. Un trabajo ha sido enviado al CE
4. El trabajo ha sido afectado por un evento causado por el LRMS del CE (por ejemplo ha sido suspendido o su prioridad ha sido bajada)
5. El trabajo ha sido parado y encolado de nuevo en el CE
6. El trabajo se ha completado
7. El trabajo ha pasado de nuevo al broker
8. Se ha obtenido la salida del trabajo y borrado la información relativa al mismo
9. El trabajo ha terminado
10. Un CE apropiado ha sido encontrado donde el trabajo puede correr

6.2.4. Descripción de Recursos

La descripción de los recursos debe ser coherente con la de los trabajos para que se puedan relacionar fácilmente, y hacer "matching" de los trabajos mandados con los recursos disponibles en cada momento.

Por ello se utiliza el Glue Schema ([51]) que ya hemos comentado en el punto de descripción de trabajos. Con este esquema es posible en la actualidad modelar los recursos computacionales (CE), los de almacenamiento (SE), y se está desarrollando en la actualidad el modelado de los recursos de red.

Los campos de este esquema son demasiados para comentarlos aquí pero resultan adecuados para expresar toda la información que necesitamos de los recursos, como por ejemplo tanto los atributos principalmente estáticos (número de procesadores total, arquitectura del sistema, sistema operativo y software disponible, LRMS instalado, colas de trabajos, etcétera); así como los atributos más dinámicos (carga del sistema, número de procesadores disponible, tiempo de respuesta, trabajos corriendo y en espera), y muchos más.

Este esquema modela los recursos, de manera que proveen esta información de un modo coherente. Cada recurso deberá publicar sus datos en el Sistema de Información mediante un su *information provider (ver 4.3.1)* adecuado. Para los Computing Elements y Storage elements actualmente se usa el de globus para MDS 2.X conforme a este esquema y genera la información basándose en los datos que recoge del sistema y los proporcinados opcionalmente por la herramienta Ganglia [53].

Esta información es publicada a través del sistema de información, que actualmente es MDS 2.X. Además existen los esquemas adaptados para utilizarlos con otros sistemas de información como R-GMA [47] pero no los hemos utilizado concienzudamente ya que no se ha probado todavía de manera estable y en producción.

Internamente el CrossBroker utilizará esta información convirtiéndola a classads, de manera que el matching es directo utilizando la librería de Condor classads cuando buscamos *matching* de 1 trabajo contra 1 recurso. Esto se utiliza por parte del módulo del Selector de Recursos (como veremos en el punto 6.2.6), y mediante unas extensiones podremos utilizar set-matching de 1 trabajo contra varios recursos para cuando necesitemos esta funcionalidad.

6.2.5. Scheduling Agent

El *Scheduling Agent* es el responsable de la planificación de los trabajos y de decidir dónde se van a ejecutar éstos. Para localizar cuál es el mejor recurso primero llamará al *Resource Searcher* para obtener la lista de candidatos dónde enviar el trabajo. Esta lista de recursos dependerá del tipo de trabajo enviado, de sus requerimientos y del estado actual de los mismos dentro del *testbed.*

Esta lista puede estar ordenada por las preferencias de selección que el usuario indica a través del atributo *Rank* del JDL, por lo que el usuario incluye su propia valoración de cuáles son los mejores recursos en los que el trabajo puede ejecutarse.
Adicionalmente el SA incluye sus propias políticas de selección de recursos para hacer el envío de trabajos efectivo, y hacer un balance correcto entre lo que es mejor para la aplicación y lo mejor para el conjunto de recursos del *testbed*.

Para los trabajos *batch* el RS ya aplica unas políticas determinadas dependiendo de los requerimientos de datos que tenga el trabajo heredadas de datagrid. Básicamente se seleccionaran primero aquellos CEs que dispongan del mayor número de ficheros requeridos de manera "local", de manera que también se puede calcular el rango estimando los tiempos de acceso a éstos mediante funciones externas (consultar el punto 6.2.6.2).

Políticas adicionales son empleadas para no enviar el trabajo a colas del CE que van a estar exhaustas enviando una ráfaga de trabajos, ya que las colas son configuradas con un número de "slots" determinados. Siempre que es posible se enviará el trabajo donde haya menos trabajos encolados.

Para los trabajos paralelos mpi además buscaremos CEs que tengan al menos el número de cpus libres que se están pidiendo. Para trabajos paralelos mpich-g2 que se pueden ejecutar en varios CEs se utiliza el siguiente criterio:

- Grupos (sets) de CEs con menor número de diferentes CEs serán seleccionados primero. Esta política tiende a correr MPI en un único cluster, lo cual elimina largas latencias en los mensajes enviados entre distintos tareas corriendo en clusters diferentes
- Si hay más de un grupo con el mismo número de CEs, el grupo que tiene mejor rango global será seleccionado primero. Los rangos son asignados a cada CE dependiendo de ciertas métricas de rendimiento como sus MFLOPS, memoria, etcétera.

Para las variantes interactivas de estos trabajos se tiene en cuenta de manera automática el estado de las colas de los CEs. Normalmente cada CE puede tener una o más colas aceptando trabajos, donde el broker los enviará hasta que se ejecuten. Estas colas son las del gestor de recursos local de cada sitio (PBS, LSF) que ya aplicará sus propias políticas de ejecución determinadas por el administrador local. Mientras que esto es válido para trabajos no interactivos batch y paralelos. En esta situación un trabajo interactivo que se enviara a estas colas podría permanecer esperando a ejecutarse bastante tiempo antes de que el usuario percibiera respuesta, aunque una vez ejecutado la interactividad con el usuario y el control puede ser directo
Esta no es una aproximación totalmente válida para los trabajos interactivos ya que el usuario pretende tener una respuesta inmediata al enviar el trabajo, por lo que el SA intenta aplicar unas políticas específicas:

- Para trabajos paralelos interactivos comprobar que se tienen las cpus libres en estos mometos, y requerir sólo estas para que no se encolen los trabajos.

- Mandar los trabajos interactivos a CEs cuyas colas están vacías, por lo que se asegura que va a ser el primer trabajo a ejecutar y se va a obtener la respuesta más rápida posible del LRMS local.
- Aplicar técnicas de prioritización de trabajos, dando a los interactivos la mayor prioridad posible. Si no hay cpus libres disponibles se utilizarán técnicas para bajar la prioridad de los trabajos ejecutandose y mandar el trabajo interactivo a estos recursos con mayor prioridad. Esto se consigue a través de métodos de preem-cion de trabajos mediante el mecanismo Glide-In de condor, pero no lo explicaremos aquí de forma extensiva.

Usar estas técnicas de prioritización es a priori complicado en un marco general ya que no existe soporte especifico por parte de los LRMS locales, ya que una de los conceptos básicos del *grid* es que no se tiene control sobre los recursos sino que están bajo el control de sus respectivos dominios administrativos. Por lo tanto cada CE y su LRMS son configurados de manera con las políticas locales que pueden no ser por ejemplo las de tener prioridad para los trabajos interactivos.

Sin embargo podemos utilizar la prioridad entre trabajos de un mismo usuario para ejecutar los interactivos de manera eficiente. Para ello podemos utilizar técnicas que nos permitan utilizar el "sitio" de un trabajo no interactivo que ya se está ejecutando para permitir la ejecución de otro interactivo en su lugar de manera más prioritaria.

Una vez que el SA decide cuál es el mejor recurso (o conjunto de ellos) en los que ejecutar el trabajo, pasa esta selección al *Aplication Launcher,* que será el encargado de enviar de forma efectiva el trabajo a estos recursos. La gestión del trabajo a partir de este momento es llevada a cabo por otros módulos, pero si por alguna razón el envío fallara, el SA seleccionaría el siguiente recurso (o conjunto de ellos) de la lista generada para volver a re-enviar el trabajo. El número de reenvios puede ser configurable de manera que tras un número de intentos determinados se pueda decidir que el trabajo ha sido o no ejecutado satisfactoriamente.

6.2.5.1. Uso de información externa

Para conocer los recursos disponibles y su estado se utiliza el sistema de información. Como hemos visto actualmente se utiliza el MDS y su implementación en LDAP. La publicación de esta información sigue en la actualidad el Glue Schema, que puede ser bastante restrictivo en algunos casos y no presentar toda las posibilidades de los recursos. Una consecuencia de que sea restrictivo es que no se publica información que es muy útil para la localización de recursos adecuados, como por ejemplo el ancho de banda o la latencia de comunicaciones entre dos Computing Elements determinados. Esta información puede ser actualmente accesible a través de algunas herramientas de monitorización de Crossgrid [56] con lo que se necesita un método para accederla desde nuestro broker para que sea disponible al mismo y al usuario final.
Se ha desarrollado un método para acceder a esta información de una manera general, a través de plugins programables a través de un API definida [57]. A través de esta interfaz se hace disponible para el broker la información de las herramientas de monitorización, y además es accesible para el usuario final a través de la descripción del trabajo en JDL.
El usuario puede expresar sus preferencias a través del campo *Rank*, llamando a las funciones provistas por los plugins. La especificación final de estas funciones y los parámetros que aceptad y metricas usadas son provistos por los desarrolladores del plugin. En la actualidad se ha implementado el plugin para el acceso a la herramienta de postprocessing [57].
En este caso por ejemplo podemos acceder a información de esta herramienta como los "idle_bogomips" que no está accesible a través de los sistemas de información. Esto se lograría por ejemplo con una expresión como la siguiente:

```
Rank = other. GlueCEStateFreeCPUs *
ppt_getClusterParameter("max","idle_bogomips",other.GlueCeUniqueId,0)
```

Esta expresión permite que los Ces sean ordenado por un valor calculado como el número de Cpus libres multiplicado por el valor de "Idle bogomips" de cada CE. Los parámetros de la función implementada ppt_getClusterParameter tendría en cuenta el CE a evaluar a través del parámetro GlueCeUniqueId. El resto de parámetros son dependientes del plugin, pero básicamente señalan la función, el atributo a buscar, y en último lugar el tiempo de predicción para los datos (0 es el dato actual).

Información muy útil para hacer set-matching de recursos entre varios Computing Elements (para trabajos mpich-g2 por ejemplo, ver 6.2.6.3), puede ser el ancho de banda entre éstos. También disponible a través de este plugin, que podria obtener los valores de la siguiente manera:

Rank = ppt_getNetworkParameter("avg","bandwidth",other.groupGlueCeUniqueId,0);

De esta manera se obtendría un valor del ancho de banda agregado entre los grupos de CEs considerados en cada momento. Para ello introducimos un nuevo atributo que sirve para señalar estos grupos, *groupGlueCeUniqueId*.
Este valor se refiere a la lista de CEs que es evaluada por el Resource Selector en cada momento al construir los grupos válidos, e internamente la pasará al plugin como tal (una lista de la forma { "CE1", "CE", ..., "CEn"}.

6.2.5.2. Servicio de reservas

Debido a la propia naturaleza de los Servicios de Información, se puede acceder a la misma a través de dos métodos. Preguntando al GIIS o BDII de manera que es muy eficiente el acceso, peropor contrapartida se obtiene información actualizada sólo cada cierto periodo que puede ser razonablemente largo (unos minutos). Es el acceso adecuado para descubrir recursos y sobre sus atributos estáticos, que no cambian constantemente.

Sin embargo para conocer el estado más actual de cada recurso, el procedimiento más adecuado consiste en preguntar directamente a su GRIS, encargado de publicar los datos directamente y comunicarlos a su GIIS. Con este método podemos acceder a la información más actual, pero debido a la implementación esta información no es actualizada de modo atómico y puede haber situaciones en las que no exista una sincronía total entre la información publicada y la real. Esta información puede ser bastante crítica en la selección de recursos, como por ejemplo la carga del sistema o las cpus utilizadas, ya que podemos tener una situación en la que creamos que tenemos unos recursos libres cuando realmente han sido ocupados en un instante inmediatamente posterior a la consulta.

Para hacer frente a estas situaciones el broker mantiene un sistema de reservas de los recursos, utilizado internamente para evitar las inconsistencias que se puedan dar. Por ejemplo no enviar multitud de trabajos a un determinado *site* que sigue publicando cpus libres en el sistema de información aunque ya se han enviado un grupo de trabajos. Éstos pueden estar en varios lugares independientemente, desde la cola del LRMS del CE hasta realmente en ejecución ejecución, pero en una situación en la que el GRIS todavía no la ha reflejado.

Este módulo contiene la información más reciente accesible de los GRIS, así como aquella que se conoce internamente para el broker como los trabajos que han sido ya enviados al mismo. Esta información también se actualiza cada vez que hay información nueva disponible desde el recurso, sincronizándose. De esta manera podemos tener un control para evitar las situaciones descritas anteriormente. Sin embargo no podemos evitar las situaciones en las que se evite el lanzar los trabajos a través de nuestro broker y se envien directamente trabajos a los recursos mediante los comandos globus básicos, situación que es perfectamente válida. En estas situaciones deberemos contar con la información disponible y hacer frente a las posibles inconsistencias mediante otros métodos, de tolerancia a fallos de ejecución como veremos en los siguientes puntos.

6.2.6. Resource Searcher

La tarea principal del *Resource Searcher* (buscador de recursos) es la de realizar los pasos necesarios para la búsqueda de los recursos que se necesitan para cada trabajo. Este módulo recibe como entrada la descripción de un trabajo y produce como salida una lista de posibles recursos en los que ejecutar el trabajo. Se realiza un emparejamiento, en lo que se denomina *matchmaking,* entre los trabajos y los recursos de manera que se satisfacen los requerimientos de los mismos.

El trabajo ha sido especificado por el usuario por medio del lenguaje JDL (*Job Description Language)* tal como se describe en el punto 6.2.2. Este lenguaje está basado en los classads de Condor [46] , con lo cual se puede realizar una transformación directa para proporcionarla como entrada al RS.
Los recursos también están caracterizados por una descripción como un classad. En el punto 6.2.4 se demuestra cómo se hace la especificación de los recursos y su conversión a éstos.

El proceso de emparejamiento básico es realizado de manera que se puede utilizar la funcionalidad de *matchmaking [45]* de la misma librería de condor Classads. Mediante esta librería, tanto trabajos como recursos son especificados mediante *classads*, de manera que dos de estos classads evalúan a cierto (son compatibles o hacen pareja) si los requerimientos de uno se evalúan a cierto en el contexto del otro ClassAd. De esta manera se pueden establecer requerimientos de los trabajos a satisfacer por los recursos, pero también requerimientos de los recursos a satisfacer por los trabajos.

El lenguaje de ClassAds y la librería de matchmaking fue diseñada para seleccionar a máquinas en las cuales correr trabajos, y por lo tanto permite hacer emparejamientos 1 a 1, es decir asignar un trabajo a un recurso. En nuestros requerimientos figuran la ejecución de trabajos paralelos en varias máquinas con lo cual no es posible utilizar directamente estos métodos, y hemos añadido varias extensiones para ser utilizadas cuando un trabajo requiere múltiples recursos computacionales (CE o *Computing Elements*). Estas extensiones son llevadas a cabo dentro del algoritmo de *set-matching 6.2.6.3*

Los recursos son principalmente los *Computing Elements* a los que finalmente se enviará el trabajo, pero también se consideran recursos de almacenamiento (SE o *Storage Elements*) de forma secundaria. No se van a emparejar únicamente trabajos con SEs, sino con CEs que requieran SEs (por ejemplo cercanos o con mayor ancho de banda con respecto a este CE).

6.2.6.1. Algoritmo de matchmaking

La base de este algoritmo proviene del Matchmaker del proyecto Datagrid. El algoritmo realiza en la práctica debe realizar los pasos 1 al 4 descritos en el capítulo 3.4 (Etapas de la planificación sobre Grid), dividiéndolo en dos fases diferentes: la fase de comprobación de requerimientos y la fase de cómputo del ranking.

En la primera fase de *comprobación de requerimientos*, primero se hace un descubrimiento de los recursos que son potencialmente válidos para su utilización. Para ello se consulta al sistema de información, a través de un submódulo que se encargaría de interaccionar con el sistema de información utilizado, actualmente MDS, pero que se puede adaptar para trabajar por ejemplo con R-GMA [47].

Durante este primer paso se contacta a GIIS global para obtener información sobre aquellos recursos que cumplen los requerimientos básicos. El requerimiento principal es que usuario que manda el trabajo esté autorizado en esos recursos, cosa que se publica en el mismo GIIS para cada recurso. La primera aproximación fue publicar los DN (*Distinguished Name)* de los usuarios que se aceptan, pero actualmente se utiliza la VO bajo la cual el usuario está mandando el trabajo.

Una vez se han obtenido los recursos en los que potencialmente el usuario puede ejecutar, se realiza un matchmaking de los requerimientos de las aplicaciones contra los recursos. El usuario había definido los requerimientos en el campo asociado del JDL, por lo que estos requerimientos son contrastados con los de los recursos.

Debido a que los atributos que se indican en los requerimientos suelen referirse a datos estáticos (como la información del sistema operativo, el software disponible, o los rendimientos globales expresados por ejemplo como SPecInt) consultar directamente al GIIS es la mejor opción en términos de eficiencia, ya que constituye un acceso rápido a esta información en vez de preguntar directamente a cada GRIS del recurso.
En el caso de aplicaciones paralelas mpi también se utiliza este algoritmo ya que la aplicación se enviará a un CE, y una vez allí ya se enviará a tantos WNs como sea necesario dependiendo de las cpus requeridas. Para estas aplicaciones el RS comprueba que el CE al menos tiene tantas cpus como el trabajo mpi está pidiendo.

Finalmente se obtiene una lista de recursos (*suitableCEs)*, como una lista de ClassAd representando a los mismo, y que son válidos para ejecutar el trabajo.

La segunda fase trata *de computar el ranking* de estos recursos, para obtener la lista ordenada de los mismos. Esta fase puede ordenar esta lista de recursos de acuerdo a unas preferencias establecidas por el usuario a través del campo *Rank* como se indica en el punto 6.2.2.
Para llevar a cabo esta ordenación el RS contacta directamente los GIIS de los CEs obtenidos en la anterior fase, es decir aquellos que se encuentran en la lista de *suitableCE.*

Uno de los campos que había sido obtenido en la anterior fase es la dirección de contacto de cada GIIS que normalmente es un estándar que comprende el URL de este CE y un puerto por defecto, pero que puede indicarse en este campo destinado a tal efecto.

Por lo tanto el algoritmo iterará por la lista preguntando directamente a cada GIIS los atributos que necesite para calcular el rango, ya que son normalmente atributos dinámicos como por ejemplo la cantidad de memoria disponible, o las cpus libres. Para ello se utiliza la misma librería de classads para llevar a cabo esta operación de cálculo con cada uno de los CEs, teniendo como resultado una lista de classAd ordenada numéricamente de acuerdo con los atributos indicados por el usuario.
Para las aplicaciones mpi en esta fase calcula las cpus libres que tiene el CE, descartándolo de la lista si es menor de las cpus requeridas.

Adicionalmente se puede cambiar la política de ordenación de recursos, cosa que puede hacer el módulo del Scheduling Agent dependiendo de diferentes variables.

6.2.6.2. Algoritmo de matchmaking con requerimientos de datos

Tanto en el proyecto Datagrid como en el Crossgrid, se tienen aplicaciones de computación intensiva en datos. Es por lo tanto esencial tener estos en cuenta en la selección de recursos y por lo tanto si el usuario indica la necesidad de estos datos se utilizará un algoritmo diferente.
Para ello se puede utilizar la interfaz que ofrece los servicios de Replica Management del Datagrid [43] y el algoritmo original que sirve a nuestros efectos, con algunas modificaciones.
El usuario puede describir con atributos de JDL los requerimientos que tiene en cuando a los datos:

OutputSE: representa el Storage Element donde la salida de datos debe ser enviada cuando acabe el trabajo
InputData: representa los ficheros de entrada que son necesarios para la ejecución del trabajo, indicados mediante LFNs, GUIDs. (puede encontrarse más información en las referencias [43])
DataAccessProtocol: indica el protocolo que se utiliza por la aplicación para acceder a estos ficheros (gsiftp, srm, etc..)

Las dos fases de requerimientos y rango expuestas en el punto anterior son mantenidas, pero el RS las ejecuta incluyendo los requerimientos de acceso a los datos. Para ello se realiza una fase de pre-matching para encontras aquellos CEs que satisfagan los requerimientos de acceso a datos y de autorización del usuario.
Durante esta fase se contacta con el Replica Location Service de Datagrid para resolver los nombres lógicos de los ficheros (LFNs) y recolectar información sobre los SEs que contienen al menos un fichero de entrada de los que ha indicado el usuario.
Después el RS realiza la primera fase de clasificación en la que se contacta el GIIS, añadiendo a los requerimientos los ya comentados de tener el OutputSE “cerca” de el CE que se está seleccionando. Usando la información anterior, el RS clasifica estos CEs dependiendo del número de ficheros de entrada guardados en los SEs que estan cerca del CE y que se pueden comunicar en al menos un protocolo de los especificados en el campo DataAccessProtocol.

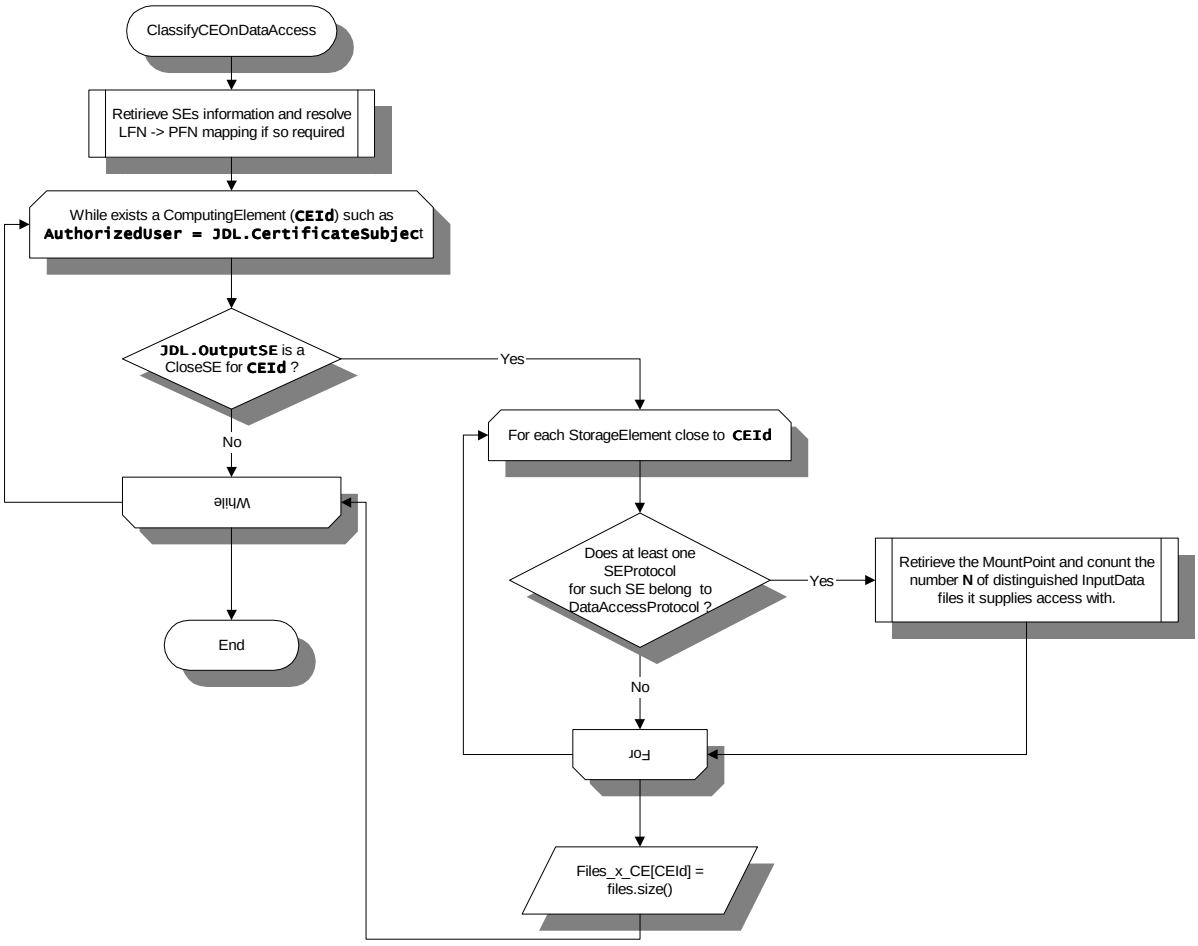
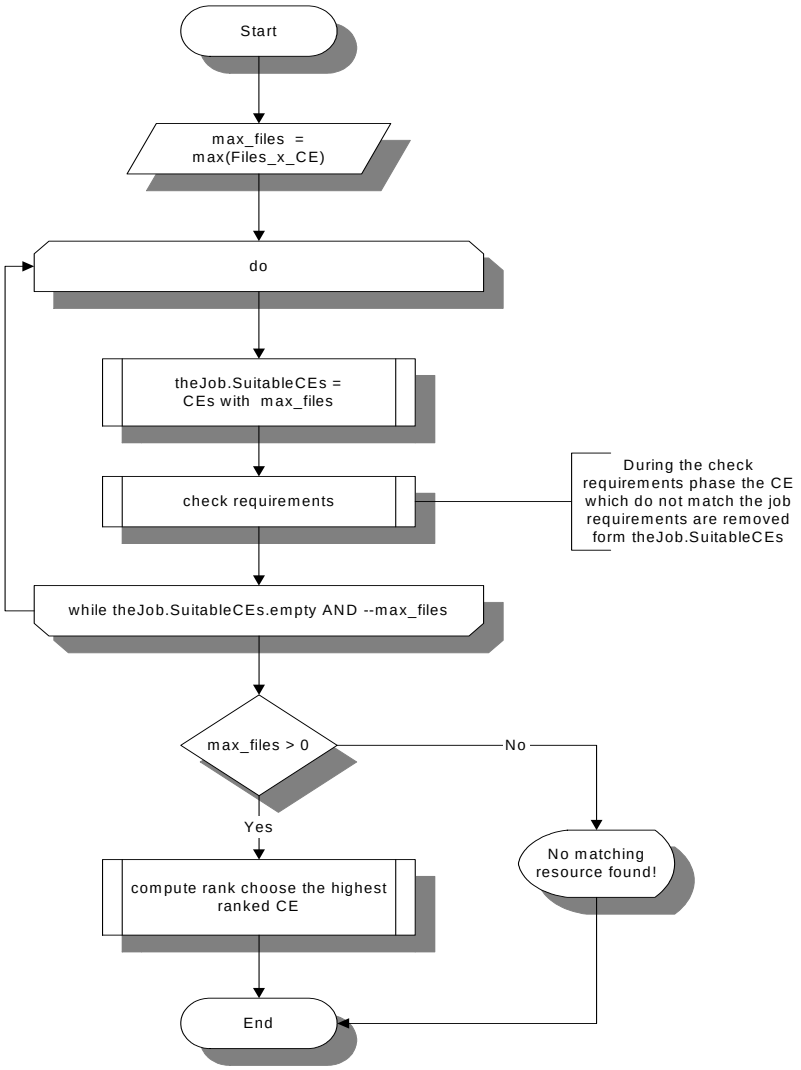


Ilustración 1

Después el RS realiza la fase real de matchmaking y comprueba los requerimientos para los CEs obtenidos en la fase de pre-match anterior, empezando por aquellos que pueden acceder al mayor número de ficheros distintos. Si un CE no satisface los requerimientos especificados por el usuario es quitado de la lista. Esta fase es realizada hasta que al menos se tiene un CE que haga matching.

Para el caso de aplicaciones paralelas mpi, el acceso a los datos se puede modelar de la misma forma, ya que aunque los procesos pueden residir en distintas cpus, todas serán WNs de un mismo CE. La aplicación accederá a los datos desde un único WN y se compartirán mediante directorios comunes o a través de la propia aplicación, por lo que el modelado para estimar el coste es el mismo.



En la última fase de ranking se realiza de forma similar al caso sin datos, aunque un caso especial es que el usuario puede especificar que desea que se ordene la lista de CEs por el coste de acceso a los datos especificados indicándolocon: Rank = other.DataAccessCost.

En este caso el CE es elegido entre aquellos que cumplen los requerimientos del usuario indicados en el JDL, y la elección del mejor CE es delegado a la función *getAccessCost* que proporciona también el WP2 de Datagrid, y que selecciona el CE cuyo coste de acceso a los datos es el menor.

6.2.6.3. Algoritmo de set-matching

Veíamos cómo el algoritmo anterior de matchmaking se utilizaba para hacer *matching* de un trabajo con un Computing Element, es decir 1 a 1. Sin embargo como hemos visto podemos querer ejecutar aplicaciones en más de un recurso distribuído, por lo cual necesitamos un método para hacer matching de un trabajo con varios CEs, es decir matching de 1 a muchos, o *set-matching*.

Ahora representamos un conjunto de recursos como un ClassAd set (un grupo de ClassAds), de manera definimos que buscaremos emparejamiento correctos entre un ClassAd (el JobAd que representa el trabajo) y este ClassAd set (representado un conjunto de CEs). Algunos atributos del JobAd se utilizan para poner restricciones individuales en cada uno de los classAds que forman parte del set (por ejemplo, el Sistema Operativo de cada CE debe ser Linux 2.4, o deben tener más de 1 GB de memoria total).
Después se utilizan otros atributos del JobAd para poner restricciones sobre el conjunto representado por el ClassAd set (por ejemplo el total de cpus libres debe ser igual o mayor que las requeridas por el JobAd).

La entrada de este algoritmo seguirá siendo un classAd que representa el trabajo (JobAd), pero ahora la salida no es una lista classAds (CeAd) representando los recursos, sino una lista de setAds que representa el conjunto de sets de recursos que son válidos para ejecutar nuestro trabajo.

Para ello la comprobación de requerimientos debe llevarse en dos pasos:

- Paso 1: comprobar los requerimientos individuales de cada CE.
- Paso 2: agrupar los CEs en classad sets y comprobar los requerimientos agregados.

El primer paso del algoritmo es similar al listado en los puntos anteriores, la fase de *comprobación de requerimientos* de los algoritmos de matchmaking 1 a 1. A diferencia del anterior de esta primera fase no obtenemos la lista de CEs que es capaz de ejecutar nuestro trabajo, sino la lista de CEs que cumplen los requerimientos que debe cumplir cada uno de ellos individualmente, y que por lo tanto son capaces de formar parte de un setAd. A esta lista la seguimos denominando *suitableCEs*.
Por lo tanto esta primera fase constituye una pre-selección de requerimientos, en la cual se sigue consultando el GIIS para buscar los CEs, y se realiza el matching 1 a 1 contra aquellos atributos que se deben cumplir por todos los CEs.

En la segunda fase se utiliza esta lista de *suitableCEs* obtenida de la fase anterior, para construir grupos de CEs (los classad set) de manera cumplan conjuntamente los requisitos colectivos. Por ejemplo, se intenta conseguir que el número total de CPUs disponibles agregado sea mayor o igual al número de cpus requeridas en el JobAd. Es en esta fase donde se realiza el set-matching, de 1 a muchos.

En la práctica esta segunda fase se realiza a la vez que la fase de *cálculo del rango*, de manera que a partir de la lista de *suitableCEs* posibles, se construirá la lista de setAds.. Para ello en esta fase se consulta el GRIS de cada CE que pertenece a esta lista obteniendo la información mas actualizada posible de cada recurso.

Se va construyendo por un proceso iterativo la lista de setAds, a través de dos estructuras de datos para cada JobAd analizado:

- *matchedClassAdSets*: un vector que contiene los setAds que ya cumplen todos los requerimientos colectivos..
- *partiallyMatchedClassAdSets:* un vector que contiene los setAds que todavía no cumplen los requisitos globales, pero que son candidatos para hacerlo añadiendo nuevos CEs a estos sets.

Empezando por el primer CE de la lista de *suitableCEs,* una vez se ha obtenido la información más actualizada se evalúa si cumple todos los requerimientos colectivos. Si esto es cierto tendremos un setAd con un único CE que ya es válido por lo cual lo incluiremos en la primera entrada del vector *matchedClassAdSets,* que corresponde con sets compuestos por un único CE. El proceso continúa con el siguiente elemento de la lista de *suitableCEs*.

Si el CE no cumple los requerimientos él solo, se deben buscar grupos de CEs que junto con éste sí que los cumplan. Para ello el algoritmo recorre el vector *partiallyMatchedClassAdSets* para ver si añadiendo este CE a cada uno de los sets almacenados en el vector puede construir un nuevo set con este CE añadido que sí que los cumpla.
Si este nuevo set cumple los requerimientos colectivos entonces es añadido a la entrada correspondiente del vector *matchedClassAdSets*, de acuerdo con el número de elementos del set (2ª entrada para los sets de 2 CEs, 3ª entrada para los de 3 CEs, etcétera).
Si este nuevo *set* no cumple los requerimientos colectivos, puede ser que todavía se le puedan añadir más CEs en sucesivas iteraciones y así los cumpla (por ejemplo, se requiere 4 GB de memoria agregada, y el set tiene en estos momentos 2 CEs con 1 GB cada uno). De esta manera se incluye el set en la entrada correspondiente del vector *partiallyMatchedClassAdSets* para tenerlo en cuenta en la siguiente iteración cuando se pase al siguiente CE de la lista de *suitableCEs*.

Como se puede observar el algoritmo no hace una búsqueda exhaustiva, ya que no se computa todos los grupos de CEs posibles. En particular, significa que el algoritmo no considera soluciones en las que un subconjunto de CEs ya ha sido incluido en un set que cumple los requisitos, por lo que es más eficiente.
Por ejemplo si tenemos 4 CEs en la lista de suitablesCEs (CE1, CE2, CE3, y CE4) puede darse una situación en la que tengamos tres soluciones posibles, es decir 3 sets que cumplen los requerimientos:

- {CE2}
- {CE1 + CE3}
- {CE1 + CE4}

Pero el algoritmo no computaría el set { CE1 + CE3 + CE4 } ya que al menos un sub-conjunto {CE1 + CE3} ya era una solución válida.

El rango es calculado en esta segunda fase para los sets válidos, de manera que se tiene en cuenta el rango individual de cada CE y se pondera por el número de cpus libres que están aportando al grupo total. En el ejemplo 6.2.6.4 puede verse esta situación.

Finalmente la salida del algoritmo proporcionará el vector *matchedClassAdSets,* que contendrá todos los conjuntos de CEs que son compatibles con los requerimientos que se habían pedido.

6.2.6.4. Ejemplo de búsqueda de recursos

En este punto se ilustra la utilización del algoritmo de en la práctica, dado unos requerimientos específicos. Por ejemplo en la siguiente figura se incluye el JDL de un trabajo paralelo mpich-g2 que necesita 10 cpus. Estas tipo de aplicaciones puede ejecutarse sobre 1 o varios CEs distintos por lo que internamente el CrossBroker seleccionará el algoritmo de *set-matching*.

Adicionalmente el usuario requiere que las colas sean del tipo PBS, y se prefieren aquellos recursos que tengan un mayor índice en la escala estándar *SpecInt* de rendimiento.

```
Executable = "mpi_app";
JobType = “mpich-g2”;
NodeNumber = 10;
Arguments = "-n";
StdOutput = "std.out";
StdError = "std.err";
Requirements = other.GlueCEInfoLRMSType=="pbs";
Rank = other.GlueHostBenchmarkSI00;
OutputSandbox  = {"std.out","std.err"};
```

Suponemos que disponemos de los siguientes CEs, cuya información general está accesible al RS a través de consultas al GIIS:

- *ce001.grid.ucy.ac.cy:2119*
- *cluster.ui.sav.sk:2119*
- *zeus24.cyf-kr.edu.pl:2119*
- *cagnode45.cs.tcd.ie:2119*
- *ce100.fzk.de:2119*
- *ce01.lip.pt:2119*
- *cg01.ific.uv.es:2119*
- *cgce.ifca.org.es:2119*

Entonces dependiendo del estado actual del testbed en el que se realice la consulta sobre los recursos disponibles, podríamos obtener la siguiente respuesta:

```
Connecting to host cg07.ific.uv.es, port 7772
*****************************************
             GROUPS OF CE IDs LIST
 The following groups of CE(s) matching your job requirements have
been found:

*Groups with 1 CEs*   *TotalCPUs* *FreeCPUs*

[Rank=650]
 ce001.grid.ucy.ac.cy:2119    10         10
[Rank=630]
 cluster.ui.sav.sk:2119       16         16
[Rank=400]
 zeus24.cyf-kr.edu.pl:2119    58         57

*Groups with 2 CEs*   *TotalCPUs* *FreeCPUs*

[Rank=440 TotalCPUs=12 FreeCPUs=12]
 cagnode45.cs.tcd.ie:2119      4          4
 ce100.fzk.de:2119             8          8
[Rank=498 TotalCPUs=10 FreeCPUs=10]
 ce01.lip.pt:2119              2          2
 ce100.fzk.de:2119             8          8

 *Groups with 4 CEs*  *TotalCPUs* *FreeCPUs*

[Rank=435.6 TotalCPUs=10 FreeCPUs=10]
 cagnode45.cs.tcd.ie:2119       4          4
 ce01.lip.pt:2119               2          2
 cg01.ific.uv.es:2119           2          2
 cgnode00.di.uoa.gr:2119        2          2
```

En este caso podemos comprobar que tenemos varios resultados, sets de recursos agrupados por el número de CEs y encontramos:

- Grupos de elementos que contienen un único CE, al que el trabajo sería enviado únicamente a este recurso. En esta situación por ejemplo, el CE **ce001.grid.ucy.ac.cy:2119** tiene libres las 10 cpus y un rango global (basado en el SI00 de este CE) de 650. Este recurso sería seleccionado primero por el Scheduling Agent con la política por defecto de seleccionar aquellos sets con menos elementos (en este caso un único elemento) ya que la latencia de las comunicaciones sería menor; y con mayor rango calculado.
- Después de los sets que contienen 1 CE, se forman sets con varios elementos. Encontramos 2 grupos con 2 CEs válidos para ejecutar el trabajo del usuario. El mejor, de acuerdo con el rango calculado, es el formado por **cagnode45.cs.tcd.ie:2119** *y* **ce100.fzk.de**. El rango computado de 440 no es la media de los rangos de los componentes (400 y 460, que sería 430) sino el rango ponderado considerando el número de cpus libres de cada componente.
- Como se ve no hay grupos posibles con 3 CEs que cumplan los requerimientos y agrupen el número de cpus requerido, y no hayan sido tenidos en cuenta en previos sets (con 1 o 2 CEs). Sin embargo podemos encontrar 1 grupo con 4 CEs.
- El CE **cgce.ifca.org.es:2119** no forma parte de ningún set ya que, aunque en principio cumplía los requerimientos individuales, en el momento de realizar la búsqueda no disponía de ninguna cpu libre y por lo tanto no es tenido en cuenta.

6.2.7. Application Launcher

Este módulo es el encargado de lanzar los trabajos paralelos a los recursos que han sido seleccionados. Para ello se siguen diferentes aproximaciones dependiendo del tipo de trabajo. Básicamente se envía el trabajo a las máquinas especificadas usando los mecanismos de gestión de trabajos de Condor-G [48]. Condor-G es un gestor de trabajos que hace de interfaz para lanzar los mismos via globus GRAM, y que en la práctica sustituye la interfaz en línea de comandos de globus, y que además aprovecha los métodos internos de Condor en lo que respecta a emisión fiable y colas de trabajos.

Las aplicaciones paralelas MPI que se consideran pueden estar compiladas con mpich (ch-p4 device) o con mpich-g2 [49] (Globus2 device), dependiendo de las necesidades del usuario y los recursos disponibles.

Por una parte mpich-p4 permite la ejecución intra-cluster, es decir entre máquinas de un mismo cluster comunicados por una LAN. En este caso las aplicaciones pueden estar compiladas estáticamente contra esta librería, o se dispone de ella en los WNs donde se ejecutará la aplicación.
Por otra parte, las aplicaciones mpich-g2 pueden ejecutarse entre varios clusters, comunicándose los procesos mediante los mecanismos de globus, donde han sido localizados los recursos por el algoritmo de

set-matching del RS. Estas aplicaciones no requieren que a librería esté instalada para la ejecución, sino que la librería mpich-g2 se compila dinámicamente y el lanzador específico de estas aplicaciones se encarga de la gestión de los sub-trabajos.

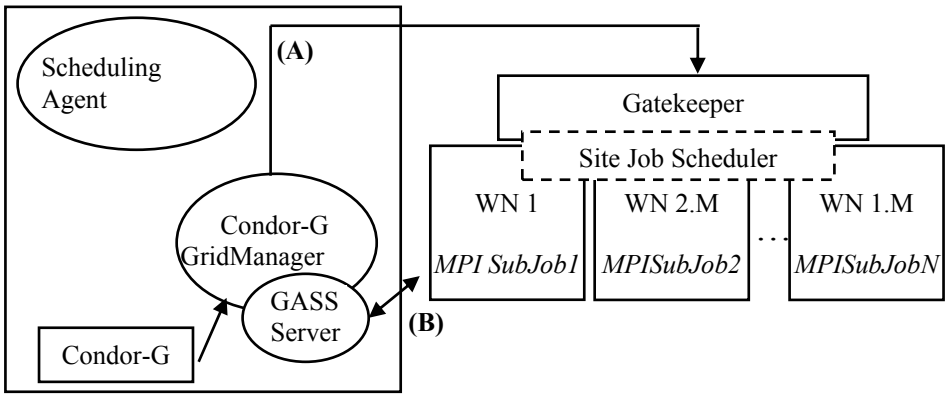
Un trabajo MPI enviado al grid puede fallar por varias causas, por lo que es fundamental tener servicios tolerantes a fallos. Nuestro lanzador de trabajos mpich-g2, junto con los mecanismos de Condor-G garantiza la co-asignación de recursos, el restablecimiento en caso de error y semánticas de exactamente una ejecución en las máquinas. Constituye por lo tanto un servicio confiable para el envío de estas aplicaciones paralelas mpich-g2 y sustituye los servicios provisto en el toolkit de globus.
Teniendo en cuenta las limitaciones de ejecución de mpich-g2 entre máquinas con IPs privadas, el RS busca los recursos que dispongan conectividad hacia fuera y IPs públicas. Esta limitación no existe para aplicaciones mpich-p4.

Esta información es tenida en cuenta por el RS gracias a su publicación en los servicios de información, pero desafortunadamente puede darse la situación de que esta información no sea correcta debido a la incorrecta configuración de los sites. En esta situación, los sub-trabajos que sean enviados a alguna de estas máquinas con IPs privadas no empezarán correctamente, bloqueando los otros sub-trabajos de la aplicación. Como se explica después, la detección de esta situación es dejada al AL, que reaccionará correctamente con los métodos más adecuados.

6.2.7.1. Gestión de trabajos mpich-p4

Las aplicaciones paralelas mpich-p4 serán ejecutadas en un único CE, como se muestra en la figura. Una vez que el SA recibe una aplicación de este tipo para ser ejecutara, se realiza una petición al RS y para que haga el proceso de matchmaking de los recursos para determinar cuál es el mejor en ese momento para la ejecución. Después se invocará al AL para que envíe el trabajo, ejecutando 2 pasos:

- Usando Condor-G, un script es enviado al sitio seleccionado (Flecha A en la figura). Este script es enviado a una de las colas del sitio controladas por el LRMS local (como por ejemplo PBS como el caso anterior). En la petición se hace referencia al número de máquinas (worker Nodes) que el trabajo mpi necesita y que serán servidas por el LRMS.
- Una vez el LRMS permite la ejecución del trabajo, habrá reservado las máquinas correspondientes que son indicadas a través de una variable interna. El script será ejecutado en una de estas máquinas, por ejemplo el WN1. El script es el encargado de obtener el ejecutable del trabajo en si (flecha B de la figura), así como los ficheros necesarios para su ejecución descriptos en el atributo *InputSandbox* del JDL. Esto es realizado a través de de un servicio GASS de ficheros de Globus. Una vez que se ha llegado a este punto, se realizará una llamada al *mpirun* para la ejecución del código paralelo en el número determinado de trabajadores.



Con esta aproximación, se asume que todos los Worker Nodes comparten el sistema de ficheros donde se ejecutan los trabajos (tradicionalmente el espacio de usuario bajo /home). Por lo tanto sólo se necesita ejecutar el binario del ejecutable y los ficheros a una maquina, desde donde ya es accesible para el resto. Adicionalmente se debe tener configurado correctamente el método de comunicación entre procesos, que

suele ser *rsh* o *ssh*. En este último caso se deben configurar todos los WN para aceptar comunicaciones entre ellos sin pedir ninguna password para que *mpirun* pueda empezar su ejecución automáticamente.

6.2.7.2. Gestión de trabajos mpich-g2

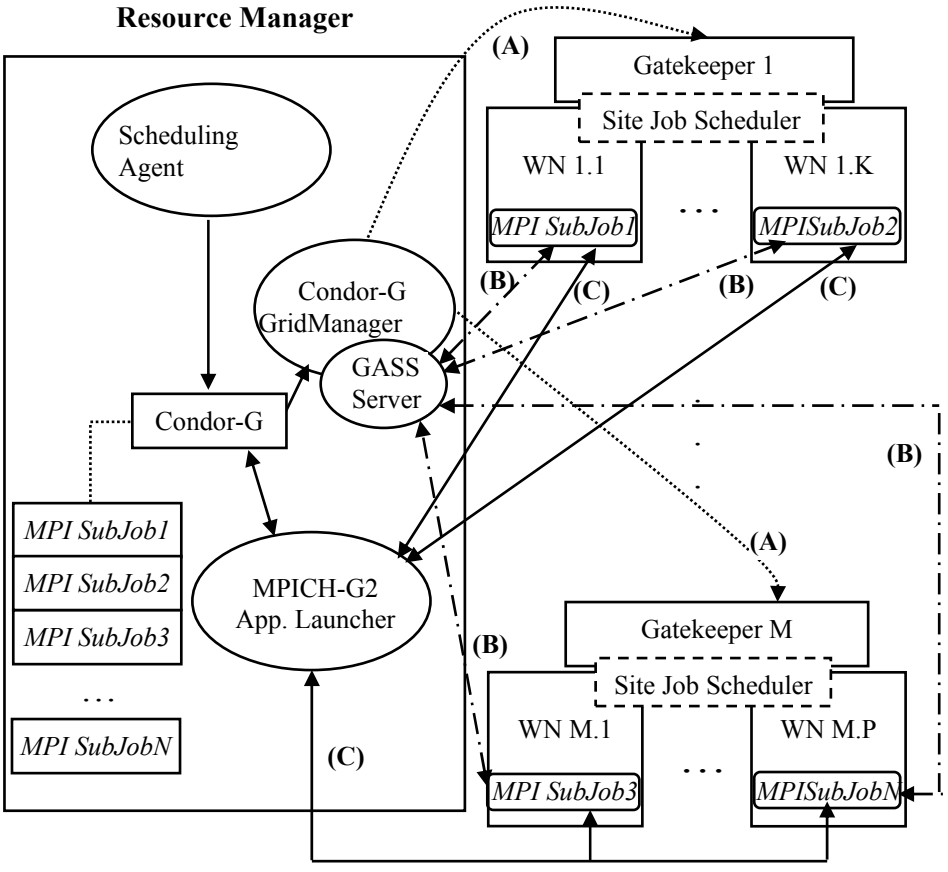
La gestión de los trabajos paralelos puede requerir más máquinas que las que son provistas por un único site, o CE, por lo que entonces se necesita una ejecución en múltiples sitios. Usando mpich-g2, una aplicación puede ejecutarse en una WAN comunicándose sus trabajadores a través de los mecanismos seguros de globus.

Una aplicación mpich-g2 puede ejecutarse usando el comando *globusrun del globus toolkit*, a través de la especificación de los distintos sites donde se quiere ejecutar usando el fichero *.rsl* (por ejemplo: globusrun –s –w –f app.rsl). El comando globusrun invoca los servicios de co-reserva (*co-allocation*) de *la* librería DUROC [50] para sincronizar todos los subtrabajos a través de un mecanimo de barrera. Pero cuando se ejecutan los trabajos con globusrun, el usuario debe especificar los recursos y además preguntar por el estado de su aplicación, reenviandola si algo ha ido mal. Para liberar al usuario de estas responsabilidades, proponemos usar Condor-G para la ejecución fiable de los trabajaos en múltiples sitios. El Lanzador de aplicaciones específico para mpich-g2 maneja la sincronización de manera nativa usando los mismos servicios de DUROC, pero obteniendo a la vez los beneficios de usar Condor-G.

Una vez que el SA detecta que la aplicación mpich-g2 necesita ser enviada, pasa la petición con los recursos adecuados al lanzador específico (MPI-AL). Este lanzador es ejecutado a través de Condor-G, de manera que primero se ejecuta en la máquina local del Crossbroker una instancia para cada trabajo enviado, y éste se encarga de enviar todos los subtrabajos a través de Condor-G de nuevo, siguiendo un procolo de *commit* de dos fases:

- Primero todos los subtrabajos son enviados a través de condor-G
- Un segundo paso garantiza que todos los subtrabajos tieen una máquina donde se ejecutan, y que han ejecutado la llamada a *MPI_Init* necesaria. Esta llamada invoca a su vez a DUROC, y la sincronización es conseguida a través de la barrera que levantará el MPI-AL cuando todos los subtrabajos ya han hecho la llamada inicial.

Después de la sincronización y que el MPI-Al levante la barrera, los subtrabajos son permitidos para que se ejecuten. La figura muestra como se ejecutan sobre múltiples CEs, de manera que tenemos N subtrabajos que forman parte de la aplicación mpich-g2. Los subtrabajos serán ejecutados en varios sitios diferentes, aunque por simplicidad en la figura se muestran sólo 2 sitios.



Las flechas marcadas como (A), muestran la fase de 2 pasos de *co-allocation*. El servidor GASS es contactado para enviar los ficheros y los ejecutables a los WN remotos donde finalmente se ejecutará cada subtrabajo, y al final para recoger los ficheros de salida de los mismoa. Esto es mostrado por las flechas marcadas como (B).

Puede darse la situación de que algunos subtrabajos ya hayan realizado la llamada MPI_Init y otros no se estén ejecutando todavía. Esto puede deberse a que algunos subtrabajos estén esperando en las colas para la ejecución, debido que que hay otros trabajos antes que éstos.
En el peor caso se pueden dar interbloqueos (deadlocks) distribuido. Por ejemplo en la situación en la que todos los subtrabajos de una aplicación1 se están ejecutando en un site A y el último subtrabajo está en la cola de del site B. Si entonces tenemos una aplicación 2 que tiene todos los subtrabajos ejecutándose en B, y el último subtrabajo esperando en la cola del sitio A, tendremos interbloqueo ya que la situación no puede avanzar.
Para evitar estas situaciones, el MPI-Al establece un tiempo durante el cual se esperará la llamada MPI-Init de todos los subtrabajos, cancelando todos en el caso de que salte el *time-out* y no se hallan recibido.

Cada subtrabajo es enviado de forma independiente a través de condor-G a la máquina correspondiente, aunque se puede optimizar el envío de los mismos de manera que se envíen conjuntamente todos los subtrabajo correspondientes a un mismo CE a través de un envío de un trabajo que pida el número de nodos igual al de subtrabajos. Con esto se consigue no saturar las colas de los LRMS con subtrabajos, y además que la ejecución sea más efectiva ya que cuando entre éste todos los subtrabajos tienen garantizada la máquina correspondiente y la aplicación completa tendrá mas opciones de ejecutarse.

Una vez que los subtrabajos están ejecutándose en los WN, MPI-AL monitoriza su ejecución y escribe el *log* de la aplicación global, dando una información completa sobre la ejecución de los subtrabajos. Esta monitorización es mostrada a través de las flechas C en la figura, y constituye la clave para proveer una ejecución fiable de las aplicaciones así como su robustez.
Si la aplicación acaba correctamente o si existe algun problema en la ejecución de algún trabajo, el lanzador MPI-Al lo comunica al SA, a través de ficheros de log que son comprobados por el SA. El SA llevará a cabo la acción correcta, de acuerdo con esta información.

Para garantizar la ejecución fiable, el SA monitoriza todas las instancias de los MPI-AL que están corriendo en este momento. Cuando una de esta finaliza, comprueba el correspondiente fichero de log de este lanzador para ver cuál ha sido el resultado.
Los problemas detectados pueden ocurrir en diferentes momentos, y pueden ser o temporales o permanentes. La siguiente tabla muestra dos problemas que pueden aparecer antes de que el protocolo de dos fases acabe. Si el lanzador MPI-Al en si mismo falla, el SA emitirá otra instancia que controlará los sub-trabajos de la aplicación. Si la aplicación termina correctamente o si hay alguna terminación anormal de un subtrabajo (debido a un error del programa por ejemplo), el SA notificará al usuario.

Situación detectada por el MPI-AL	**Acción del SA**
- Un subtrabajo no se ejecutó porque el recurso Globus falló	Marcar el recurso como no disponible, para que no sea elegible por los siguientes trabajos durante cierto tiempo. Repetir la llamada al RS.
- Un subtrabajo no empezó su ejecución aunque el recurso globus estaba corriendo.	Se reintentará el reenvio a los mismos recursos, pero si la misma situación se repite puede ser debido a un firewall en la máquina remora. La aplicación repetirá la llamada al RS
- El lanzador MPI-AL falla	El SA mandará otra instancia que se adueñará de los subtrabajos que se han lanzado. Si los subtrabajos han ejecutado todos el protocolo commit de dos pasos, continuan la ejecución sin darse cuenta, de otra manera los trabajos son eliminados y el envío será repetido
- El trabajo falla anormalmente	El usuario será notificado

Para resumir, podemos concluir que el MPI realizará las siguientes acciones:

- Ejecución fiable de aplicación paralelas con semántica de única ejecución (once-only)
- Ejecución coordinada de las aplicaciones, lo que significa que el trabajo ejecutará cuando todo los subtrabajos tengan recursos para la ejecución.
- Un buen uso de los recursos. Si un subtrabajo no puede ser ejecutado la aplicación entera fallará, por lo que las máquinas no estarán bloqueadas y estarán listas para la ejecución de otros trabajos de otros usuarios.

6.2.8. Job Controller

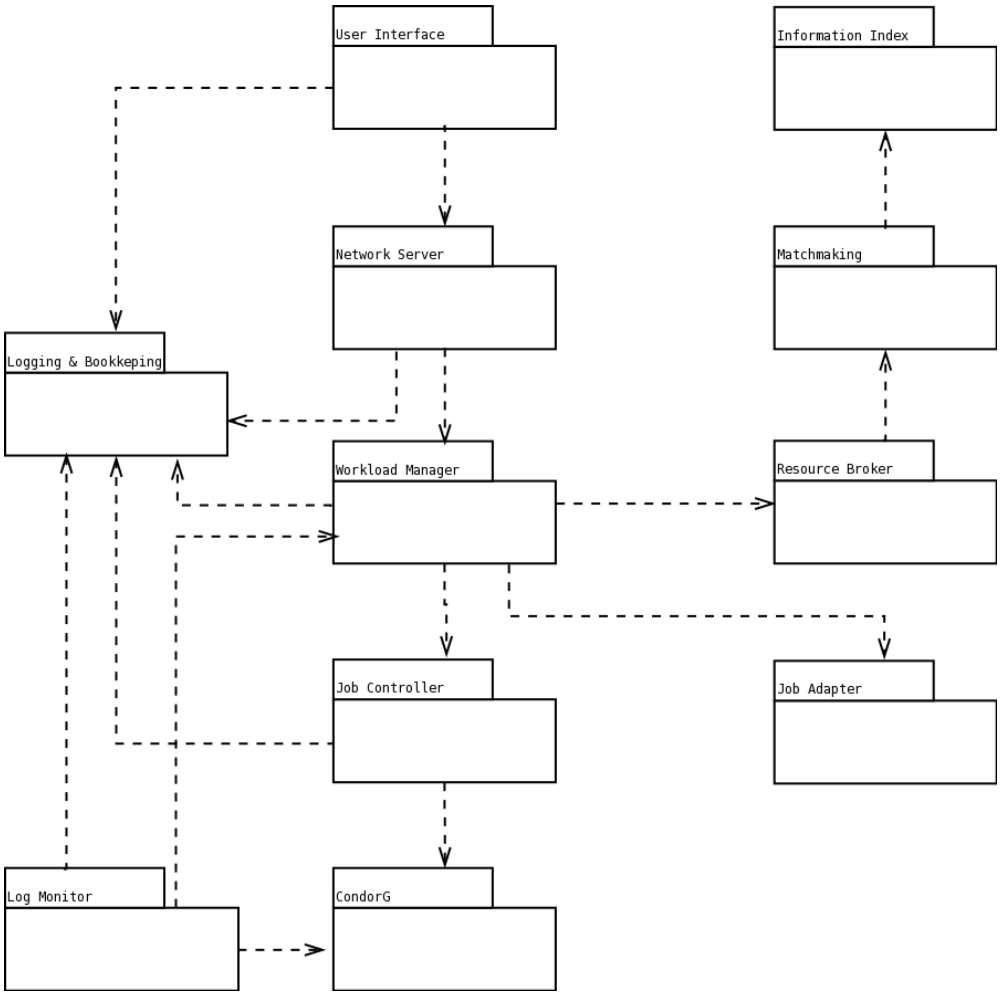
Este módulo realiza las tareas de control necesarias para cada trabajo y además hace de interfaz con el sistema de envío de trabajos subyacente, que en este caso es Condor-G [48] . De esta manera es posible serializar las operaciones sobre los trabajos del CrossBroker sobre la interfaz de Condor-G en linea de comandos.
Mediante este módulo podemos enviar trabajos, preguntar el estado de los mismos y cancelarlos. Asimismo proveen de información sobre la ejecución al resto de módulos, que realizarán las acciones oportunas.
Debido a las limitaciones de Condor-G no se puede monitorizar directamente toda la información, por lo que además este módulo se encargará de controlar continuamente la salida de los logs para interceptar los eventos relacionados con un trabajo determinado y reportarlos al sistema de Logging and Bookeping externo.

7. IMPLEMENTACIÓN DEL CROSSBROKER

La implementación de nuestro CrossBroker se basa en un diseño cliente-servidor, dividido en varios módulos. Se ha desarrollado desde un punto de vista funcional, reutilizando la mayor parte de software disponible que se adaptara a nuestras necesidades. En la capa más básica tenemos los servicios de globus 2.X que proveen los medios básicos de autenticación, de gestión básica de recursos (gram), y de envío de ficheros a través de GridFtp y del *gass-server.*
Como gestor básico de trabajos utilizamos Condor-G, que sustituye los comandos de globus para el envío de trabajos y la ejecución fiable de trabajos simples. Es además utilizado por el broker de Datagrid que desarrollado para el envío de trabajos simples.
Los módulos específicos se han desarrollado sobre estos componentes, manteniendo los componentes externos con interfaces compatibles con éstos para su adaptación a futuras versiones. En algunas partes ha sido necesaria la modificación del código fuente, especialmente del código del broker de Datagrid, para hacer las extensiones necesarias de forma que fuera eficiente.

La implementación se ha desarrollado en varios lenguajes siendo las más importantes:

- L&B client and server (código externo no modificado) : C/ C++
- Network Server client and server(User Acces Module) : C / C++, APIs de globus
- Workload manager(Scheduling Agent Module, Queue Manager), Matchmaker (Resource Selector), Aplication Launcher: C/C++, APIs de globus
- UserInterface (User Acces Module): API C++ / API Java/ JNI technology / command line en Python, APIs de globus
- ThirdParty Software no modificado : ByPass in C / C++ ; GlobusGridFtpServer in C ; Globus-SSL-Utils in C ; Trio in C ; binarios de Condor-G
- Partes específicas como los *JobWrappers* enviados a los CEs y ejecutados en los WN son shell scripts y perl scripts.



7.1.Broker

7.1.1. Diagramas UML

Resource Selector

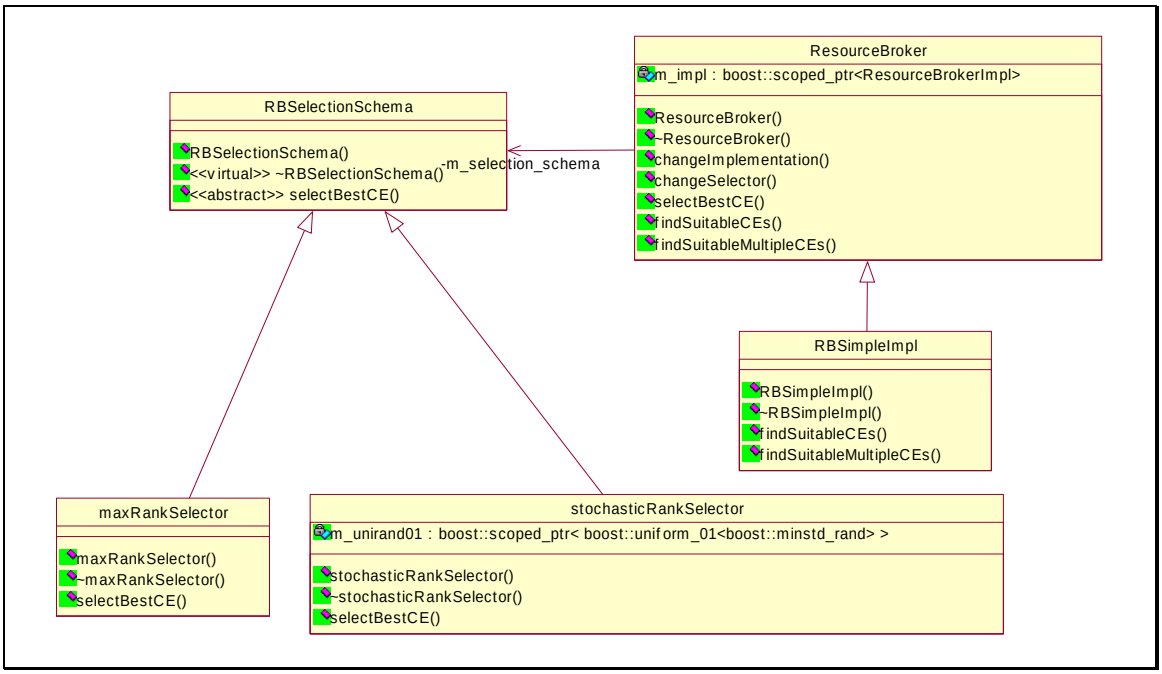


Figure 1, ResourceBroker

ResourceBrokerImpl

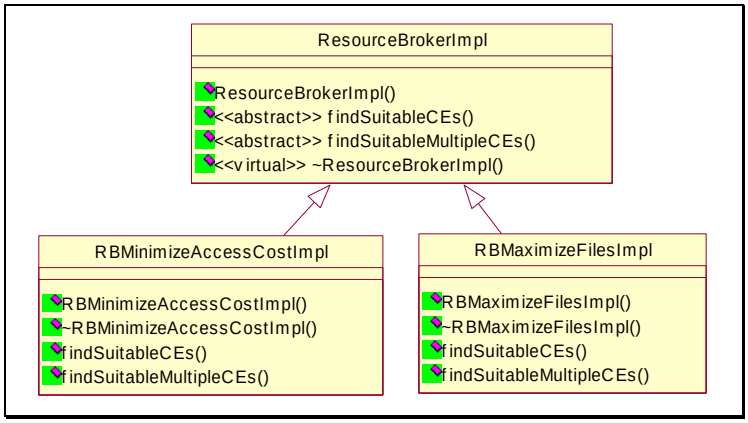


Figure 2, ResourceBrokerImpl

Excepciones y otros

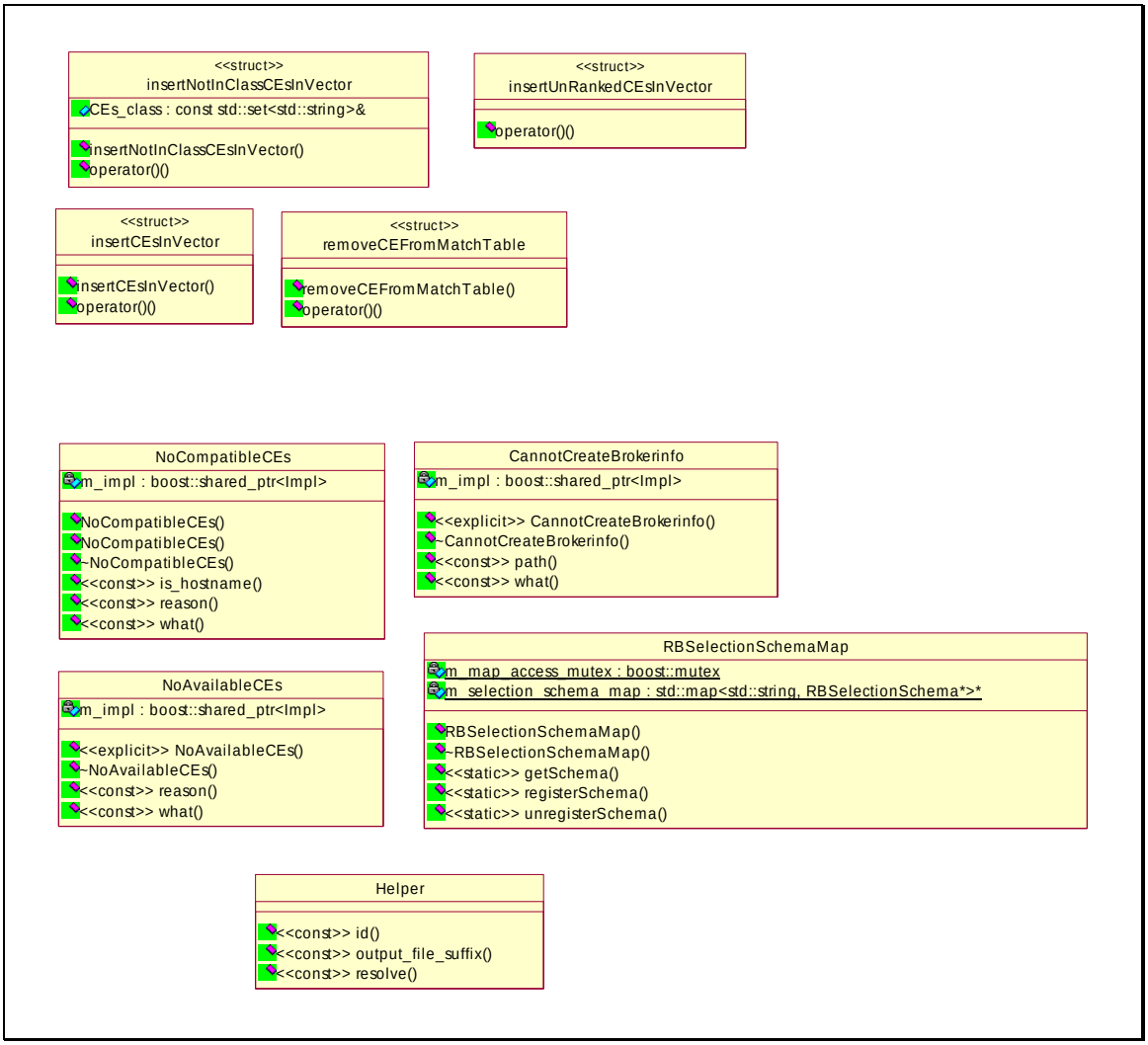


Figure 3, Excepciones

7.1.2. Clases Implementadas

ResourceBroker

- Esta clase es la principal del Resource Selector, y representa el R*esourceBroker*, una clase que contiene una implementación específica de un selector de recursos (*ResourceBrokerImpl*) ,que los obtiene dependiendo qué tipo de búsqueda de recursos queramos llevar a cabo. Además contiene un esquema de selección de política (*RbSelectionSchema)* que permite describir cuáles son las políticas de selección aplicadas en cada momento.

ResourceBrokerImpl

- Representa una clase abstracta de una implementación, que será usada por la clase anterior. De ella derivan las siguientes implementaciones.

RBSimpleImpl

- Contiene la implementación para buscar recursos cuando no se tienen requerimientos de datos, para seguir el algoritmo descrito en el punto 6.2.6.1.

RBMinimizeAccessCostImpl

- Contiene la implementación para buscar recursos con requerimientos de datos, de manera que se buscan aquellos que tienen el mínimo coste de acceso a los ficheros especificados. Para ello se utiliza las funciones de estimación de coste de acceso a ficheros proporcionadas por elementos externos asociados al *Replica Location Service.*

RBMaximizeFilesImpl

- Implementa el algoritmo descrito en 6.2.6.2, de manera que se buscan los recursos que dispongan de el mayor número de ficheros disponibles de forma local en sus testbeds. Para ello también se contacta con el *Replica Location Service* externo

RBSelectionSchema

- Representa una clase abstracta, que define la interfaz que debe tener los esquemas de políticas de selección de recursos, de la cual se derivan las diferentes implementaciones.

RBSelectionSchemaMap

- Clase que sirve para registrar nuevos esquemas dinámicamente, y hacer una selección de los mismos disponibles para que sean utilizados por la clase anterior (*RBSelectionSchema*)

maxRankSelector

- Esquema de selección por defecto, que hereda de la clase *RBSelectionSchema,* y que selecciona el mejor recurso obtenido de acuerdo con la política especificada por el usuario a través del campo *Rank*. Este campo puede adicionalmente tener otras políticas añadidas por el *Scheduling Agent*

stochasticRankSelector

- Selector estocástico del mejor recurso, de acuerdo al campo *Rank*. En este caso si varios recursos tienen el mejor rango se hace una selección estocástica para seleccionarlo, lo que permite hacer una distribución correcta de los trabajos en los recursos, en el caso de enviar sucesivamente varios trabajos con las mismas características.

helper

- Clase extra para realizar las funciones asociadas de llamadas a otros módulos, que no se encuadran en las anteriores.

.

7.1.3. Ficheros adicionales

Exceptions.*

- Contiene las excepciones que se lanzan cuando ocurre algun error en este módulo.

Utility.h

- Utilidades del código que para mejorar la legibilidad se agrupan en este fichero .

7.1.4. MatchMaking

Diagramas UML

Excepciones

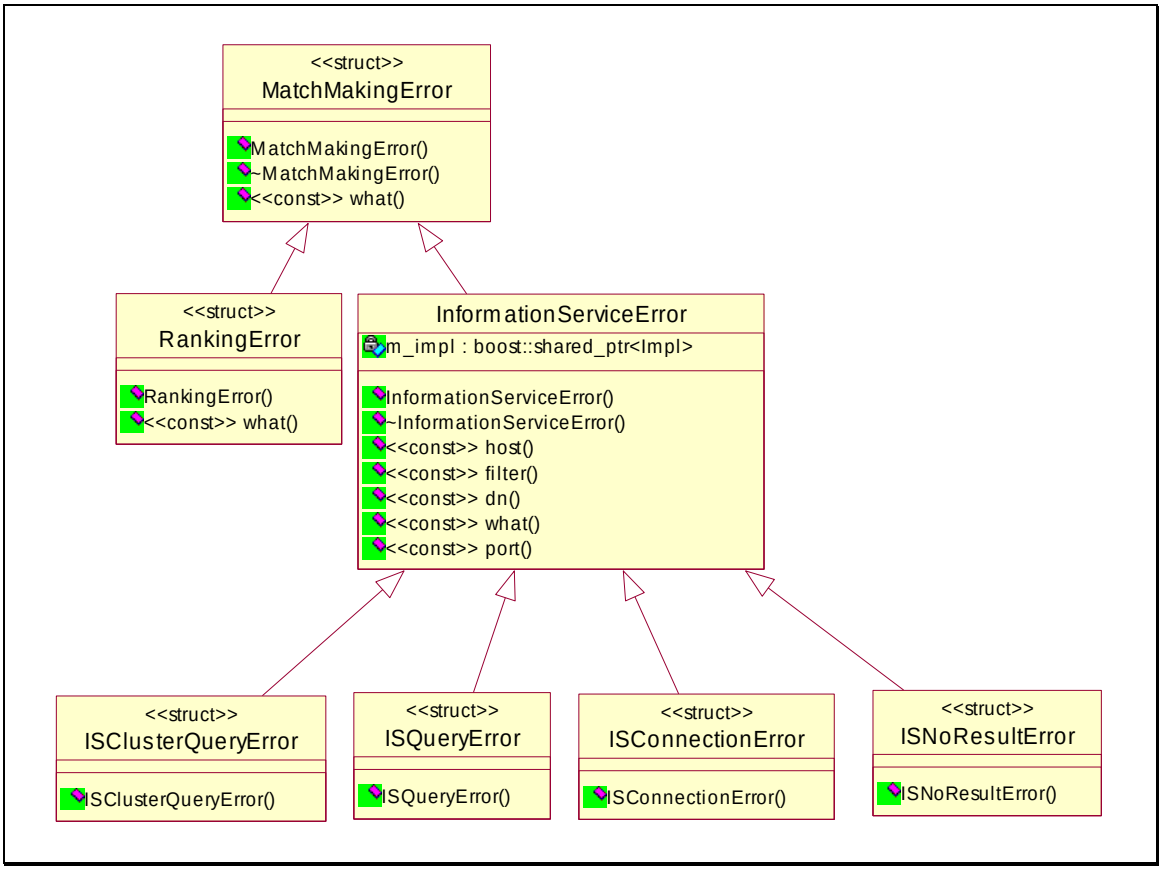


Figure 4, Errors

MatchMaker

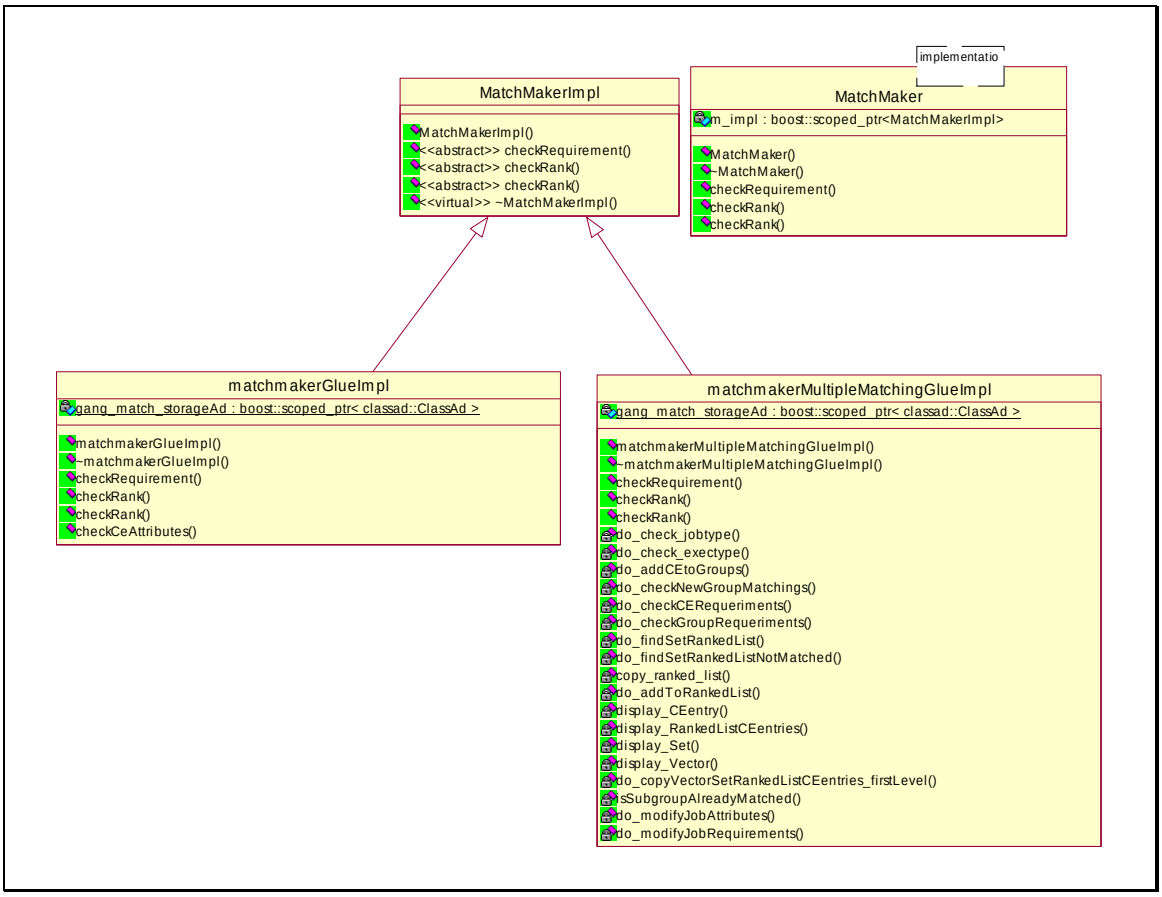


Figure 5, MatchMaker

Modelado de Recursos simples y conjuntos

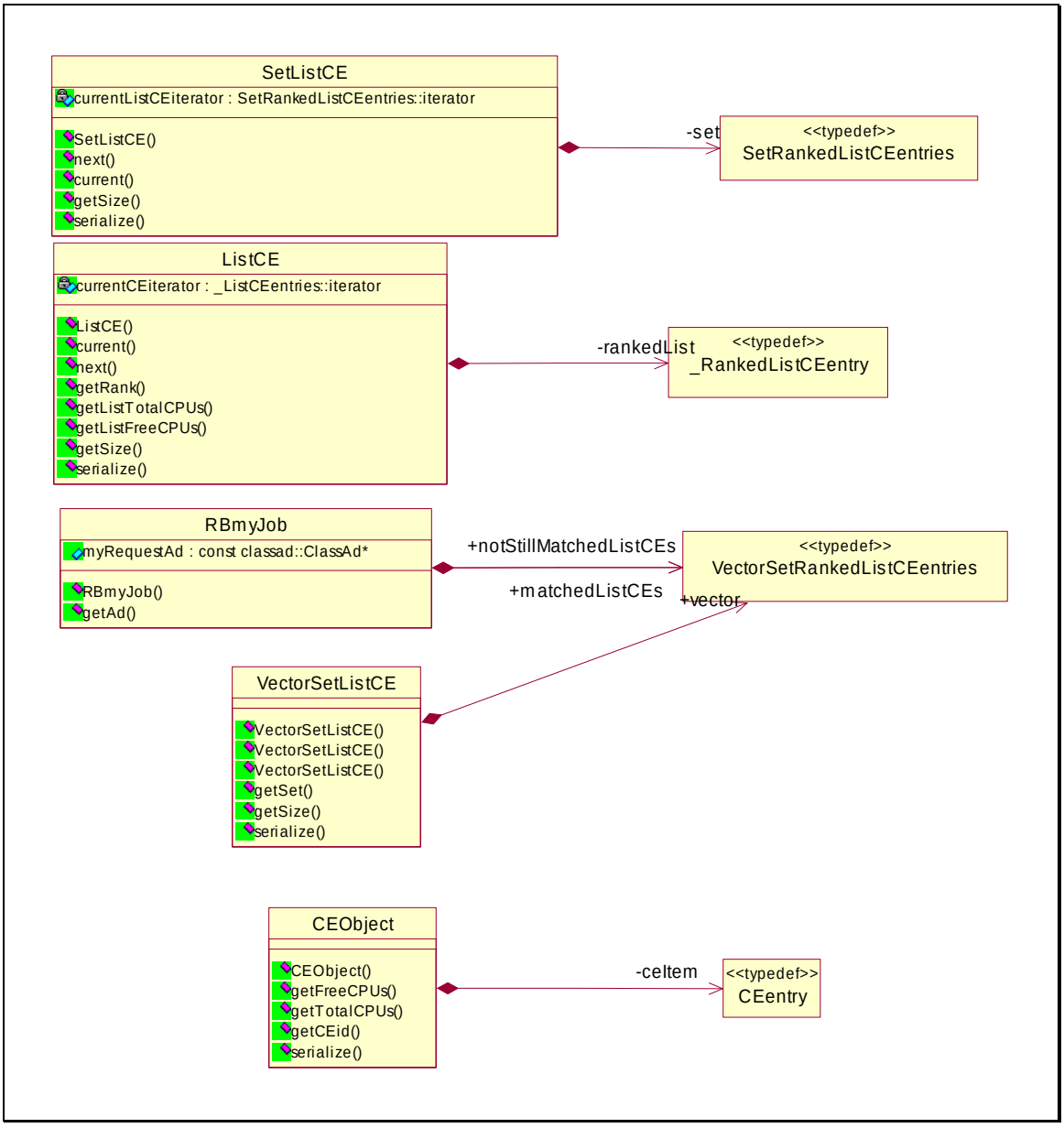


Figure 6, Lists, Objects and Vectors

Estructuras y typedef

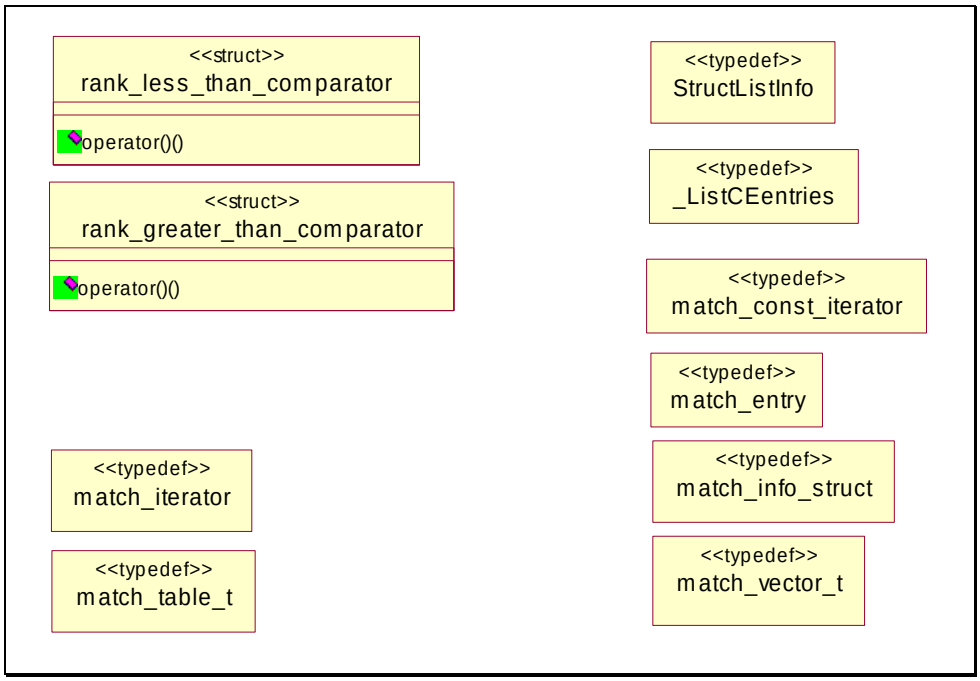


Figure 7, Estructuras y typedefs

Classes

CEObject

- Representa un recurso (Computing Element) con representación interna como un classAd y que provee de funciones para conocer los datos más importantes. Es usado para construir posteriormente listas de los mismos.

ListCE

- Representa una lista de recursos, agrupados para formar grupos de recursos compatibles.

SetListCE

- Representa un conjunto de listas de recursos.

VectorSetListCE

- Vector formado por elementos de la anterior clase (SetListCE), y que agrupa las listas de recursos por el número de componentes de sus conjuntos.

MatchMaker

- Clase que representa el emparejador de trabajos con recursos, dependiendo del tipo de trabajo se hará un emparejamiento 1 a 1, o 1 a muchos. Hace referencia a una implementación, dependiendo de cómo se realiza el matchmaking y cómo se obtiene la información de los recursos.

matchmakerImpl

- Clase abstracta que representa la implementación del matchmaker, y que contiene la interfaz que debe ser implementada por los diferentes matchmakers.

matchmakerGlueImpl

- Matchmaker que obtiene la información del Sistema de información MDS, organizada en el *Glue Schema*

matchmakerMultipleMatchingGlueImpl

- Hereda del anterior, y provee las funciones adecuadas para el *matching* múltiple de recursos.

matchmakerInteractiveGlueImpl

- Hereda del anterior, aplicando las políticas adecuadas de selección de recursos cuando lanzamos trabajos interactivos.

BrokerInfo

Uml Diagram

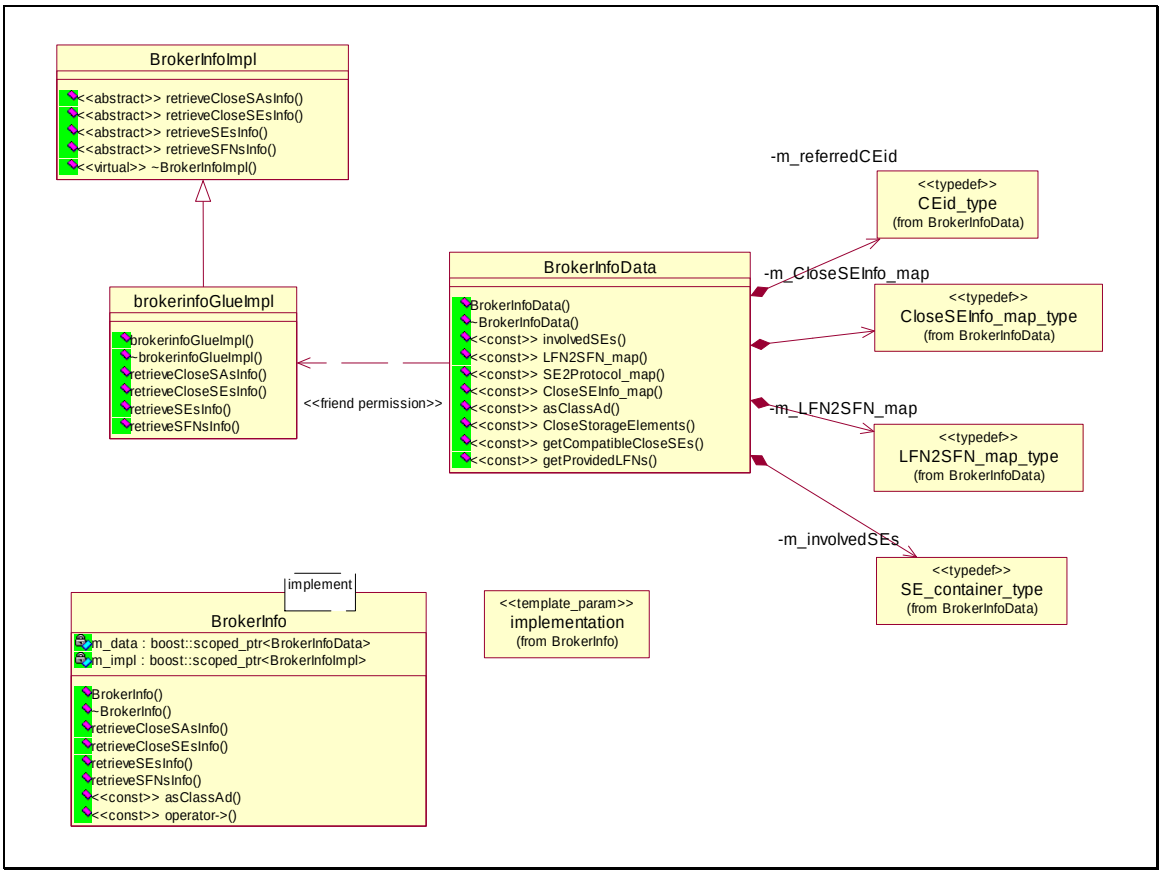


Figure 8, Broker Info

Classes

brokerInfo

- Clase que contiene información sobre las réplicas disponibles cuando el trabajo tiene requerimientos de datos, de manera que se puede serializar y tener la información disponible cuando se ejecute el trabajo en el recurso remoto, y así pueda acceder a la información ya resuelta por el broker. Contiene una implementación que lleva a cabo las funciones adecuadas.

brokerInfoImpl

- Implementación abstracta del brokerInfo

brokerInfoGlueImpl

- Implementación basándose en la información provista por los Storage Elements publicada en el Glue Schema.

brokerInfoData

- Contiene la información obtenida para cada fichero de entrada requerido, la resolución de los Logical File Names, etcétera.

7.1.5. Integración con Replica Location Service

Diagrama Uml

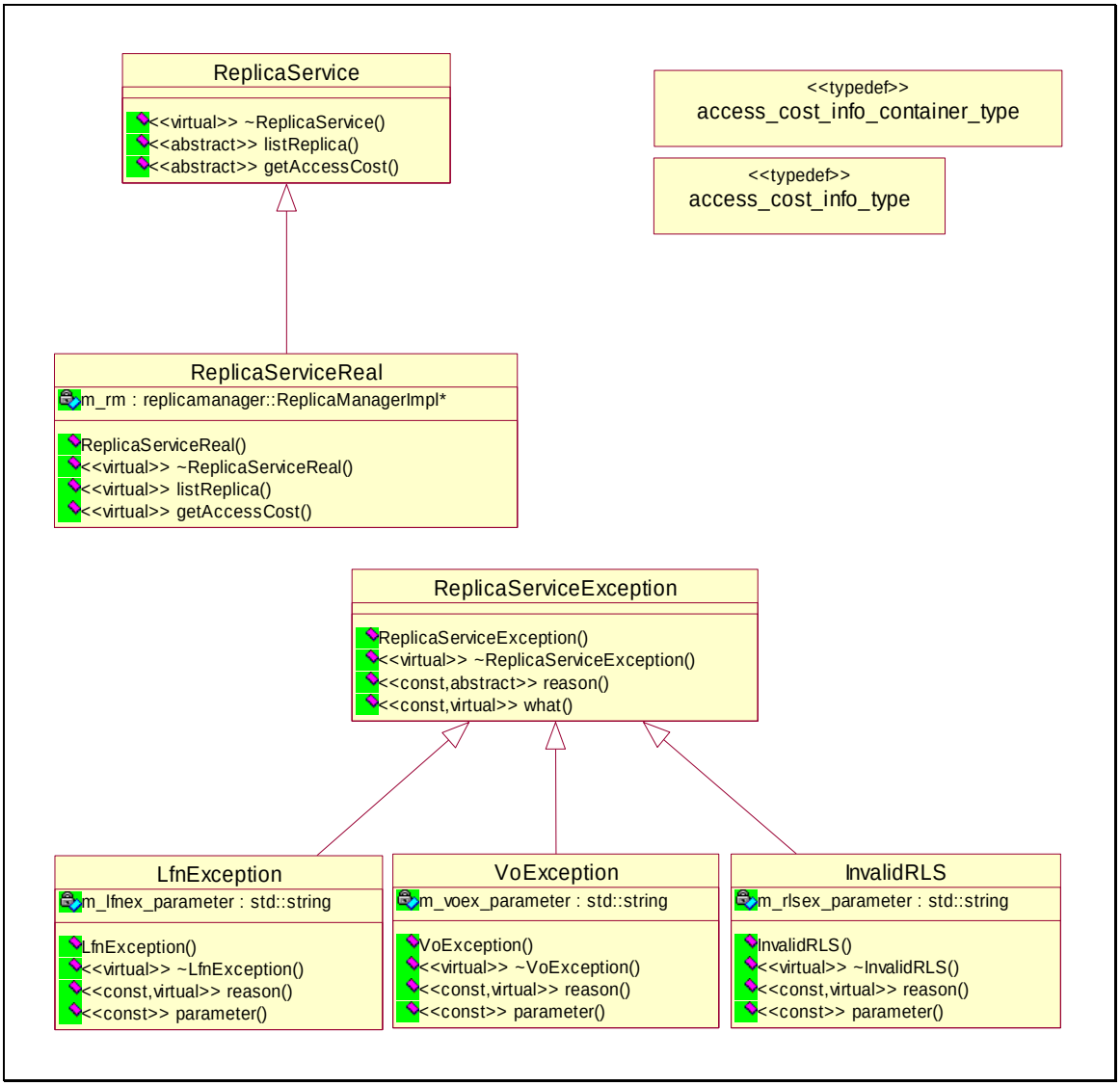


Figure 9, RLS Namespace

Clases

ReplicaService

- Clase virtual con la especificación de la interfaz que debe contener las implementaciones del servicio de replicas, para resolver las cuestiones relacionadas con los datos distribuídos.

ReplicaServiceReal

- .Implementación de la interfaz anterior, que hace de interfaz con el *Replica Location Service* Externo.

ReplicaServiceException

- Representa una excepción producida en las consultas al *RLS.*

LfnException

- Excepción al resolver un LFN

VoException

- Excepción relacionada con errores derivados de las VO.

InvalidRLS

- Excepción que determina un punto de RLS inválido

7.2.Common

7.2.1. Diagramas UML

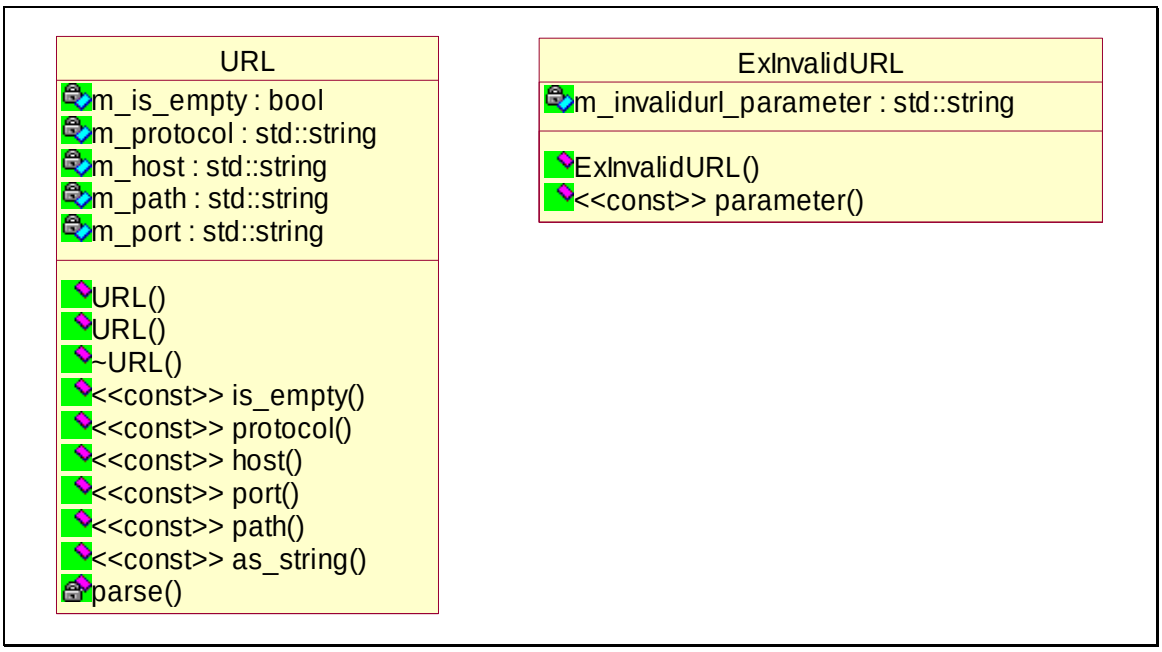


Figure 10, Common classes

7.2.2. Classes

URL

- Clases para manejar funcionalidades communes y excepciones relacionadas con URLs.

ExInvalidURL

- Exepción en el tratamiento de URLs

7.3.Helper

7.3.1. Diagramas UML

Helper y Helper Factory

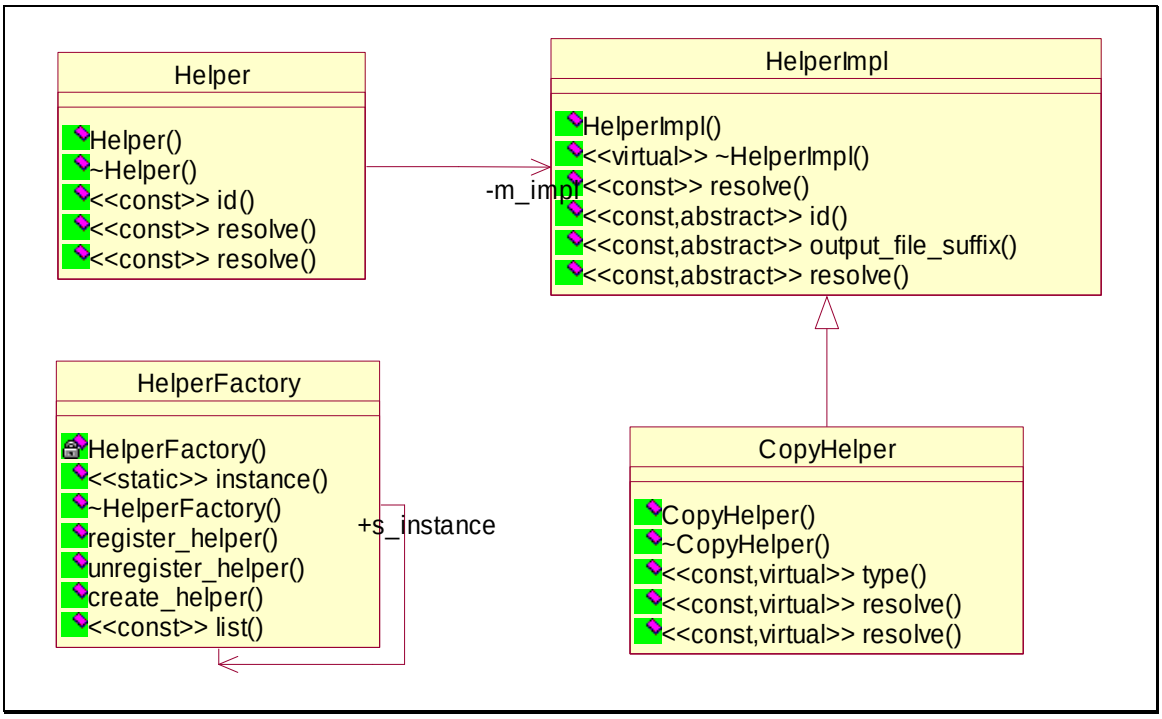


Figure 11, Helper and Helper Factory

Excepciones y Errores

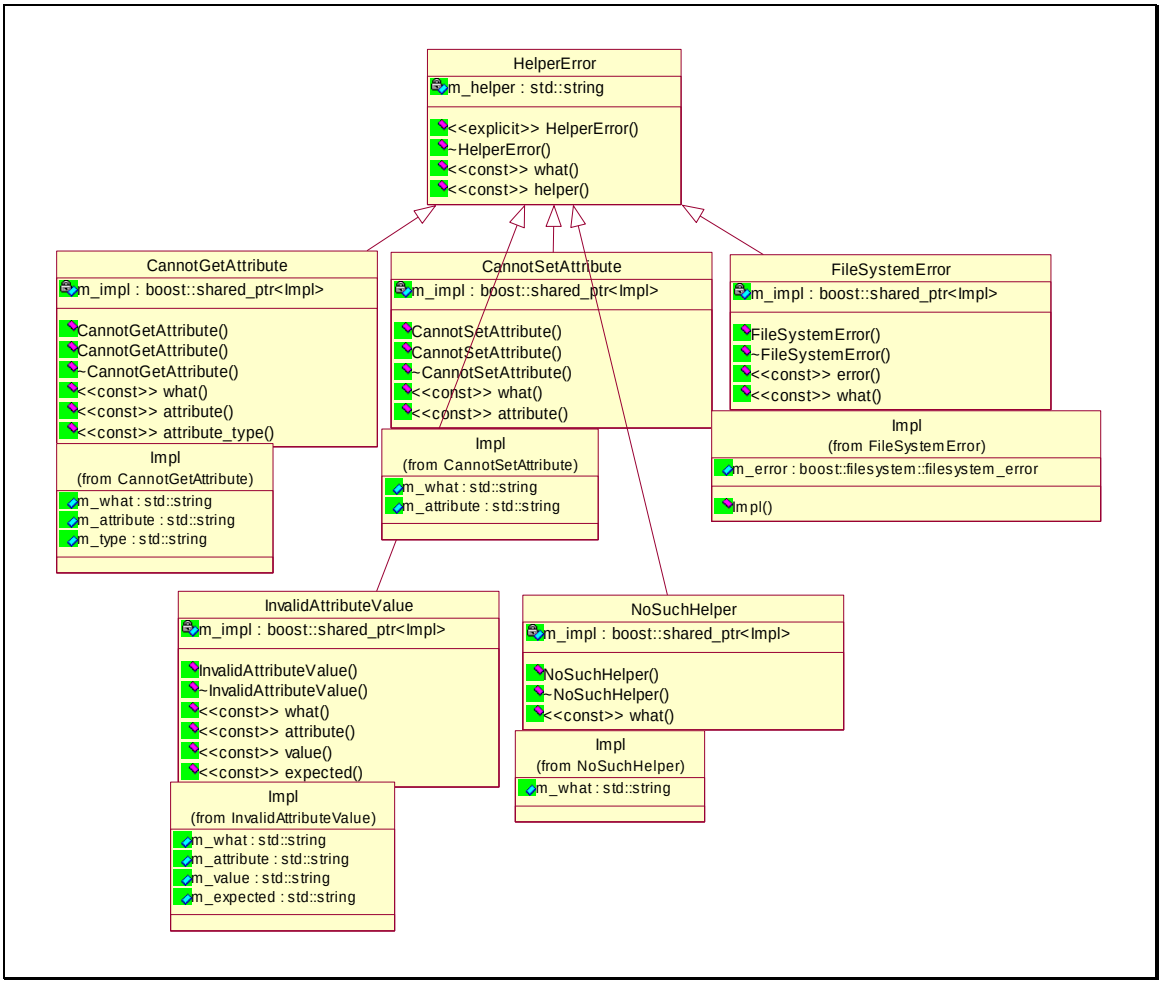


Figure 12, Errors

Request

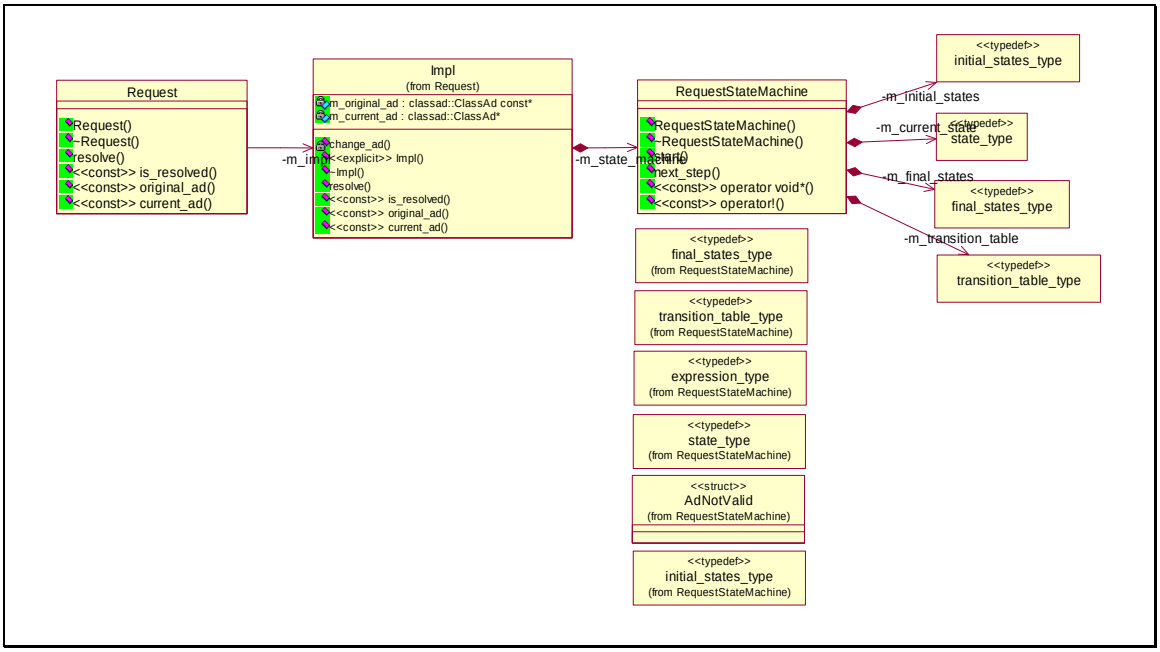


Figure 13, Request

ClassAds

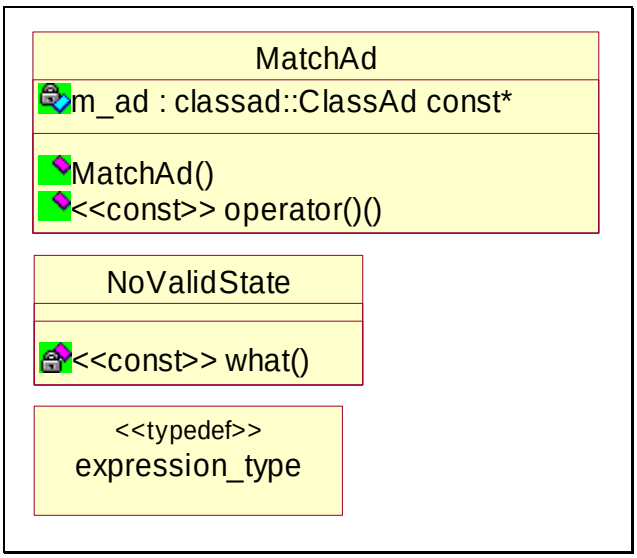


Figure 14, Others

7.3.2. Classes

Helper

- Esta clase representa el punto de acceso para resolver un trabajo, y debe ser implementado por clases que se deriven de ésta.

HelperImpl

- Implementación basándose en la representación de trabajos como classads.

HelperFactory

- Factoría de creación de objetos *Helper*, a partir de su definición como classads

CopyHelper

- Clase para copiar objetos *Helper*

HelperError

- Excepción en la factoría o al ejecutar alguna de las funciones asociadas de *Helper*

CannotGetAttribute

- Excepción

CannotSetAttribute

- Excepción

FileSystemError

- Excepción

InvalidAttributeValue

- Excepción

NoSuchHelper

- Excepción

RequestStateMachine

- Representa una máquina de estados por la que pasa el trabajo desde su recepción hasta su finalización.

7.4.JobAdapter

7.4.1. Diagrama UML

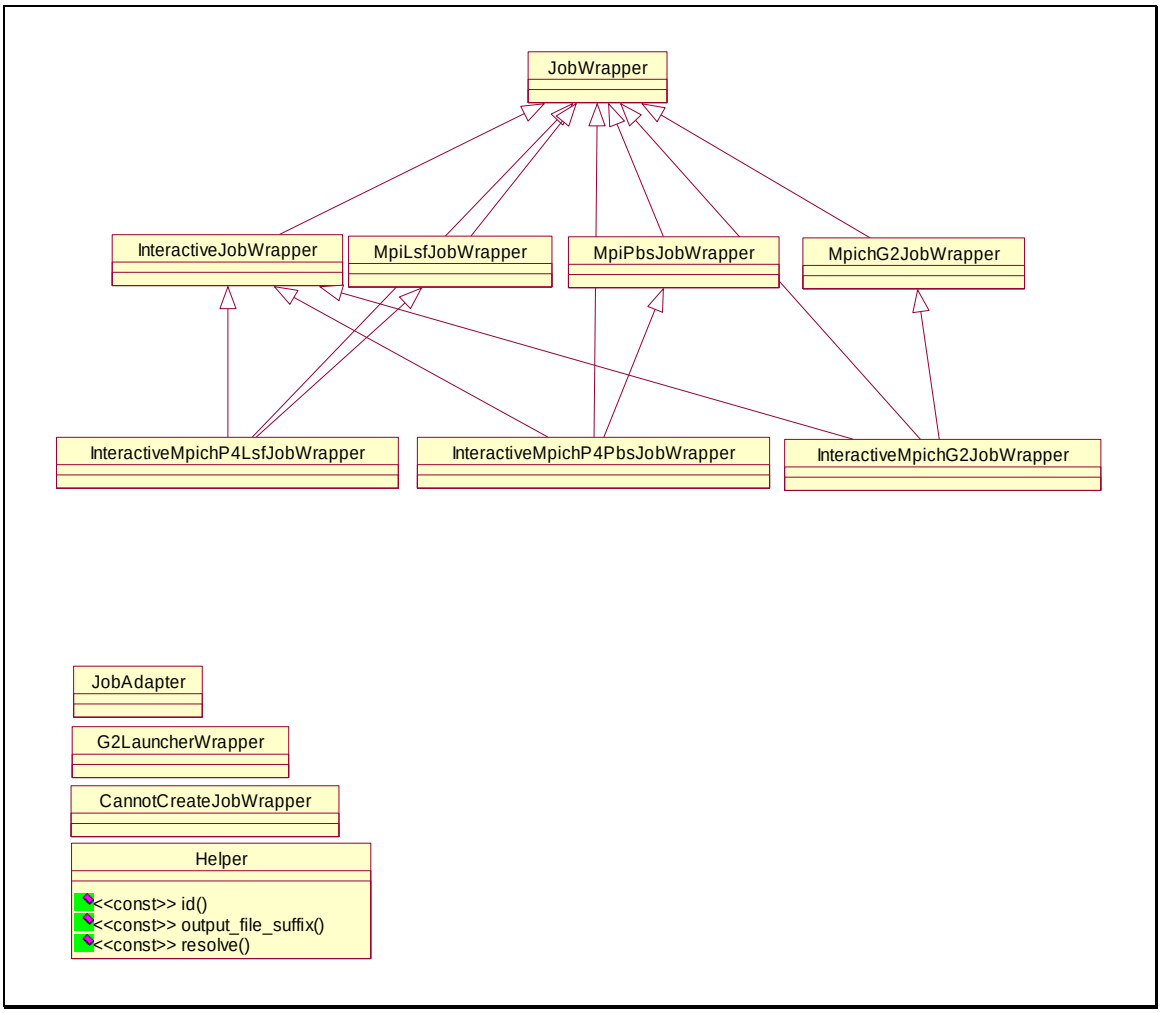


Figure 15, Job Wrappers

Interactive Jobs

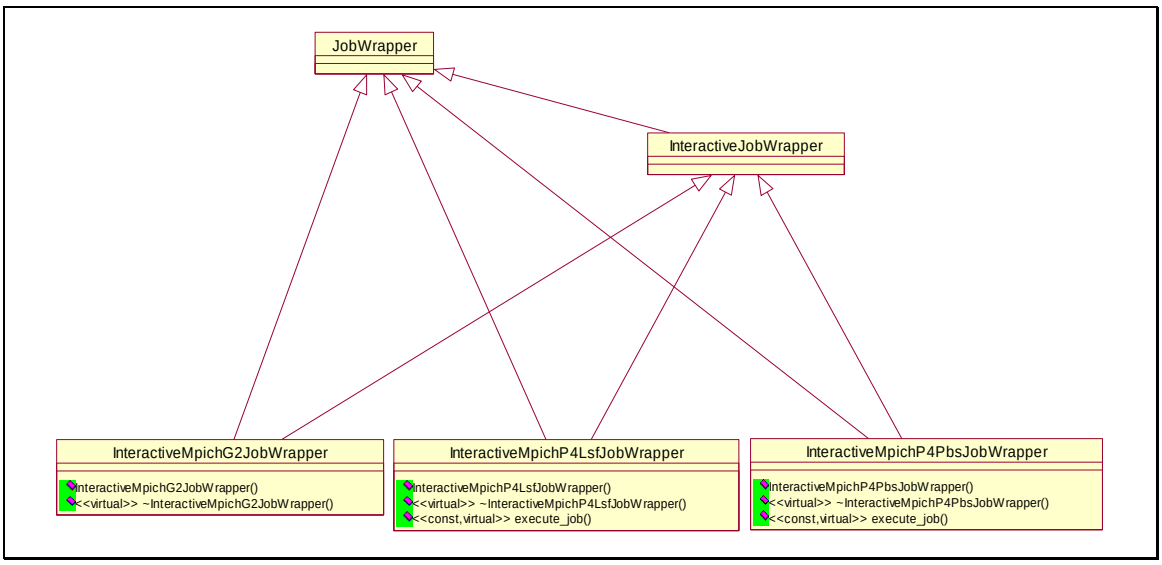


Figure 16, Interactive Jobs

Interactive MPI Jobs

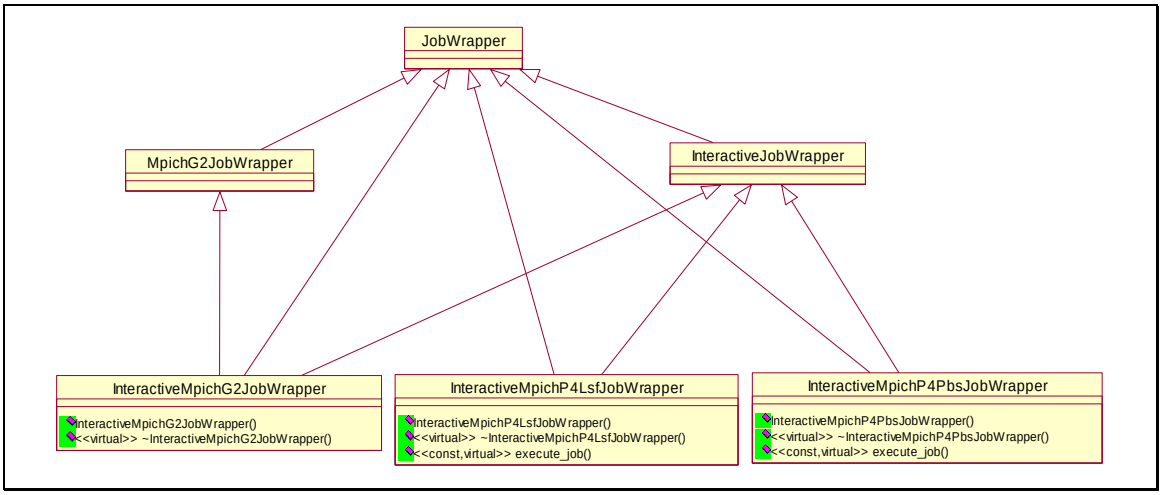


Figure 17, Interactive MPI Jobs

7.4.2. Clases

JobWrapper

- Clase abstracta que representa un wrapper de trabajos que se ejecutarán en la máquina remota para realizar las acciones adecuadas

InteractiveJobWrapper

- Deriva de la *JobWrapper* y representa trabajos interactivos.

MpilSfJobWrapper

- Deriva de *JobWrapper* y representa trabajos paralelos mpi (mpich-p4) ejecutados en Computing Elements con LRMS de tipo LSF.

MpiPbsJobWrapper

- Deriva de *JobWrapper* y representa trabajos paralelos mpi (mpich-p4) ejecutados en Computing Elements con LRMS de tipo PBS.

MpiChG2JobWrapper

- Deriva de *JobWrapper* y representa trabajos paralelos mpi (mpich-p4) ejecutados en varios computing elements. Cada uno de los subtrabajos está implementado por este wrapper.

InteractiveMpichP4LsfJobWrapper

- Deriva de Interactive*JobWrapper* y de MpilSfJobWrapper y representa trabajos paralelos mpi (mpich-p4) interactivos.

InteractiveMpichP4PbsJobWrapper

- Deriva de Interactive*JobWrapper* y de MpiPbsJobWrapper y representa trabajos paralelos mpi (mpich-p4) interactivos.

InteractiveMpichG2JobWrapper

- Deriva de MpichG2JobWrapper y de MpiPbsJobWrapper y representa trabajos paralelos mpi (mpich-p4) interactivos.

G2LauncherWrapper

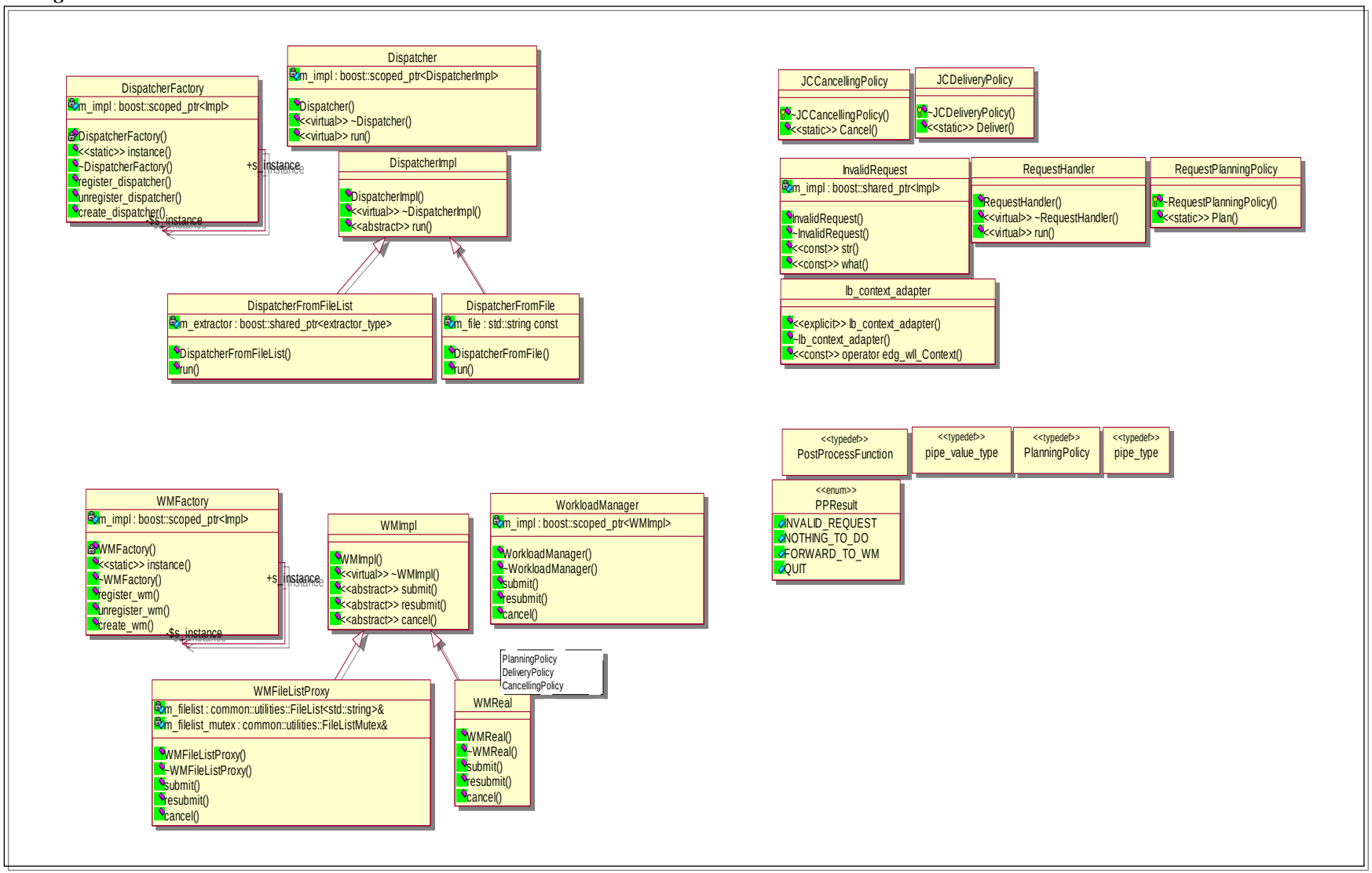
- Wrapper del Application Launcher propio de las aplicaciones mpich-g2

CannotCreateJobWrapper

- Excepción

7.5.Manager (Scheduling Agent)

7.5.1. Diagramas UML



WorkloadManager

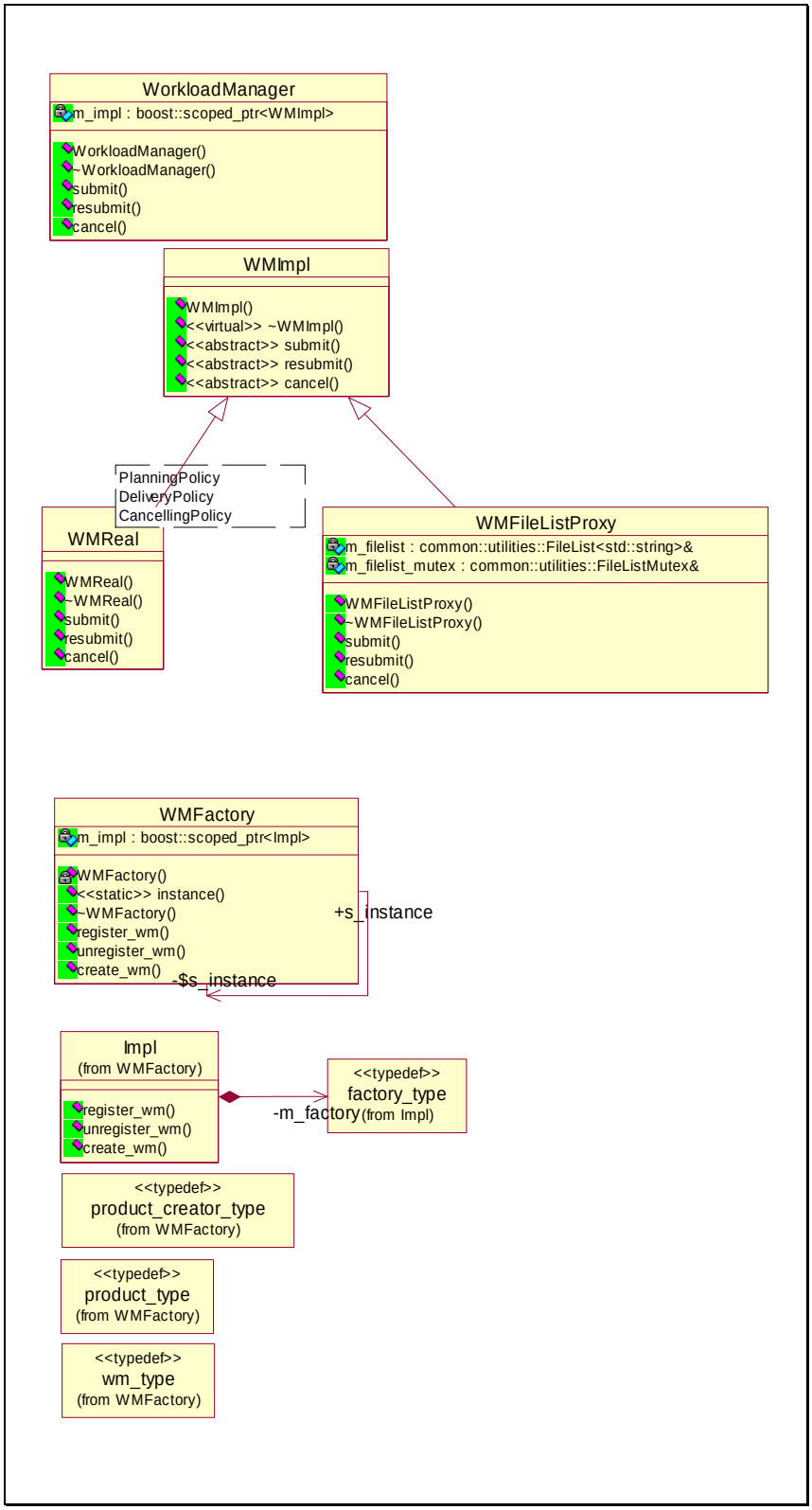


Figure 18, WorkloadManager

Dispatcher

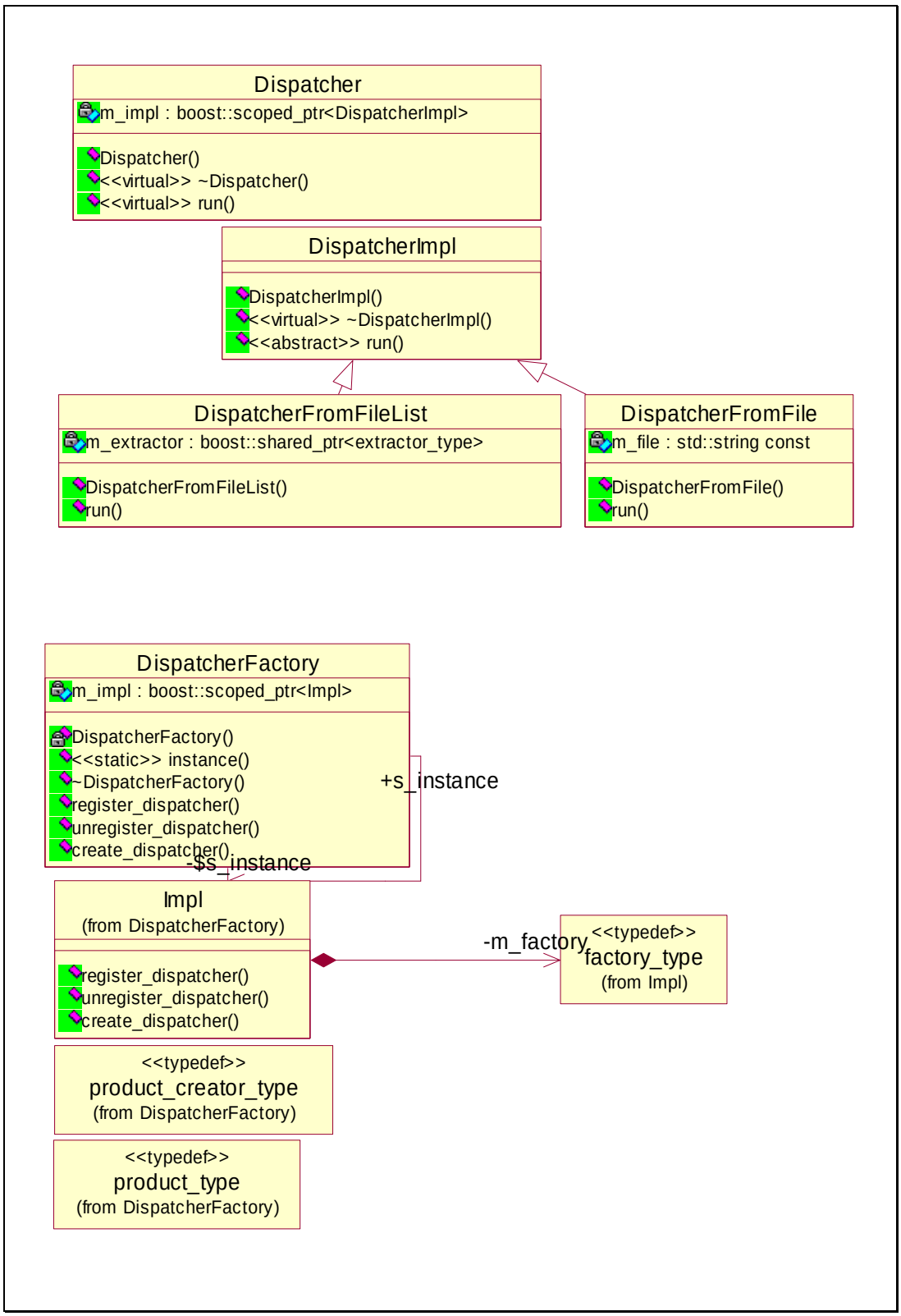


Figure 19, Dispatcher

Varios

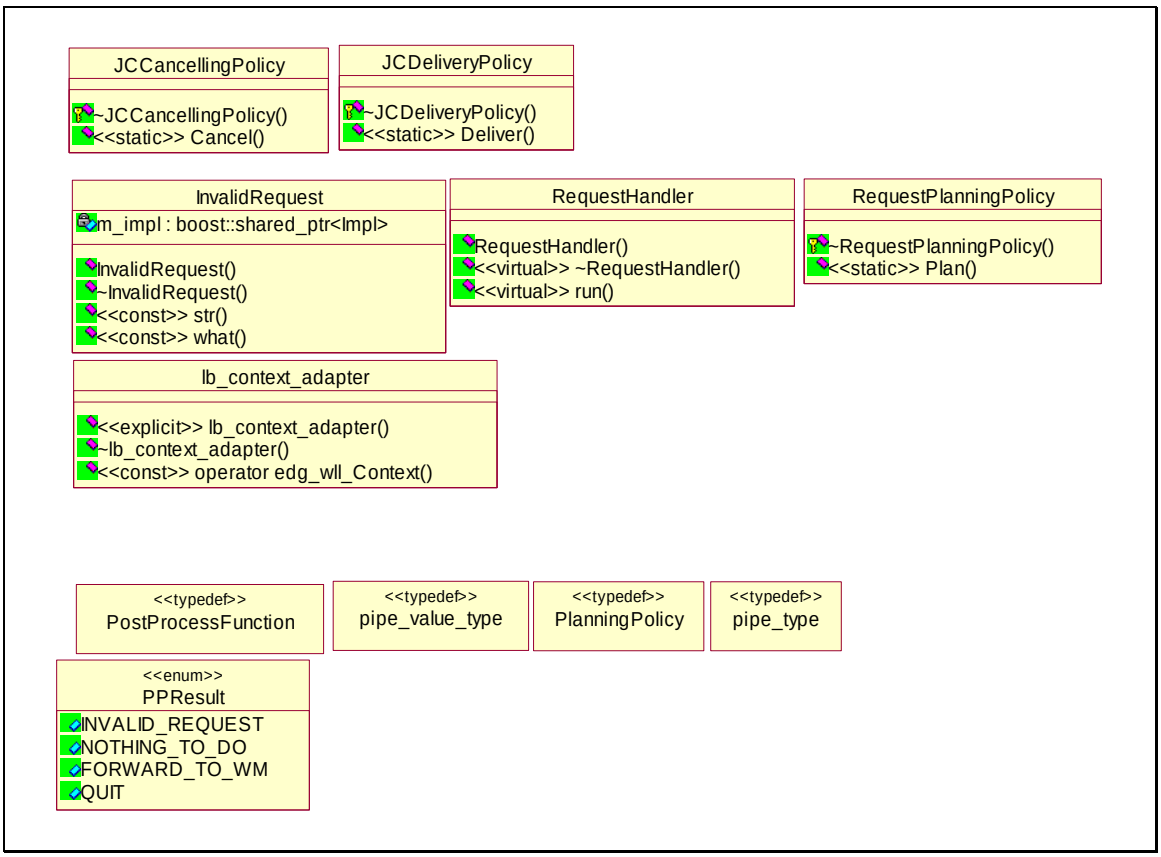


Figure 20, Others and various

7.5.2. Clases

Dispatcher

- Clase que representa un manejados de un trabajo de la cola para realizar las funciones adecuadas. Tiene asociado una implementacion particular dependiendo donde cómo se almacene la cola de trabajos

DispatcherImpl

- Implementación abstracta, sirve de base para las implementaciones reales y especifica una interfaz que deben seguir éstas.

DispatcherFromFile

- Manejador de trabajos desde ficheros

DispatcherFromFileList

- Lista de manejador Manerador de trabajos

DispatcherFactory

- Constructor de manejadores.

WorkloadManager

- Representa un WM abstracto

WmImpl

- Clase abstracta que representa una implementacion

WmReal

- Representa el Scheduling Agent de nuestra arquitectura.

JCCCancellingPolicy

- Politicas de cancelación de trabajos.

JCDDeliveringPolicy

- Politicas de envio de trabajos.

7.5.3. Ficheros

CommandAndManipulation.*

- Agrupación de los commandos de ejecución

7.6.PlugIn

7.6.1. Diagramas UML

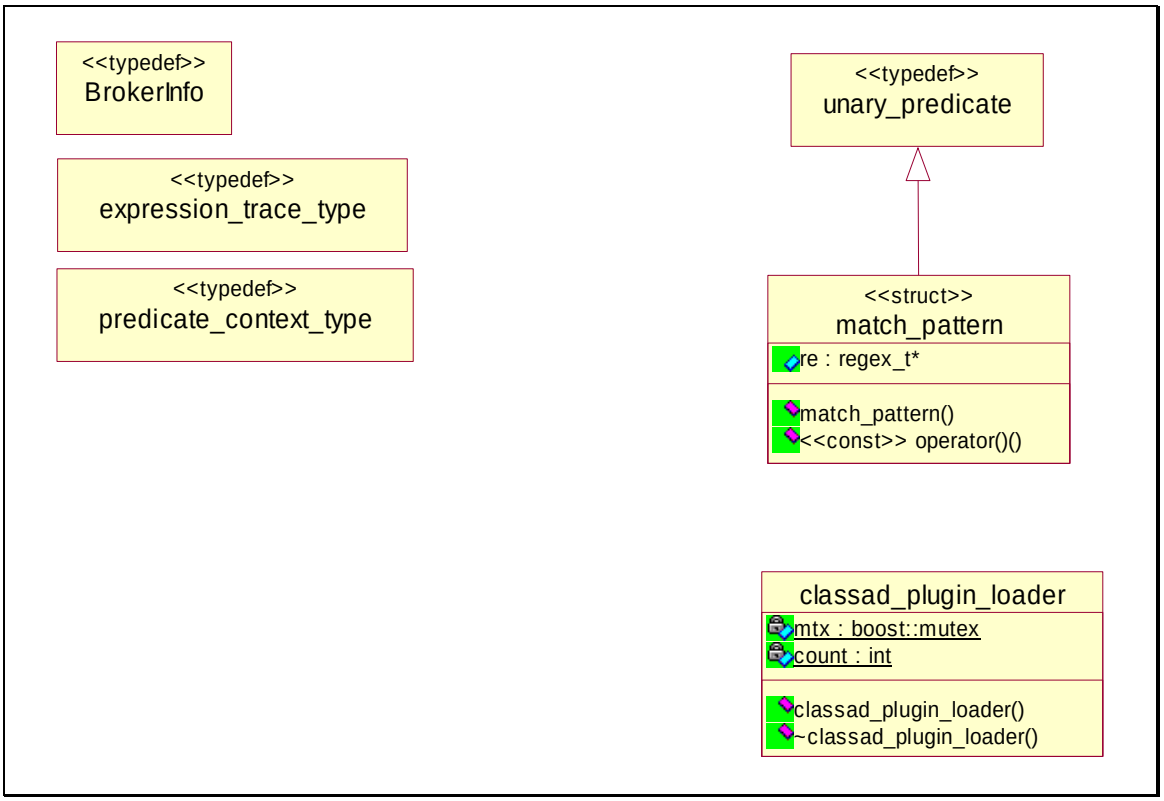


Figure 21, Overall Architecture

7.6.2. Clases

Classad_plugin_loader

- Cargador de plugins para ser utilizados por el MatchMaker

Cg-Classad-plugin

- Plugin específico para la utilización de herramintas de monitorización y *postprocessing* de Crossgrid

8. CONCLUSIONES Y TRABAJO FUTURO

En este trabajo se ha realizado una presentación de las características principales que tienen los sistemas grid, y la arquitectura de los mismos, basándonos en el que es el estándar de facto ahora mismo, Globus. Se hace especial hincapié en el área de la planificación y la gestión de recursos en estos sistemas, ya que es un área muy importante en la que se está trabajando activamente.
Posteriormente se describe cuál es la arquitectura y los principales componentes del sistema de gestión de recursos que hemos desarrollado en el proyecto *Crossgrid*. El *CrossBroker* representa un *super-planificador* que soporta de manera automática y eficaz todo tipo de tareas, pero que está especialmente orientado a los trabajos paralelos en un entorno distribuido y multiusuario.
Nuestro sistema consta de varios componentes, siendo los principales el *Scheduling Agent, el Resource Searcher* y el *Aplication Launcher*. También existen varios módulos que son necesarios para permitir ejecutar las tareas de los anteriores, como el *User Access Module,* el *Queue Manager,* y el *Job Controller.*

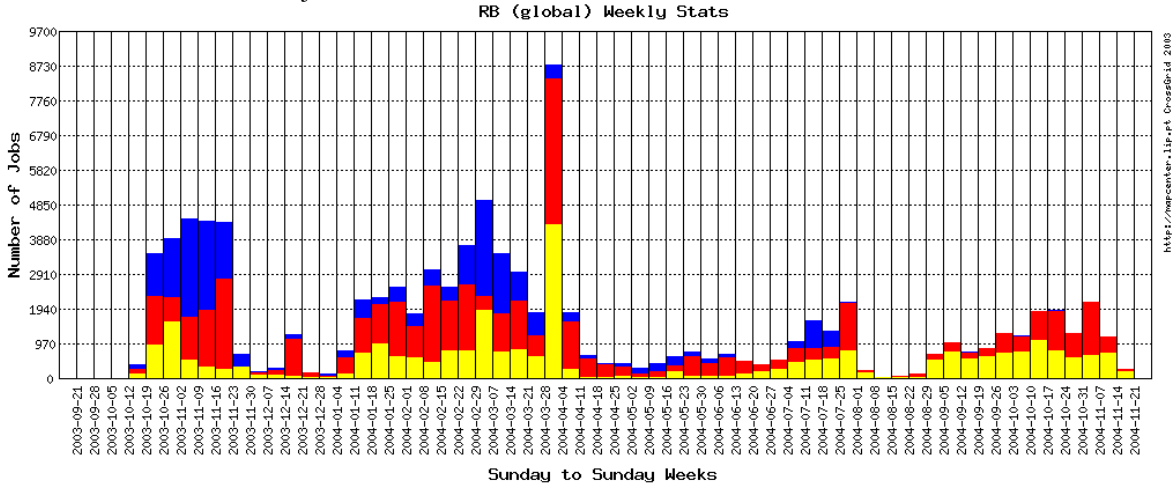
El *User Access Module*, se encarga de hacer de interfaz con los usuarios a través de un API y unos comando definidos. Los trabajos recibidos son gestionados en una cola persistente por el *Queue Manager*, que hace de repositorio. El *Scheduling Agent* es el elemento central que lleva el control de las acciones para ejecutar los trabajos. Éste contacta con el *Resource Searcher* para obtener los recursos disponibles dependiendo del tipo y los requerimientos de cada trabajo. Finalmente a través del *Application Launcher* específico se realiza el envió fiable del trabajo a los recursos correspondientes. El *Job Controller* hace de interfaz con los sistemas subyacentes como Condor-G para ejecutar la monitorización de los trabajos.

El *CrossBroker* utiliza software como globus, condor-g, y está desarrollado sobre el Resource Broker de Datagrid de manera que constituye una extensión al mismo, proporcionando la funcionalidad deseada para ejecutar trabajos simples, y extendiéndolo para ejecutar también trabajo paralelos (mpich-p4, y mpich-g2).

Nuestro software se ha integrado con diferentes herramientas dentro de Crossgrid. Se ha desarrollado una interfaz general de plugins, que ha sido implementada para la monitorización de los elementos del testbed dando una información más completa. Estas herramientas de monitorización permiten dar predicciones sobre el comportamiento futuro, lo que puede mejorar la planificación de los trabajos.
Además se ha integrado con el *Migrating Desktop*, un escritorio virtual que permite un interfaz amigable para el usuario final del grid.

Nuestro sistema de gestión de recursos está siendo utilizado en la actualidad por los diferentes usuarios de las cuatro aplicaciones claves de Crossgrid: biomedicina, análisis distribuido en física de altas energías (HEP), prevención de inundaciones, y análisis de polución. A través de nuestro *broker* se pueden mandar trabajos a las máquinas del *testbed* distribuido que sigue la arquitectura presentada , y que reúne a 21 partners de 11 países diferentes.

Como estadística podemos mostrar los datos recogidos por el sistema de monitorización del testbed [58] en cuanto a número de trabajos enviados a través de nuestro CrossBroker:



En la gráfica se muestran los trabajos enviados agrupados por semanas, desde Octubre de 2003 hasta Diciembre de 2004. En azul se muestran los trabajos recibidos, en rojo aquellos que se han ejecutado satisfactoriamente, y en amarillo aquellos cuyo usuario ha consultado su salida.
Como podemos observar se ha han lanzado más de 80.000 trabajos con alto grado de éxito. Conforme nuestro software iba estando más maduro los trabajos fallados (Diferencia entre el área azul y la Roja) van disminuyendo, hasta un éxito de casi un 100% .

En la actualidad nuestro software está siendo evaluado para su integración dentro del proyecto europeo EGEE (Enabling Grids for E-Science in Europe). Este proyecto del sexto programa marco, tiene como objetivo establecer un grid a nivel europeo y de otros países como Rusia, Israel, etcétera, involucrando hasta 70 partners diferentes.

Como trabajo futuro, la tarea inmediata consistiría en utilizar el futuro globus 4.X, todavía en desarrollo, y que se basa en una arquitectura de servicios. Éste es el estándar que parece se va a seguir a partir de su *release* oficial y en la que se van a basar los desarrollos. Nuestra arquitectura permite utilizar globus a través de condor-g con lo que éste último también debe soportarlo para que sea factible su utilización directa. El resto de elementos son adaptables con algunos cambios, especialmente en la interfaz de usuario, que ahora se basa en globus 2.
En cuanto a nueva funcionalidad, se está trabajando en el soporte automático para la planificación y ejecución de *workflows* de trabajos modelados con DAGS, y a mejorar el soporte a la interactividad de los trabajos. Adicionalmente, se están evaluando la integración de sistemas de accounting, y de reserva global y en avance de recursos. Éste último es un campo que en la actualidad no está muy maduro, y en el que probablemente va a haber más investigación en los próximos años.

9. REFERENCIAS

[1] **The Anatomy of the Grid: Enabling Scalable Virtual Organizations**. I. Foster, C. Kesselman, S. Tuecke. *International J. Supercomputer Applications*, 15(3), 2001.

[2] **A resource management architecture for metacomputing systems**. K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke.

[3] K. Czajkowski, I. Foster, C. Kesselman., V. Sander, and S. Tuecke**. SNAP: A protocol for negotiating service level agreements and coordinating resource management in distributed systems**. In D. Feitelson and L. Rudolph, editors, Job Scheduling Strategies for Parallel Processing (Proceedings of the Eighth International JSSPP Workshop; LNCS #2537), pages 153–183. Springer-Verlag, 2002

[4] R. Raman, M. Livny, and M. Solomon. **Matchmaking: Distributed resource management for high throughput computing**. In Proceedings of the Seventh IEEE International Symposium on High-Performance Distributed Computing (HPDC-7), 1998.

[5] K. Czajkowski, I. Foster, and C. Kesselman. **Co-allocation services for computational Grids**. In Proceedings of the Eighth IEEE International Symposium on High Performance Distributed Computing (HPDC-8), August 1999.

[6] I. Foster, M. Fidler, A. Roy, V. Sander, and L. Winkler. **Endto-end quality of service for high-end applications**. Computer Communications, Special Issue on Network Support for Grid Computing, 2002.

[7] F. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao. **Application-level scheduling on distributed heterogeneous networks**. In Proceedings of SuperComputing (SC'96), 1996.

[8] G. Aloisio and M. Cafaro. **Web-based access to the Grid using the Grid Resource Broker Portal. Concurrency and Computation: Practice and Experience**, Special Issue on Grid Computing Environments, 14:1145–1160, 2002.

[9] K. Kurowski, J. Nabrzyski, and J. Pukacki. **User preference driven multiobjective resource management in Grid environments**. In Proceedings of the First IEEE International Symposiumon Cluster Computing and the Grid (CCGrid'01), May 2001.

[10] S. J. Chapin, D. Katramatos, J. F. Karpovich, and A. S. Grimshaw. **Resource management in Legion. Future Generation Computing Systems**, 15:583–594, October 1999.

[11] F. Berman. High performance schedulers. In Ian Foster and Carl Kesselman, editors, The Grid: **Blueprint for a New Computing Infrastructure**, chapter 12, pages 279–309. Morgan Kaufmann, 1999.

[12] Jennifer M. Schopf and Bill Nitzberg. **Grids: The top ten questions**. Scientific Programming, Special Issue on Grid Computing, 10(2):103–111, August 2002.

[13] **Global Grid Forum (GGF).** http://www.ggf.org.

[14] **Global Grid Forum Scheduling and Resource Management Area** (SRM). http://www.mcs.anl.gov/~jms/ggf-sched.

[15] **Grids y e-Ciencia.** Jesús Marco, Jornadas Técnicas redIris, Mallorca 2003.

[16] **"The Grid: Blueprint for a New Computing Instraestructure".** I.Foster, C. Kesselman. Morgan-Kaufmann 1999

[17] **Globus.** http://www.globus.org

[18] **The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration.** I. Foster, C. Kesselman, J. Nick, S. Tuecke, Open Grid Service Infrastructure WG, Global Grid Forum, June 22, 2002

[19] **Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List Profile (RFC 3280)**, http://www.ietf.org/rfc/rfc3280.txt

[20] **Generic Security Service Application Program Version 2, Update 1 (RFC 2743),** http://www.ietf.org/rfc/rfc2743.txt

[21] Neuman, B. Clifford, **"Proxy-Based Authorization and Accounting for Distributed Systems"**, In Proceedings of the 13th International Conference on Distributed Computing Systems, pages 283-291, May 1993.

[22] **Internet X.509 Public Key Infrastructure (PKI) Proxy Certificate Profile, (RFC 3820),** http://www.ietf.org/rfc/rfc3820.txt

[23] **A Resource Management Architechture for Metacomputing Systems**. K. Czajkowski, Ian Foster *y* Nicholas Karonis *y* Carl Kesselman, Stuart Martin *y* Warren Smith *y* Steven Tuecke. Proc. IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing, pg. 62-82, 1998.

[24] **GENIAS Software GmbH. CODINE: Computing in distributed networked environments**, 1995.http://www.genias.de/genias/english/codine.html.

[25] The PSCHED APIWorking Group. **PSCHED: An API for parallel job/resource management** version 0.1, 1996. http://parallel.nas.nasa.gov/PSCHED/.

[26] M. Litzkow, M. Livny, and M. Mutka. **Condor - a hunter of idle workstations**. In *Proc. 8th Intl Conf. on Distributed Computing Systems*, pages 104{111}, 1988.

[27] R. Henderson and D. Tweten. **Portable Batch System: External reference specification.** Technical report, NASA Ames Research Center, 1996.

[28] S. Zhou. **LSF: Load sharing in large-scale heterogeneous distributed systems**. In *Proc. Workshop on Cluster Computing*, 1992

[29] J. Weissman and A. Grimshaw. **A federated model for scheduling in wide-area systems**. In *Proc.5th IEEE Symp. on High Performance Distributed Computing*, 1996.

[30] David A. Lifka. **The ANL/IBM SP scheduling system**. In *The IPPS'95 Workshop on Job Scheduling Strategies for Parallel Processing*, pages 187{191, April 1995.

[31] **Grid Information Services for Distributed Resource Sharing**. K. Czajkowski, S. Fitzgerald, I. Foster, C. Kesselman. *Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10)*, IEEE Press, August 2001.

[32] S. Gullapalli, K. Czajkowski, C. Kesselman, and S. Fitzgerald. **The grid notification framework. Grid Forum Working Draft** GWD-GIS-019, June 2001. http://www.gridforum.org.

[33] **EU-DATAGRID project**, www.eu-datagrid.org

[34] **EU-CROSSGRID project**, www.eu-crossgrid.org

[35] **Task 1.1. Biomedical application** . Software Requirements Specification Draft, CG-Task1.1-SRS001-0.1-DRAFT_c.doc

[36] **Task 1.2. Flooding Crisis Support** . Software Requirements Specification Draft, CG-1.2-SRS001-0.1-DRAFT_c.doc

[37] **Task 1.3. Distributed Data Analysis in HEP** . Software Requirements Specification Draft, Task1[1].3srsv1.doc

[38] **Task 1.4. Air pollution, weater forecasting and sea wave modelling** . Software Requirements Specification Draft, CG-1.4-SRS-DRAFT.doc

[39] **Task 1.4b. Data Mining for Weather Forecasting** . Software Requirements Specification Draft, Task1[1].3srsv1.doc

[40] **Migrating Desktop**, http://ras.man.poznan.pl/crossgrid/

[41] F Giacomini, F.Prelz. **Definition of architecture, technical plan and evaluation criteria for scheduling, resource management, security and job description, http://server11.infn.it/workload-grid/docs/DataGrid-01-D1.2-0112-0-3.pdf**

[42] **Datagrid WP1 – Workload Management Installation Guide**, http://server11.infn.it/workload-grid/docs/DataGrid-01-TEN-0118-0_4-Document.pdf

[43] **D2.2 Architecture and Design Document**, http://edg-wp2.web.cern.ch/edg-wp2/docs/DataGrid-02-D2.2-0103-1_2.pdf

[44] **Job Description Language**, http://server11.infn.it/workload-grid/docs/DataGrid-01-TEN-0102-0_2-Document.pdf

[45] Rajesh Raman, Miron Livny, and Marvin Solomon, "**Matchmaking: Distributed Resource Management for High Throughput Computing**", Proc. of the Seventh IEEE International Symposium on High Performance Distributed Computing, July 28-31, 1998

[46] **Condor Classads**, http://www.cs.wisc.edu/condor/classad/

[47] **R-GMA: Relational Grid Monitoring Architecture**, http://www.r-gma.org/

[48] J. Frey, T. Tannenbaum, I. Foster, M. Livny, S. Tuecke, "**Condor-G: A Computation Management Agent for Multi-Institutional Grids**". Journal of Cluster Computing, vol. 5, pages 237-246. 2002.

[49] N. Karonis, B. Toonen, I. Foster, **"MPICH-G2: A Gridenabled implementation of the message passing interface"**. Journal of Parallel and Distributed Computing, 62, pp.551-563. 2003.

[50] . K. Czajkowsi, I. Foster, C. Kessekman. "**Co-allocation services for computational Grids".** Proceedings of the Eighth IEEE Symposium on High Performance Distributed Computing, IEEE Computer Society Press, Silver Spring MD, 1999.

[51] **Glue Schema specification version 1.2 draft 1** (29 Oct. 2004), http://infnforge.cnaf.infn.it/docman/view.php/9/65/GLUEInfoModel_V_1_2_draft_1.pdf

[52] **The Globus Alliance. Glue Schema page**. http://www.globus.org/mds/glueschemalink.html.

[53] **Ganglia, distributed monitoring tool**. http://ganglia.sourceforge.net/

[54] **RFIO: Remote File Input/Output** http://doc.in2p3.fr/doc/public/products/rfio/rfio.html

[55] **LCG2 Users guide**, https://edms.cern.ch/file/454439//LCG-2-UserGuide.html

[56] **Postprocessing Monitoring Tools**, http://www.eu-crossgrid.org/user_manuals/CG5.2-TEM-v0.3-CYF782-UserManualGMDAT.pdf

[57] **Crossgrid WP3.2 Monitoring Plugin Support**, http://savannah.fzk.de/distribution/crossgrid/crossgrid/wp3/wp3_2-scheduling/docs/CG3.2-v1.2-MonitoringPlugin.pdf

[58] **Crossgrid Mapcenter**, http://mapcenter.lip.pt/