UNIVERSIDAD POLITÉCNICA DE VALENCIA

MASTER EN COMPUTACIÓN PARALELA Y DISTRIBUIDA

WINGS: Worflow In Next generation Grids

Autor: Miguel Caballer Fernández

Director: Dr. Ignacio Blanquer Espert

Contenido

1 Introducción

Actualmente con el desarrollo de las tecnologías grid se han ido desplegando una serie de infraestructuras a nivel mundial que han permitido el acceso a los investigadores a un número enorme de recursos.

Muchos de los investigadores no tienen los conocimientos necesarios para el desarrollo de aplicaciones que sean capaces de aprovechar todos estos recursos que se han puesto a su alcance. Además la existencia de diferentes middlewares e infraestructuras desplegadas aumenta la complejidad para aprovechar el mayor número de recursos posibles.

La mayoría de las aplicaciones científicas implican una sucesión de múltiples etapas que deben coordinarse, en las que los resultados de unas tareas son los datos de entrada de otras tareas. Esto provoca que se creen una serie de grafos de dependencias entre las diferentes tareas, que pueden llegar a ser muy complejos haciendo que el proceso de ejecución manual de estas aplicaciones de sea muy tedioso.

Por eso se empieza a aplicar el concepto de workflow (que aparece en los años 70 en el ámbito empresarial) para la automatización de procesos en ámbitos científicos Con la aparición de las tecnologías grid y la dificultad de la migración de aplicaciones se adapta el concepto de workflow a los entornos grid. Las herramientas de workflow en grid, tratan de facilitar la migración y diseño de aplicaciones grid, proporcionando una capa de alto nivel que permita aislar a los investigadores de la complejidad interna del funcionamiento de los middleware grid.

Aunque en la actualidad Globus Toolkit [1] puede ser visto como el estándar “de facto” de los middleware grid, existen muchas otras infraestructuras desplegadas usando diferentes middlewares. Algunos de ellos están construidos usando el Globus Toolkit como base, pero añadiendo nuevas características como gLite [2] o el OSG Software Stack [3], y otros son utilizados en entornos empresariales como Fura [4] o directamente usando servicios web. Además muchos investigadores también tienen acceso a recursos locales ejecutando comandos directamente o usando algún sistema de colas (Local Resource Management Systems, LRMS) como PBS/Torque, SGE, LSF, etc. Uno de los principales objetivos de diseño del sistema workflow es poder acceder a todos estos entornos de ejecución de forma transparente permitiendo el acceso a todas las infraestructuras de computación al alcance de usuario.

Los middleware grid están en continua evolución, por tanto es importante para un sistema workflow la capacidad de adaptarse a estos cambios y poder añadir nuevos entornos de ejecución para acceder a las infraestructuras disponibles actualmente. Por tanto la extensibilidad es una característica clave para un sistema workflow.

En la presente tesis se plantea como objetivo general el desarrollo de un sistema workflow para la ejecución de tareas en diferentes sistemas de ejecución, usando un sistema de definición de workflows de alto nivel y que además permita, de forma sencilla, la adición de nuevos sistemas de ejecución por parte de terceros desarrolladores.

Las aportaciones más destacables de esta tesis son las siguientes:

- Definición de un lenguaje de definición de workflows de alto nivel, que permite al investigador definir sus tareas de forma sencilla, aislándose de los detalles de los recursos donde se vayan a ejecutar. Además este lenguaje es extensible para permitir la adición de nuevos sistemas de ejecución de forma sencilla.
- Diseño de un sistema workflow modular que permite la fácil integración de nuevos sistemas de ejecución, para permitir la evolución del sistema en el futuro.
- Implementación del motor de ejecución con una serie de módulos que permitan ejecutar en sistemas de ejecución grid como GT2 o Fura, otros sistemas como SSH y uno adicional que permite la ejecución de sub-workflows.

El contenido de la tesis está estructurado de la siguiente manera: En la introducción se comentan los antecedentes y los objetivos propuestos en esta tesis. En el estado del arte se comentan todas las tecnologías utilizadas en el desarrollo de la misma: grid y workflow. En la parte de grid se describen los dos middlewares que han sido implementados en el motor de ejecución del sistema. En el apartado de workflow se muestran las diferentes aproximaciones desarrolladas en la actualidad para comprobar sus puntos fuertes y sus defectos para poder desarrollar un sistema workflow que permita, cumplir los objetivos propuestos. En el tercer apartado se muestra el sistema WINGS, primero describiendo los conceptos utilizados, luego el lenguaje de definición de workflows, y por último el motor de ejecución. El cuarto apartado muestra un caso de uso utilizado para demostrar la funcionalidad del sistema en un caso concreto de una aplicación real. Finalmente se muestran las publicaciones producidas por el trabajo desarrollado en la tesis y las conclusiones y trabajos futuros.

1.1 Objetivos

Como objetivo principal de esta tesis de master se propone desarrollar una herramienta workflow que permita la ejecución de aplicaciones científicas definidas a través de un workflow con tres características fundamentales que no proporcionan las herramientas workflow actuales:

- Definición del workflow sencilla y de alto nivel.
- Ejecución multi-entorno: Todas las tareas puedan ser ejecutadas en diferentes infraestructuras, ya sean con acceso local, o mediante uso de sistemas LRMS o cualquier middleware grid, de forma transparente al usuario.
- Facilidad de extensión: De forma sencilla permita la adición de nuevos sistemas de ejecución por parte de terceros desarrolladores, permitiendo la continua evolución.

2 Estado del Arte

2.1 Tecnología Grid

El término Grid [5] fue acuñado en la mitad de la década de los 90 para denotar una infraestructura de computación distribuida para los ámbitos de la ciencia y la tecnología. Inicialmente el concepto de grid era una infraestructura hardware y software que proporciona acceso dependiente, consistente, generalizado y económico a capacidades computacionales de altas prestaciones.

La idea de un Grid es que sea una arquitectura fiable, consistente y accesible. Aunque hay múltiples definiciones un sistema grid puede resumirse mediante las siguientes características [6]:

- *Recursos coordinados que no están sujetos a un control centralizado*: un sistema grid está integrado por un conjunto de recursos en diferentes dominios de administración y debe respetar sus políticas internas de seguridad.
- *Uso de interfaces y protocolos estándar, abiertos, y de propósito general*: Debe usar protocolos abiertos para evitar crear sistemas específicos de una sola aplicación y facilitar la integración con otros sistemas.
- *Para proporcionar calidad de servicio no trivial*: un sistema grid debe permitir el uso de los recursos que lo integran de manera coordinada para proporcionar varios parámetros de calidad de servicio, como pueden ser el tiempo de respuesta, disponibilidad, alta productividad, etc. no alcanzable de forma independiente por los reursos individuales.

De estas definiciones se puede extraer un conjunto de características indispensables deben estar presentes en las herramientas Grid:

- Gestión local de los recursos: Interconectar recursos en diferentes dominios de administración respetando sus políticas internas de seguridad y su software de gestión de recursos en la Intranet
- Transparencia: Los datos en los diferentes formatos y tipos de ficheros han de ser integrados en una base de datos virtual, de manera que el usuario pueda manejarlos con independencia de la fuente de que provengan.
- Ubicuidad: Posibilidad de acceder a los recursos desde cualquier sitio, lo que implica un middleware preparado para soportar una amplia diversidad de plataformas y sistemas operativos.
- Uniformidad: El usuario de Grid ha de ver los diferentes recursos y datos como un único recurso.
- Organizaciones Virtuales: Los usuarios del grid se agrupan en organizaciones virtuales (VO de su acrónimo en inglés) multi-institucionales para resolver problemas en las mismas áreas de interés, gestionando de forma conjunta la mayor parte de las políticas de autorización.
- Fiabilidad: Disponibilidad permanente, que exige especial cuidado en la tolerancia a fallos y la redundancia, con un almacenamiento robusto.
- Seguridad: Es fundamental tanto en los datos como en el acceso a los recursos compartidos.

Más adelante y con la aparición de los Servicios Web el concepto de Grid se generaliza y aparece el Servicio Grid. El término Servicio Grid se usa en diferentes contextos y en general es cualquier servicio ofrecido a los clientes en un entorno Grid. Un Servicio Grid también puede ser visto como un Servicio Web adaptado a los requerimientos de los entornos Grid y con las características antes comentadas que debe tener una arquitectura Grid. Para ello aparece OGSA (Open Grid Services Architecture) [7], [8] para normalizar los servicios habituales en una aplicación Grid (gestión de tareas, de recursos, seguridad...) definiendo los interfaces a estos servicios. Más adelante aparece WSRF (Web Services Resource Framework) [9] como implementación de los servicios Web requeridos por OGSA. WSRF proporciona un conjunto de operaciones que los servicios web compatibles pueden implementar para convertirse en servicios web con estado.

2.1.1 Globus Toolkit

En la actualidad Globus Toolkit (GT) puede ser visto como el estándar de facto de los middleware grid. El Globus Toolkit es un proyecto open source que incluye una serie de servicios software y librerías para la monitorización, descubrimiento, gestion de recursos y gestión de ficheros. Todo ello dentro de un entorno de seguridad controlado.

La primera versión de Globus apareció en 1998 en el "Argonne National Laboratory". En 2002 apareció la versión 2.0 (GT2) que supuso un importante avance en las tecnologías grid y que todavía es ampliamente utilizado en numerosas infraestructuras de todo el mundo. Tras la evolución de los Servicios Web y la orientación a servicios de las tecnologías grid, apareció la versión 3.0, que tuvo poco éxito y un corto periodo de vida, ya que la implementación de los servicios web ofrecida no respetó ningún estándar perteneciente a una arquitectura orientada a servicios. Este inconveniente produjo el rápido desarrollo e implantación de la versión 4 (GT4). A finales del año 2009 apareció la versión 5 del Globus Toolkit, en la cual se descarta toda la parte relacionada con Servicios Web y se vuelve a un sistema similar al usado en el GT2, que es la versión actual en desarrollo.

2.1.1.1 GT2

Globus Toolkit 2 [10] es un conjunto de servicios y librerías software que proporcionan un entorno Grid permitiendo el lanzamiento de aplicaciones Grid. Globus provee características de seguridad, descubrimiento de información, manejo de recursos, manejo de datos, comunicación, detección de fallos y portabilidad.

GT2 está compuesto de una serie de módulos. Cada módulo define una interfaz, que es utilizada por los servicios de nivel superior, y provee una implementación que utiliza las operaciones apropiadas a bajo nivel para implementar dichos módulos. Los módulos de los que se compone GT2 son los siguientes:

- *Localización y asignación de recursos*: Provee mecanismos para expresar los requerimientos de recursos de una aplicación, para identificar recursos que cumplan los requerimientos, y la planificación del uso de los recursos una vez localizados. Los mecanismos de localización son necesarios ya que, en general, las aplicaciones no conocen la localización exacta de los recursos necesarios.
- *Comunicaciones*: Provee de los mecanismos básicos de comunicación. Estos mecanismos deben permitir una implementación eficiente de un amplio rango de métodos de comunicación, incluyendo el paso de mensajes, llamada a procedimiento remoto, etc.
- *Servicio unificado de información sobre recursos*: Provee mecanismos para obtener información en tiempo real sobre la estructura y estado del meta-sistema global.
- *Interfaz de autenticación*: Provee métodos de autenticación y autorización que puedan ser utilizados para validar tanto a usuarios como a recursos.
- *Creación de procesos*: Es utilizado para iniciar la computación en un recurso una vez ha sido localizado y asignado. Esta tarea incluye preparar los ejecutables, creación de un entorno de ejecución, lanzar el ejecutable, paso de argumentos, integrar el nuevo proceso con el resto de la computación, y controlar la finalización del proceso.
- *Acceso de datos*: Es el responsable de proveer acceso de alta velocidad a datos remotos. El módulo de acceso de datos de Globus permite abordar el problema de conseguir buenas prestaciones al acceder a sistemas de ficheros paralelos y sistemas E/S en red como el HPSS (High Performance Storage System).

Todos los módulos de Globus pueden ser utilizados juntos para definir una máquina virtual. La visión de esta máquina global facilita el desarrollo de aplicaciones y aumenta la portabilidad, permitiendo a los programadores pensar en un conjunto de recursos geográficamente distribuidos y heterogéneos como entidades unificadas.

Ahora vamos a describir con más detalle los componentes que forman el Globus Toolkit 2:

2.1.1.1.1 Seguridad

El Globus Toolkit usa GSI (Grid Security Infrastructure) para permitir autenticación y comunicaciones sobre una red abierta de forma segura. GSI provee varios servicios para el Grid que permiten autenticación única.

Las motivaciones principales de GSI son:

- Comunicación segura entre los elementos del grid.
- Soporte de registro único (single sign-on), incluyendo delegación de credenciales para realizar cómputos que involucren múltiples recursos y sitios. La funcionalidad Single Sign-On (SSO) es un procedimiento de autenticación que habilita al usuario para acceder a varios sistemas con una sola instancia de identificación.

- Ofrecer seguridad a través de diversos dominios administrativos. No existe un sistema de seguridad centralizado.

GSI se basa en la encriptación de clave pública (o encriptación asimétrica), certificados X.509, y el protocolo de comunicaciones SSL (Secure Sockets Layer). Extensiones para estos estándares han sido añadidas para permitir autenticación única y delegación.

GSI proporciona la capacidad de delegación: una extensión del protocolo SSL que reduce el número de veces que el usuario debe introducir su clave de acceso. La computación Grid requiere el uso de diversos recursos (los cuales requieren autenticación), o la necesidad de usar agentes (locales o remotos) que pidan servicios en nombre del usuario, la necesidad de reintroducir la clave del usuario para cada uso es evitada creando un *proxy*.

Un *proxy* consiste en un certificado (con la clave pública incluida dentro) y una clave privada, el cual contiene la identidad del usuario. El certificado es firmado por el usuario. El certificado también incluye una marca temporal después de la cual el *proxy* dejara de ser válido.

2.1.1.1.2 Manejo de Datos

Globus proporciona un protocolo de transferencia de ficheros de altas prestaciones, seguro, fiable y optimizado para redes de área ancha. El GridFTP está basado en el protocolo FTP. Las características principales del GridFTP son:

- Seguridad GSI en los canales de control y datos.
- Múltiples canales de datos para transferencias de ficheros en paralelo.
- Transferencias parciales de ficheros.
- Transferencias entre dos servidores "third-party".
- Canales de datos autentificados.
- Canales de datos reutilizables.

2.1.1.1.3 Control de Recursos

Globus incluye un conjunto de servicios llamados GRAM (Globus Resource Allocation Manager). GRAM simplifica el uso de sistemas remotos mediante el uso de una interfaz estándar para la petición y uso de recursos remotos para la ejecución de "trabajos". Entendiendo por "trabajos" el lanzamiento de un ejecutable con una serie de parámetros de entrada, un conjunto de ficheros de entrada, y unos resultados. El uso principal de GRAM es el envío y control de trabajos de forma remota. Su uso más frecuente es para el desarrollo de aplicaciones distribuidas.

GRAM reduce el número de mecanismos necesarios para el uso de recursos remotos. Los sistemas locales pueden usar diferentes métodos de control (planificadores, sistemas de colas,...), pero los usuarios y desarrolladores sólo necesitan conocer el uso de GRAM para acceder a todos ellos.

GRAM provee un mecanismo de autorización simple basado en las identidades GSI y un mecanismo para mapear identidades GSI con usuarios de los recursos locales.

Para facilitar la portabilidad y la ejecución de aplicaciones que usan E/S al entorno Grid, Globus proporciona otro servicio llamado GASS (Global Access to Secondary Storage). El GASS está diseñado para evitar la necesidad de autentificarse manualmente por medio de sistemas como ftp, e instalar sistemas de ficheros distribuidos. El API está diseñado para reutilizar programas que utilizan la E/S estándar de C sin necesidad de modificaciones de código. GASS también permite el envío de los ejecutables a todos los recursos Grid utilizados.

2.1.1.1.4 Servicios de Información

Globus posee una serie de servicios destinados a mantener y proveer información sobre los recursos del Grid para permitir a las aplicaciones elegir aquellos que sean más adecuados a sus necesidades, sin necesitar un conocimiento previo de la estructura del Grid. Para ello provee del servicio MDS (Monitoring and Discovery Service).

MDS usa un entorno extensible para el manejo de información estática y dinámica sobre el estado del Grid y todos sus componentes: redes, nodos de computación, sistemas de almacenamiento.

MDS provee de herramientas para construir una infraestructura de información para Grid basada en LDAP (Lightweight Directory Access Protocol). MDS define una forma de presentar los datos a los usuarios (usando el protocolo LDAP con algún esquema en particular) usando dos tipos de servidores para mantener la información GIIS y GRIS.

El GRIS (Grid Resource Information Service) provee un sistema uniforme para informar sobre la configuración, capacidades y estado de los recursos del Grid. El GRIS es un servicio de información distribuida que puede responder preguntas, sobre un recurso en particular, accediendo directamente a un proveedor de información integrado dentro los servicios de Globus en un recurso Grid.

El GIIS (Grid Index Information Service) se encarga de combinar varios servidores GRIS para proveer de una visión global del sistema Grid. El GIIS provee un mecanismo para identificar recursos para un uso determinado.

Globus provee de un sistema para mantener información sobre colecciones de ficheros repartidas entre los diferentes recursos del Grid. El Replica Catalog soporta el control de réplicas de ficheros manteniendo un mapeo entre el nombre "*lógico*" de un fichero con las direcciones físicas de todas sus posibles copias. Esto permite mantener grandes volúmenes de información repartidos entre los diferentes recursos del Grid, pero manteniendo una visión global del mismo.

2.1.1.2 GT4

Globus Toolkit 4 [11] es un conjunto de servicios, contenedores y librerías orientada para el desarrollo de aplicaciones Grid de alto nivel. GT4 es un software de base, no orientado a usuarios finales, sino a desarrolladores de aplicaciones e integradores de sistemas.

- Servicios: Servicios encargados de proporcionar las funcionalidades de: control de ejecuciones (GRAM), acceso de datos y transferencia de ficheros (GridFTP, RFT, OGSA-DAI), replica de ficheros (RLS, DRS), monitorización y descubrimiento (Index, Trigger, WebMDS), manejo de credenciales (MyProxy, Delegación, SimpleCA). La mayoría de ellos son Servicios Web implementados en Java.
- Contenedores: tres contenedores para desplegar servicios desarrollados por los usuarios en Java, Python y C. Estos contenedores tienen soporte para las especificaciones de los Servicios Web, incluyendo WRSF, WS-Notification y WS-Security.
- Librerías: un conjunto de librerías en Java, Python y C que permite invocar tanto los servicios GT4 como los desarrollados por los usuarios.

2.1.1.2.1 Arquitectura

En la Figura 1 podemos ver los componentes que incluye. En ella se distinguen los componentes WS y los pre-WS que son los pertenecientes a GT2 que se mantienen por compatibilidad.

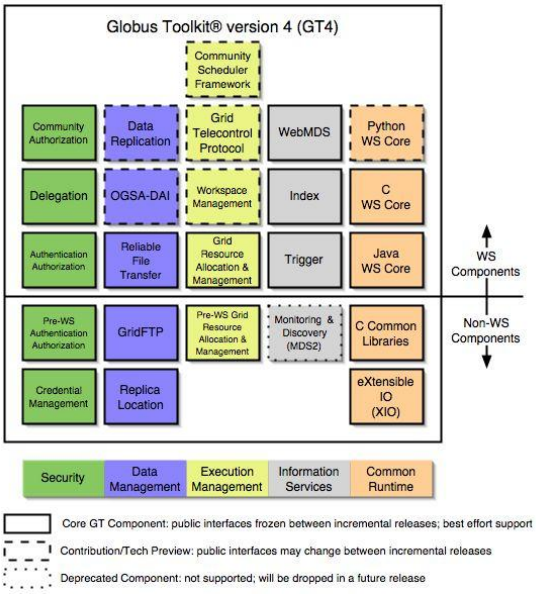

Figura 1. Arquitectura de Servicios de Globus 4.

Gestión de Ejecuciones

Para lanzar un trabajo o desplegar un servicio en un recurso remoto es necesario acceder al recurso, configurarlo para que se adapte a nuestras necesidades, enviar los archivos necesarios para la ejecución, lanzar, monitorizar y gestionar los resultados de la ejecución.

Grid Resource Allocation and Management (GRAM) se encarga de todas esas tareas permitiendo lanzar, monitorizar, y cancelar trabajos en recursos Grid. GT4 incorpora hasta dos implantaciones de GRAM:

- Construida usando interfaces WS (WS GRAM).
- Basada en un protocolo pre-WS (Pre-WS GRAM).

Transferencia de datos

GridFTP: GT4 al igual que su predecesor proporciona GridFTP como protocolo de transferencia de ficheros de altas prestaciones, seguro, fiable y optimizado para redes de área ancha.

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

Replica Location Service (RLS): es un sistema descentralizado para mantener y proveer acceso a la información sobre las diferentes ubicaciones de las réplicas de ficheros o conjuntos de datos.

Data Replication Servicer (DRS): combina RLS y GridFTP para proveer toda la funcionalidad necesaria para la replicación de datos.

OGSA-DAI: El objetivo de OGSA-DAI es proveer métodos uniformes para descubrir, acceder e integrar recursos de datos heterogéneos, principalmente bases de datos, en el Grid.

Monitorización y Descubrimiento de Servicios

WS Monitoring and Discovery System (MDS4): Se trata de un monitor de nivel de Grid basado en WSRF, usado para descubrimiento y manejo de recursos. Sirve para obtener un gran rango

de información relativa a los recursos básicos y colas, y puede servir de interfaz para monitorización de sistemas cluster como Ganglia.

Cada servicio basado en WSRF ofrece bastante información de monitorización sobre sí mismo, de tal manera que permite a WS MDS usar sus datos.

Igual que cualquier sistema de monitorización Grid, MDS4 ofrece un servicio de indexado donde los datos son recogidos y almacenados. Además, MDS4 almacena información en la base de datos Xindice, basada en XML, y tiene un interfaz web para poder ver los datos fácilmente.

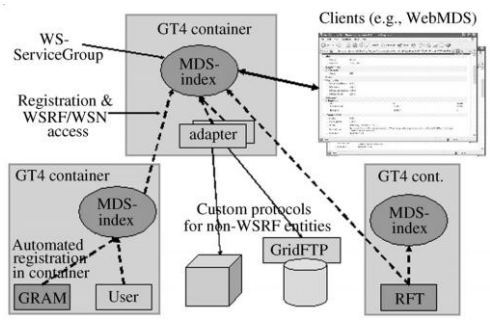

Figura 2. Monitorización y descubrimiento en GT4.

Seguridad

WS Authentication & Authorization: Contiene las librerías que hacen posible la autenticación la autorización para los componentes WS. Esto incluye funcionalidades como autenticación basada en X.509 en GSI SecureConversation, protección de mensajes, y todo un framework de autorización con varios mecanismos.

2.1.1.3 GT5

La última versión del Globus Toolkit aparece a finales de 2009. Se vuelve a utilizar una arquitectura similar a la versión 2, se elimina toda la parte de WS que se separa en un proyecto nuevo denominado Globus Crux [12].

En la parte de gestión de ejecuciones se desarrolla GRAM5 como una evolución de la versión 2 del servicio GRAM. En la transferencia de datos desaparece el Reliable File Transfer (RFT) y es sustituido por un nuevo servicio denominado Globus online [13]. La seguridad sigue basándose en el uso de certificados X509, pero añaden el uso de servidores MyProxy para el almacenamiento y obtención de credenciales de usuario. Finalmente, como se puede ver en la Figura 3, desaparece la parte de monitorización y descubrimiento de servicios, descartando las versiones anteriores de los servicios MDS.

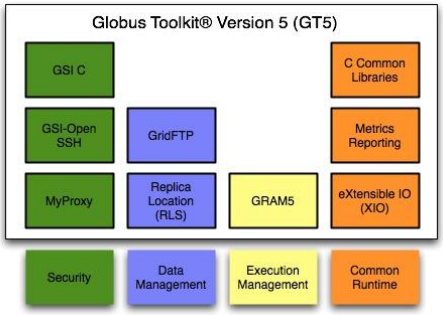

Figura 3. Arquitectura Globus Toolkit 5.

2.1.2 Fura

Fura [4] (anteriormente llamado InnerGRID) de la empresa GridSystems, en un middleware grid de entorno empresarial. Fura es un conjunto de herramientas que permite construir y

gestionar de forma conjunta una plataforma heterogénea de ordenadores. El objetivo de esta plataforma consiste en aprovechar los recursos de cómputo, haciendo colaborar a los ordenadores que la componen para realizar tareas de cálculo intensivo

2.1.2.1 Conceptos

La funcionalidad en Fura viene definida por tres conceptos principales: módulo, tarea y microtarea:

- Modulo: Un módulo es un script, librería o aplicación ejecutable que representan los trabajos que se van a lanzar al Grid. La definición de los módulos se completa indicando los parámetros de entrada que necesitará para la ejecución. Esos parámetros pueden ser un valor simple o un rango de iteración.
- Tarea: sobre el módulo se apoya el concepto de tarea, que consiste en una ejecución particular de un módulo del repositorio en el que se instancias los parámetros de entrada del módulo a una serie de valores concretos. En caso de que el parámetro sea simple se indicará un valor, si es un rango se indicarán los valores necesarios para instanciar el rango (un directorio y un filtro para ficheros o un valor inicial, valor final y valor de paso para rangos numéricos, etc.).

  Es importante ver una tarea como una ejecución completa que proporciona resultados finales y que será Fura quien se encargue de dividir la ejecución de esta tarea en microtareas.
- Una microtarea (µtarea) es una parte de la ejecución de una tarea que será lo que realmente se ejecutado. La µtarea surge de la iteración, por parte del servidor Fura, de los diferentes rangos de la tarea a la que pertenece, de tal manera que cada parámetro de entrada tenga un valor concreto.

2.1.2.2 Arquitectura

Fura está compuesto básicamente por tres componentes (Figura 4): un servidor que gestiona la ejecución de las tareas. Una serie de agentes encargados de contactar con el servidor para indicarle el estado del nodo y solicitar tareas que, una vez asignadas, pondrá en ejecución. Por último el tercer componente es un portal web que actúa como interfaz de usuario.

A continuación se comentan un poco más en detalle las características y funcionalidades de los componentes mencionados:

- Servidor: Es el responsable de la gestión de cada uno de los nodos que lo forman y de la coordinación del conjunto. Se encarga de monitorizar el estado de los agentes, y de seleccionar las microtareas a ejecutar en caso de que el agente se encuentre en estado inactivo y disponga de los recursos necesarios para llevarlas a cabo (requerimientos de memoria, almacenamiento en disco, etc.).

  Adicionalmente se puede incluir una cadena de servidores redundantes para aumentar la disponibilidad y robustez del sistema. El primer servidor será el gestor principal mientras que el resto monitorizará su estado y actualizará una copia propia del sistema de ficheros. En caso de parada (voluntaria o accidental) del nodo principal, el siguiente servidor de la lista pasará a ser el responsable de la gestión del sistema hasta que el servidor principal se recupere y retome sus responsabilidades.

  Una vez asignada la microtarea a un agente, monitoriza la progresión en la ejecución de la misma y, una vez se ha completado la ejecución, se encarga de recuperar los ficheros resultantes del cómputo.

  En caso de que la ejecución de la microtarea falle (el ordenador se apague, se detenga el agente, etc.), el servidor se encargará de reubicarla en otro agente disponible.

Como tarea complementaria, el servidor recoge estadísticas de ejecución de las distintas microtareas, con el fin de mejorar la asignación de ejecuciones a los nodos y ofrecer al administrador del sistema una visión global del mismo.

El servidor Fura se encarga, además, de implementar el sistema de ficheros de repositorio y garantizar la seguridad en los accesos a los datos, por medio de un sistema de permisos definido para los distintos usuarios y roles activados por el administrador.

- Agente: Es una aplicación ligera encargada de la ejecución de la microtareas asignadas por el servidor, de tal manera que no afecten a la usabilidad del ordenador, por parte del usuario local al mismo.

  Esta aplicación se encarga de monitorizar el estado del ordenador en el cual está instalada, para informar al servidor de la disponibilidad de recursos y solicitar la colaboración en la ejecución de las tareas.

  Una vez asignada una microtarea al nodo, el agente es el responsable de descargarse el archivo ejecutable que implementa la microtarea, y ponerlo en funcionamiento con los parámetros indicados por el servidor.

  Así mismo, el agente realiza un seguimiento de la ejecución de la microtarea, comprobando la ocupación del procesador, el espacio en disco consumido, etc., y finalmente recogiendo el código de salida del programa lanzado.

  Para completar el proceso, se encargará de informar al servidor de los ficheros que se han generado durante la ejecución, para que éste solicite el envío de aquellos que considere necesarios (aquéllos indicados por el usuario).

- Portal: Es un conjunto de aplicaciones web que actúan como interfaz gráfica para el acceso a toda la funcionalidad del servidor: la gestión de módulos y tareas, el acceso al sistema de ficheros, monitorización de los recursos, etc.

Para completar el sistema Fura proporciona una API para poder acceder a toda la funcionalidad del servidor, de tal manera que se puedan crear programas externos que lancen trabajos al sistema Fura sin necesidad de pasar por el Portal.

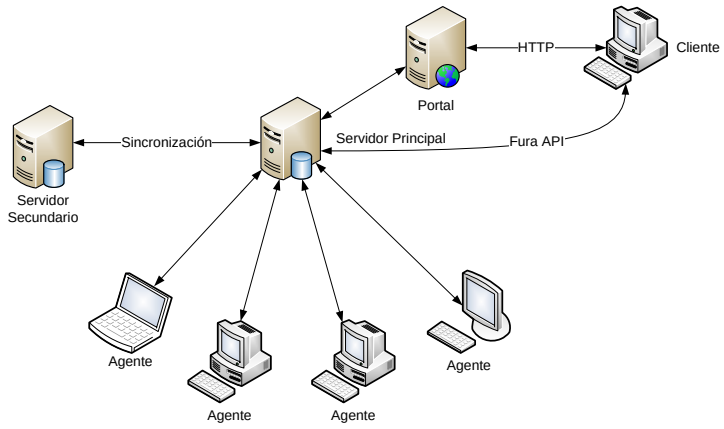

Figura 4. Arquitectura de Fura.

2.2 Workflow

2.2.1 Introducción

Existen varias definiciones del concepto de workflow, pero la más extendida es la dada por la Workflow Management Coalition (WfMC) [14] como organización referencia en la definición de estándares sobre workflow:

"La automatización de un proceso de negocio, por completo o una parte, donde los documentos, la información o las tareas se pasan de un participante a otro para ser procesados, según un sistema de reglas de procedimiento para conseguir un objetivo global".

El concepto de workflow aparece en los años 70 aplicado en el ámbito empresarial, cuyo principal objetivo era la estructuración y programación de tareas, incluyendo las interacciones humanas de dichos procesos, así como la organización del flujo de datos. Inicialmente el tránsito de datos se producía en formato papel, aunque con el desarrollo de las TIC, han sido sustituidos de forma progresiva por formatos electrónicos.

Workflow está normalmente asociado a procesos de reingeniería de los procesos de negocio, es decir con el análisis, modelización, definición y posterior implementación de los procesos de negocio principales de una organización.

Tambien la WfMC en [15] define un sistema de gestión de workflow como:

"Un sistema que define, gestiona y ejecuta workflows de forma completa, mediante la ejecución de software cuyo orden de ejecución es dirigido por la representación por un ordenador de la lógica del workflow".

Por tanto un sistema de gestión de workflow es aquel que proporciona una automatización de un proceso de negocio controlando una secuencia de actividades de trabajo y la invocación de las adecuadas interacciones humanas y/o de recursos computacionales asociados con cada una de las actividades. Estos sistemas workflow deben proporcionar soporte en tres áreas funcionales:

- Definición y modelización del workflow y de todas las actividades que lo componen.
- Gestión de los procesos de workflow en un entorno operacional de manera que se ordenen las actividades para ser llevadas a cabo como parte de cada proceso.
- Gestión de las interacciones con los usuarios humanos y las aplicaciones para el procesado efectivo de todos los pasos de las actividades.

2.2.1.1 Workflow Grid

Con el desarrollo de la computación distribuida y las tecnologías Grid, estos conceptos se han aplicado en el campo de la computación científica. En este campo un sistema de gestión de workflow se encarga de la definición, gestión y ejecución de los workflows en recursos computacionales. La aplicación de las técnicas de workflow para la composición de aplicaciones en entornos Grid, permite crear aplicaciones dinámicas que coordinen la ejecución de tareas en distintos recursos distribuidos.

En [16] se reformula la definición de workflow para adaptarla al entorno grid:

"La automatización de procesos, que incluye la orquestación de un conjunto de servicios grid, agentes y actores que deben ser combinados para resolver un problema o crear un nuevo servicio."

También en [17] se da otra descripción centrada en el ámbito de la e-Ciencia.

"Un workflow es una especificación de alto nivel de un conjunto de tareas y dependencias entre ellas que se deben satisfacer para obtener un objetivo concreto."

Según el Grid Workflow Forum (http://www.gridworkflow.org/) las principales diferencias entre el uso clásico del workflow y su aplicación en el ámbito de las tecnologías Grid son:

- Mantenimiento de estado: Esta característica básica dentro del entorno de los servicios Grid, y que lo diferencia de los clásicos servicios Web, debe ser contemplada en los workflows Grid.

- Fiabilidad: Los recursos Grid pueden fallan durante la ejecución de un workflow. Esto debe ser tenido en cuenta teniendo una tolerancia a fallos avanzada (workflow checkpointing, control de procesos…).
- Rendimiento: Uno de los objetivos de la computación en Grid es la de obtener alta prestaciones. Por tanto los workflow deben permitir la utilización del paralelismo, selección de recursos, balanceo de carga…
- Flujos de datos muy grandes: En los entornos Grid, tanto la cantidad de datos a tratar como el tamaño de los flujos pueden llegar a ser muy grandes.

Con las descripciones anteriores la Figura 5 muestra un esquema general de lo que sería un sistema workgflow grid. En la parte superior se encuentran las herramientas de definición de wokflow. Estas serían las encargadas de crear la especificación del workflow usando algún tipo de notación: de forma gráfica, con lenguajes de definición, etc. El componente de enactment sería el encargado de tomar esa especificación y llevar a cabo el lanzamiento de las diferentes tareas, gestionando el movimiento de datos necesario y teniendo un control de fallo de las ejecuciones. Por ultimo en las capas inferiores se encontrarían los diferentes middleware grid que darían en acceso efectivo a los recursos disponibles.

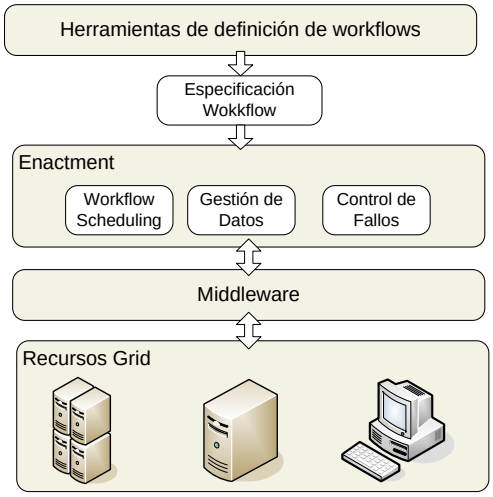

Figura 5. Esquema de un Sistema de Workflow Grid.

2.2.2 Proyectos

Existen diferentes proyectos que están trabajando en el desarrollo de los diferentes componentes necesarios para la creación de un sistema workflow orientado a aplicaciones grid. En [18] se hace una taxonomía de los diferentes sistemas workflow en entornos grid existentes.

A continuación vamos a mostrar una descripción de los más importantes, mostrando finalmente un análisis de las características de cada uno, para comprobar si cumplen con los objetivos que nos hemos fijado en esta tesis de master.

2.2.2.1 Globus

Dentro del proyecto Globus, se han realizado trabajos sobre la aplicación de workflows en los entornos grid. Estos trabajos han sido desarrollados dentro del componente denominado Java CoG kit. El Java Commodity Grid (CoG) Kit de Globus es un conjunto de librerías que permite el desarrollo de aplicaciones y componentes de alto nivel para el Globus Toolkit. Además proporciona otra serie de funcionalidades, entre las que destaca el soporte de workflows.

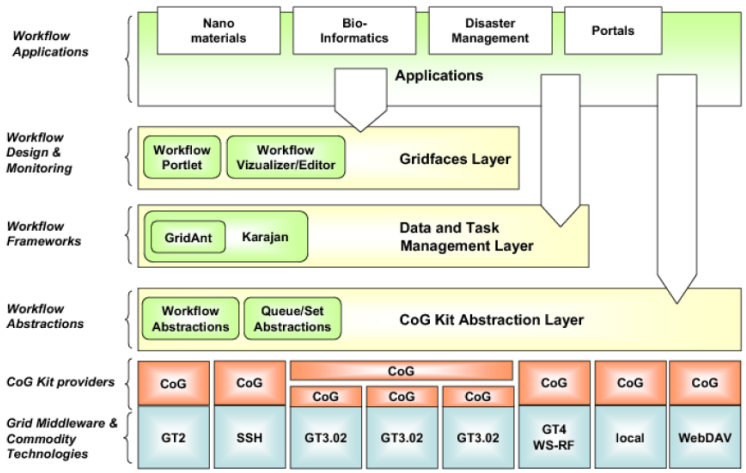

Figura 6. Esquema capas del Java CoG Kit.

Componentes:

GridAnt [19]: fue desarrollado como una manera sencilla y rápida de obtener un sistema workflow basándose en un lenguaje XML. En este caso se utilizó como base el lenguaje Ant, ya que se puede considerar como el estándar de facto para los procesos de compilación en proyectos Java. Este lenguaje tiene una serie de limitaciones, pero puede proporcionar un buen marco de trabajo para casos sencillos.

Karajan [20]: aparece como la evolución del GridAnt, tomando su nombre del famoso director de orquesta. La principal diferencia con su predecesor es la introducción de las secuencias de control, como los bucles y las condiciones, así como la reimplementación de los bloques secuenciales y paralelos de tareas.

2.2.2.2 Triana [21]

Triana es un entorno de resolución de problemas gráfico basado en workflows que fue desarrollado inicialmente para proporcionar una herramienta de análisis para datos de ondas gravitacionales en asociación con el proyecto GEO 600, aunque puede ser utilizado para resolución de problemas de cualquier tipo. Inicialmente se las ejecuciones se realizaban en local o remotamente usando Java RMI. Actualmente se han añadido una serie de componentes distribuidos como trabajos grid o web services. Al igual que la mayoría de los diferentes proyectos analizados utiliza XML como base para la definición del leguaje de definición de workflows. Para las ejecuciones grid utiliza una capa software llamada Grid Application Toolkit (GAT) desarrollada dentro del proyecto GridLab [22] para distribuir las tareas entre los diferentes recursos Grid.

2.2.2.3 Askalon

Askalon es un entorno de programación cuyo principal objetivo es facilitar el desarrollo de aplicaciones grid. Está diseñado como una arquitectura orientada a servicios distribuida, que está compuesta de los siguientes conjuntos de servicios:

- Resource Broker: encargado de la negociación y reserva de recursos para la ejecución de la aplicación grid.
- Resource Monitoring: permite el control de los recursos grid, integrando y extendiendo las actuales herramientas, así como desarrollando nuevas técnicas de monitorización.
- Information Service: un servicio de propósito general para la organización y mantenimiento de la información de los recursos y de las aplicaciones.

- Workflow Composition & Execution: permite la definición y ejecución de flujos de tareas complejos con tolerancia a fallos en entornos grid. Para la definición utiliza un leguaje de modelización basado en XMl, el Abstract Grid Workflow Language (AGWL [23]).
- Meta-Scheduler: se encarga del mapeo de las aplicaciones workflow in los recursos del grid.
- Performance prediction: servicio encargado de la estimación temporal de las actividades atómicas y transferencias de datos.
- Performance analysis: instrumentación y análisis de workflows grid con soporte de detección de cuellos de botella (excesiva comunicación, carga mal balanceada, ineficiencia…)

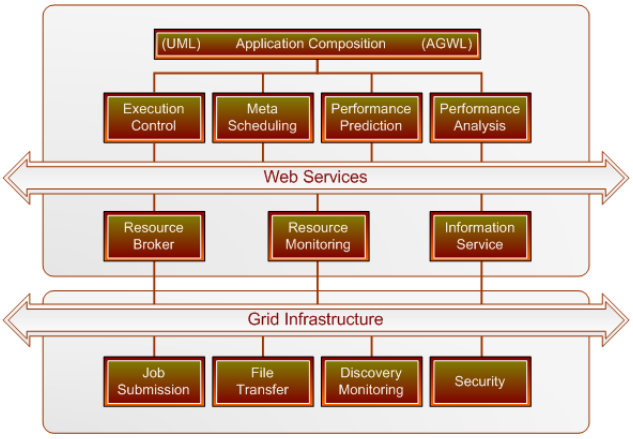

Figura 7. Arquitectura orientada a servicios de Askalon.

2.2.2.4 WFEE [24]

Este sistema workflow ha sido propuesto y desarrollado por la Universidad de Melbourne. Es el entorno de ejecución del lenguaje de definición de modelos xWFL. Las principales características del WFEE son:

- Modelo de ejecución descentralizado.
- Descubrimiento de servicios utilizando Globus MDS y Grid Market Directory.
- Traducción del leguaje de modelado (xWFL) antes de la fase de scheduling. El documento XML es transformado en un conjunto de objetos Java (tareas, parámetros…)
- Utiliza el concepto de "Tuple-Space" durante la fase de mapeo del workflow como método para controlar y responder a los eventos del workflow.
- Utiliza una estrategia de scheduling en run-time, consultando el sistema de descubrimiento de servicios para seleccionar los recursos más adecuados y controlando la transferencia de datos.

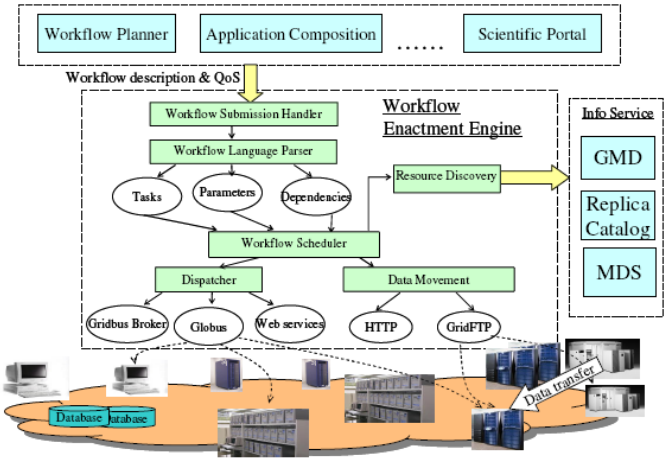

Figura 8. Arquitectura del WFEE.

2.2.2.5 Taverna [25]

Ha sido desarrollado como parte del proyecto myGrid. Permite crear y ejecutar workflows usando el lenguaje Simplified Conceptual Workflow Language (Scufl). El Scufl, es la base de Taverna y es un lenguaje que permite la definición de workflows dirigidos principalmente por los flujos de datos. Utiliza Freefluo [26] como motor de ejecución de los workflows. Como formato de intercambio utiliza una versión XML del Scufl (XScufl)

Este proyecto surge con la orientación en el uso por parte del mundo científico, en concreto en el campo de la biología.

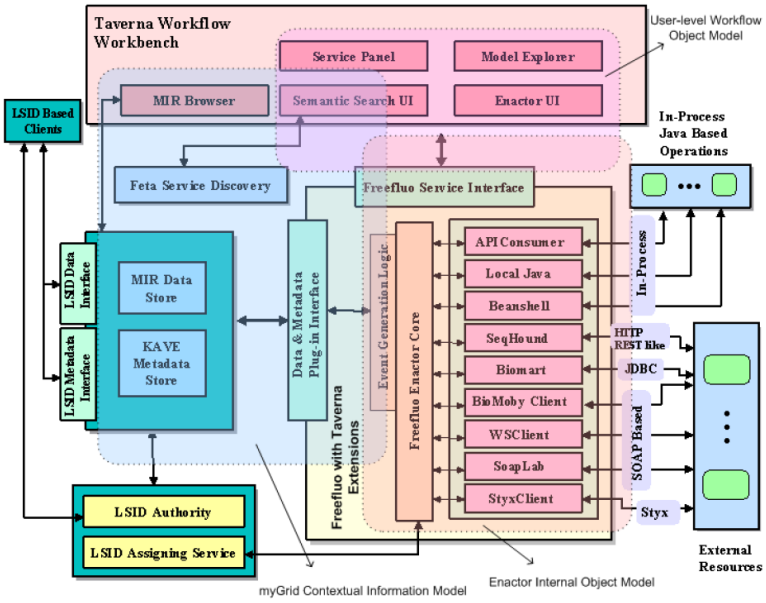

Figura 9. Arquitectura de Taverna.

La arquitectura de Taverna se muestra en la Figura 9. Ésta se basa en dos conceptos:

- Un modelo de datos de tres capas para la descripción de recursos y su interoperabilidad en diferentes niveles de abstracción, desde la perspectiva del usuario a la de los servicios:
    - La capa de flujo de datos de la aplicación está dirigida al usuario y está caracterizada por el denominado "User-Level Workflow Object Model". El propósito es mostrar los distintos workflows desde el punto de vista

orientado al problema, ocultando la complejidad de las interacciones entre los servicios.

- La capa de ejecución se encarga de ocultar los detalles de la ejecución al usuario. Es caracterizada por el "Enactor Internal Object Model" que comprende al motor de ejecución de workflows Freefluo.
- La capa de invocación de procesadores destinada para interactuar e invocar servicios concretos.

- Un Framework que proporciona tres niveles de extensibilidad:
  - Una primera capa que permite añadir nuevos paneles al GUI, facilitando al usuario derivar y gestionar extensiones de comportamiento a Taverna.
  - Una segunda capa que permite añadir nuevos tipos de procesadores, de manera que el motor de ejecución pueda reconocer e invocar nuevos tipos de servicios.
  - El tercer nivel permite la integración de componentes externos usando un interfaz de tipo evento-observador. Los eventos producidos durante la ejecución del workflow pueden ser interceptados por plugins de tipo observador de manera que puedan interactuar con servicios externos.

2.2.2.6 Kepler

Kepler [27] es un sistema workflow "científico" basado en la herramienta Ptolemy II [28]. Kepler permite capturar workflows en un formato que permite que sea fácilmente intercambiado, almacenado, versionado y ejecutado. Utiliza un modelo "orientado al actor" que permite un fácil diseño, ejecución y reutilizado de los workflows. Como formato de intercambio utiliza un lenguaje basado en XML, denominado Modeling Markup Language (MoML).

Utiliza la abstracción "Actor" definida en Ptolemy II como interfaz para poder acceder a Servicios Web, Servicios Grid, Trabajos Globus, GridFTP, además de poder definir nuevos "actores" por parte del usuario. Es posible la creación de actores compuestos de un conjunto de workflows de otros actores (denominado como graph clustering en MoML).

2.2.2.7 GridFlow

Es un proyecto colaborativo que promueve la integración de diferentes middlewares grid. Proporciona un entorno de servicios de dos capas, una capa de gestión de los workflows grid globales y una capa encargada de la gestión de los sub-workflows locales. Utiliza una especificación XML para la representación de los workflows.

Todos los componentes son controlados desde un portal que provee simulación, ejecución y monitorización de los workflows.

Utiliza una metodología basada en agentes para la gestión los workflows grid globales, en concreto usando ARMS [29]. Este sistema está integrado con un sistema de gestión de recursos local denominado Titan [30]. Tanto las funcionalidades de ARMS como de Titan están basadas en las capacidades de predicción de prestaciones de las aplicaciones del sistema PACE [31].

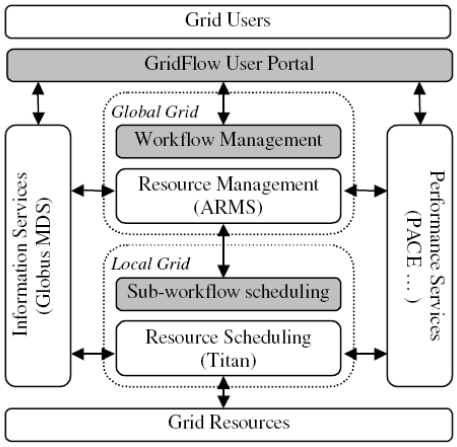

Figura 10. Esquema de sistema GridFlow.

2.2.2.8 DAGMan

DAGMan [32] (Directed Acyclic Graph Manager) es un meta-planificador para el sistema Condor [33]. Controla dependencias entre trabajos a un mayor nivel que el planificador Condor estándar. Como indica el nombre, DAGMan representa las dependencias como grafos acíclicos dirigidos.

Utiliza ficheros en formato de texto para definir los grafos donde los nodos están formados por trabajos que serán sometidos a los diferentes sistemas Condor que forman el sistema. Las aristas son las dependencias de datos.

2.2.2.9 VDS - The GriPhyN Virtual Data System

VDS, The Virtual Data System (antiguamente denominado Chimera) del proyecto GriPhyN [34], proporciona una serie de herramientas para expresar, ejecutar y monitorizar workflows.

Utiliza VDL (Virtual Data Language) como lenguaje de definición de workflows. VDL es un lenguaje de alto nivel, que proporciona la capacidad de definición de workflows sin tener que dar detalles específicos sobre localizaciones, ficheros etc.

Actualmente utiliza Pegasus (Planning for Execution in Grids) [35][33] como motor de ejecución de workflows.

Pegasus hace de puente entre las aplicaciones de ámbito científico y el entorno de ejecución, haciendo un mapeo entre las descripciones de workflows de alto nivel en las infraestructuras de computación distribuidas.

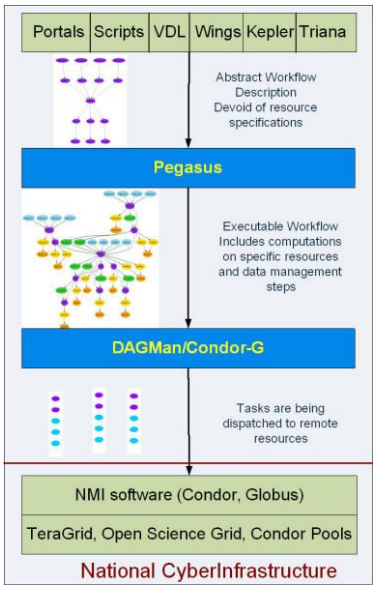

Figura 11. Esquema funcional de Pegasus.

El modelado del workflow puede realizarse de forma manual o utilizando herramientas como Chimera. El lenguaje de definición de workflows utilizado por Pegasus es DAX, un lenguaje de definición de grafos acíclicos dirigidos (DAG) en XML. El sistema produce workflows concretos, es decir, ejecutables, que son enviados al sistema DAGMan para su ejecución.

Actualmente VDS está trabajando con otro planificador experimental denominado Euryale.

2.2.2.10 Swift [36]

Es un sistema de programación paralelo que integra los siguientes elementos:

- Un lenguaje de Script denominado SwiftScript. Usa la notación XML Dataset Typing (XDTM) para definir el mapeo entre la organización local y la infraestructura física.
- Un motor de ejecución, CoG Karajan, un sistema de envío de trabajos, Falkon, y un compilador y las librerías necesarias para ejecutar tareas especificadas mediante programas SwiftScript
- Un sistema de "debug" de las ejecuciones grid, Kickstart, que captura los detalles de la ejecuciones y puede almacenarlas en un sistema de base de datos.

Swift continúa el trabajo del VDS, mejorándolo en varios aspectos, en concreto en el uso de XDTM para la definición de vistas lógicas de los conjuntos de datos.

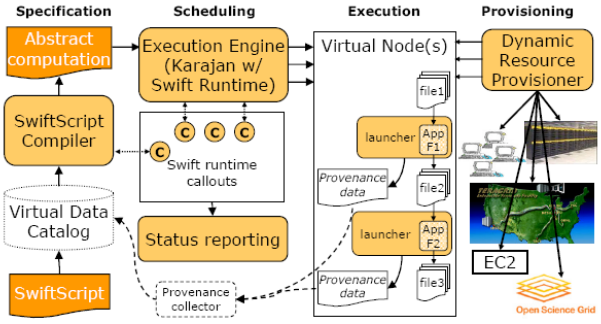

Figura 12. Diagrama del sistema Swift.

2.2.2.11 YAWL

YAWL (Yet Another Workflow Language) [37] es un lenguaje de definición de workflows desarrollado por el grupo de investigación QUT's BPM en conlaboración con la Eindhoven University of Technology. Está basado sobre dos conceptos: las redes de Petri y los workflow patterns [38]. Ha sido desarrollado tomando como base las redes de Petri y añadiendo mecanismos para permitir el soporte de patrones de workflows identificados.

Una especificación de un workflow en YAWL es un conjunto de redes workflow extendidas (extended workflow nets EWF-nets) agrupadas en forma jerárquica, formado estructuras en forma de árbol. Las tareas pueden ser atómicas o compuestas. Las atómicas son los elementos hoja de la estructura en forma de árbol. La tarea para la cual no existe ninguna tarea compuesta que la enlace será la denominada tarea raíz.

Algunas características de YAWL:

- OR, AND, y XOR splits/joins
- Web Services pueden ser utilizadas como tareas atómicas o como tareas compuestas.
- Paralelismo explícito (múltiples instancias de una tarea atómica)
- Cancelación de una parte de un workflow por una tarea particular.

2.2.2.12 P-GRADE

P-GRADE (Graphical Parallel Program Development Environment) [39] es un entorno de desarrollo de aplicaciones grid. Utiliza Globus, Condor-G y MPICH-G2 como middlewares para realizar las ejecuciones. Para proporcionar el soporte de workflows P-GRADE usa el lenguaje GRAPNEL (GRaphical Process NEt Language) que permite al programador la definición de workflows de objetos o llamadas a librerías. GRAPNEL Utiliza gráficas para expresar el paralelismo y C/C++ para la parte secuencial. Una serie de herramientas permiten generar código MPI o PVM a partir de la definición en lenguaje GRAPNEL.

Los workflows describen tanto el flujo de control como el flujo de datos de la aplicación. Un trabajo puede iniciarse cuando todos los ficheros de entrada están disponibles en la localización donde se va a realizar la ejecución.

Para planificar y controlar la ejecución de los workflows se utlizan las herramientas Condor-G y DAGMan, encargándose el sistema de la generación del fichero de descripción del sistema DAGMan y del envío de los ficheros a los nodos de ejecución.

Utiliza PROVE [40] como herramienta de visualización del progreso de las tareas que conforman el workflow. PROVE coopera con el sistema de monitorización grid Mercury/GRM (desarrollado en el proyecto EU GridLab [41]).

2.2.2.13 ScyFlow

ScyFlow [42] es un entorno que soporta la especificación, ejecución y monitorización de workflows. Está formado por dos componentes principales:

- ScyFlowVis: Interfaz visual para la especificación gráfica de los workflows. Se encarga de la traducción del grafo a formatos internos (XML, lista de dependencias, pseudo-código) para su almacenamiento y comunicación con otros componentes.
- ScyFLowEngine: Proporciona un conjunto de servicios para completa ejecución de los workflows entre los distintos recursos Grid, teniendo en cuenta las dependencias de datos y de control. Proporciona una API que permitirá a otras aplicaciones acceder al motor de ejecución de workflows.

2.2.2.14 K-Wf Grid

El proyecto K-Wf Grid (Knowledge-based Workflow System for Grid Applications) [43] introduce el concepto de workflow basado en el conocimiento (Knowledge-based). Este tipo de sistemas utiliza sistemas expertos para la ejecución de los workflows y la selección de los Servicios Grid.

La arquitectura del sistema (Figura 13) está compuesta por cuatro capas horizontales (A, B, C, D) y la capa vertical denominada del "Conocimiento" (K). Las capas horizontales son: el portal Web (A), capa de construcción de aplicaciones Grid (B), capa de control de aplicaciones (C), y por último la capa del middleware grid de bajo nivel. Cada capa solo se comunica con las capas vecinas de manera que pueden ser reemplazadas sin necesidad de cambiar el sistema completo.

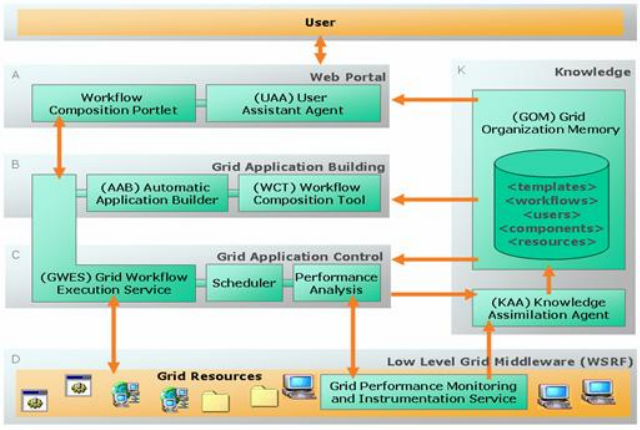

Figura 13. Arquitectura del sistema K-Wf Grid.

Como motor de ejecución de workflows utiliza el componente denominado Grid Workflow Execution Service (GWES) [44]. Implementa un concepto de workflow dinámico utilizando el lenguaje Grid Workflow Description Language (GWorkflowDL) [45], que está basado en la teoría de las redes de Petri de alto nivel. El propósito principal del GWES es permitir a los usuarios ejecutar workflows en recursos distribuidos sin necesidad de conocer detalles específicos de implementación. Proporciona métodos para iniciar e analizar workflows grid, coordinar y ejecutar los workflows en recursos distribuidos y no homogéneos teniendo en cuenta el flujo de datos y de control. El servicio soporta Web Services, Globus Toolkit 4 y es extensible a otras plataformas como UNICORE, Condor, GRIA y SGE.

2.2.3 Análisis

A continuación mostramos una tabla mostrando las características principales de todos los sistemas workflows analizados. Las cuatro primeras características han sido estructuradas según la taxonomía realizada en [18]. Las dos últimas columnas han sido añadidas para mostrar, en primer lugar que pocos de los sistemas tienen la capacidad de extensión hacia nuevos middleware de forma sencilla, en segundo lugar, que aunque la mayoría de los sistemas tienen licencias de tipo software abierto, algunas como es el caso de Askalon tiene una licencia muy restrictiva que solo permite su uso en ámbito educacional y que no permite ningún tipo de modificación ni ampliación por terceros.

	Estructura	Modelo	Definición	Arquitectura	Extensible	Licencia
GridAnt	No DAG	Concreto	Lenguaje	Centralizada	No	Globus
Karajan	No DAG	Abstracto	Lenguaje/ Gráfico	Centralizada	No	Globus
Triana	No DAG	Abstracto	Gráfico	Descentralizada	No	Apache 2
Askalon	No DAG	Abstracto	Lenguaje/ Gráfico	Descentralizada	No	Askalon

WFEE	DAG	Abstracto	Lenguaje	Descentralizada	No	N.D.
Taverna	DAG	Abstracto/ Concreto	Lenguaje	Centralizada	Si	LGPL
Kepler	No DAG	Abstracto/ Concreto	Gráfico	Centralizada	No	BSD
GridFlow	DAG	Abstracto	Lenguaje/ Gráfico	Jerarquica	No	N.D.
DAGMan	DAG	Abstracto	Lenguaje	Centralizada	No	Condor
VDS	DAG	Abstracto	Lenguaje	Centralizada	No	Apache 2
Swift	No DAG	Abstracto	Lenguaje	Centralizada	No	Globus
YAWL	DAG	Abstracto	Gráfico	Centralizada	No	LGPL
P-GRADE	DAG	Abstracto	Lenguaje/ Gráfico	Centralizada	No	GPL
ScyFlow	DAG	Abstracto	Gráfico	Centralizada	No	N.D.
K-Wf Grid	DAG	Abstracto	Lenguaje	Centralizada	No	N.D.

De los sistemas analizados algunos de ellos usan un lenguaje de bajo nivel o necesitan describir muchos detalles de implementación sobre el workflow. Otros están orientados para un tipo específico de middleware grid o solo permite usar un grupo reducido de sistemas y no permite extensiones, o lo permite de forma muy restrictiva. Este último es el principal problema encontrado en los diferentes sistemas workflow analizados, ya que, como hemos comentado, la extensibilidad es un punto clave para éxito de una herramienta workflow. Solo un sistema es extensible a nuevos middlewares: Taverna. Este sistema aunque tiene una serie de conceptos sencillos a la hora de definir un workflow, tiene sin embargo tiene una serie de inconvenientes: No permite el uso de secuencias de control, como sentencias condicionales o bucles, a la hora de definir un workflow. Además aunque permite la extensión hacia nuevos sistemas de ejecución, el usuario debe tener un conocimiento avanzado de cada uno de los plugins para poder hacer funcionar las ejecuciones, no permitiendo que se ejecute de forma indistinta en cualquier sistema.

Dado que ningún sistema de los actualmente existentes se adaptan a los objetivos presentados al inicio de esta tesis, hemos desarrollado nuestro propio sistema, tratado de aunar las mejores características de los sistemas analizados y aportando las características que estos no desarrollaban.

En cuanto a los lenguajes o les falta nivel de abstracción (hace falta muchos detalles de implementación) o les falta completitud para expresar algunos estructuras de control. Por eso también vamos a diseñar un nuevo lenguaje para poder expresar un workflow de forma completa, aislando lo máximo de los detalles de implementación y permitiendo la extensibilidad de middlewares de forma simple. De los lenguajes analizados, los dos lenguajes que más se adaptan a estas necesidades son el Scufl de taberna y el AGWL de askalon. El primero carece de instrucciones de control y el segundo tiene unas restricciones de licencia muy importantes. Por tanto tomando conceptos de ambos lenguajes vamos a definir uno nuevo que no aporte todas las características necesarias para cumplir los objetivos de esta tesis.

3 WINGS

A continuación vamos a describir el sistema WINGs (Workflow In New-generation GRIDs) desarrollado en el marco de la tesis de master. WINGS. De los sistemas analizados hemos extraído una serie de características interesantes para nuestro sistema:

- Conceptos abstractos para definir el workflow: El usuario debe ser capaz de especificar los workflows introduciendo la información de sus aplicaciones y como se conectan entre sí, minimizando la información sobre los sistemas grid subyacentes. Independencia, tanto en la definición de ejecuciones, de los despliegues donde finalmente vayan a ser ejecutados.

- Uso de un lenguaje de definición XML: El uso de un lenguaje XML para definir los workflows permite una manera sencilla de realizar intercambios con otros investigadores, sencilla extensibilidad, etc. Además como vamos a usar conceptos sencillos y abstractos será sencillo de representar gráficamente para facilitar la edición de forma gráfica por el usuario.
- Guiado por datos: En la ejecución real de un workflow son los datos los que marcan las pautas de cuando se deben ejecutar las tareas. En algunos lenguajes los envuelven usando lenguajes con complicadas sentencias de control. El uso de un lenguaje guiado por datos hace más sencilla la expresión y compresión por parte de los científicos no acostumbrados a lenguajes de programación.

Para describir el funcionamiento del sistema workflow inicialmente se van a mostrar los cuatro conceptos abstractos de alto nivel en los que se basa la definición de un workflow. Más adelante se describirá el lenguaje de definición de workflows, donde se mostrará cómo se definen cada uno de estos conceptos, como se estructuran, así como otra serie de elementos necesarios para completar la definición de un workflow. Después será descrito el motor de workflow que se encarga de la ejecución de los workflows.

3.1 Conceptos básicos

3.1.1 Actividades

Las actividades son abstracciones de las tareas grid. De forma similar a las funciones en los lenguajes de programación procedural. Una actividad se define por su interfaz (y tipo de los parámetros de entrada y salida) y por la funcionalidad que realiza. Para facilitar el despliegue en diferentes entornos de ejecución la funcionalidad se define usando un listado de despliegues en el cual se indican, para cada uno de ellos, los datos necesarios para poder ejecutar la actividad en dicha infraestructura grid. De esta manera se independizan las ejecuciones posteriores de los despliegues existentes.

3.1.2 Grupos de Datos

Los grupos datos son abstracciones para el manejo de los datos usados en el workflow. Son los puntos de comunicación entre las tareas ejecutadas. Permiten definir los parámetros de entrada del workflow, los de salida, así como "almacenar" todos los datos intermedios generados en el lanzamiento del workflow.

3.1.3 Ejecuciones

Las ejecuciones son instanciaciones de las actividades que hayan sido definidas. Las ejecuciones referencian que actividad instancian y para cada uno de los parámetros de entrada definidos en la actividad se les indica un grupo de datos donde tomar/dejar los datos y se puede indicar un valor prefijado.

3.1.4 Operaciones

Las operaciones son tareas simples y de poca necesidad computacional que normalmente son necesarias para el post/pre procesado de los datos de las ejecuciones (extraer un valor de un archivo de texto, seleccionar los ficheros de las tareas que hayan acabado con éxito, etc.). Tradicionalmente era necesario crear ejecuciones "artificiales" para realizar estas tareas. WINGS implementa una serie de este tipo de tareas para facilitar la creación de workflows sin necesidad de tener que implementar este tipo de tareas.

3.2 Lenguaje WINGS

A continuación vamos a describir el esquema completo que define el lenguaje XML WINGS para la definición de workflows. En el lenguaje de definición usamos los conceptos que se han comentado en el punto anterior, además de otros necesarios para la correcta funcionalidad del sistema.

3.2.1 Esquema del documento XML

El documento XML de definición de un workflow está estructurado en 5 partes.

```
<workflow name="example">

      <definitions>
      </definitions>

      <authorisation>
      </authorisation>

      <data>
      </data>

      <executions>
      </executions>

      <resources>
      </resources>

</workflow>
```

- definitions: parte de definición de las actividades donde se describen todas las actividades del workflow con sus parámetros y la información de los diferentes despliegues disponibles.
- authorisation: define los datos de autorización por defecto que será utilizada en la ejecución de las tareas, en caso de que no tengan definido su propia autorización.
- data: en este apartado se definen los distintos grupos de datos que será utilizado en el workflow.
- executions: lista de todas las tareas (ejecuciones y operaciones) que serán ejecutados durante el workflow.
- resources: listados de los diferentes recursos de ejecución en los que se van a lanzar las tareas.

3.2.2 Especificación de Ficheros

Durante toda la definición de los siguientes elementos del lenguaje se van a ir referenciando diferentes ficheros (ejecutables, librerías, datos, etc.). Para todos ellos se va a utilizar una notación en forma de URI (Uniform Resource Identifier) de tal manera que se puedan especificar ficheros en diferentes ubicaciones, tanto locales como remotas.

En caso de no especificar ningún tipo de protocolo: “D:/datos” o “/tmp/datos”, se entenderá que se refieren a ficheros locales. En caso contrario al usar la notación completa: “gsiftp://server:2811//file.ex”, serán los diferentes plugins del sistema de transferencias los encargados de gestionar el movimiento de datos necesario.

3.2.3 Definición de las actividades

Para definir una actividad es necesario especificar los parámetros (interfaz) de la actividad a ejecutar y los datos necesarios para los despliegues:

```
<activity name="activity1">
 <input name="parameto1" type="File"/>
 <input name="parameto2" type="File" granularity="4"/>
 <output name="salida1" type="File"/>
 <deployments>
  <furaDeployment class="wings.furaclient.FuraActivity">
    <module name="actividad1"/>
  </furaDeployment>
  <executableDeployment class="wings.globus.GlobusActivity">
    <executable file="gsiftp://server:2811//file.ex"/>
    <file file="gsiftp://server:2811//file.lib"/>
  </executableDeployment>
 </deployments>
</activity>
```

Inicialmente se indican los parámetros de entrada y salida. Para cada uno de ellos se indica un nombre y el tipo de datos del mismo. Los tipos de datos (básicos) que se pueden utilizar son: File, String, Integer, Float y Boolean. Más delante explicaremos el uso de tipo complejos.

Para los parámetros de entrada se puede indicar además la granularidad. Este atributo permite agrupar bloques de elementos de un dataGroup para lanzar una ejecución. De tal manera que en el ejemplo mostrado el parámetro de entrada "parametro2" necesitará de 4 elementos para poder ejecutar.

En el siguiente apartado se define los datos de despliegue de la actividad. En este apartado tendremos, para cada uno de los entornos de ejecución en los que se vaya a poder lanzar dicha actividad, los datos necesarios para dicho tipo de despliegue. De esta manera para que todas las ejecuciones de una actividad puedan ser ejecutadas en un nuevo middleware simplemente tendremos que añadir una nueva entrada en la sección "deployments".

El único dato que aparecerá en todos los despliegues es el atributo "class". Dicho atributo indicará el nombre de la clase responsable de la ejecución de la tarea en la infraestructura deseada. De esta manera el sistema permite la adición de nuevos entornos de ejecución, a modo de plug-in, simplemente añadiendo una clase al entorno de ejecución del workflow, creando el subsistema de "middleware engine" que explicaremos más adelante.

Actualmente se han definido cuatro tipos de deployments para la ejecución de dos tipos de tareas:

- Deployment para Fura: Fura es middleware grid orientado a entornos empresariales desarrollado por GridSystems. En el nodo furaDeployment las actividades se mapean al concepto de modulo Fura. En el caso de estar creado simplemente se indicará el nombre del módulo. Si no está creado, y en tiempo de ejecución debe ser creado, existirán dos opciones:

    - Indicar las diferentes plataformas necesarias y los ficheros para cada una de ellas:

```
<module name="actividad1">
      <platform arch="win32">
            <executable path="D:/datos" file="pr1.bat"/>
            <file name="lib1" path="D:/datos" file="lib.dat"/>
      </platform>
  </module>
```

- Indicar un fichero de importación de los datos de la actividad (creado previamente en un servidor Fura):

```
<module name="actividad2" importfile="  "/>
```

En caso de que en el servidor Fura ya exista un modulo con dicho nombre, por defecto será utilizado dicho módulo. En caso de que se quiera que el modulo sea creado y borrando el anterior, el nodo module tiene en parámetro "overwrite" de tipo booleano para indicarlo.

- Deployment de tipo "ejecutable": En el nodo executableDeployment las actividades se definen mediante la especificación de un fichero ejecutable a lanzar y un conjunto de ficheros adicionales necesarios para la ejecución. Este es un tipo es muy genérico y puede servir para muchos tipos de despliegues como pueden ser Globus o SSH, simplemente cambiado la clase que define la actividad.

```
<executableDeployment class="wings.globus.GlobusActivity">
  <executable file="gsiftp://server:2811//file.ex"/>
  <file file="gsiftp://server:2811//file.lib"/>
</executableDeployment>

<executableDeployment class="wings.globus.SshActivity">
  <executable file=" e:/datos/pr.sh "/>
  <file file="e:/datos/lib.so"/>
</executableDeployment>
```

- Deployment para SubWorkflows: Para permitir reutilizar workflows como parte de un workflow general de mayor tamaño se ha desarrollado un tipo de deployment denominado "workflowDeployment". La sintaxis será la siguiente:

```
<workflowDeployment class="wings.subworkflow.SubworkflowActivity">
  <xmlFile path="C:\datos" file="optimizacion.xml"/>
  <parameterMapping>
        <parameter name="parameto1" dataGroup="ParametrosFuncion"/>
        <parameter name="salida1" dataGroup="FicheroFinal"/>
  </parameterMapping>
</workflowDeployment>
```

En la definición del despliegue de tipo SubWorkflow además del fichero XML con la descripción del subworkflow, es necesario un mapeo entre los parámetros de entrada y salida de la actividad con algún dataGroup del subworkflow. De esta manera al ejecutarse la actividad el primer paso será "introducir" los datos de entrada de la actividad en los dataGroup indicados en el mapeo. Una vez finalizada la ejecución del subworkflow se "sacaran" los datos de los dataGroup que aparezcan en el listado mapeado a un parámetro de salida de la actividad y devolverlos como resultados de la misma.

Opcionalmente una actividad puede indicar ciertas restricciones sobre los recursos en los que se pueden lanzar sus ejecuciones. Para ello se puede añadir un elemento de tipo resource dentro del deployment de tal manera que se indiquen ciertas características del recurso o bien indicar directamente el nombre de un recurso en concreto. Cuando se indica una propiedad del recurso, a la hora del lanzamiento, el dispatcher sólo seleccionará aquellos recursos definidos que tengan la propiedad indicada, con el mismo valor (en caso de que se haya establecido).

```
<furaDeployment class="wings.furaclient.FuraActivity">
  <module name="actividad1"/>
  <resource name="furaRes2">
        <property name="prop1" value="val1"/>
  </resource>
</furaDeployment>
```

3.2.4 Autorizaciones

Una de las características primordiales en los entornos grid es la seguridad. En WINGS hemos definido cuatro tipos de autorizaciones para poder abarcar el mayor número posible de entornos.

- Userpass: Este es el tipo básico de autenticación utilizado en muchos entornos de ejecución (Fura, SSH) etc. Utiliza un nombre de usuario y una contraseña para la identificación de un usuario. En general este tipo de autenticación es poco recomendable, y en la mayoría de los casos (como ocurre con SSH) se pueden utilizar otras alternativas como los certificados, o el uso de ficheros de claves.

```
<authorisation>
  <userpass user="micafer" password="12345"/>
</authorisation>
```

- Proxy: Este tipo necesita un fichero de autenticación de tipo proxy típicamente utilizado en aplicaciones Grid del entorno Globus.

```
<authorisation>
  <proxy proxyfile="/tmp/proxy"/>
</authorisation>
```

- Certkey: Se deben especificar la ubicación de los ficheros de la clave pública y privada del usuario para su identificación.

```
<authorisation>
  <certkey certfile="cert" keyfile="key" keypasword="1234"/>
</authorisation>
```

- Userkey: Este tipo de autorización es típicamente utilizada en entornos de tipo SSH, para el acceso a recursos sin necesidad de especificar la contraseña del usuario. Se indica el nombre de usuario y la ubicación del fichero con la clave privada del mismo. Para que sea autorizado en el recurso remoto, éste debe estar configurado correctamente para el acceso a dicho usuario.

```
<authorisation>
  <userkey username="user" privateFile="/tmp/id_rsa" />
</authorisation>
```

El nodo de authorisation además permite combinar distintas autorizaciones para ser utilizadas por diferentes middlewares:

```
<authorisation>
  <deployment type="Fura"/>
    <userpass user="micafer" password="12345"/>
  </deployment>
  <deployment type="Globus"/>
    <proxy proxyfile="/tmp/proxy"/>
  </deployment>
</authorisation>
```

De esta manera sera el motor de ejecución el que seleccionará que información de autorización usar dependiendo del recurso en el que cierto trabajo sea lanzado.

3.2.5 Grupos de Datos

Los Grupos de Datos son los orígenes y los destinos de los datos. Los elementos donde las tareas envían los datos de salida para ser utilizados en las siguientes ejecuciones. Representas los datos de entrada y salida de todas las ejecuciones, y los datos de inicio y fin del workflow completo.

Hay cinco tipos estándar de datos para usar en los dataGroups: Boolean, Integer, Float, String and File. Hay cuatro tipos distintos de grupos de datos que se pueden usar en la definición de los workflows:

- Values: Este tipo de datos se utiliza en el caso de que el tipo de datos no sea de tipo fichero, de tal manera que se indican una serie de valores del tipo indicado.

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

- Range: Este tipo de datos se utiliza en el caso de que el tipo de datos sea o Integer o Float. De forma similar al funcionamiento de Fura se indica un valor de inicio, uno de finalización y un valor de paso.

```
<dataGroup name="dg" type="Integer">
<range initialValue="1" step="1" finalValue="10"/>
</dataGroup>
```

- Filerange: Para la definición de los tipos de datos de ficheros vamos a utilizar un esquema similar al caso de Fura. Se define un rango de ficheros indicando la ruta del directorio contenedor de los ficheros y el filtro para obtener los ficheros.

```
<dataGroup name="dg" type="File">
  <filerange name="fr1" path="D:/datos" file="datos*.txt"/>
</dataGroup>
```

- Filter: El tipo filter se utiliza para la recogida de los datos de salida de las distintas ejecuciones. Permite recoger de forma selectiva un grupo de ficheros definiendo un filtro.

```
<dataGroup name="dg" type="File">
  <filter in="*result.txt"/>
</dataGroup>
```

  En el caso de que no se indique a un dataGroup de tipo File ningún filtro se presupone que todos los ficheros resultantes de la tarea.

```
<dataGroup name="dg" type="File">
</dataGroup>
```

  En el caso de que se aplique un filtro, además se puede crear una réplica de los ficheros de recogidos en una ubicación externa (por ejemplo un servidor ftp, u otro servidor Fura distinto, etc.).

```
<dataGroup name="dg" type="File">
  <filter in="*.txt">
    <replica path="/results" dest="ftp://server.algo.com"/>
  </filter>
</dataGroup>
```

En el caso de que se trate de un dataGroup que almacena datos intermedios y no necesite ningún tipo de filtrado o replica, simplemente indicando el nombre y el tipo del mismo es suficiente para su definición.

3.2.5.1 Tipos de datos Complejos

Para el caso en el que el middleware usado para procesar las ejecuciones lo soporte, se pueden usar tipos de datos "complejos" definidos a modo de estructuras de datos.

En el caso que en la definición de una actividad o de un dataGroup se utilice un tipo distinto de los básicos (File, String, Integer, Float y Boolean), se asume que es un tipo de datos complejo. Para definir un tipo complejo lo haríamos de la siguiente manera:

```
<dataGroup name="i2" type="Complejo">
  <values>
    <complexValue>
      <field name="campo1" value="val1"/>
      <field name="campo2" value="val2"/>
      <field name="campo3" value="val3"/>
    </complexValue>
  </values>
</dataGroup>
```

Dentro del nodo values de un dataGroup se podrá definir un nodo de tipo complexValue que tendrá una seria de nodos "field" para cada uno de los campos del tipo. Cada nodo "field" tendrá un nombre y el valor a establecer a dicho campo.

Dependiendo de cada middleware estos datos serán interpretados de forma diferente. En el caso de Fura, tiene una característica que le permite registrar tipos datos en el servidor. De esta manera cuando se use un tipo complejo en un despliegue de tipo Fura, la estructura definida en el XML deberá coincidir con la registrada en el servidor Fura.

3.2.6 Ejecuciones

Las ejecuciones son instanciaciones de las actividades que hayan sido definidas en el apartado de "definitions". Cada ejecución debe indicar el nombre de la actividad que ejecutará y los parámetros den entrada y salida (que deben coincidir con los de la actividad que instancia). Los parámetros de entrada y salida indicarán mediante en parametro source y destination respectivamente los dataGroup donde leerán o almacenarán los datos. En el caso de los parámetros de entrada en vez del atributo "source" se puede usar el atributo "value" el cual permite especificar un valor fijo que será usado en todas las ejecuciones.

De igual forma que la actividades, las ejecuciones pueden tener un nodo resource en el que definir una serie de restricciones sobre los recursos a utilizar en el lanzamiento de dicha ejecución.

```
<execution name="ejecicion1" activity="activity1">
  <input name="parametro1" source="input1"/>
  <input name="parametro2" value="fijo"/>
  <output name="salida1" destination="outpaso1"/>
</execution>
```

Además cada ejecución puede tener un nodo autorisation con las credenciales a utilizar en la ejecución. Esto permite que ciertas ejecuciones se puedan lanzar con diferentes usuarios, para

casos en los que el usuario “general” del workflow no tenga permisos sobre determinado modulo, o de acceso a determinados ficheros, etc.

Para poder permitir que los diferentes tipos de despliegues puedan añadir información específica del tipo de middlware que la va a ejecutar, cada execution puede tener 1 o varios nodos deploymentData. Podrán haber tantos como tipos de despliegues disponibles. Y será la clase encargada de la ejecución la que procesará dicha información. Para el caso concreto de Fura tendría la siguiente forma:

```
<deploymentData type="Fura">
    <resourceFilter>
       (Properties.Memory >= 10) AND (Properties.FreeDiskSpace >= 10)
    </resourceFilter>
    <groupAllocationMode>
       ALLOCATION_EXCLUSIVE
    </groupAllocationMode>
    <groups>
       <group>
             itecbanv01
       </group>
    </groups>
</deploymentData>
```

3.2.7 Recursos

El sistema workflow permite lanzar trabajos a diferentes recursos grid. En el elemento “resources” se pueden indicar todos los diferentes recursos que podrán ser utilizados por el dispatcher del motor de ejecución a la hora de lanzar los trabajos. El concepto de recurso dentro del motor de workflow no es el “estándar” de los entornos grid. El recurso es un punto de acceso a una infraestructura grid, que puede un simple recurso de computación, como ocurre en el caso de Globus, un servidor que da acceso a todos los agentes de cálculo a su cargo en el caso de Fura, o un servidor GridWay con el que se puede acceder a un conjunto de recursos de tipo Globus.

Dependiendo del tipo de despliegue los datos para especificar un recurso son diferentes. En el caso de Fura los datos a indicar son los necesarios para establecer una conexión con un servidor Fura.

```
<furaResource name="fres1" server="serv" port="9112" ssl="false"/>
```

En el caso de Globus los datos serían los siguientes:

```
<globusResource>
  <gram server="serv" port="9112" jobmanager="jobmanager-pbs"/>
</globusResource>
```

Y en el caso de un recurso SSH se definiría de esta forma:

```
<sshResource host="serv" port="22"/>
```

Como se ha comentado en las definiciones de las actividades y ejecuciones, los recursos pueden definir una seria de propiedades que los caracterizan y permiten agruparlos. De tal forma que el dispatcher puede trabajar con un grupo de recursos a la hora de la selección para determinadas tareas (por ejemplo que servidores tienen creado un determinado modulo, o que servidores tienen instalado cierto plugin, o un nodo Globus con unas características específicas).

```
<furaResource name="furaRes2" server="serv" port="9112" ssl="false">
      <property name="prop1" value="val1"/>
</furaResource>
```

3.2.8 Operaciones

Las operaciones son ejecuciones sencillas y ligeras que realizan ciertas funciones de pre o post proceso de las ejecuciones, sin necesidad de que el usuario tenga que crear ejecuciones artificiales para realizarlas.

3.2.8.1 Implementadas

A continuación vamos a mostrar varios ejemplos de operaciones implementadas:

matches: tiene como entrada un DG de tipo File o de tipo String y devuelve como resultado otro DG del mismo tipo que el de entrada, en el que todas los valores cumplen la expresión regular indicada en la operación. En el caso de los ficheros la expresión regular se refiere al contenido de los mismos.

```
<operation name="op1">
  <input source="inoperm"/>
  <output name="out2" destination="dgm"/>
  <matches regexp=".*OK.*"/>
</operation>
```

getDataFields: tiene como entrada un DG de tipo File y puede devolver un DG de cualquier tipo menos de tipo File. Se encarga de obtener el contenido de cada uno de los ficheros que tenga como entrada y extraer los campos indicados en la operación. La operación debe indicar un campo de separador, para extraer los campos dentro de una línea. Después se va indicando todos los campos que se desean extraer. De cada campo se puede indicar el número del campo, es decir, la posición que ocupa dentro de la línea, y la línea del mismo. En caso de que no se indique línea se extrae dicho campo de todas las líneas de los ficheros de entrada. Además se encargará de realizar la conversión al tipo del DG que se haya indicado como DG de salida de la operación, obteniendo un listado de valores de dicho tipo.

```
<operation name="op1">
  <input source="inoper"/>
  <output name="out2" destination="dg1"/>
  <getDataFields>
    <fields separator=" ">
      <field number="1" line="1"/>
      <field number="4"/>
    </fields>
  </getDataFields>
</operation>
```

reduction: puede tener como entrada un número indefinido de DG (de tipo numérico), y como salida se obtiene un DG con tantos valores como DG de entrada se hayan pasado. La operación lo que hace es reducir todos los valores de cada uno de los DG a un solo valor, usando la operación aritmética indicada. En el ejemplo se pasan 3 DG de entrada, por tanto el DG de salida tendrá 3 valores, el 1º con la suma de todos los valores del DG dg1 y así sucesivamente.

```
<operation name="op4">
  <input name="in1" source="dg1"/>
  <input name="in2" source="dg2"/>
  <input name="in3" source="dg3"/>
  <output name="out" destination="dg4"/>
  <reduction operator="+"/>
</operation>
```

arithmentic: puede tener como entrada un número indefinido de DG (de tipo numérico), y como salida se obtiene un DG de tipo numérico. El número de valores que contendrá el DG de resultado dependerá del número de valores de todos los DG de entrada y del campo

“cardinality” indicado en la operación. En el caso de que todos los DG tengan el mismo número de elementos, el DG final siempre tendrá ese mismo número de elementos, obteniendo el valor de la formula indicada en la operación, usando los valores de cada uno de los DG de entrada de la misma posición para obtener el valor de dicha posición en el resultado final. El campo “cardinality” se aplica en los casos en que los DG de entrada tienen distintos números de elementos entre sí, pudiendo tener tres valores:

- Zero: En el caso que algún DG tenga menos elementos, el resultado tendrá tanto elementos como el DG que más elementos tenga, y resto se completarán con ceros para alcanzar dicho número de elementos.
- Min: EL resultado final tendrá tantos elementos como el menor de todos los DG de entrada.
- Error: En caso de que todos los DG de entrada no tengan la misma cardinalidad la función dará error.

```
<operation name="op4">
  <input name="in1" source="dg1"/>
  <input name="in2" source="dg2"/>
  <input name="in3" source="dg3"/>
  <output name="out" destination="dg4"/>
  <arithmetic function="(%%1 + %%2) / %%3" cardinality="Zero"/>
</operation>
```

splitFile: tiene como entrada y salida un DG de tipo fichero. Se encarga de coger todos los ficheros del DG de entrada y separarlos en parte de tamaños iguales, o bien con el mismo número de líneas. Los ficheros resultantes tendrán el siguiente formato:

S[numero_parte][nombre_original]

```
<operation name="op2">
  <input source="outoperiter"/>
  <output name="out2" destination="outsplit"/>
  <splitFile bytes="55"/>
</operation>
```

popData: esta operación se encarga de devolver los datos que han sido responsables de la generación del dato que le llega como entrada. Cuando aparece un dato nuevo en el DG de entrada de esta operación (en este caso OutCondFinal) la operación se encarga de rastrear entre toda la línea de ejecución del workflow para sacar el dato que ha generado dicho dato en un DG de un paso anterior (en el ejemplo Result) y meterlo en el DG de salida (FicheroFinal).

```
<operation name="final">
      <input name="in" source="OutCondFinal"/>
      <output name="out" destination="FicheroFinal"/>
      <popData dataGroupOrigin="Result"/>
</operation>
```

mergeFiles: tiene como entrada un conjunto de DGs de tipo Fichero y como salida un único DG también de tipo fichero. Se encarga de tomar los ficheros de entrada de cada uno de los DG de entrada y fusionarlos en un único fichero de salida. El fichero de salida usará el parámetro “template” para darle nombre. El campo template usa una sucesión de símbolos “#” para definir la parte que la función usará para generar caracteres aleatorias para el nombre del fichero.

```
<operation name="op2">
  <input source="in1"/>
  <input source="in2"/>
  <input source="in3"/>
  <output name="out1" destination="out"/>
  <mergaFiles template="result.####"/>
</operation>
```

3.2.9 Instrucciones de control

Para los casos en que con la funcionalidad comentada no se puedan expresar la funcionalidad requerida con el lenguaje de definición de workflows.

3.2.9.1 Condiciones

Toda operación y ejecución, de forma implícita, va devolver un valor booleano (como una función en C, o una ejecución de un programa en Linux). Como caso general, en el caso de que el resultado producido por la ejecución/operación sea vacio devolverá "false" y si devuelve algún valor será "true".

Ciertas operaciones van a tener tendrán un significado especifico que cambiará este funcionamiento general. Son operaciones cuyo resultado sea un DG de tipo booleano, en este caso el resultado será cierto en caso de que el resultado de todos los valores del resultado sean ciertos, y falso en caso contrario, es decir cuando alguno sea falso (asumiendo un AND como operación de reducción).

3.2.9.2 Operador "if"

El esquema de la instrucción "if" es el siguiente:

```
<if name="">
      <condition>
            <operation name="op4">
            </operation>
      </condition>
      <then>
            <executions>
            </executions>
      </then>
      <else>
            <executions>
            </executions>
      </else>
</if>
```

El funcionamiento es el siguiente: el operador if se lanzará una vez que los datos de entrada definidos como datos de entrada al mismo, tengan datos para operar. En ese momento se ejecutará la operación para evaluar la condición. En caso de que la operación se evalúe a cierto se pasará a ejecutar la parte "then" del operador o la "else" en caso contrario. En cuanto se vuelvan a producir datos en los DG de entrada del "if" y por tanto nuevas ejecuciones del bloque de instrucciones se volverá a evaluar la condición, con los datos de entrada que tenga en ese momento. Dentro de los bloques "then" y "else" se definirán los dataGroup que será usados en el interior de los bloques (al igual que en los lenguajes de programación las variables internas a una función). Dentro de cada bloque solo se podrán usar estos DG definidos o los datos de entrada o salida al "if" (al igual que en una función en lenguajes de programación).

3.2.9.3 Operador "iterator"

El esquema de la instrucción "iterator" es el siguiente:

```
<iterator condition="booleanop4" name="">
      <input name="i1" source="" iterated="true" step="2"/>
      <input name="i2" source="" iterated="false"/>
      <output name="i2" destination=""/>
      <condition>
            <operation name="op4">
            </operation>
      </condition>
```

```
        <iteratedBlock>
              <data>
              </data>
              <executions>
              </executions>
        </iteratedBlock>
  </iterator>
```

Al igual que en el caso del "if" el bloque del operador "iterator" se ejecutará una vez tengan datos sus DG de entrada. En ese momento se evaluará la condición de entrada, y en caso de ser cierta se entrará dentro del bloque de ejecuciones. Un dato de entrada puede estar marcado como iterable, indicando además el "paso" de iteración. Esto permite que, al ejecutar el interior del bucle, se tenga el nivel de paralelismo indicado por el step. En caso de que se ponga un valor de 1, los datos será leídos de uno en uno y por tanto el bloque será "secuencializado". El step es indicado para cada uno de los parámetros "iterados" de tal manera que el nivel de paralelismo viene indicado por la combinación de todos los step de los DG de entrada que son iterados. Una vez finalizadas todas las tareas del interior de bucle se volverá a evaluar la operación de la condición para comprobar si se vuelve a entrar en el bucle o no.

3.3 Motor de Ejecución de WINGS

Usando el lenguaje especificado, la definición de un workflow se describe por medio de una secuencia de "dataGroup – ejecución/operación – dataGroup". Dos ejecuciones u operaciones nunca se conectan directamente. Este esquema dirigido por los datos simplifica la descripción del workflow y aumenta su expresividad. También permite que cualquier ejecución/operación pueda ser lanzada cuando todos sus parámetros de entrada tengan al menos un dato para procesar, permitiendo el lanzamiento en paralelo de las tareas.

Tanto el lenguaje de definición como el motor de ejecución de WINGS han sido diseñados e implementados teniendo en cuenta los objetivos marcados: capacidades multi-grid y facilidad de extensión. Por ello el motor de ejecución ha sido diseñado de forma modular, para facilitar la extensión del mismo de manera sencilla (Figura 14).

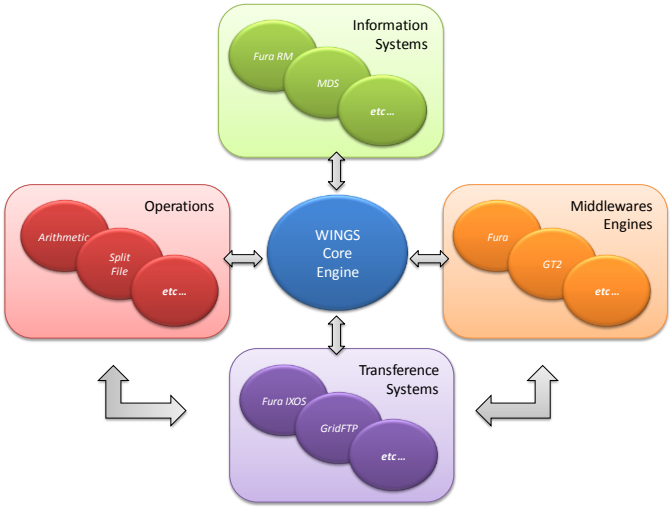

Figura 14. Arquitectura del motor de ejecución de WINGS

Cuatro subsistemas proporcionan la funcionalidad que permite la ejecucion de las tareas, junto con el componente central que los coordina. Todos ellos usan un sistema de "plug-ins" para permitir que la adición de nuevos componentes a cada uno de ellos para poder ser utilizados por el sistema completo.

- Middleware engines: Estos plug-ins proporcionan la funcionalidad para acceder a los diferentes middleware grid. Estos plug-ins son los encargados de afrontar los problemas

específicos de cada plataforma grid (envío de trabajos, control del estado de las tareas, etc.) de forma transparente al "core" de motor de ejecución.

- Transference system: Permite hacer las transferencias de datos entro los diferentes recursos grid. Cada plug-in proporciona la capacidad de transferencia que proporciona indicando un conjunto de protocolos. El motor central es el encargado de seleccionar en cada momento el elemento necesario para hacer las transferencias requeridas por las tareas en ejecución o para la realización de las replicas de datos. El "core" se encarga de analizar los protocolos de conexión de los dos "endpoints" implicados en la transferencia y consulta al conjunto de plug-ins de transferencia para obtener el o los plug-ins adecuados para realizar la transferencia.

  Cuando la transferencia se realice entre dos protocolos soportados por un solo plug-in, éste se encargará de todo el proceso de transferencia. En el caso de que un plug-in soporte el protocolo de origen y otro el de destino, el motor de ejecución se encargará de descargarse los ficheros usando el primer plug-in y subirlos los ficheros al destino con el segundo.

- Operaciones: Conjunto de operaciones disponibles para ser utilizadas en la definición de los workflows.

- Information systems: Conjunto de plugins que permite obtener información sobre los recursos disponibles en cada una de las infraestructuras de ejecución, para que el motor de workflow pueda elegir el mejor recurso donde lanzar los trabajos. En el caso de que no existiera un sistema de información para alguno de los despliegues siempre sería seleccionado en último lugar. Y si no hubiese ningún plug-in, los recursos serían seleccionados de forma aleatoria.
- Core Engine: El componente central a cargo de la coordinación de todos los elementos del sistema de plug-ins es el que se ha denominado como "core engine". Él es el encargado de la lectura del fichero XML con la definición del workflow y posteriormente del control de la ejecución.

3.3.1 Core Engine

El componente central o "core" se encarga del control y gestión de los diferentes plugins del sistema para conseguir el funcionamiento deseado. Una vez leído y analizado el fichero XML con la descripción del workflow se inicia la ejecución del workflow:

- Selección de tareas a lanzar: Se analizan todas las ejecuciones y operaciones para comprobar si tienen satisfechas todas sus dependencias de datos. Todas las tareas que las tengan satisfechas serán lanzadas de forma paralela.
- Ejecución de la tarea:
  - Operación: En el caso de ejecutar una operación el "core" debe seleccionar el plug-in adecuado para cada operación. Éste será el encargado de descargase los datos de entrada necesarios haciendo llamadas a subsistema de transferencias. Al finalizar devolverá la lista de resultados al dataGroup indicado.
  - Ejecución:
    - Selección recurso: Para ello se debe obtener el conjunto de recursos disponibles (especificados en la definición del workflow) para cada uno de los despliegues para los que haya sido definida la actividad que instancia la ejecución a lanzar. Para cada uno de ellos obtener su información usando el plug-in de sistema de información adecuado y seleccionar el de mejor prestaciones.
    - Lanzar la petición al plug-in adecuado: Una vez seleccionado el recurso, el lanzamiento del trabajo será enviado al plug-in encargado de lanzar al tipo de middleware de dicho recurso. El plug-in será el encargado de obtener los ficheros de entrada (usando el sistema de

transferencias), de lanzar de forma efectiva la tarea y controlar su ejecución, para finalmente devolver la lista resultados al dataGroup especificado en el workflow.

- Creación de Replicas: En caso que se haya indicado en la definición del workflow, al finalizar una tarea y enviar la lista de resultados al dataGroup de salida de la misma, se realizará una réplica de los resultados producidos al destino indicado en la réplica.
- A medida que los datos de salida de las tareas finalizadas van "llegando" a los dataGroups de salida de dichas tareas, se va analizando el workflow para comprobar que tareas pueden ser lanzadas. En el caso de que no queden datos nuevos por procesar en ninguno de los dataGroups y ya hayan finalizado todas las tareas se dará por finalizado el workflow.

3.3.2 Sistema de Plugins

El motor de ejecución de workflow ha sido diseñado para hacer los más sencillo posible la adición de nuevos componentes al sistema. Para añadir un nuevo plugin, del tipo que sea, simplemente es necesario crear una clase Java que implemente la interfaz específica de cada plugin. A continuación se detalla cada uno de los sistemas de plugins.

3.3.2.1 Plugin de Ejecución

Para crear una clase para el subsistema de ejecución es necesario implementar una clase que herede de la clase "GenericActivity". Esta clase implementa parte de la funcionalidad para el lanzamiento de trabajos, pero debe ser completada con una serie de funciones específicas para cada sistema de ejecución.

setResource: Está función se encarga de recibir los datos del recurso seleccionado por el dispatcher del "core" de ejecución para su posterior uso en el lanzamiento de la tarea. Los datos del recurso serán específicos para cada uno de los despliegues, y el plugin será responsable de interpretar correctamente su información a la hora del lanzamiento de la tarea.

prepareActivity: Realiza todos los preparativos necesarios para el lanzamiento de la tarea, dependiendo de cada entorno esto implica unas operaciones distintas: subir los archivos ejecutables , crear un módulo en Fura, etc.

prepareInitialData: Se encarga de enviar, al recurso donde se va a realizar la ejecución, todos los datos de entrada de la tarea a ejecutar.

execute: Ésta es la tarea que, de forma efectiva, lanza la tarea al entorno de ejecución, y la monitoriza hasta comprobar su finalización.

getOutputData: Una vez finalizada la tarea devuelve la lista de ficheros que han sido generados. En el caso de ficheros esta función no implica ninguna transferencia, solo se devuelven las referencias a los ficheros en la ubicación en la que hayan sido generados.

clean: Limpia todos los ficheros utilizados durante la ejecución, de tal manera que la máquina de destino quede sin ningún fichero.

cancel: Cancela la ejecución.

3.3.2.2 Plugin Sistema de transferencias

En el caso del sistema de transferencias se debe crear una clase que implemente el interfaz Transmitter, para el que se deben implementar las siguientes funciones:

isAvailable: esta función recibe como parámetro dos cadenas de texto que definen los protocolos a usar como puntos de origen y destino de una trasferencia, y debe indicar si la clase es capaz de entenderlos y de realizar dicha transferencia.

copyFiles: recibe dos TransferenceEndpoints como parámetros que le indican el origen y destino de los datos para efectuar la transferencia.

numberOfFiles: recibe un TransferenceEndpoint como parámetro y devuelve el numero de ficheros que se encuentran en dicha ubicación.

Una vez creada la clase se debe registrar dicha clase dentro del sistema para que pueda ser utilizado. Para ello se debe realizar una llamada similar a esta, preferiblemente desde la clase de la actividad.

```
static {
  FileTransferer.addTransmitter(FuraFileTransferer.class);
}
```

3.3.2.3 Plugin Operaciones

El caso de las operaciones es similar al de las ejecuciones pero con la particularidad de que la funcionalidad viene implícita con la operación. Una operación debe implementar una clase que herede de la clase "GenericFunction". Esta clase implementa parte de la funcionalidad para la ejecución de operaciones, pero debe ser completada con una serie de funciones específicas para cada operación.

executeOperation: realiza la función definida por la operación.

prepareInitialData: obtiene todos los datos de entrada (descargando en caso necesario) para la ejecutar operación.

getOutputData: Una vez finalizada la tarea devuelve la lista de ficheros que han sido generados.

3.3.2.4 Plugin Sistemas de Informacion

Por último el subsistema de información lo forman las clases que heredan de la clase "GenericDispatcher".

getResource: esta función toma la lista de recursos disponibles en el sistema, y aquellos que son del entorno de ejecución para el que ha sido programado, seleccionar el mejor asignándole una valoración.

3.3.3 Plugins para Fura

Para integrar el middleware Fura dentro del sistema WINGS se han desarrollado los plugins para los 3 subsistemas del motor de ejecución:

- Sistema de transferencia: Este plugin se encarga de gestionar la transferencia de ficheros con el sistema de ficheros propio de Fura. Para ello el plugin hace uso de las librerías que proporciona Fura con el conjunto de APIs para el acceso a su sistema de ficheros propietario denominado IXOS.
- Middleware de ejecución: Para integrar Fura, ha sido necesario el uso de las APIs de acceso al sistema de ejecución de Fura. Los pasos necesarios para lanzar una ejecución en Fura implican la creación de un módulo (en la función *prepareActivity*), para luego instanciarla en una tarea concreta con los datos de entrada subidos con la función *prepareInitialData* y finalmente ejecutar la tarea. En el caso de Fura es el propio sistema el que se encarga de generar todas las micro-tareas necesarias para procesar todos los datos de entrada especificados en la tarea, descargando de este trabajo al plugin.
- Sistema de información: Este plugin se encarga de obtener toda la información de los nodos de cálculo disponibles en un determinado despliegue Fura. De esta manera se

pueden obtener el número de nodos disponibles, sus características, y permitir al motor de workflow seleccionar el mejor recurso a la hora de lanzar las ejecuciones.

3.3.4 Plugins para SSH

En el caso de SSH solo han sido implementados dos de los plugins. El plugin de información no ha sido desarrollado y se deja al sistema la elección del mejor recurso:

- Sistema de transferencia: El plugin permite el acceso a nodos usando el protocolo "scp" utilizado en todos los recursos que usan SSH.
- Middleware de ejecución: Para la ejecución en sistemas con SSH el proceso es el siguiente: en la fase de preparación se crean los directorios de trabajo y se suben todos los ficheros necesarios para la ejecución (ejecutable, librerías, etc.), después tras la subida de los datos de entrada, el motor de workflow debe lanzar todas las ejecuciones que se correspondan con la combinación de todos los datos de entrada. En el caso de Fura esta parte es gestionada por el propio motor de ejecución de Fura, pero en este caso es el plugin el que debe realizar todas las combinaciones y lanzar el equivalente a las micro-tareas de Fura.

3.4 Control de Errores

El sistema workflow no se encarga del control de errores para gestionar el caso de que una tarea acabe de forma incorrecta. WINGS no trata de hacer tareas que son propias de la capa de middleware, sólo se encarga de la ejecución del flujo de trabajos. En algunos casos, como Globus Toolkit, el propio middleware no tiene ningún tipo de gestión de errores, pero otros (Fura, gLite, etc.) si lo gestionan y se encargan de relanzar las tareas finalizadas con error.

3.5 Selección de Recursos

El motor de ejecución de WINGS se encarga de ir seleccionando donde lanzar las tareas dependiendo de los recursos especificados en la definición de workflow y los despliegues para los que haya sido definida dicha tarea.
WINGS tiene una selección de recursos sencilla basada en el sistema de plugins de tal manera que son ellos los responsables de una selección más o menos avanzada. Aunque WINGS desde su diseño inicial no está pensado como un selector de recursos, ya que como se ha comentado anteriormente los recursos en WINGS no son simples recursos sino puntos de acceso a infraestructuras. De tal manera que el selector de recursos no trata de reemplazar a los actuales grid schedulers (como GridWay o Fura…) sino proporcionar un acceso homogéneo y transparente a todos ellos.

4 Casos de Uso

Para demostrar la funcionalidad del sistema WINGS vamos a mostrar dos casos de uso: el primero va a mostrar un ejemplo sintético, en el que se simula el caso de un proceso de optimización, utilizando muchas de las facilidades aportadas por WINGS para modelar un workflow, en el segundo caso se mostrará un caso práctico real dentro del campo de la biomedicina.

4.1 Caso sintético

En este primer caso vamos a crear un workflow de un caso genérico de un proceso de optimización, para demostrar cómo se puede modelar este tipo de procesos de manera sencilla usando los conceptos definidos en WINGS. En un proceso de optimización se desea obtener un conjunto de valores de entrada que consiguen obtener el mejor valor de una función de optimización. En el proceso existen dos procesos: la ejecución de la función de optimización que obtiene un valor numérico a partir de unos valores de entrada, y el generador de parámetros

que, tomando el mismo conjunto de valores de entrada que la función de optimización y el resultado obtenido en la misma, genera otro conjunto de valores de entrada que, en principio, obtendrá un valor mejor de la función. Por tanto el proceso completo es un proceso iterativo en el que se van ejecutando los dos pasos del proceso hasta que el valor obtenido por la función de optimización se encuentra dentro de un determinado rango.

El workflow completo aparece mostrado en la Figura 15. En él se muestran los dos pasos del proceso de optimización, completados con las operaciones necesarias para el funcionamiento efectivo del mismo. En el anexo se puede encontrar el código XML completo de este ejemplo.

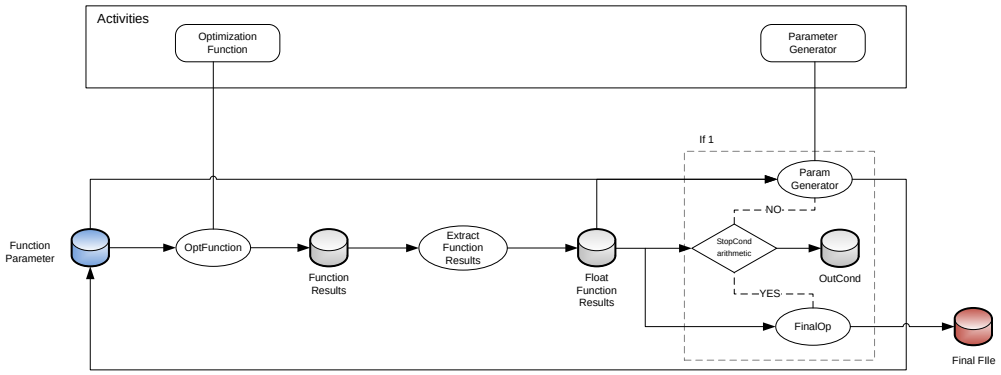

Figura 15. Ejemplo de workflow de optimización.

El funcionamiento del workflow sería el siguiente:

- Los datos iniciales se encuentran modelados por la fuente de datos "Function Parameter", y son representados como una cadena separada por espacios de una secuencia de números. El resto de dataGroup inicialmente se encontrarán vacíos. El dataGroup indicado es la entrada de la función "OptFunction" (una ejecución de la actividad "Optimization Function"), y por tanto, al tener datos disponibles, podrá ser lanzada. Como resultado de esta función obtendremos un fichero con un número obtenido por la función de optimización y que sería almacenada en el dataGroup "Function Results".
- Esto permite el lanzamiento de la operación "Extract Function Results". Esta operación es una instancia de la operación "getDataFileds" descrita anteriormente, que permite extraer valores numéricos de un fichero de texto. El valor extraído del fichero es almacenado en el dataGroup "Float Function Results".
- El siguiente paso es la ejecución de la estructura "if", ejecutando la operación aritmética "StopCond" para evaluar si el valor de entrada se encuentra por debajo de un determinado valor.
    - En caso negativo el proceso de optimización finalizaría y sería ejecutada la operación "FinalOp". La función es una instancia de la operación "popData" que permitiría obtener el fichero con los datos de entrada de la iteración actual usada en la operación "OptFunction" que sería almacenado en el dataGroup "Final File" y seria el dato de salida del workflow, dando por finalizada la ejecución.
    - En caso positivo el proceso de optimización continuaría llamando a la ejecución Param Generator (instancia de la actividad "Parameter Generator"). Esta ejecución puede ser lanzada puesto que tiene datos disponibles para calcular y se encarga de generar un nuevo valor de entrada en el dataGroup "Function Parameter", iniciando de nuevo el proceso completo.

4.2 Worflow de Proceso de Imágenes Médicas

Este segundo caso es una aplicación de co-registración de imagen médica. La co-registración de imágenes consiste en la alineación de los voxels de dos o más imágenes en el mismo espacio geométrico usando las transformaciones necesarias para hacer que todas las imágenes sean lo más parecido posible a la imagen de referencia. El proceso de co-registración puede ser rígido o elástico. En el caso rígido solo pude usar transformaciones afines (desplazamientos, rotaciones, escalados, etc.) a las imágenes. En el elástico se pueden utilizar también transformaciones de tipo elástico. El proceso de co-registración puede ser aplicado en 2D (individualmente en cada corte) o en 3D (usando el estudio completo). En el caso propuesto se van usar los dos tipos de transformaciones en su caso 2D.

El workflow completo es el mostrado en la Figura 16. Está compuesto por tres fases, la co-registración rígida, la elástica (de mayor consumo de CPU), y un último proceso en el que se transponen los N estudios (con K cortes) en K estudios con N cortes. Esto permite ver la evolución temporal de cada una de las áreas (cortes) de los estudios. En el anexo se puede encontrar el código XML completo de este ejemplo.

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

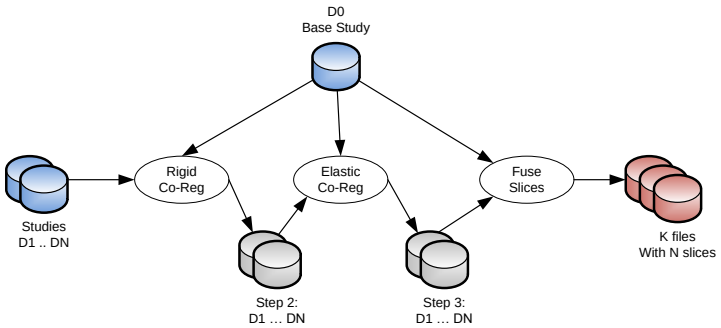

Figura 16. Ejemplo de workflow de co-registración.

En este segundo caso se ha utilizado un testbed mayor combinando dos tipos de infraestructuras (Globus y Fura) para probar la capacidad multigrid del sistema desarrollado. La infraestructura de Fura está compuesta por un servidor y dos agentes Linux. El primer agente (F1) es un AMD Opteron 2.4GHz con 1GB de RAM, y el segundo (F2) un AMD Opteron 2.2GHz con 1GB de RAM. En el caso de Globus se ha usado un pequeño clúster compuesto por dos nodos de cálculo, siendo cada uno de ellos (GN) un Pentium Xeon 2 GHz con 512 MB de RAM. Todos los elementos del testbed están conectados usando una red Gigabit Ethernet.

Para demostrar la capacidad multigrid y de adaptación a los diferentes entornos de ejecución que hayan disponibles en cada momento, se han realizado tres pruebas: la primera usando la infraestructura de Fura, la segunda con la de Globus, y la tercera combinado ambas.

Los resultados de las tres pruebas son mostrados en la Tabla 1. La fase de ejecución muestra el tiempo de las tres fases del workflow, solo mostrando el tiempo de de uso de CPU. La tabla muestra el tiempo de cada una de las ejecuciones, así como el nodo de las diferentes infraestructuras en la que ha sido ejecutado. El tiempo total de esta fase es la suma de todas las ejecuciones no paralelas de cada uno de los pasos. La operación de limpieza es realizada al finalizar el workflow, para borrar todos los archivos intermedios utilizados en las ejecuciones. En el caso de Globus, este paso es bastante costoso debido a la operación de borrado recursivo de directorios remotos. Esta operación no es soportada de forma nativa en el protocolo

GridFTP, por tanto, es necesario hacerlo de forma manual accediendo al directorio, listando los ficheros y borrándolos uno por uno. Finalmente el tiempo de "Overhead" incluye todo el tiempo invertido en la transferencia de ficheros entre los diferentes nodos, asi como los propios overheads introducidos por los middlewares y el propio sistema workflow. Como se ve en los resultados, en el caso de Globus el tiempo de overead el mecho mayor debidos a que sus protocolos son más pesados que en el caso de Fura.

La distribución de las tareas entre los diferentes recursos. En el caso de Fura, el "dispatcher" del servidor envía los trabajos a los mejores servidores disponibles, seleccionando primero el nodo F1 y luego el F2. En el caso de Globus los trabajos son enviados a un sistema de colas PBS, que se encarga de seleccionar el nodo final de ejecución. En este caso los nodos son homogéneos asi que no es relevante en que nodo se ejecutan los trabajos. En el caso mixto las ejecuciones de la co-registración rígida, al tener una corta duración, han sido lanzadas en los dos recursos Fura (los de mejores prestaciones). En el caso de la elástica, al tener mayor duración, las dos primeras ejecuciones son lanzadas a los nodos de Fura y, una vez que esos recursos se encuentran completos, las dos restantes son lanzadas en la infraestructura de Globus. La última tarea, también de corta duración, es lanzada en un recurso Fura.

	Fase de ejecución						
	CoReg Rig.	CoReg Elástica	Fuse Slices	Total	Limpieza	Overhead	Total
Fura	2'' (F1) 2'' (F2) 2'' (F1) 2'' (F2)	40' 16'' (F1) 43' 43'' (F2) 40' 05'' (F1) 41' 23'' (F2)	1'' (F1)	84' 6''	4''	55''	85' 5''
Globus	4'' (GN) 4'' (GN) 4'' (GN) 4'' (GN)	50' 23'' (GN) 50' 26'' (GN) 51' 39'' (GN) 50' 24'' (GN)	5'' (GN)	102' 14''	1'28''	3'28''	107'10''
Combinada Fura/Globus	2'' (F1) 2'' (F2) 2'' (F1) 2'' (F2)	39' 38'' (F1) 41' 32'' (F2) 50' 56'' (GN) 51' 09'' (GN)	1'' (F1)	51' 12''	23''	2'25''	54''

Tabla 1. Tiempos de ejecución workflow co-registración.

Esta prueba trata de demostrar el funcionamiento del WINGS usando un testbed combinando varios tipos de middleware. Como se puede ver en el código (adjunto en el anexo final) la especificación de la actividad es sencilla y simplemente es necesario proporcionar unos datos mínimos para cada uno de los despliegues disponibles. De esta manera si el usuario tuviera acceso a una nueva infraestructura que utilice un middleware distinto, las modificaciones necesarias en el código del workflow serían pequeñas.

Los resultados han mostrado el correcto funcionamiento del workflow ejecutado usando cualquiera de las tres alternativas. Lógicamente el caso combinado obtiene el mejor tiempo de respuesta dado que se puede utilizar toda la infraestructura, accediendo a un número mayor de recursos, lo que permite lanzar todas las tareas en paralelo. También se muestra como en el caso de Globus, el tiempo de overhead es superior debido al uso de protocolos más pesados, que en el caso de Fura.

5 Publicaciones y Proyectos

El trabajo realizado en esta tesis se inició enmarcado en el Proyecto CENIT en Cooperación "ITECBAN: Infraestructura Tecnológica y Metodológica de Soporte para un Core Bancario". Programa Nacional de Tecnologías De La Información y las Comunicaciones, Orden ITC72759/2005 en el que colaboraba el GRyCAP en colaboración con GridSystems. Como resultado de dicha colaboración WINGS fue incorporado a la versión 1.7 del middleware Fura (puede ser descargado de http://sourceforge.net/projects/fura/files/).

Mas adelante se continuó el trabajo enmarcado en el poyecto ngGrid – Componentes de Nueva Generación para la Explotación Eficiente de Infraestructuras Grid en e-Ciencia, TIN2006-

12890) donde se continuó su evolución para utilizarlo en un conjunto de aplicaciones mas extenso.

Como resultado del trabajo realizado en esta tesis de master se han realizado las siguientes publicaciones en congresos internacionales:

- Carlos de Alfonso; Miguel Caballer; Vicente Hernández. "WINGS: Versatile Workflow for the Grid". ADVCOMP 2008. pp 51 – 56.
- Carlos de Alfonso; Miguel Caballer; Vicente Hernández. "WINGS: A Multigrid Workflow Engine". Cracow Grid Workshop 2008. pp 129 – 136.
- C. de Alfonso; M. Caballer; V. Hernández; E. Martí. "Globus plug-in for WINGS Wokflow Engine". Cracow Grid Workshop 2009. pp 179 – 187.

6 Conclusiones y trabajos futuros

La posibilidad de acceder a diferentes tipos de despliegues así como una sencilla extensibilidad a nuevos entornos son características claves a la hora del desarrollo de un sistema workflow. Con esas ideas, en la presente tesis de master, se han analizado diferentes sistemas workflow actuales detectando sus características y sus puntos débiles para el desarrollo de un sistema workflow con las características deseadas.

Se han descrito los conceptos para el modelado de un workflow mediante un lenguaje sencillo, de alto nivel, y abstracto que permita a los investigadores describir los flujos de trabajo de sus tareas sin necesidad de conocer los detalles de los recursos que van a utilizar para su ejecución efectiva. Para ello se ha creado el lenguaje WINGS donde el flujo de ejecución es puramente controlado por los datos, de manera que es más intuitivo para la descripción de workflows y proporciona una serie de características adicionales como paralelismo implícito, bucles de datos automáticos, etc. El lenguaje se ha completado con una serie de estructuras de control para darle mayor potencia y flexibilidad. Se ha utilizado XML como lenguaje base para permitir una sencilla extensión en el futuro.

Se ha implementado un motor de ejecución de los workflows definidos con el lenguaje WINGS, diseñado de forma modular, de manera que sea sencilla su extensión para nuevos entornos de ejecución. Se han creado un sistema de plugins que permite añadir diferentes entornos de ejecución, protocolos de transferencia de ficheros, acceso a sistemas de información de forma independiente, de manera que la adicción de nuevos componentes al sistema se hace más sencilla.

Se han mostrado de casos de uso: uno teórico para mostrar la modelización de un workflow en el caso de un proceso de optimización, y un caso real donde se ha modelado un workflow de un caso de procesamiento de imagen médica. En este último caso se ha usado una plataforma mixta con un conjunto de recursos Fura y Globus para demostrar la interoperabilidad del sistema, obteniendo buenos resultados. Con estos casos de uso se ha podido comprobar la capacidad descriptiva del lenguaje definido y la funcionalidad del sistema, cumpliendo ambos con los objetivos marcados al inicio de la tesis.

En cuanto a las líneas de trabajo futuro hay dos principales: La primera implica la mejora y extensión del motor de workflow. En este parte se pueden citar la mejora del sistema de información para una mejor selección de los recursos disponibles, la creación de interfaz gráfico para el diseño y seguimiento de la ejecución del workflow de forma general, la adición de nuevos entornos de ejecución como gLite, PBS, etc.

La segunda línea va encaminada hacia una de las líneas de trabajo principal del grupo de investigación al que pertenezco: las tecnologías cloud. Este tipo de tecnologías se basan en la virtualización para la creación de infraestructuras de computación, así como su adaptación de forma elástica en tiempo de ejecución. En este campo las herramientas workflow se deben adaptar para permitir, no solo lanzar ejecuciones en los recursos disponibles, si no también, en

caso necesario aumentar o decrecer el número de recursos de acuerdo a las necesidades del usuario. La adaptación de WINGS para abarcar estos nuevos conceptos implicará un cambio importante tanto en el lenguaje de definición de workflows, para poder expresar las necesidades de elasticidad de la infraestructura del usuario, como en el motor de ejecución para permitir interactuar con los diferentes middleware cloud que puedan existir.

7 Referencias

[1] Globus Alliance – Globus Toolkit. http://www.globus.org/toolkit.

[2] EGEE – gLite - Lightweight Middleware for Grid Computing. http://glite.cern.ch/.

[3] Open Science Grid - OSG Software Stack - http://www.opensciencegrid.org/OSG_at_work/Software

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

[5] Foster I., Kesselman C. "The Grid: Blueprint for a New Computing Infrastructure". Morgan Kauffman, 1999.

[6] Foster I. "What is the Grid? A three point checklist". Grid Today 22 Julio 2002.

[7] The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. I. Foster, C. Kesselman, J. Nick, S. Tuecke, 2002.

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

[9] The WS-Resource Framework. K. Czajkowski, D. F. Ferguson, I. Foster, J. Frey, S. Graham, I. Sedukhin, D. Snelling, S. Tuecke, W. Vambenepe. March 5, 2004.

[10] Foster, I., Kesselman, C. “Globus: A Metacomputing Infrastructure Toolkit”. Intl J. Supercomputer Applications, 11(2):115-128, 1997.

[11] Foster, I. “Globus Toolkit Version 4: Software for Service-Oriented Systems”. IFIP International Conference on Network and Parallel Computing, Springer-Verlag LNCS 3779, pp 2-13, 2005.

[12] Globus Alliance – Crux Toolkit – http://confluence.globus.org/display/whi/Crux+Documents.

[13] I. Foster, "Globus Online: Accelerating and Democratizing Science through Cloud-Based Services," IEEE Internet Comput., vol. 15, pp. 70-73, 2011

[14] Workflow Management Coalition, “Glossary: A Workflow Management Coalition Specification”, Workflow Management Coalition Standard, WfMC-TC-1011, 1994.

[15] Workflow Management Coalition, “The Workflow Reference Model”, Workflow Management Coalition Standard, WfMC-TC-1003, 1994.

[16] Fox, G., Gannon, D. “Workflow in Grid Systems”. Concurrency and Computation: Practice and Experience. Vol. 18 Iss. 10. pp 1009 – 1019. 2006.

[17] E. Deelman, D. Gannon, M. Shields, Ian Taylor. “Workflows and e-Science: An overview of workflow system features and capabilities” Future Generation Computer Systems 25 (2009) 528 – 540.

[18] Yu J, Buyya R. A taxonomy of workflow management systems for Grid computing. Technical Report GRIDS-TR-2005-1, Grid Computing and Distributed Systems Laboratory, University of Melbourne, 2005. Available at: http://www.gridbus.org/reports/GridWorkflowTaxonomy.pdf.

[19] Kaizar Amin, Mihael Hategan, Gregor von Laszewski, Nestor J. Zaluzec, Shawn Hampton, and Al Rossi. GridAnt: A Client-Controllable Grid Workflow System. *In 37th Hawaii*

International Conference on System Science, Island of Hawaii, Big Island, 5-8 January 2004.

[20] G. Von Laszewski. Java CoG Kit Workflow Concepts for Scientific Experiments. Technical Report, Argonne National Laboratory, Argonne, IL, USA, 2005.

[21] Taylor I, Shields M, Wang I, Rana O. Triana applications within Grid computing and peer to peer environments. Journal of Grid Computing 2003; 1(2):199–217.

[22] Allen G et al. Enabling applications on the Grid: A GridLab overview. International Journal of High Performance Computing Applications (Special Issue on Grid Computing: Infrastructure and Applications) 2003; 17(4):449–466.

[23] Thomas Fahringer, Jun Qin and Stefan Hainzer Specification of Grid Workflow Applications with AGWL: An Abstract Grid Workflow Language. *Proceedings of Cluster Computing and Grid 2005 (CCGrid 2005)*

[24] Jia Yu and Rajkumar Buyya. A Novel Architecture for Realizing Grid Workflow using Tuple Spaces. Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04). 119 – 128.

[25] Tom Oinn, Mark Greenwood et al. "Taverna: lessons in creating a workflow environment for the life sciences". CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE 2006. 18:1067–1100

[26] Freefluo Workflow Enactor. http://freefluo.sourceforge.net/

[27] Bertram Ludäscher, Ilkay Altintas, Chad Berkley, Dan Higgins, Efrat Jaeger, Matthew Jones, Edward A. Lee, Yang Zhao, Scientific Workflow Management and the Kepler System. Concurrency and Computation: Practice and Experience; 18(10): 1039 - 1065.

[28] Ptolemy II project and system. Department of EECS, UC Berkeley, 2004. http://ptolemy.eecs.berkeley.edu/ptolemyII/.

[29] J. Cao, S. A. Jarvis, S. Saini, D. J. Kerbyson, and G. R. Nudd, "ARMS: an Agent-based Resource Management System for Grid Computing", Scientific Programming, Special Issue on Grid Computing, Vol. 10, No. 2, pp. 135 - 148, 2002.

[30] D. P. Spooner, J. Cao, J. D. Turner, H. N. Lin Choi Keung, S. A. Jarvis, and G. R. Nudd, "Localised Workload Management Using Performance Prediction and QoS Contracts", in Proc. of 18th Annual UK Performance Engineering Workshop, Glasgow, UK, pp. 69-80, 2002.

[31] G. R. Nudd, D. J. Kerbyson, E. Papaefstathiou, S. C. Perry, J. S. Harper, and D. V. Wilcox, "PACE – A Toolset for the Performance Prediction of Parallel and Distributed Systems", Int. J. High Performance Computing Applications, Special Issues on Performance Modelling – Part I, Vol. 14, No. 3, pp. 228-251, 2000.

[32] E. Deelman, Tevfik Kosar, Carl Kesselman, Miron Livny, What makes workflows work in an opportunistic environment?, Concurrency and Computation: Practice and Experience. Vol. 18 Iss. 10. pp 1187 – 1199.

[33] Condor Project Home Page. http://www.cs.wisc.edu/condor/

[34] GriPhyN - Grid Physics Network. http://www.griphyn.org/

[35] E. Deelman, et al., "Pegasus : Mapping Scientific Workflows onto the Grid," Proceedings of 2nd EUROPEAN ACROSS GRIDS CONFERENCE, Nicosia, Cyprus, 2004.

[36] Zhao Y., Hategan M., Clifford B., Foster I., von Laszewski G., Raicu I., Stef-Praun T., Wilde M. "Swift: Fast, Reliable, Loosely Coupled Parallel Computation". 2007 IEEE International Workshop on Scientific Workflows.

[37] W.M.P. van der Aalst and A.H.M. ter Hofstede. YAWL: Yet Another Workflow Language . Information Systems, 30(4):245-275, 2005.

[38] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow Patterns. Distributed and Parallel Databases, 14(1):5–51, 2003.

[39] Dozsa G, Kacsuk P, Lovas R, Podhorszki N, Drotos D. P-GRADE: A graphical environment to create and execute workflows in various Grids. Proceedings of Workflow in Grid Systems Workshop in GGF10, Berlin, Germany, March 2004. GGF: Berlin, 2004.

[40] Z. Balaton, P. Kacsuk, and N. Podhorszki, "Application Monitoring in the Grid with GRM and PROVE", Proc. of the Int. Conf. on Computational Science – ICCS 2001, San Francisco, pp. 253-262, 2001

[41] GridLab project, www.gridlab.org

[42] *Karen M. McCann, Maurice Yarrow, Adrian DeVivo, Piyush Mehrotra. ScyFlow*: an environment for the visual specification and execution of scientific workflows. Concurrency and Computation: Practice and Experience. Vol. 18 Iss. 10. pp 1155 – 1167.

[43] Knowledge-based Workflow System for Grid Applications (K-Wf Grid). http://www.kwfgrid.net

[44] The Grid Workflow Execution Service (GWES). http://www.gridworkflow.org/kwfgrid/gwes/docs/

[45] The Grid Workflow Description Language Toolbox. http://www.gridworkflow.org/kwfgrid/gworkflowdl/docs/

A Anexo

A.1 Código XML Caso Sintético

```
<workflow name="teorico">

  <definitions>

    <activity name="FuncionOptimizacion">
      <input name="parameto" type="String"/>
      <output name="salida1" type="File"/>
      <deployments>
        <furaDeployment class="wings.core.furaclient.FuraActivity">
          <module name="Optimizacion"/>
        </furaDeployment>
      </deployments>
    </activity>

    <activity name="GeneradorParametros">
      <input name="DatosEntradaFunc" type="String"/>
      <input name="ResultadoFunc" type="Float"/>
      <output name="salida1" type="File"/>
      <deployments>
        <furaDeployment class="wings.core.furaclient.FuraActivity">
          <module name="Generador"/>
        </furaDeployment>
      </deployments>
    </activity>

  </definitions>

  <authorisation>
    <userpass user="micafer" password="passwd"/>
  </authorisation>

  <data>

    <dataGroup name="ParametrosFuncion" type="String">
      <values>
        <value value="3 3 3 3 3 3 3 3 3 3"/>
      </values>
    </dataGroup>

    <dataGroup name="ResultadosFuncion" type="File"/>
    <dataGroup name="ResFuncionFloat" type="Float"/>
    <dataGroup name="ResGenParametros" type="File"/>
    <dataGroup name="OutCondFinal" type="Float"/>
    <dataGroup name="FicheroFinal" type="File">
      <filter in="*">
        <replica path="d:/datos/results"/>
      </filter>
    </dataGroup>

  </data>

  <executions>

    <execution name="FuncionOpt" activity="FuncionOptimizacion">
      <input name="parametro1" source="ParametrosFuncion"/>
      <output name="salida1" destination="ResultadosFuncion"/>
    </execution>

    <operation name="ConvertResultadosFuncion">
      <input name="parametro1" source="ResultadosFuncion"/>
      <output name="salida1" destination="ResFuncionFloat"/>
```

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

    <if name="if1">
      <condition>
        <operation name="CondParada">
          <input name="parametro1" source="ResFuncionFloat"/>
          <output name="salida1" destination="OutCondFinal"/>
          <arithmetic function="0.001 > %%1 => %%1"/>
        </operation>
      </condition>
      <then>
        <operation name="final">
          <input name="in" source="OutCondFinal"/>
          <output name="out" destination="FicheroFinal"/>
          <popData dataGroupOrigin="ResGenParametros"/>
        </operation>
      </then>
      <else>
        <execution name="GeneParam" activity="GeneradorParametros"
                   paramComposition="one-to-one">
          <input name="DatosEntradaFunc" source="ParametrosFuncion"/>
          <input name="ResultadoFunc" source="ResFuncionFloat"/>
          <output name="salida1" destination="ResGenParametros"/>
        </execution>
      </else>
    </if>

    <operation name="ConvertResultadosGeneradorParam">
      <input name="parametro1" source="ResGenParametros"/>
      <output name="salida1" destination="ParametrosFuncion"/>
      <getDataFields>
        <fields separator="\n">
          <field number="1"/>
        </fields>
      </getDataFields>
    </operation>
  </executions>

  <resources>
    <furaResource server="server1" port="9112" ssl="false"/>
  </resources>


</workflow>
```

A.2 Código XML Caso Corregistración de Imágenes Médicas

```
<workflow name="real">
  <definitions>
    <activity name="CoRegistracionRigida">
      <input name="basal_in" type="File"/>
      <input name="estudios_in" type="File"/>
      <input name="ficheroSalida" type="String"/>
      <output name="estudios_out" type="File"/>
      <deployments>
        <furaDeployment class="wings.core.furaclient.FuraActivity">
          <module name="RigidReg">
            <platform arch="linux">
              <executable path="E:/real" file="RigidReg.sh"/>
              <file path="E:/real" file="RigidReg.ex"/>
            </platform>
```

```
        </module>
      </furaDeployment>
      <executableDeployment class="wings.core.globus.GlobusActivity">
        <executable file="E:/real/RigidReg.sh"/>
        <file file="E:/real/RigidReg.ex"/>
      </executableDeployment>
    </deployments>
  </activity>

  <activity name="CoRegistracion">
    <input name="basal_in" type="File"/>
    <input name="estudios_in" type="File"/>
    <input name="ficheroSalida" type="String"/>
    <input name="p1" type="String"/>
    <input name="p2" type="String"/>
    <input name="p3" type="String"/>
    <input name="p4" type="String"/>
    <output name="estudios_out" type="File"/>
    <deployments>
      <furaDeployment class="wings.core.furaclient.FuraActivity">
        <module name="reggrid">
          <platform arch="linux">
            <executable path="E:/real" file="RigidReg.sh"/>
            <file path="E:/real" file="RigidReg.ex"/>
          </platform>
        </module>
      </furaDeployment>
      <executableDeployment class="wings.core.globus.GlobusActivity">
        <executable file=" E:/real/RigidReg.sh"/>
        <file file=" E:/real/RigidReg.ex"/>
      </executableDeployment>
    </deployments>
  </activity>

  <activity name="FuseSlices">
    <input name="cortes" type="String"/>
    <input name="outfile" type="String"/>
    <input name="basal" type="File"/>
    <input name="estudios" type="File" granularity="4"/>
    <output name="resultado" type="File"/>
    <deployments>
      <furaDeployment class="wings.core.furaclient.FuraActivity">
        <module name="FuseSlices">
          <platform arch="linux">
            <executable path="E:/real" file="FuseSlices.sh"/>
            <file path="E:/real" file="FuseSlices.ex"/>
          </platform>
        </module>
      </furaDeployment>
      <executableDeployment class="wings.core.globus.GlobusActivity">
        <executable file=" E:/real/FuseSlices.sh"/>
        <file file=" E:/real/FuseSlices.ex"/>
      </executableDeployment>
    </deployments>
  </activity>
</definitions>

<authorisation>
  <deployment type="Fura">
    <userpass user="micafer" password="passwd"/>
  </deployment>
  <deployment type="Globus">
    <certkey certfile="C:\usercert.pem" keyfile="C:\userkey.pem"/>
  </deployment>
</authorisation>

<data>
```

```
    <dataGroup name="basal" type="File">
      <filerange name="basal" path="E:/ref/" file="*.tgz"/>
    </dataGroup>

    <dataGroup name="estudios" type="File">
      <filerange name="estudio1" path="E:/otras" file="0810x02b.tgz"/>
      <filerange name="estudio2" path="E:/otras" file="0810x03b.tgz"/>
      <filerange name="estudio3" path="E:/otras" file="0810x04b.tgz"/>
      <filerange name="estudio4" path="E:/otras" file="0810x05b.tgz"/>
    </dataGroup>

    <dataGroup name="outCoRegRig" type="File">
      <filter in="*res_coreg_rig*.tgz"/>
    </dataGroup>

    <dataGroup name="outCoReg" type="File">
      <filter in="*res_coreg_ela*.tgz"/>
    </dataGroup>

    <dataGroup name="datosfin" type="File">
      <filter in="*dinam_reor*.img">
        <replica path="E:/real/results"/>
      </filter>
    </dataGroup>

  </data>

  <executions>

    <execution name="exCoRegRig" activity="CoRegistracionRigida">
      <input name="parametro1" source="basal"/>
      <input name="parametro2" source="estudios"/>
      <input name="ficheroSalida" value="res_coreg_rig"/>
      <output name="salida1" destination="outCoRegRig"/>
    </execution>

    <execution name="exCoRegistracion" activity="CoRegistracion">
      <input name="parametro1" source="basal"/>
      <input name="parametro2" source="outCoRegRig"/>
      <input name="ficheroSalida" value="res_coreg_ela"/>
      <input name="p1" value="0.05"/>
      <input name="p2" value="1.5"/>
      <input name="p3" value="1.5"/>
      <input name="p4" value="20"/>
      <output name="salida1" destination="outCoReg"/>
    </execution>

    <execution name="exFuseSlices" activity="FuseSlices">
      <input name="parametro1" value="5"/>
      <input name="parametro2" value="dinam_reor"/>
      <input name="parametro3" source="basal"/>
      <input name="parametro4" source="outCoReg"/>
      <output name="salida1" destination="datosfin"/>
    </execution>

  </executions>

  <resources>
    <globusResource>
      <gram server="server1" port="2119" jobmanager="jobmanager-pbs"/>
    </globusResource>
    <furaResource server="server2" port="9112" ssl="false"/>
  </resources>

</workflow>
```