Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

Framework para el despliegue automático de workflows científicos en entornos Grid

Proyecto Fin de Carrera
Ingeniería Informática

Autor

Sergio Hernández de Mesa

Directores

Francisco Javier Fabra Caro
Pedro Javier Álvarez Pérez-Aradros

Grupo de Integración de Sistemas Distribuidos y Heterogéneos (GIDHE)
Departamento de Informática e Ingeniería de Sistemas
Escuela de Ingeniería y Arquitectura
Universidad de Zaragoza

Curso 2010/2011
Septiembre 2011

RESUMEN

Framework para el despliegue automático de *workflows* científicos en entornos *Grid*

La heterogeneidad de las infraestructuras de computación existentes para el despliegue y ejecución de *workflows* científicos, junto con la complejidad de los *workflows* utilizados en *Grid*, producen que el proceso de *scheduling* sea una tarea muy complicada. Un enfoque que ayuda a resolver esta tarea es la utilización de un *broker* de recursos, pero esto provoca que las infraestructuras Grid asociadas estén fuertemente acopladas con el *middleware* subyacente. Por tanto, la integración de diferentes *middlewares* representa un desafío que sigue abierto.

En este proyecto se presenta un enfoque flexible para el despliegue de *workflows* científicos en entornos *Grid* heterogéneos. La propuesta realizada se fundamenta en la utilización de un sistema de coordinación basado en el modelo de Linda y un conjunto de mediadores que abstraen al sistemas de los detalles de los diferentes *middlewares* utilizados. Como resultado, la infraestructura permite ejecutar *workflows* científicos en diferentes entornos *Grid* de forma completamente transparente para el usuario. Además, la infraestructura desarrollada proporciona mecanismos que permiten escalar y extender la misma fácilmente, haciendo que el sistema sea adecuado para una gran variedad de escenarios.

Asimismo, el *framework* resuelve algunos de los problemas actuales en lo referente al modelado de *workflows* científicos. Los lenguajes de modelado utilizados en la actualidad son muy dependientes del sistema de gestión de *workflows* utilizado, dificultando la utilización del mismo en otros entornos y la compartición de los experimentos. Como solución, la infraestructura propuesta permite integrar diferentes entornos de modelado utilizando un lenguaje independiente del entorno de ejecución para describir las tareas que constituyen un *workflow*.

Las prestaciones del sistema se han analizado y evaluado en términos de rendimiento y escalabilidad, comparando los resultados obtenidos con otro sistema de gestión de *workflows* científicos como es el caso de Taverna.

Finalmente se presenta un caso de estudio correspondiente al *First Provenance Challenge*, por su importancia dentro del campo de las ciencias de la vida, y que sirve para demostrar la viabilidad de la propuesta desarrollada. Además, se proporcionan diferentes implementaciones del problema, mostrando la flexibilidad del sistema en lo que respecta a la utilización de diferentes entornos de modelado. Taverna y las redes de referencia se utilizan para modelar los *workflows*, y se integran varios *grids* de forma transparente, desplegando el *workflow* en las infraestructuras disponibles: Hermes, Aragrid, Piregrid y Amazon EC2.

Agradecimientos

Quiero agradecer a mis directores, por darme la oportunidad de involucrarme en este maravilloso proyecto y por la dedicación tanto profesional como personal que me han brindado, ha sido inmejorable.

A mis padres y a mi hermana, por su apoyo durante este proyecto y durante toda la carrera.

A mis compañeros de laboratorio, por las largas horas que hemos pasado juntos y todo lo que hemos disfrutado en esas cuatro paredes.

A mis compañeros de carrera, por todos los buenos momentos que hemos pasado durante estos años, tanto dentro como fuera del CPS, y por todos los que quedan.

Al resto de amigos y familia, por el apoyo brindado y por interesaros por mi proyecto aunque no os enteraseis de nada.

A Natalia por todo.

Índice

Índice de figuras

Índice de tablas

Capítulo 1 - Introducción

Las ciencias de la vida han experimentado una gran evolución en los últimos años. Diversas disciplinas como la biología, la genética, la química o la meteorología han mostrado un enorme cambio en como se desarrollan sus investigaciones. La forma de realizar los experimentos ha evolucionado drásticamente, pasando de pequeños experimentos llevados a cabo en el laboratorio o con computadores personales a grandes experimentos que necesitan de grandes infraestructuras de computación. De esta forma, la computación científica se ha convertido en el tercer pilar de la ciencia junto con el estudio teórico y la experimentación [B1].

Como respuesta a las crecientes necesidades de la comunidad investigadora, las infraestructuras de computación han ido evolucionando de acuerdo a los nuevos avances tecnológicos que han ido apareciendo. Supercomputación, computación distribuida, computación *Grid* y computación *Cloud* son algunas de las soluciones más empleadas.

Asimismo, las necesidades de los científicos no sólo han crecido en relación a la potencia de cálculo necesaria para realizar sus experimentos, sino que también han aumentado en cuanto a la complejidad de los mismos. Actualmente se necesitan complejos *workflows* científicos que coordinen un elevado número de tareas, gestionen complicadas relaciones entre los datos, etc. Como respuesta a esta necesidad, han aparecido los sistemas de gestión de *workflows* científicos como una herramienta muy útil para modelar y ejecutar los mismos en grandes infraestructuras de supercomputación.

Sin embargo, todavía quedan desafíos abiertos en este campo, como la necesidad de incluir sistemas heterogéneos de una forma transparente para formar infraestructuras de computación más potentes o la posibilidad de modelar los *workflows* científicos independientemente del entorno de ejecución utilizado.

1.1 - Contexto general

Los primeros experimentos científicos que necesitaban una gran potencia de computación recurrieron a la utilización de supercomputadores. Sin embargo, esta solución presentaba grandes problemas. En primer lugar, el elevado coste de estas infraestructuras hacía el uso de las mismas inalcanzable para la gran mayoría de investigadores. Por otra parte, la ejecución de las aplicaciones involucradas en los experimentos era tremendamente dependiente del supercomputador, por lo que llevaba mucho tiempo ponerla en marcha y el código generado no podía ser utilizado en otras infraestructuras.

En aquel momento, el mundo empresarial se enfrentaba al problema de automatizar sus procesos de negocio utilizando para ello aplicaciones informáticas. Este hecho provocó la aparición de los primeros *workflows* dentro del ámbito computacional. Un *workflow* se define como un conjunto de tareas, de cualquier tipo, conectadas entre sí en un determinado orden de acuerdo a las dependencias existentes entre las mismas. Dentro del ámbito de la computación, estas tareas se refieren a actividades computacionales o a otro *workflow* de las mismas características.

De la misma manera, la comunidad investigadora empezó a adoptar los *workflows* como herramienta para modelar sus experimentos. Estos ***workflows* científicos** se caracterizan por estar compuestos por un gran número de tareas, computacionalmente muy costosas, y un manejo complejo de los datos involucrados en las mismas. Los primeros *workflows* se representaban utilizando lenguajes muy complejos o a través de *scripts* que automatizaban la ejecución de las tareas.

Posteriormente, el auge de la computación distribuida provocó un movimiento hacia este nuevo paradigma ya que resultaba mucho más accesible para los investigadores y ofrecía grandes oportunidades. La adopción del mismo provocó que la utilización de *scripts* para ejecutar *workflows* quedase obsoleta, ya que no permite controlar y coordinar la ejecución de las tareas en diferentes recursos que se encuentran distribuidos a través de la red. Como solución, aparecieron los **sistemas de gestión de *workflows*** como sistemas capaces de controlar y coordinar la ejecución de estas tareas utilizando mecanismos de comunicación como llamadas a procedimientos remotos (*Remote Procedure Call*, RPC), tecnologías de objetos distribuidos y sistemas de ficheros y bases de datos distribuidos.

En la actualidad, el desarrollo de los sistemas distribuidos ha evolucionado en lo que hoy se conoce como **computación *Grid*,** un paradigma de computación distribuida en el cual un conjunto de recursos altamente heterogéneos y geográficamente distribuidos se engloban de forma transparente para trabajar como uno solo [B2]. Las capacidades de las infraestructuras Grid han derivado en entornos de ejecución adecuados para *workflows* científicos, provocando que la comunidad científica haya adoptado este nuevo paradigma como base para sus investigaciones. Por su parte, las tecnologías de comunicación han evolucionado hacia arquitecturas orientadas a servicios web (SOAP y REST), lo que también ha influido en los sistemas actuales de gestión de *workflows* científicos.

Sin embargo, la heterogeneidad de las infraestructuras *Grid* y la propia complejidad que presentan los *workflows* científicos provocan que todavía existan importantes problemas. Por su naturaleza, los *Grids* son infraestructuras altamente heterogéneas, sus recursos son muy dispares y dinámicos, y los elementos que los forman pueden cambiar constantemente. De estos aspectos surge la necesidad de utilizar un *middleware* que facilite el control y la coordinación de los trabajos en los diferentes recursos. Este *middleware* es el encargado de conocer las características y el estado de los recursos, mover los datos necesarios entre los mismos, gestionar la ejecución de los trabajos y gestionar los fallos que aparezcan en la ejecución de un trabajo, principalmente.

Por su parte, los sistemas de gestión de *workflows* incorporan un elemento, conocido como ***broker* de recursos**, que se encarga de gestionar el estado de los recursos en cada momento. Para ello, resulta clave la utilización de la información suministrada por el *middleware*. El *broker* se utiliza para tomar la decisión de en qué recurso debe ejecutarse cada tarea (***scheduling***), decisión que puede tomarse tanto de forma estática (antes de ejecutar el *workflow*) como de forma dinámica (en el momento de ejecutar cada tarea). Sin embargo, la naturaleza evolutiva y cambiante del sistema aconseja la utilización de un *scheduling* dinámico.

Estas circunstancias provocan que el *broker* de recursos y, en consecuencia, el sistema de gestión de *workflows*, sean altamente dependientes del *middleware* utilizado. Esto se traduce en que migrar el sistema de un *middleware* a otro, o ajustar el mismo para trabajar simultáneamente con recursos de diferentes *middlewares*, es un tarea muy complicada [B3]. A su vez, esto imposibilita la integración de diferentes entornos *grid* para crear infraestructuras computacionales mucho más potentes.

Muy recientemente ha surgido el paradigma de la **computación** *Cloud* o computación en la nube como alternativa a la computación *Grid*. Este nuevo paradigma se basa en la publicación y consumo de servicios y el aprovisionamiento de recursos de computación a través de Internet, bajo demanda y de una forma flexible, adaptable y altamente configurable. Una de las principales diferencias de la computación *Cloud* respecto a la computación *Grid* es que, en el primero, el usuario no conoce la ubicación real de los recursos computacionales que está utilizando si no que en su lugar trabaja en una máquina virtual que puede corresponder a cualquiera de los recursos del proveedor del servicios. Esto posibilita un estilo de computación distribuido, flexible y escalable, gestionado de forma transparente al usuario final. Aunque la computación *Cloud* está teniendo un gran éxito en el mundo empresarial, la adopción en el ámbito científico es todavía muy baja por la falta de infraestructuras que permitan la gestión de *workflows* científicos en infraestructuras de computación *Cloud* y el coste de las mismas.

Esta heterogeneidad en los recursos de computación y, en consecuencia, en los sistemas de gestión de *workflows* científicos, unida a la falta de estándares para el modelado de los mismos, provoca que los experimentos programados por los científicos e ingenieros sean muy dependientes del entorno de ejecución. Como consecuencia, la programación de *workflows* científicos independientemente del entorno en el que van a ser ejecutados y la portabilidad de los mismos entre entornos de ejecución diferentes son desafíos que todavía no se han resuelto.

1.2 - Motivación

La aparición de nuevos paradigmas de computación como la computación *Grid* y de sistemas de gestión de *workflows* ha permitido que los científicos realicen experimentos cuya ejecución era impensable hace unos pocos años, favoreciendo enormemente el avance de la ciencia. Sin embargo, todavía existen serios problemas que limitan el avance de este campo. Principalmente, la heterogeneidad de las diferentes infraestructuras *Grid* existentes provoca que los sistemas de gestión de *workflows* sólo sean capaces de interactuar con un único *grid*, o como mucho con *grids* gestionados por el mismo *middleware*. Este aspecto limita enormemente la capacidad de cálculo que los investigadores tienen a su alcance.

Cada día es más habitual que los investigadores tengan acceso a varias infraestructuras de computación, gestionadas por centros, grupos o instituciones diferentes. Sin embargo, la heterogeneidad habitual entre dichas infraestructuras imposibilita la utilización de las mismas de forma conjunta, por lo que el usuario se ve obligado a elegir una de ellas para realizar su experimento. Si además tenemos en cuenta que estos sistemas tienen un gran número de usuarios que ejecuta trabajos a la vez, las posibilidades de que los trabajos tarden más en empezar a ejecutarse o sufran expulsiones y tengan que ser reiniciados crece enormemente, provocando que la duración del experimento se prolongue más de lo realmente necesario.

Un sistema que permita la integración de varios entornos de ejecución diferentes para que se comporten como uno sólo, permitiría solventar estos problemas. En primer lugar, ofrecería una mayor potencia de cálculo haciendo que el investigador pueda aprovechar todos los recursos que se encuentran a su alcance. En segundo lugar, se minimizaría la espera necesaria para ejecutar un trabajo y el riesgo de sufrir expulsiones al contribuir a que los sistemas tengan una menor carga local, al ejecutar las diferentes tareas del *workflow* de una forma balanceada entre los diferentes recursos existente. Esto último está estrechamente relacionado con los diferentes mecanismos de prioridad que se asignan en las infraestructuras a los usuarios que menos estresan el sistema. Una política adecuada de despliegue de trabajos en base a su carga computacional favorecerá la prioridad que se le dé al usuario en los diferentes sistemas, contribuyendo a mejorar la calidad del servicio.

Por otra parte, un aspecto fundamental en el proceso de investigación en cualquier disciplina y, en especial, en el campo de las ciencias de la vida, es la capacidad de reutilizar y compartir los experimentos realizados. Una fase habitual de la investigación es la de búsqueda de técnicas, herramientas y métodos que permitan resolver dichos problemas mediante el análisis de diferentes publicaciones. Sin embargo, en este proceso surge habitualmente el problema derivado de la complejidad de evaluar, comprobar y comparar los resultados de una publicación con los resultados de otra. Esta dificultad surge por la heterogeneidad de las técnicas actuales de modelado. Cada sistema utiliza su propio lenguaje para modelar los *workflows* que ejecuta, siendo éste muy dependiente del mismo y, por tanto, muy dependiente de la infraestructura de ejecución. Este aspecto provoca que la repetición del mismo experimento en otra infraestructura diferente sea muy complicado.

La utilización de un lenguaje de modelado independiente de la infraestructura aporta valiosas mejoras a este proceso de comprobación y evaluación así como la compartición de experimentos entre los diferentes investigadores, lo que facilitaría a su vez las labores de investigación y favorecería al avance de la ciencia. Igualmente, el desarrollo de un sistema de gestión de *workflows* que permitiese diferentes lenguajes de modelado conseguiría el mismo efecto.

1.3 - Objetivos de este proyecto

Este proyecto se enmarca dentro del Grupo de Integración de Sistemas Distribuidos y Heterogéneos (GIDHE) [W1], el cual forma parte del Departamento de Informática e Ingeniería de Sistemas (DIIS) de la Universidad de Zaragoza. Una de las líneas principales de este grupo está orientada al análisis y estudio de posibles soluciones para resolver los diferentes problemas comentados anteriormente.

El objetivo principal de este Proyecto Final de Carrera es desarrollar un *middleware* que permita la integración de diferentes infraestructuras de computación heterogéneas. Para ello se propone la utilización de un *broker* de mensajes basado en el modelo de coordinación Linda [B4, B5] y un conjunto de mediadores que encapsulan los diferentes entornos de ejecución, basándose en la experiencia del grupo GIDHE en este tipo de soluciones en otros ámbitos de aplicación.

La utilización de un conjunto de mediadores nos permite integrar varios *middleware* diferentes de una forma sencilla. Cada mediador encapsula y gestiona las características específicas del entorno de ejecución, de una manera completamente transparente para el usuario, haciendo que el mismo no tenga que preocuparse de la heterogeneidad de los recursos de computación utilizados. Como resultado, los *workflows* implementados utilizando la infraestructura propuesta pueden ejecutarse en diferentes *grids*, como el *cluster* Hermes [W2] del Instituto Universitario de Investigación en Ingeniería de Aragón (I3A) [W3], que es manejado por Condor [W4], o los *grids* Piregrid [W5] y Aragrid [W6], gestionados por gLite [W7]. De la misma forma, diferentes tareas de un mismo experimento podrían ejecutarse en diferentes infraestructuras sin ningún tipo de esfuerzo adicional por parte del usuario.

El paradigma de las redes en redes [B6] y la herramienta Renew [B7] se utilizan para lograr la independencia entre el modelado de *workflows* científicos y el sistema responsable del entorno Grid y para permitir la portabilidad de los *workflows* desarrollados entre diferentes entornos de ejecución. Las redes de referencia, pertenecientes al paradigma de las redes en redes, una subclase de las redes objeto basada en el formalismo de las redes de Petri de Alto Nivel, permiten representar, modelar e implementar sistemas concurrentes y distribuidos. La herramienta Renew proporciona un entorno gráfico de modelado y ejecución utilizando este paradigma. En cualquier caso, la utilización de estas tecnologías no es obligatoria y se posibilita tanto la integración del sistema en otros entornos como Taverna [W8] como la traducción de *workflows* científicos modelados utilizando otros lenguajes.

Concretamente, el *framework* abarcará los siguientes aspectos:

- Coordinación mediante un *broker* de mensajes basado en Linda. Como se ha comentado anteriormente, se utilizará un *broker* de mensajes basado en el modelo de coordinación Linda para permitir la integración de diferentes entornos *Grid* de una forma transparente.
- Escalabilidad. El sistema debe ser escalable con respecto a las infraestructuras de computación que pueden ser añadidas al mismo, el número de componentes adicionales que pueden ser utilizados y el número de tareas que pueden ser gestionadas.
- Flexibilidad. El *framework* debe ser flexible desde varios puntos de vista:
    1. Debe ser flexible desde el punto de vista de la integración de distintos componentes al sistema. La incorporación de los mismos al *framework* debe ser transparente para el resto de componentes involucrados.
    2. Debe ser flexible desde el punto de vista de los entornos de ejecución que se pueden utilizar. No deben existir limitaciones sobre las características de los entornos de ejecución ya que se pretende lograr una infraestructura en la que el entorno de ejecución utilizado sea completamente independiente del resto de componentes del sistema.
    3. Debe ser flexible desde el punto de vista de los *workflows* que se pueden ejecutar. El sistema debe ser capaz de ejecutar cualquier tipo de *workflow* con las únicas limitaciones de que el modelo del mismo sea correcto y exista como mínimo una infraestructura de computación que sea capaz de ejecutarlo.

4. Debe ser flexible desde el punto de vista de los usuarios que pueden acceder al sistema. El *framework* puede ser utilizado por diferentes usuarios que pueden tener permisos de acceso a diferentes entornos de ejecución. Por tanto, el sistema debe ser capaz de gestionar los diferentes usuarios existentes junto con los diferentes recursos accesibles por parte de cada usuario. Existen diversas formas de implementar esta funcionalidad, como los perfiles de usuario o las políticas de personalización de usuario.

- Adaptabilidad y tolerancia a cambios dinámicos en el entorno. El *framework* debe ser capaz de adaptarse a las condiciones cambiantes del entorno en lo que se refiere a usuarios existentes, infraestructuras disponibles y aplicaciones definidas. De esta forma, debe ser posible definir nuevos usuarios, entornos de ejecución y aplicaciones sin que la ejecución del sistema se vea afectada.
- Gestión de los fallos encontrados. El sistema debe ser capaz de gestionar los fallos que aparezcan al ejecutar un *workflow*, ya sean fallos motivados por un error en el modelo o por un fallo en la infraestructura de computación.
- Incorporación con los entornos de computación disponibles. El *framework* debe incorporar las diferentes infraestructuras *Grid* de computación disponibles como son el caso de Hermes, Piregrid y Aragrid. Además, debe permitir la inclusión de infraestructuras de computación confiables, en las que se garantiza que no van a existir fallos en los recursos de computación. En el caso que nos ocupa se utilizará Amazon EC2 (*Amazon Elastic Compute Cloud*) [W9] como infraestructura confiable.
- Independencia entre la capa de modelado y la capa de despliegue y ejecución. El modelado de *workflows* debe ser independiente de la infraestructura concreta en la que se va a ejecutar el mismo.
- Compatibilidad con otras herramientas de modelado de *workflows* científicos. Gracias a la independencias entre la capa de modelado y la capa de ejecución, el sistema debe ser capaz de desplegar *workflows* modelados con otra herramienta diferente.
- Facilidad de utilización y configuración. La configuración y utilización del sistema debe ser sencilla. No debe ser necesaria la intervención de una persona especializada en el manejo de la infraestructura para configurar el sistema o para ejecutar los *workflows*.

1.4 - Organización de la memoria

El resto de esta memoria se organiza de la siguiente forma:

- **Capítulo 2. Puesta en contexto**. Se describen los conceptos fundamentales propios del contexto en el que se enmarca el proyecto.
- **Capítulo 3. Estado del arte**. Se analizan los diferentes sistemas existentes en la actualidad y se realiza una comparación exhaustiva de los mismos.
- **Capítulo 4. Modelado e implementación**. Se presenta el diseño y la implementación del sistema.
- **Capítulo 5. Evaluación del sistema**. Se analiza el sistema desde el punto de vista de rendimiento y escalabilidad, comparando el mismo con otros sistemas existentes.

- **Capítulo 6. Caso de estudio: First Provenance Challenge**. Se proporciona un caso de estudio en el área de la obtención de imágenes cerebrales de alta resolución.
- **Capítulo 7. Conclusiones y trabajo futuro**. Se ofrecen las conclusiones, tanto a nivel técnico como personal, a las que se ha llegado tras la realización del proyecto. Asimismo, se describen posibles líneas futuras de trabajo.

Adicionalmente, se incluyen una serie de anexos que ofrecen información adicional para completar los contenidos ofrecidos en la memoria principal:

- **Anexo I - Gestión del proyecto.** Se describe la metodología de trabajo utilizada, la gestión del tiempo y el esfuerzo y las herramientas de gestión utilizadas.
- **Anexo II - Tecnologías utilizadas.** Se enumeran las principales tecnologías utilizadas durante el desarrollo de este proyecto, añadiendo una breve descripción de las mismas e indicando en qué parte del sistema se han utilizado.
- **Anexo III - Manual de usuario.** Se proporciona el manual de usuario que describe como instalar y utilizar el sistema.
- **Anexo IV - Registros del sistema.** Se detallan los aspectos más relevantes de los registros de información utilizados en el sistema. Se proporcionan los esquemas que deben seguir los ficheros de configuración y ejemplos de los mismos.
- **Anexo V - Broker de coordinación.** Se describen las modificaciones realizadas en el *broker* de coordinación utilizado.
- **Anexo VI - Componente de movimiento de datos.** Se detalla el componente de movimiento de datos, indicando su utilidad y detalles acerca de la implementación del mismo.
- **Anexo VII - Componente de gestión de fallos.** Se detalla el componente de gestión de fallos, indicando los fallos que son gestionados, las políticas de recuperación ante dichos fallos y detalles sobre la implementación del componente.
- **Anexo VIII - Mediadores.** Se detallan los aspectos más relevantes de la utilización e implementación de los diferentes mediadores utilizados.
- **Anexo IX - Evaluación del sistema**. Se analiza el sistema desde el punto de vista de rendimiento y escalabilidad, comparando el mismo con otros sistemas de gestión de *workflows* científicos.
- **Anexo X - Caso de estudio: First Provenance Challenge.** Se proporciona un caso de estudio en el área de la obtención de imágenes cerebrales de alta resolución.

Además, se incluyen dos secciones adicionales tras los anexos:

- **Glosario**: Listado de términos y abreviaturas utilizadas frecuentemente en esta memoria junto con su significado.
- **Bibliografía**: Listado de referencias bibliográficas y referencias web utilizadas en esta memoria.

Capítulo 2 - Puesta en contexto

Hay una gran cantidad de términos y conceptos necesarios para comprender el contexto de este proyecto y las diferentes magnitudes del mismo. En este capítulo, vamos a explicar los más relevantes para facilitar la comprensión tanto de la memoria como del proyecto en general.

2.1 - Computación *Grid*

La computación *Grid* es un paradigma de computación distribuida en el cual un conjunto de recursos altamente heterogéneos y geográficamente distribuidos se engloban de forma transparente para trabajar como uno solo. De esta forma, se logra la visión de tener un solo computador muy potente, con recursos muy elevados, tanto en lo que respecta a potencia de cálculo como a memoria o a capacidad de almacenamiento.

En general, este tipo de sistemas son sistemas de alta disponibilidad y de propósito general que dan servicio a un gran número de usuarios. Los usuarios que tienen características, intereses y objetivos comunes forman una organización virtual (*Virtual Organization,* VO) y definen su propia forma de compartir información. Las VOs facilitan la compartición de contenidos entre sus usuarios, de una forma segura, sin que los usuarios de otras VOs tengan acceso a dichos contenidos.

Una de las características clave de este tipo de infraestructuras es la heterogeneidad de los recursos que la componen. Un *grid* puede estar compuesto por ordenadores de muy diversas características utilizando diferentes Sistemas Operativos. Otro aspecto clave es la naturaleza dinámica de este tipo de sistemas. Los recursos que componen el *Grid* varían constantemente, haciendo que las características del mismo estén en continuo cambio. Debido a las dos circunstancias anteriores, los sistemas *Grid* están sujetos a la aparición de fallos. Los cambios que surgen en el entorno de computación pueden provocar fallos en los trabajos que se están ejecutando en el *Grid*, por ejemplo, si el trabajo se está ejecutando en un nodo que se desconecta del *Grid*. Asimismo, debido al elevado número de usuarios que acceden al *Grid*, es posible que el trabajo de un usuario sea expulsado de un nodo para ejecutar en el mismo un trabajo de otro usuario con mejor prioridad.

Los tres aspectos anteriores hacen necesaria la utilización de un *middleware* que gestione la ejecución de trabajos en la infraestructura. Un *middleware* es un sistema que se encarga de englobar y gestionar todos los servicios y recursos que ofrece el *Grid* bajo una única interfaz común. Algunos de los aspectos principales de los que se encarga el *middleware* son: gestión dinámica de los recursos, despliegue de los trabajos en los diferentes nodos del *Grid*, gestión de los fallos producidos, control de acceso y seguridad, etc.

2.2 - Computación *Cloud*

La computación *Cloud* o computación en la nube es un nuevo paradigma de computación que para muchos corresponde con la evolución natural de la computación *Grid*. Este nuevo paradigma se basa en el ofrecimiento de servicios y aprovisionamiento de recursos de computación a través de Internet bajo demanda y de una forma flexible, adaptable y altamente configurable.

Una de las principales diferencias de la computación *Cloud* respecto a la computación *Grid* es que, en este nuevo paradigma, el usuario no conoce la ubicación real de los recursos computacionales que está utilizando si no que en su lugar trabaja en una máquina virtual que puede corresponder a cualquiera de los recursos del proveedor del servicio de *Cloud*. La principal ventaja que aporta es la posibilidad de solicitar recursos bajo demanda de una forma flexible. El usuario solicita recursos de computación para satisfacer las necesidades que tiene en un momento determinado, pagando solamente por el consumo efectuado y no de una forma global. De esta forma, el usuario puede ejecutar sus experimentos en una infraestructura que se adapta a las necesidades del mismo provocando un mejor aprovechamiento de los recursos. Por contra, uno de sus principales problemas es que la información gestionada por el usuario también se almacena en la nube, es decir, en los servidores del proveedor. Esto implica que ahora el usuario se hace dependiente del proveedor del servicio de *Cloud* ya que éste almacena su información. Otro de los problemas que presenta esta aproximación es que, al almacenar el proveedor la información y gestionar la misma, el usuario pierde el control sobre su información con los consiguientes problemas de seguridad y propiedad que esto conlleva.

La computación *Cloud* está teniendo un gran éxito en el mundo empresarial. Sin embargo, la adopción en el ámbito científico es todavía muy baja por la falta de infraestructuras que permitan la gestión de *workflows* científicos en infraestructuras de computación *Cloud* y por el coste asociado de este nuevo paradigma. Por ello, su utilización actual se limita a cubrir necesidades puntuales para poder cumplir fechas límites.

2.3 - *Workflows* científicos

Un *workflow* se define como un conjunto de tareas, de cualquier tipo, conectadas entre sí en un determinado orden de acuerdo a las dependencias existentes entre las mismas. Dentro del ámbito de la computación, estas tareas se refieren a actividades computacionales o a otro *workflow* de las mismas características. En el ámbito de las ciencias de la vida los *workflows* se denominan *workflows* científicos y se caracterizan por estar compuestos por un gran número de tareas, computacionalmente muy costosas, y un manejo complejo de los datos involucrados.

Los *workflows* se han establecido como una de las herramientas principales para representar los experimentos realizados por los investigadores de los diferentes campos de las ciencias de la vida. Sin embargo, no existen lenguajes, metodologías o herramientas estándares para su modelado. Por tanto, han surgido una gran variedad de alternativas para el mismo, complicándose enormemente compartir los modelos realizados entre diferentes investigadores.

Por su parte, los sistemas de gestión de *workflows* científicos ofrecen su propio lenguaje de modelado. Por cuestiones de simplicidad, los usuarios utilizan el lenguaje asociado al sistema de gestión que les permite ejecutar sus experimentos debido a que la traducción entre lenguajes de modelado no es siempre sencilla e introduce un paso extra en el proceso. Esto provoca que los modelos sean muy dependientes del sistemas y, en consecuencia del *Grid* en el que se ejecuten los mismos, complicando su reutilización por parte de otras personas.

Los *workflows* se pueden clasificar en dos grandes grupos: *workflows* dirigidos por el control y *workflows* dirigidos por los datos. En los primeros, las interacciones entre las diferentes tareas que forman el *workflow* definen las dependencias entre las mismas. En los

segundos, las dependencias representan el flujo de los datos entre las tareas que producen los mismos y las tareas que los consumen. Asimismo, también existen lenguajes híbridos que mezclan características de ambos grupos.

No existe un consenso sobre que tipo de *workflows* son mejores, si no que suele depender del tipo de problema o del dominio de aplicación del mismo. El problema reside en que en muchas ocasiones el lenguaje proporcionado por el sistema no es adecuado para representar el *workflow* lo que da lugar a problemas en el modelado del mismo.

2.4 - Sistemas de gestión de *workflows* científicos

Los sistemas de gestión de *workflows* ofrecen herramientas y servicios para modelar, desplegar y ejecutar *workflows* científicos de un modo automático. El objetivo principal de estos sistemas consiste en facilitar la labor del usuario abstrayendo al mismo de la complejidad inherente que presentan los recursos físicos en los que se despliegan las tareas. Con ese mismo objetivo, estos sistemas automatizan algunas tareas como el movimiento de datos que hay que realizar entre los recursos para permitir la ejecución de las tareas, la obtención de los *logs* o registros de ejecución de las tareas ejecutadas o la coordinación de las diferentes tareas de acuerdo a las dependencias establecidas por el usuario en el modelo.

Este tipo de sistemas surgió como respuesta a las necesidades de los científicos a la hora de ejecutar sus experimentos en infraestructuras de computación de alto rendimiento. En este tipo de entornos de ejecución, la coordinación de las diferentes tareas de un *workflow* o el movimiento de datos entre los diferentes nodos de la infraestructura se vuelve un aspecto complicado. Asimismo, los sistemas de gestión de *workflows* científicos también intentan facilitar la labor del usuario automatizando procesos repetitivos como puede ser la visualización de los resultados o la recuperación de los *logs* de ejecución.

Para entender este tipo de sistemas, es necesario conocer cómo es el usuario que utiliza los mismos. Normalmente los usuarios son científicos de diferentes campos de las ciencias de la vida como biología, geología, genética, astronomía, etc. En general, los científicos de estos campos tienen un bajo conocimiento de aspectos informáticos. Esto se traduce en que en general el científico no está dispuesto a realizar un esfuerzo extra para aprender como se utilizan diferentes sistemas y valorar cuál es el más adecuado para sus intereses. En su lugar, en general, los científicos se limitan al aprendizaje y utilización de un único sistema.

Por tanto, un requisito fundamental en el desarrollo de estos sistemas es la simplicidad en lo que se refiere a la interacción con el usuario, ya que este es un aspecto clave para el mismo. Por esta razón, en general, los sistemas de gestión de *workflows* científicos proporcionan interfaces gráficas para componer los mismos de una forma sencilla e intuitiva. Sin embargo, esto provoca que los *workflows* desarrollados no puedan ser utilizados por otros sistemas, lo que dificulta la compartición de los mismos. Debido a esto, algunos sistemas se han orientado hacia un grupo determinado proporcionando servicios avanzados orientados al desarrollo de *workflows* dentro de ese campo de aplicación, es el caso de Taverna [W8] que está orientado al campo de la bioinformática. Este último aspecto es una de las claves de la existencia de numerosos sistemas de gestión de *workflows* científicos y de que no haya ningún sistema que se haya impuesto al resto.

Por otra parte, como contraposición a la sencillez que buscan los sistemas en la interacción con los usuarios, estos sistemas utilizan técnicas complejas para coordinar y ejecutar los *workflows*. Las infraestructuras de computación son altamente heterogéneas y se gestionan a través de *middlewares* muy diferentes, esto provoca que los sistemas de gestión de *workflows* sean muy dependientes del entorno de ejecución y hace necesario la utilización de mecanismos complicados para permitir la automatización de los diferentes aspectos ya comentados. A su vez, esto introduce un problema adicional: la configuración de los sistemas en lo que respecta al entorno de ejecución es muy compleja y requiere de personal especializado, lo que dificulta la puesta en marcha del sistema.

Las diferentes alternativas que han aparecido siguen todas la misma filosofía: proporcionar un lenguaje de modelado sencillo de utilizar y que a la vez permita modelar *workflows* científicos complejos y sacar el máximo rendimiento posible de la infraestructura de computación a la hora de desplegar y ejecutar los *workflows*. Sin embargo, precisamente por esto surgen dos grandes problemas:

- El modelado de los *workflows* es altamente dependiente del sistema. Como consecuencia, la compartición de *workflows* y la ejecución del mismo *workflow* en otra infraestructura se vuelve muy complicada.
- El acoplamiento entre el sistema de gestión de *workflows* científicos y la infraestructura de computación es muy alto. De esta forma, la configuración del sistema en la infraestructura es compleja, la migración del sistema de una infraestructura a otra es difícil y la integración de diferentes infraestructuras de computación dentro del mismo sistema no es posible.

En el *Capítulo 3* se realiza una comparación de los sistemas de gestión de *workflows* científicos más destacables de acuerdo a sus características más relevantes.

2.5 – El modelo de coordinación Linda

El modelo de coordinación Linda, propuesto originalmente por Carriero y Gelernter [B4, B5], se basa en el principio de comunicación generativa, es decir, en el consumo de una serie de estructuras pasivas de datos (*tuplas*, similares al concepto de listas en el lenguaje LISP) que son producidas por los participantes involucrados en la coordinación. Linda se compone de un espacio global de tuplas o *pizarra* y un conjunto de procesos que acceden al espacio de tuplas utilizando un conjunto reducido de operaciones atómicas: una operación *out* de escritura de tuplas en el espacio, una operación *in* para realizar la lectura destructiva de tuplas del espacio y una operación *rd* para realizar la lectura no destructiva. Las operaciones de lectura reciben como parámetro un patrón y devuelven una tupla del espacio que coincide con la descripción dada en el patrón, bloqueando al proceso invocante durante su ejecución. El éxito del modelo de coordinación Linda en los entornos de sistemas distribuidos se ha debido, principalmente, a su conjunto reducido de operaciones, a su característica de coordinación basada en los datos, y en la comunicación desacoplada en el tiempo entre procesos que pueden cooperar entre sí sin tener que adaptarse o anunciarse.

En el caso de las operaciones de lectura, se pueden utilizar comodines que sustituyan tuplas enteras o atributos. Al utilizar un patrón sin comodines existirá un emparejamiento si

hay una tupla exactamente igual en el espacio de tuplas. Si, por el contrario, se utiliza un patrón con comodines, se pueden utilizar diferentes políticas de emparejamiento (políticas de *matching*) para determinar si existe concordancia. Las dos políticas más habituales son:

- *Emparejamiento fuerte*. Los comodines del patrón sólo pueden corresponderse en la tupla emparejada con elementos atómicos (números enteros, número reales, cadenas, etc.) no con otras tuplas.
- *Emparejamiento débil*. Los comodines del patrón pueden emparejarse tanto con elementos atómicos como con otras tuplas.

Existen diversas implementaciones, tanto centralizadas como distribuidas, del modelo de coordinación Linda. En este proyecto utilizaremos WS-PTRLinda [B9], una implementación realizada en el marco del grupo GIDHE [W1] que utiliza las redes en redes [B6], la herramienta Renew [B7] y que se basa en la implementación anterior del grupo, RLinda [B2, B10].

2.6 - MyExperiment

Uno de los aspectos fundamentales para los investigadores es la adquisición y compartición de conocimiento. El trabajo de investigación consiste, en muchas ocasiones, en buscar técnicas, herramientas y datos y combinarlos para dar solución a algún problema determinado. El desarrollo de Internet y el crecimiento de la comunidad investigadora han provocado que exista una mayor cantidad de información en la red y que sea muy fácil acceder a la misma. Sin embargo, todavía existen barreras que limitan la compartición, principalmente de herramientas, ejemplos y resultados entre investigadores. De la misma forma, el grado de comunicación y colaboración entre los diferentes investigadores es todavía muy escaso.

Para dar solución a estos problemas nace myExperiment [B8, W8], una red social para investigadores que fomenta la compartición de toda clase de información: publicaciones, *workflows* científicos, datos, resultados, *logs* de ejecución, etc. Actualmente (Agosto de 2011) cuenta con más de 3000 miembros, 200 grupos, 1000 *workflows*, 300 ficheros y 100 *packs* (conjuntos completos de información). El principal objetivo de esta comunidad es eliminar las barreras existentes en el mundo de la investigación.

Uno de los principales avances que introduce es la oportunidad de compartir los experimentos realizados junto con sus datos, resultados y *logs* de ejecución (en los denominados *packs*), permitiendo al resto de investigadores acceder no sólo a su técnica o método si no también al experimento que lo muestra y con el que se han presentado los resultados. Esto fomenta la compartición, la reutilización y la comprobación de que los resultados son correctos. Asimismo, myExperiment da la oportunidad de crear grupos y comunidades y entablar una comunicación fluida con los diferentes investigadores de la comunidad.

MyExperiment ha sentado las bases para lo que muchos consideran una nueva forma de investigación que regirá el mundo de la investigación científica, por lo menos en el ámbitos de las ciencias de la vida, en los próximos años. De hecho, ya han surgido algunos proyectos colaborativos basados en esta idea como, por ejemplo, SALAMI [W11], un proyecto multidisciplinar para el análisis de grandes cantidades de información musical.

Capítulo 3 - Estado del arte

La heterogeneidad de las infraestructuras de computación, la propia complejidad de los *workflows* científicos y la diversidad de áreas de aplicación de los mismos han provocado la existencia de una gran variedad de herramientas para modelar, desplegar y ejecutar *workflows* en entornos *Grid*. En este apartado, vamos a analizar los sistemas más empleadas y comúnmente aceptados dentro de la comunidad científica de acuerdo a las taxonomías presentadas en [B11, B12]. Se abordarán los diferentes enfoques existentes desde el punto de vista del modelado de *workflows* científicos y desde el punto de vista del despliegue y ejecución de dichos *workflows*. Esto corresponde con la visión que tanto los usuarios finales como los desarrolladores tienen de este tipo de infraestructura. Finalmente compararemos dichos sistemas de acuerdo a la clasificación realizada.

3.1 - Herramientas de modelado de workflows científicos

Desde el punto de vista del modelado, los diferentes sistemas pueden clasificarse de acuerdo a las siguientes características:

- Estructura. Un *workflow* científico está formado por una serie de tareas que deben ejecutarse en un cierto orden, de acuerdo a las dependencias existentes entre las mismas, ya sean éstas dependencias de datos o dependencias de control. A nivel conceptual, existen dos representaciones posibles:
    1. Estructura de *grafo acíclico dirigido* (DAG). Estos *workflows* se caracterizan porque sólo permiten operaciones de secuencia, elección y paralelización, no permiten la utilización de bucles.
    2. Estructura de *grafo*. Estos *workflows* se caracterizan por permitir la utilización de cualquier tipo de estructura de control, bucles incluidos.
- Modelo. En lo que hace referencia al modelo o especificación del *workflow*, existen dos posibilidades: un modelo abstracto o un modelo concreto.
    1. En un *modelo abstracto*, el *workflow* se define independientemente de la plataforma de ejecución.
    2. En un *modelo concreto*, las tareas del *workflow* están ligadas a recursos físicos concretos en los que se ejecutarán posteriormente.
- Sistema de composición. Respecto al sistema de composición de los *workflows*, podemos dividir los sistemas en sistemas de composición automática y sistemas dirigidos por el usuario.
    1. Los *sistemas automáticos* generan *workflows* automáticamente a partir de una serie de requisitos indicados por el usuario.
    2. Los *sistemas dirigidos por el usuario* requieren la edición directa del *workflow*. Dentro de este grupo, se pueden identificar dos técnicas de composición:
        1. Técnicas basadas en representación textual, ya sea mediante un lenguaje de marcado tipo XML u otro tipo de lenguaje.

2. Técnicas basadas en la edición gráfica del grafo que modela el *workflow*, ya sea mediante Redes de Petri, diagramas UML o diagramas de componentes propios.

- Restricciones de calidad de servicio (*Quality of Service*, QoS). Otro aspecto interesante reside en si el propio sistema de modelado del *workflow* permite recoger restricciones relacionadas con calidad de servicio. Los sistemas que incorporan esta característica permiten asignar restricciones de calidad de servicios a dos niveles diferentes:

  1. A *nivel de tarea*. Se indican los requisitos de QoS de forma individual para todas las tareas del *workflow*. Los requisitos globales se pueden obtener a través de los requisitos individuales de cada tarea.

  2. A *nivel de workflow*. Se indican los requisitos de QoS para el *workflow* completo.

En la tabla 1 se recoge una lista de las diferentes herramientas de modelado de *workflows* analizadas junto con sus características en lo que respecta a los aspectos anteriores.

Herramienta	Estructura	Modelo	Sistema de Composición	Restricciones de QoS
Askalon [W12]	Grafo	Abstracto	Dirigido: gráfico y textual	Definidas por el usuario o preestablecidas
Condor [W4]	DAG	Abstracto	Dirigido: textual	El usuario define los recursos deseados
Gridbus Workflow [W13]	DAG	Abstracto. Concreto	Dirigido: textual	Fecha límite. Minimización del coste.
Globus [W14]	DAG	Abstracto	Dirigido: textual	Tiempo de ejecución estimado de la aplicación
Kepler [W15]	Grafo	Abstracto. Concreto	Dirigido: gráfico	No Disponible
Pegasus [W16]	DAG	Abstracto	Automático. Dirigido: textual	No Disponible
Taverna [W8]	DAG	Abstracto. Concreto	Dirigido: gráfico y textual	No Disponible
Triana [W17]	Grafo	Abstracto	Dirigido: gráfico	No Disponible
UNICORE [W18]	Grafo	Concreto	Dirigido: gráfico	No Disponible

Tabla 1: Herramientas de gestión de *workflows* científicos en *Grids*. Modelado

En la sección 3 dentro de este mismo capítulo se realiza un análisis de los resultados presentados en la tabla anterior.

3.2 - Herramientas de despliegue y ejecución de workflows científicos

Desde el punto de vista del despliegue y ejecución de *workflows* científicos vamos a centrarnos en tres aspectos fundamentales: el mecanismo de *scheduling* utilizado para desplegar las tareas, el mecanismo de coordinación utilizado entre los diferentes elementos del sistema y los aspectos más importantes relacionados con la gestión de los datos y los fallos producidos.

Scheduling

El *scheduling* es el proceso que determina en qué recurso concreto se va a realizar la ejecución de una tarea concreta. Se trata de un aspecto clave para aspectos como el rendimiento o la satisfacción de los requisitos definidos para el *workflow* científico. Los aspectos fundamentales que caracterizan este proceso son los siguientes:

- Arquitectura del scheduler. La arquitectura del *scheduler* es muy importante de cara a aspectos como el rendimiento y la escalabilidad del sistema. Existen tres posibilidades:
  1. *Arquitectura centralizada*. En este tipo de arquitectura, un único *scheduler* central toma toda las decisiones para el despliegue de todas las tareas del *workflow*.
  2. *Arquitectura jerárquica*. En una arquitectura jerárquica existe un *scheduler* global que controla a una serie de *schedulers* de menos nivel. El *scheduler* global asigna subworkflows a los *schedulers* locales que despliegan las tareas de los mismos.
  3. *Arquitectura descentralizada*. En una arquitectura distribuida, existen varios *schedulers*, todos al mismo nivel, que se comunican entre sí y que despliegan las tareas, seleccionándose en cada momento el *scheduler* con menor carga.
- Tipo de *scheduling*. Dependiendo del momento en el que se realiza el *scheduling* y las características del mismo podemos distinguir entre:
  1. *Scheduling estático*. El *scheduling* se realiza antes de la ejecución del *workflow*. Puede clasificarse a su vez en otros dos tipos.
     1. Dirigido por el usuario. El usuario puede decidir cómo mapear las tareas en los recursos disponibles, de acuerdo a sus propios criterios.
     2. Basado en simulación. El *scheduling* se realiza en base a una simulación previa de las tareas del workflow. Esta simulación puede basarse en información estática o en resultados de rendimiento de ejecuciones previas.
  2. *Scheduling dinámico*. El *scheduling* se realiza en tiempo de ejecución. Hay dos clases:
     1. Basado en predicción. Es similar al *scheduling* estático basado en simulación, se mezcla información dinámica con resultados basados en predicción.
     2. *Scheduling* bajo demanda. La decisión se realiza en el momento de ejecutar la tarea utilizando la información actual del estado del sistema.
- Estrategia de *scheduling*. Desplegar un *workflow* científico en un entorno distribuido es un problema NP-Completo. Las diferentes estrategias empleadas utilizan heurísticas de acuerdo a criterios de QoS como son el presupuesto disponible o el cumplimiento de fechas de entrega. Las diferentes estrategias posibles son:
  1. *Estrategias dirigidas por el rendimiento* que tratan de conseguir un tiempo mínimo de ejecución del *workflow*.
  2. Estrategias *dirigidas por el coste* que minimizan el gasto realizado.
  3. Estrategias *dirigidas por la confianza* que buscan incrementar la fiabilidad al ejecutar las tareas que componen el *workflow*.

Coordinación

Para realizar las diferentes tareas necesarias para coordinar y gestionar la ejecución de un *workflow*, es necesaria la interacción de varios componentes independientes como son el *scheduler* o el gestor de recursos. Además, estos sistemas pueden encontrarse distribuidos por lo que la coordinación y comunicación entre estos elementos se vuelve un aspecto fundamental.

- Mecanismo de coordinación. Existen tres posibles mecanismos de coordinación:
  1. Coordinación *basada en mercado*. En este paradigma de coordinación, los recursos computacionales son vistos como productos de un mercado virtual en el cual se produce la compra y venta de los mismos para permitir la asignación de recursos de una manera eficiente.
  2. Coordinación *basada en grupos*. Se basa en la formación de grupos, denominados Organizaciones Virtuales (VOs), entre usuarios con áreas de investigación similares. Dentro de estros grupos se permite la compartición de recursos e información entre los miembros de los mismos.
  3. Coordinación *basada en un espacio compartido*. En este tipo de coordinación existe un espacio compartido para todos los participantes del sistema y que puede ser accedido concurrentemente por los mismos.

Gestión de datos y fallos

El último aspecto importante reside en cómo se realiza la gestión de los datos y qué mecanismos se utilizan para recuperarse de los fallos que se puedan producir en el sistema.

- Recuperación de información. Los sistemas de gestión de *workflows* no ejecutan las tareas en sí, si no que coordinan la ejecución de las mismas en los recursos disponibles. Para ello es necesario recuperar la información de los recursos de una forma adecuada. Hay tres aspectos a tener en cuenta en cuanto a la recuperación de información:
  1. *Información estática*. La información estática se refiere a la información que no varía con el tiempo. Este tipo de información incluye aspectos relacionados con la infraestructura (número de procesadores, memoria disponible, ...), configuración (Sistema Operativo, librerías, ...), etc.
  2. *Información dinámica*. El sistema necesita almacenar información dinámica como la accesibilidad de los recursos, la carga de los mismos o el rendimiento de la red.
  3. *Información histórica*. Se trata de información que proviene de ejecuciones previas de los *workflows*. Algunos datos, como informes del rendimiento, de los errores producidos, de los resultados de la ejecución, se pueden almacenar y utilizar para predecir el comportamiento futuro de los experimentos.
- Movimiento de datos. Antes de ejecutar las diferentes tareas, es necesario situar los datos de entrada y los ejecutables en los recursos utilizados para ejecutar las mismas. Para gestionar el movimiento de los datos, existen tres alternativas:

1. Aproximación centralizada. En esta aproximación, se transfieren los datos intermedios entre los recursos utilizando un gestor central.
2. Aproximación mediada. En esta aproximación, en lugar de utilizar un gestor central para la transferencia, se utiliza un sistema de gestión distribuido.
3. Aproximación p2p. En esta aproximación, se transfieren los datos intermedios entre los diferentes recursos directamente.

- Tolerancia a fallos. En entornos *Grid*, pueden producirse fallos por múltiples circunstancias como errores en la red, sobrecarga en los recursos, fallos en los nodos, etc. Por tanto, los sistemas de gestión de *workflows* deben ser capaces de gestionar los fallos que sucedan y recuperarse de los mismos. Las técnicas de gestión de fallos pueden clasificarse en:
  1. Técnicas a *nivel de tarea*. Estas técnicas han sido ampliamente estudiadas en sistemas paralelos y distribuidos y se clasifican en:
     - Reintento. Si una tarea falla se vuelve a ejecutar en el mismo recurso en el que se estaba ejecutando.
     - Uso de recursos alternativos. Si una tarea fallo se vuelve a ejecutar en un recurso diferente del que se estaba ejecutando.
     - *Checkpointing* y reinicio. Se guardan checkpoints, imágenes del estado de la tarea en un instante temporal, y en caso de fallo se reinicia la tarea desde el último punto de guardado.
     - Replicación. Esta técnica consiste en ejecutar la misma tarea en recursos diferentes para aumentar la posibilidad de que al menos una de las réplicas finalice correctamente.
  2. Técnicas a *nivel del workflow*. Las técnicas a nivel de *workflow* pueden clasificarse en:
     - Tareas alternativas. Si una tarea falla, se ejecuta una implementación diferente de la misma tarea.
     - Redundancia. Se ejecutan múltiples tareas alternativas al mismo tiempo.
     - Gestión de errores definida por el usuario. Permite al usuario definir rutinas de error para el caso de que una tarea falle.
     - Rescate del workflow. Se genera un *workflow* con información sobre las tareas que fallaron en la primera ejecución del mismo. Mediante este workflow de rescate, se puede reiniciar la ejecución justo desde el punto en el que falló la ejecución del mismo.

En la tabla 2 se recoge una lista de las diferentes herramientas de modelado de workflows que se han analizado junto con sus características en lo que respecta a los aspectos anteriores.

Finalmente, en la siguiente sección se realiza un análisis de los resultados presentados tanto en este apartado como en el apartado anterior

Herramienta	*Scheduling*			Coordinación	Movimiento de datos	Gestión de fallos
	Arquitectura	Tipo	Estrategia			
Askalon [W12]	Descentralizada	Dinámico: Basada en predicción y bajo demanda	Dirigida por el rendimiento. Dirigida por el coste	Basada en mercado. Basada en grupos	Centralizado. Dirigido por el usuario	Nivel tarea: Recursos alternativos, reintento. Nivel *workflow*: *workflow* de rescate
Condor [W4]	Centralizada	Dinámico: bajo demanda	Dirigida por el rendimiento	Basada en grupos	Dirigido por el usuario	Nivel tarea: Recursos alternativos, reintento. Nivel *workflow*: *workflow* de rescate
Gridbus Workflow [W13]	Descentralizada	Estática: Dirigido por el usuario. Dinámica: bajo demanda	Dirigida por el coste	Espacio compartido	Centralizado. P2P	Nivel tarea: Recursos alternativos
Globus WMS [W14]	Centralizada	Dinámico: bajo demanda	Dirigida por el rendimiento	Basada en grupos. Basada en mercado.	Dirigido por el usuario	Nivel tarea: Recursos alternativos. Reintento.
Kepler [W15]	Centralizada	Definido por el usuario	Definida por el usuario	Basada en grupos.	Centralizado. Mediado. P2P	Nivel tarea: Recursos alternativos. Nivel *workflow*: Definida por el usuario, *workflow* de rescate.
Pegasus [W16]	Centralizada	Estático: Dirigido por el usuario. Dinámico: bajo demanda	Dirigida por el rendimiento	Basada en grupos	Mediado	Basado en Condor DAGMan
Taverna [W8]	Centralizada	Dinámico: bajo demanda	Dirigida por el rendimiento	Basada en grupos	Centralizado	Nivel tarea: Recursos alternativos, reintento.
Triana [W17]	Descentralizada	Dinámico: bajo demanda	Dirigida por el rendimiento	Basada en grupos	P2P	Basado en GAT Manager
UNICORE [W18]	Centralizada	Estático: Dirigido por el usuario.	Definida por el usuario	Basada en grupos	Mediado	Definido por el usuario

Tabla 2: Herramientas de gestión de *workflows* científicos en *Grids*. Despliegue y ejecución.

3.3 - Comparación de los diferentes sistemas

En lo que respecta al modelado de *workflows* científicos, las alternativas existentes son bastante similares. Las diferencias existentes entre las mismas vienen provocadas por el aumento de complejidad introducido al permitir utilizar un lenguaje de modelado de *workflows* más potente. La utilización de un grafo completo, en lugar de un grafo acíclico dirigido, para representar la estructura del *workflow*, convierte el problema de *scheduling* en un problema NP-Completo, lo que dificulta el despliegue de las tareas de forma eficiente. Asimismo, introduce la necesidad de disponer de expresiones de control que indiquen el final del *workflow*, en el caso de que existan ciclos en el mismo.

Los modelos utilizados son en su mayoría abstracto, para independizar al modelo de los detalles de ejecución, debido a la naturaleza heterogénea y dinámica del entorno de ejecución. Para componer el modelo del *workflow*, se utilizan, en general, técnicas textuales en las que el usuario específica mediante algún tipo de lenguaje de marcado basado en XML como, por ejemplo, JDL [W19] o JSDL [W20]. La utilización de estas técnicas viene dada por su sencillez, sin embargo, la utilización de modelos gráficos facilita enormemente el modelado del *workflow* por lo que los sistemas actuales tienden a utilizar este sistema de cara a facilitar la labor del usuario.

El aspecto en el que existe una mayor variedad es en lo referente a la posibilidad de indicar aspectos de calidad de servicio dentro del modelo. La mayoría de sistemas no permiten esta posibilidad o lo hacen de una manera limitada como ocurre en el caso de Condor. Sin embargo, esto no significa que no se utilicen requisitos de QoS, si no que éstos están incluidos dentro del propio sistema, normalmente, en términos de eficiencia, productividad y utilización del sistema. En cualquier caso, la inclusión de requisitos de QoS se antoja un aspecto muy importante en la actualidad, sobre todo con la aparición del paradigma de computación *Cloud*, y los sistemas más modernos, como *GridBus*, ya incorporan esta característica.

El *scheduler* es uno de los componentes fundamentales de los sistemas de gestión de *workflows* científicos. Su importancia se debe tanto a la propia decisión sobre el recurso en el que ejecutar una determinada tarea, como al impacto que tiene el mismo en aspectos claves del sistema, como el rendimiento y la escalabilidad. En lo que se refiere a su arquitectura, las últimas propuestas promueven la utilización de *schedulers* distribuidos para obtener una mayor eficiencia y dotar al sistema de mayor escalabilidad, sin embargo, la mayoría de sistemas utilizan un *scheduler* centralizado por cuestiones de simplicidad. En cuanto a la estrategia utilizada, los sistemas se reparten entre una alternativa orientada a maximizar el rendimiento de las tareas ejecutadas y una alternativa orientada a minimizar el coste de ejecución de los mismos, lo que se traduce en buscar una utilización óptima de los recursos del *Grid*.

Otro aspecto clave reside en el tipo de *scheduling* realizado, el cual puede ser estático o dinámico. Los métodos estáticos se benefician de conocer la estructura del *workflow* pudiendo optimizar el despliegue de las tareas en base a esa información. Sin embargo, debido a la naturaleza cambiante y a la propia complejidad de los *workflows* (los datos podrían conocerse sólo en tiempo de ejecución), se recomienda la utilización de métodos dinámicos. Este tipo de *scheduling* es más complejo e impone algunas dificultades [B13]. Por una parte, los sistemas de gestión de *workflows* ofrecen un componente conocido como *broker* de recursos que tiene un

conocimiento completo del estado de los recursos y la capacidad de tomar decisiones de acuerdo a ese conocimiento. La naturaleza distribuida y heterogénea de los *Grids* complica enormemente el proceso de crear y actualizar esta base de conocimiento. Como consecuencia, los brokers se integran, o al menos están fuertemente ligados, con el *middleware* del *Grid* y, por tanto, no es trivial migrar un *broker* de un *middleware* a otro [B3]. Por otra parte, el *scheduling* dinámico se aprovecha del conocimiento actualizado del entorno de ejecución pero no es consciente de la estructura completa del *workflow* y sus implicaciones de cara a conseguir una distribución óptima de las tareas. Independientemente del tipo de *scheduling*, se pueden utilizar técnicas de simulación para evaluar y determinar cuál es la estrategia óptima de despliegue para cada problema [B14, B15]. En cualquier caso, existen propuestas intermedias que buscan aprovechar las ventajas de ambos tipos de scheduling [B15, B16].

El sistema de coordinación de los diferentes sistemas de *workflows* ha sido tradicionalmente basado en grupos para facilitar y fomentar la compartición de información entre los diferentes miembros de una misma VO. Sin embargo, existen nuevas apuestas que optan por utilizar un espacio compartido. Por ejemplo, el sistema Gridbus utiliza un modelo de coordinación basado en Linda para comunicar eventos entre los diferentes componentes del sistema [B13]. Las ventajas fundamentales de esta alternativa residen en la flexibilidad del mecanismo y en que permite desacoplar los elementos del sistema haciéndolos menos dependientes del entorno concreto de ejecución.

En lo referente al movimiento de datos no hay un consenso en lo que respecta a la mejor alternativa para realizar el mismo. En general, los sistemas tienden a utilizar modelos dirigidos por el usuario por la dificultad de conocer la ubicación exacta de los datos. Los grandes avances en este aspecto han surgido por la introducción de protocolos de transferencia específicos para infraestructuras *Grid*, como GridFTP [B17].

Debido a la naturaleza de los *Grids*, la posibilidad de que la ejecución de una tarea falle, ya sea por un fallo en el nodo, un fallo de la red u otra circunstancia, es un aspecto a tener en cuenta. Existen una serie de técnicas sencillas como volver a ejecutar la tarea, ya sea en ese mismo nodo o en otro diferente, que son adoptadas en la mayoría de los casos ya que permiten la recuperación del fallo de forma satisfactoria en muchas ocasiones. Sin embargo, otras veces la naturaleza del fallo requiere de mecanismos más complejos como generar un *workflow* de recuperación que permita volver a ejecutar el mismo desde el punto en el que falló. Finalmente, otro mecanismo posible consiste en utilizar algún tipo de *scheduling* basado en seguridad que ante una ejecución problemática vuelva a ejecutar la tarea en un recurso de computación fiable.

Capítulo 4 - Modelado e implementación

En este capítulo se analizarán los aspectos más relevantes de las fases de análisis, diseño e implementación del sistema. Para ello se explicará la arquitectura utilizada en el sistema y se detallaran cada una de las capas que componen la misma, así como los diferentes componentes desarrollados.

4.1 - Arquitectura del sistema

En la figura 1 se muestra la arquitectura del sistema. Como puede observarse, se compone de tres capas diferentes: la capa de modelado, la capa de ejecución y la capa de infraestructuras de computación.

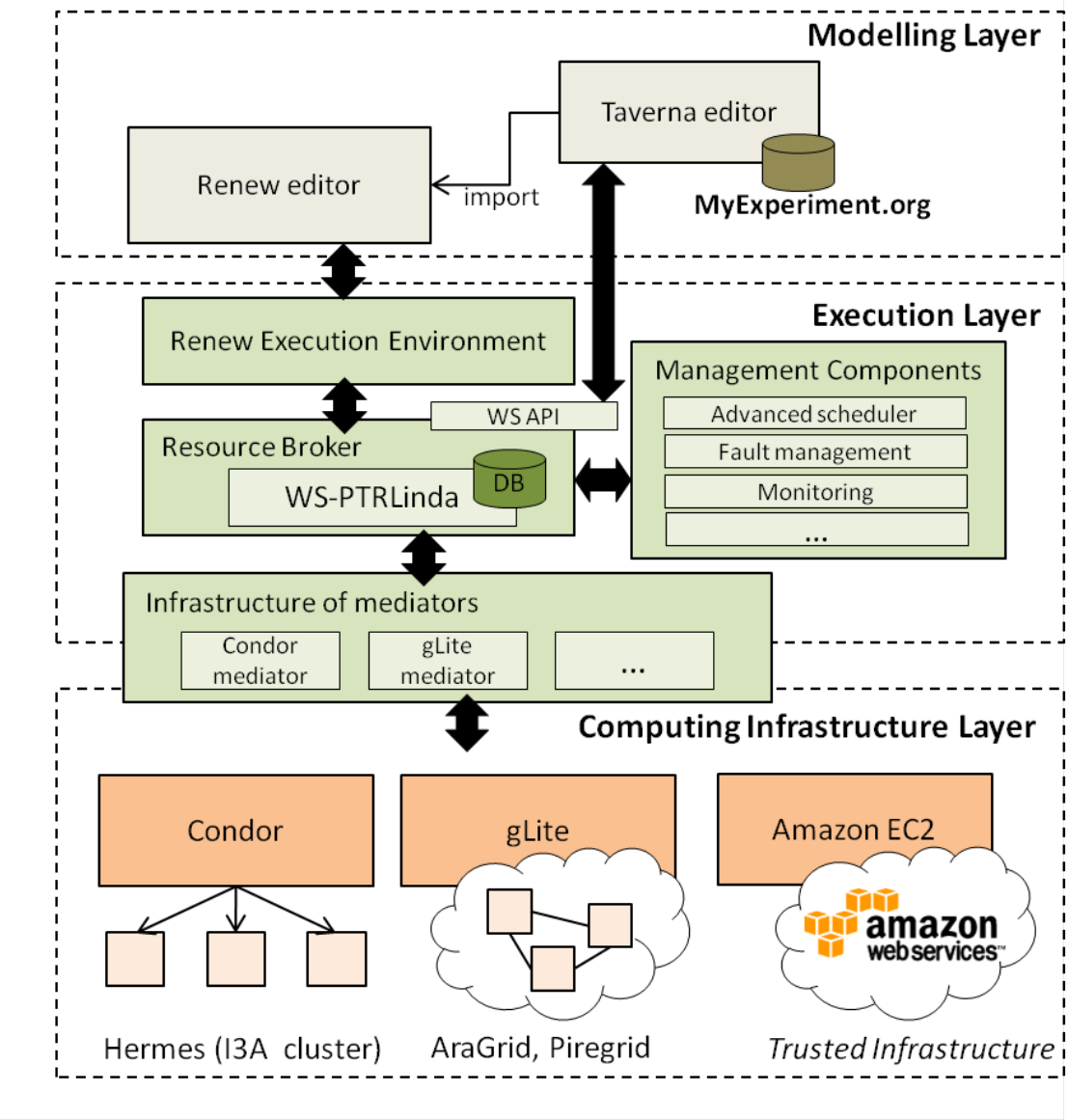

Figura 1: Arquitectura del sistema.

Capa de modelado

En primer lugar, la capa de modelado ofrece una serie de herramientas para programar *workflows* científicos, ya sea a través de Taverna [W8] u otro sistema de gestión de *workflows* existente o bien a través de Renew [B7]. Este aspecto es muy importante porque en el ámbito de las ciencias de la vida, los usuarios de este tipo de sistemas suelen tener dificultades a la hora de aprender nuevos lenguajes o técnicas de modelado. Por tanto, al permitir al usuario utilizar su lenguaje de modelado habitual, dotamos al sistema de una mayor usabilidad, ya que no se requiere que el usuario aprenda un nuevo lenguaje de modelado, si no que puede utilizar el lenguaje con el que trabaja habitualmente.

La primera de las alternativas que ofrecemos para el modelado de *workflows* científicos es la utilización de algún sistema de gestión de *workflows* ya existente, como por ejemplo Taverna. Este aspecto es realmente novedoso y no se presenta en los sistemas de gestión existentes. En nuestro caso, surge como algo natural gracias a que la capa de modelado y la capa de ejecución se encuentran desacopladas, por lo que cualquier sistema de modelado puede utilizar la infraestructura de ejecución desarrollada. Para ello, el *broker* de recursos ofrece una interfaz de servicios Web que permite la incorporación del sistema a otro entorno ya existente.

La segunda alternativa propuesta es la utilización de las *reference nets* o redes de referencia, una subclase de las redes de Petri de alto nivel, para modelar *workflows* científicos desde la perspectiva del paradigma de las redes-en-redes [B6]. La utilización de las redes de Petri como lenguaje de modelado de *workflows* ya ha sido estudiada y sus beneficios se muestran en [B18]. Asimismo, algunas ventajas adicionales son: la posibilidad de modelar *workflows* de forma independiente a la infraestructura de computación y la posibilidad de modelar cualquier tipo de *workflow*, tanto orientado a datos como orientado al control, gracias a la potencia del formalismo. Para esta alternativa, se propone la utilización de Renew como herramienta de modelado. Algunas de las principales ventajas que nos proporciona Renew con respecto a otras herramientas de modelado es:

- Independencia de la plataforma. Renew es una herramienta desarrollada en Java por lo que nos proporciona independencia de la plataforma en la que vamos a trabajar. Este aspecto es fundamental ya que necesitamos una herramienta que pueda ser utilizada en la heterogeneidad de sistemas utilizados por los usuarios.
- Interfaz gráfico y manejo intuitivo. A diferencia de otras herramientas de modelado de *workflows* científicos, Renew nos proporciona un interfaz gráfico para modelar los mismos. Además, se trata de un interfaz muy intuitivo y fácil de utilizar, lo que hace de Renew una alternativa estupenda para el ámbito en el que nos encontramos, en el que los usuarios son poco experimentados y necesitan de mecanismos sencillos que les permitan trabajar cómoda y rápidamente.
- Incorporación de un motor de ejecución. Renew incorpora un motor que nos permite realizar la ejecución efectiva y directa del sistema modelado. De esta forma, además de modelar el sistema, podemos comprobar que hemos realizado el modelado correctamente de forma empírica. Otra opción muy interesante es la simulación temporizada, en lugar de ejecutar el modelo.

- Posibilidad de ejecutar código Java. Otra de las características clave que hacen de Renew una herramienta perfecta para el sistema desarrollado, es la posibilidad de ejecutar código Java embebido dentro del sistema modelado (Renew soporta un amplio subconjunto de instrucciones de Java). Esta característica nos va a permitir interactuar de una forma sencilla y natural con el resto de componentes del sistema sin necesidad de utilizar herramientas intermedias o de modificar la herramienta utilizada.

Finalmente, otra alternativa sería traducir el modelo desarrollado, por ejemplo, con Taverna, a *reference nets*. Este aspecto no introduce una gran complejidad pudiendo realizarse esa traducción de forma manual. Además, como posible trabajo futuro se plantea el desarrollo de un *plugin* que realice esta traducción de forma automática.

Capa de ejecución

La capa de ejecución es la encargada de permitir la ejecución de los diferentes trabajos definidos en el modelo del *workflow científico* en los recursos de computación.

En un primer paso, el entorno de ejecución de Renew, o del sistema de gestión utilizado para modelar los experimentos, permite ejecutar de forma efectiva el modelo desarrollado. En esta primera versión del *framework*, el entorno de ejecución de Renew utiliza directamente las interfaces de programación del *broker* de mensajes, en lugar de utilizar la interfaz de servicio Web del mismo por cuestiones de simplicidad y eficiencia (en esta versión todos los componentes se encuentran en la misma máquina). En cualquier caso, esta segunda posibilidad será desarrollada en el futuro y puede ser utilizada actualmente si todos los componentes están en la misma máquina.

La coordinación y comunicación entre los diferentes componentes de un sistema de gestión de *workflows* científicos se ha solventado tradicionalmente utilizando componentes que se encuentran muy acoplados unos con otros. Además, este acoplamiento afecta tanto a los componentes de nivel de ejecución del sistema como a los componentes encargados del modelado de los problemas. De esta forma, la integración de nuevos componentes tiene un impacto muy grande en todo el sistema, dificultando enormemente esta tarea.

El uso de un *broker* de mensajes basado en el modelo de coordinación Linda aporta un nuevo enfoque en este aspecto. Linda promueve la coordinación entre los diferentes componentes a través de tuplas que son depositadas y extraídas de un espacio compartido. Además, cada componente puede estar distribuido, es decir, puede encontrarse en una máquina diferente y utilizar Linda para comunicarse con el resto de componentes del sistema. Esta característica permite desacoplar los diferentes elementos que forman el sistema haciendo que el mismo sea más flexible y adaptable.

En nuestro caso hemos utilizado como *broker* de mensajes WS-PTRLinda [B2, B9], una implementación del modelo de coordinación Linda del grupo de investigación GIDHE [W1]. Las principales mejoras que aporta esta implementación son la adición al modelo de Linda de una capa de temporización, permite que las tuplas sólo sean válidas durante un período de tiempo determinado, y una capa de persistencia, permite almacenar las tuplas en una base de datos. Asimismo, WS-PTRLinda incluye una capa que permite interaccionar con el sistema como un servicio web SOAP o REST, haciendo que el mismo sea accesible desde cualquier punto.

La utilización de Linda nos permite conectar al sistema un número variable de componentes que aporten diferentes funcionalidades. Posibles componentes serían un componente que gestione los fallos que se produzcan al ejecutar los trabajos, un componente de monitorización que permita obtener información detallada de los trabajos que se están ejecutando o un componente de *scheduling* avanzado que permita realizar el despliegue de los trabajos en base a diferentes parámetros y configuraciones.

En esta primera versión se ha desarrollado el componente de gestión de fallos para ilustrar cómo se pueden integrar diferentes componentes en el sistema. Se eligió implementar este componente por su importancia dentro del ámbito de la computación *Grid*. La aparición de fallos es un elemento habitual debido a la naturaleza de las infraestructuras de computación y a la complejidad de los *workflows* científicos. Estos fallos pueden deberse tanto a errores en el modelado del *workflow* científico, fallos propios de la aplicación que se está ejecutando o fallos provocados por la propia infraestructura.

Los errores que pueden aparecer son de naturaleza muy diversa, produciéndose por una mala definición del trabajo que se quiere ejecutar, un error propio del trabajo que se está ejecutando o un fallo de la infraestructura. Por ejemplo, si un nodo del *grid* falla, todos los trabajos que se estén ejecutando en el nodo fallarán. Por tanto, la gestión de estos fallos es un aspecto crítico que no puede ser obviado y que hay que gestionar de una manera adecuada.

Capa de infraestructuras de computación

La capa de más bajo nivel es la capa de infraestructuras de computación. Esta capa representa los diferentes *grids* en los que se van a ejecutar los trabajos. Como ya se ha comentado a lo largo de esta memoria, las infraestructuras de computación utilizadas en él ámbito científico son altamente heterogéneas. Por tanto, para gestionar de una manera automática y transparente para el usuario la ejecución de las tareas en los recursos disponibles, se utiliza un conjunto de mediadores que interactúan con el *middleware* del *grid.* Estos mediadores se encargan de la ejecución de los trabajos junto con un componente que mueve los datos necesarios entre los diferentes *grids* e incluso entre los *grids* y servidores externos.

La utilización de un conjunto de mediadores permite que el sistema pueda integrar cualquier tipo de g*rid*, independientemente del *middleware* que utilice. Para ello, sólo es necesario implementar un mediador que gestione y controle la ejecución de los trabajos en ese entorno de ejecución concreto, gracias a que la descripción de los trabajos y la coordinación de los componentes es independiente de la infraestructura de computación. En nuestro caso, hemos desarrollado un mediador para el *middleware* Condor y otro mediador para el *middleware* gLite, ya que son los *middlewares* de computación que gestionan las infraestructuras *grid* a las que tiene acceso el grupo de investigación GIDHE. La implementación de un nuevo mediador, que gestione la ejecución en otro sistema, puede realizarse de forma similar a la utilizada para implementar los mediadores ya desarrollados, incluyendo las particularidades de ejecución que presente ese sistema.

Por otro lado, uno de los aspectos más importantes de los que se encarga un *middleware* es de realizar el movimiento de los datos entre los diferentes recursos del *grid*. La necesidad de este movimiento viene determinada por la abstracción de que todo el *grid* es un

único supercomputador. Para lograr la misma, el movimiento de los datos necesarios para ejecutar un trabajo debe ser completamente automático y transparente para el usuario. El auge de las infraestructuras de computación *Grid* ha propiciado la aparición de protocolos de transferencia de ficheros especializados. Este es el caso de GridFTP [B17] que rápidamente se ha establecido como estándar para el movimiento de datos dentro del *grid*.

En nuestro caso, el problema cambia significativamente. El sistema desarrollado no pretende sustituir al *middleware* utilizado en el *grid*, si no englobar diferentes *middlewares* para integrar varios *grids*. Esto nos permite aprovechar las características de los *middlewares* como es el caso de la gestión de los datos dentro de los recursos del *grid*. Por tanto, nuestro problema se refiere al movimiento de datos entre diferentes *grids* de forma automática.

4.2 - Modelado de workflows científicos

El objetivo fundamental del sistema en lo que se refiere al modelado de *workflows* científicos aborda dos cuestiones fundamentales. En primer lugar, se pretende conseguir que el modelado sea independiente del entorno de ejecución utilizado. En segunda lugar, se pretende que el sistema sea capaz de utilizar modelos realizados con diferentes lenguajes de modelado.

Para conseguir el primero de los objetivos hay aspectos clave como permitir identificar los trabajos de forma estándar, mientras que para el segundo objetivo es fundamental el modelo arquitectural del sistema que permite que los componentes del mismo se encuentren desacoplados.

Interoperabilidad con diferentes herramientas de modelado

Hay una gran variedad de sistemas de *workflows* científicos más o menos asentados en la comunidad científica. Además, los usuarios de los mismos suelen tener dificultades a la hora de aprender el funcionamiento de un nuevo sistema y se muestran reacios a aprender nuevos lenguajes de modelado. Por ello, uno de nuestros objetivos fundamentales ha sido permitir ejecutar *workflows* científicos modelados con diferentes técnicas y herramientas. En este aspecto, el uso de Linda como sistema de coordinación nos abstrae del sistema y lenguaje de modelado utilizado ya que la introducción en el sistema de las tuplas que describen los trabajos es independiente del contexto en el que se generan las mismas.

El *broker* WS-PTRLinda, a través de su interfaz de servicios web, nos facilita las operaciones de escritura, lectura destructiva y lectura no destructiva del modelo de coordinación Linda. Sin embargo, la utilización directa de esta interfaz no es conveniente por dos razones fundamentales. En primer lugar, la utilización directa de esta interfaz implica que el usuario tiene que ser consciente de aspectos internos del sistema, como las palabras claves usadas para identificar las tuplas en los componentes (estos aspectos se abordan en la sección siguiente). En segundo lugar, es necesario comprobar que las tuplas utilizadas en las operaciones están bien formadas y no contienen errores.

Por tanto, se ha desarrollado una clase que ofrece tres operaciones para facilitar la labor del usuario. Se trata de la clase `Broker` que proporciona los siguientes métodos estáticos para facilitar la utilización de la infraestructura desde cualquier sistema de modelado:

- `out(tupla)`. Esta operación permite desplegar un trabajo con el formato especificado introduciendo la tupla indicada en el *broker*. En caso de que la tupla no esté bien formada, se generará una tupla de error que será tratada por el componente de gestión de fallos. Se trata de una operación no bloqueante de forma que es necesario utilizar una operación `in` posteriormente para recuperar el fin del trabajo.
- `in(patrón)`. Esta operación permite recuperar el fin de un trabajo. Además, realiza tareas adicionales, como almacenar el *log* del trabajo, de forma completamente transparente al usuario.
- `outIn(tupla)`. Esta operación permite desplegar un trabajo al igual que la operación `out` con la diferencia de que en este caso se trata de una operación bloqueante. Esta operación engloba una primera operación `out` y una segunda operación `in`.

En las anteriores operaciones se utiliza la operación de lectura destructiva para recuperar las tuplas del *broker*. La razón de la utilización de lectura destructiva se debe a que no tiene sentido que la tupla que describe un determinado trabajo continúe en el *broker* de mensajes una vez que la misma ha sido tratada por el mediador correspondiente.

Definición de los trabajos

El *broker* Linda utiliza la tupla como elemento de comunicación. Por tanto, es indispensable que las tuplas utilizadas para describir los trabajos a ejecutar utilicen un lenguaje de descripción totalmente independiente de la infraestructura de ejecución. Para la elaboración de dicho formato se ha seguido el formato JSDL [W20], un estándar para la especificación de tareas, particularmente en entornos *grid* promovido por el *Open Grid Forum* [W22].

De esta forma, el formato utilizado para la descripción de las tuplas aborda cuatro aspectos fundamentales:

1. Descripción del trabajo. La primera cuestión se refiere a la descripción tanto del trabajo como de los ficheros involucrados en la ejecución del mismo. En este aspecto se deben definir cuatro elementos:
   - Nombre de la aplicación utilizado para identificar la misma.
   - Argumentos utilizados para la ejecución de la aplicación (pueden ser nulos).
   - Lista de ficheros de entrada separados por un espacio en blanco (puede ser nula).
   - Lista de ficheros de salida separados por un espacio en blanco (puede ser nula).
2. Descripción de la entrada y salida de la aplicación. En las tareas ejecutadas en infraestructuras *Grid* no se permite la interacción con el usuario a través de la pantalla y el teclado. Para permitir la interacción se utilizan ficheros. Para ello debe indicarse:
   - Fichero utilizado como entrada estándar (puede ser nulo).
   - Fichero utilizado como salida estándar (puede ser nulo).
   - Fichero utilizado como salida de error (puede ser nulo).

3. Requisitos de calidad de servicio. Para definir los requisitos de calidad de servicio se contempla la utilización de dos campos:
   - Fecha límite en la que debe finalizar la tarea.
   - Presupuesto máximo que puede ser utilizado para ejecutar la tarea.
4. Entorno de ejecución. En este último campo se incluye la información relativa al entorno utilizado para la ejecución de la aplicación. Los aspectos a indicar son:
   - Usuario que desea ejecutar el trabajo.
   - Infraestructura de computación utilizada para ejecutar el trabajo. Si no se especifica ninguna infraestructura (a través de la cadena vacía o la cadena "null") el sistema elegirá una de las posibles de acuerdo a las características de la aplicación.

En la figura 2, puede observarse el formato de una tupla utilizada para describir un trabajo según las especificaciones anteriores:

```
[ [aplicación, argumentos, lista ficheros entrada, lista ficheros salida],
  [entrada estándar, salida estándar, salida de error],
  [fecha límite, presupuesto],
  [usuario, Grid] ]
```

Figura 2: Formato de tupla utilizado para describir un trabajo.

Para que la representación sea independiente de la infraestructura es necesario indicar la localización exacta de los ficheros involucrados en la ejecución del trabajo. Para ello se utiliza la URI que identifica el recurso. Este aspecto se detalla en la sección 4.7, de este mismo capítulo.

Identificación de los trabajos

Como se ha comentado anteriormente, hay dos formas de realizar el despliegue de un trabajo. La primera de ellas es asíncrona, realizando una operación `out` seguida de una operación `in`. La segunda forma es síncrona, utilizando la operación `outIn`. Sin embargo, existe un problema a la hora de realizar este despliegue, la semántica de Linda no establece mecanismos para sincronizar las operaciones `in` y `out` que corresponden a un mismo trabajo.

Para realizar esta identificación se valoraron tres alternativas:

- Utilizar un identificar numérico. La utilización de un identificador numérico que permita correlar las operaciones de escritura y lectura es una solución sencilla al problema. Sin embargo, esta alternativa presenta el problema de que es necesario asignar dicho identificador desde el propio sistema de modelado, delegando esta responsabilidad al usuario. Esto hace que no sea una opción viable ya que nuestro objetivo es que la identificación sea transparente al usuario. Además, esto podría limitar la interoperabilidad del sistema con otros lenguajes de modelado si los mismos no permiten incluir esta característica de una forma sencilla.
- La segunda alternativa consistía en utilizar la misma tupla tanto para la operación de lectura como para la operación de escritura. Esta alternativa permite identificar claramente cuáles son las operaciones relacionadas. Sin embargo, se descartó la misma porque se utiliza mucha información para identificar la tupla.

- La tercena alternativa se basa en la opción anterior. En esta alternativa se busca utilizar un identificador de longitud mínima en la tupla de lectura para identificar el trabajo al que hace referencia. Dicho identificador está formado por el usuario que despliega el trabajo y los ficheros de salida generados en el mismo (tanto la lista de ficheros de salida como la salida estándar y la salida de error).

Finalmente, se eligió la tercera alternativa porque no implica un trabajo extra para el usuario y es más simple que la segunda opción ya que necesita indicar menos información, favoreciendo también que el modelo sea más claro y legible.

Hay que mencionar que tanto la opción elegida como la segunda opción presentan el problema de que no aseguran la unicidad, ya que el usuario podría desplegar varios trabajos iguales o que utilizarán los mismos ficheros de salida. Sin embargo, este problema no es crítico porque no tiene sentido que el mismo usuario ejecute a la vez trabajos que utilicen los mismos ficheros de salida ya que la información de los diferentes trabajos se sobreescribiría perdiendo los resultados.

El identificador definido se utiliza en la operación de lectura `in` que detecta el final de un trabajo. Además de dicho identificador debe añadirse un comodín (carácter ?) que recupere el estado de finalización del trabajo. Este campo se utiliza para detectar si el trabajo ha finalizado correctamente o si ha finalizado con algún tipo de error de cara a permitir y facilitar el tratamiento y la detección de errores en el modelo. En la figura 3, se muestra el formato que debe ser utilizado para una operación de lectura.

```
[ usuario, lista de ficheros de salida, salida estándar, salida de error, ? ]
```

Figura 3: Formato utilizado para una tupla de lectura.

4.3 - Registros del sistema

Existen diferentes maneras de gestionar la información relativa al funcionamiento de un sistema de gestión de *workflows*. Algunos sistemas optan por almacenar determinada información y acceder a la misma cuando la necesitan, otros optan por no almacenar la información y realizar consultas al *middleware* para obtener la misma cuando la necesitan. Independientemente de la alternativa utilizada, existe información que es necesario almacenar ya que no puede ser obtenida bajo demanda, como los usuarios que tienen acceso a la infraestructura, por lo cual la utilización de registros de información se hace obligatoria.

El *framework* propuesto debe manejar información referente a los *grids* disponibles, tanto fiables como normales, usuarios que pueden ejecutar en los *grids* y aplicaciones que puede ejecutar cada *grid*. Por tanto, se hace necesaria la utilización de diferentes almacenes para los diferentes tipos de información tratada.

Las características que deben tener estos almacenes de información vienen dictadas por los objetivos del proyecto. En primer lugar, se necesita que el sistema sea flexible y adaptable frente a cambios en el entorno. Por tanto, debe ser posible modificar la información contenida en estos registros en tiempo de ejecución sin que afecte a la utilización del sistema. Asimismo, el sistema debe ser fácil de configurar, por lo que un usuario cualquiera debe ser capaz de configurar el mismo rellenando correctamente los registros del sistema. Para facilitar las

cuestiones de configuración se proporcionan *scripts* que permiten configurar los diferentes almacenes de una forma sencilla. Estos *scripts* permiten insertar nueva información, modificar la existente y eliminar la información actual. Su utilización puede consultarse en el *Anexo III - Manual de Usuario*.

En lo que respecta a almacenar la información de usuarios, aplicaciones y *grids*, queda claro que ésta debe almacenarse de forma separada ya que se trata de información muy diferente. Sin embargo, separar la información relativa a los *grids* fiables y la información relativa a los *grids* normales no estaba tan claro. Finalmente, se decidió separar la información ya que se considera que es información de distinto tipo. Además, separar la misma va a permitir realizar una gestión de la información más sencilla, tener un acceso más eficiente a la misma y personalizar cada tipo de información por separado de acuerdo a sus necesidades.

Para almacenar la información se decidió utilizar ficheros XML por cuestiones de simplicidad y facilidad en cuanto a manejo de la información y compartición de la misma. Una característica que se introduce en los ficheros para facilitar la comprensión de los mismos por parte del usuario, es que no se permiten elementos optativos, todos los elementos son obligatorios, es decir, no puede haber elementos que no aparezcan en el fichero de configuración. Sin embargo, sí que se permite que haya determinados atributos con un valor indefinido. Cuando se quiere expresar esta característica se utiliza el carácter '*' o la cadena 'null' para representar que el valor del atributo es indefinido.

A pesar de que cada tipo de registro trata diferente tipo de información, en todos los registros se incluye un campo para almacenar opciones personalizadas. Este campo se utiliza para indicar opciones personalizadas para la ejecución de una determinada aplicación, todas las aplicaciones de un usuario en un *grid* o todos los trabajos realizados al *grid*, dependiendo del registro que se trate. Esta característica proporciona al usuario una mayor flexibilidad, pudiendo adaptar la ejecución de cada trabajo de una forma más precisa si lo desea. Estas opciones deben proporcionarse en un formato que sea entendible por el *middleware* del *grid* ya que no se realiza ningún tratamiento de las mismas, si no que se añaden directamente.

Como existe la posibilidad de que estas opciones personalizadas se solapen, se impone el siguiente mecanismo de prioridad: las opciones menos prioritarias son las indicadas en el registro del usuario, después las indicadas en el registro del *grid* y, finalmente, las más prioritarias son las específicas de la aplicación. Por tanto, si en algún momento dichas opciones se contradicen se tomará la alternativa fijada por el registro con mayor prioridad. En cualquier caso, la configuración propia del trabajo indicada por el mediador tiene mayor prioridad que las opciones personalizadas de los registros para impedir comportamientos erróneos o malévolos.

En lo que se refiere al diseño de los registros, se utiliza un componente diferente para gestionar cada uno de ellos si bien todos son análogos. La arquitectura de estos componentes se presenta en la figura 4. En lo que respecta al nombre de cada uno de los subcomponentes, la palabra *Element* se sustituye por la palabra adecuada dependiendo del componente del que se trate. El componente *ElementDiscovery* es el encargado de consultar la información almacenada en el registro permitiendo consultar por un elemento determinado o por toda la lista de elementos. Por su parte, los elementos *ElementRegister* y *ElementEraser* son los que permiten añadir y modificar y eliminar información del registro, respectivamente.

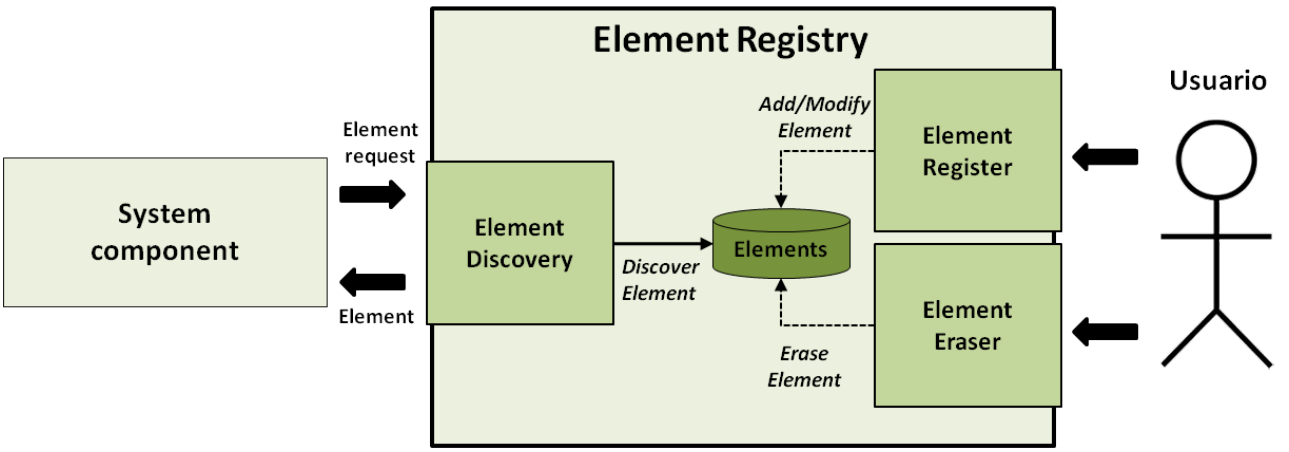

Figura 4: Arquitectura genérica de los componentes que implementan los registros

Una descripción más detallada de cada uno de los registros del sistema pueden consultarse en el *Anexo IV – Registros del sistema*.

4.4 - Broker de coordinación

La utilización de Linda permite desacoplar la comunicación entre los diferentes componentes del sistema haciendo que la misma sea asíncrona. Cuando ocurre un cierto evento en un componente, éste deposita una tupla en el espacio de tuplas gestionado por el coordinador de Linda. Esa tupla será leída en cualquier momento por otro componente utilizando una tupla patrón que concuerde con la tupla anterior.

Como pudo observarse en la figura 1, que mostraba la arquitectura del sistema, existen varios componentes que utilizan el *broker* para coordinarse y comunicarse. En primer lugar, el entorno de ejecución de Renew, o de otro sistema de gestión de *workflows* científicos, deposita una tupla en el *broker*. Posteriormente, esa tupla es leída por alguno de los mediadores. Cuando el trabajo finaliza, el mediador correspondiente deposita otra tupla en el *broker* para que el entorno de ejecución del sistema de *workflows* científicos recoja el resultado del trabajo. Adicionalmente, las tuplas pueden ser leídas por componentes adicionales.

Se realizan diferentes acciones para que cada componente sea capaz de distinguir y detectar qué tuplas debe gestionar. En primer lugar, cada uno de los componentes añade al principio de la tupla una palabra clave que identifica el evento que se ha producido. El entorno de ejecución utiliza la palabra clave `DEPLOYED` para indicar que esa tupla solicita la ejecución de un trabajo. Este tipo de tuplas son detectadas y tratadas por los diferentes mediadores definidos. Estos, a su vez, depositan tuplas en el *broker* con palabra clave `EXECUTED`, para indicar que el trabajo se ha ejecutado y devolver sus resultados, o `ERROR`, para indicar que el trabajo ha sufrido algún fallo. Si el trabajo ha finalizado correctamente el entorno de ejecución del sistema de gestión de *workflows* científicos detecta la tupla y continúa con la ejecución del mismo. Si por contra, el trabajo ha fallado, es el componente de gestión de fallos el que recoge la tupla. Este componente gestiona el fallo y puede depositar en el *broker* una tupla indicando que se vuelva a ejecutar el trabajo (palabra clave `DEPLOYED`) o una tupla indicando que ha finalizado la ejecución porque el fallo no es corregible (palabra clave `EXECUTED` con estado de error).

De esta forma, la adición de nuevos componentes es muy sencilla. Tan sólo es necesario que el nuevo componente utilice otra palabra clave para identificar los eventos adecuados. De igual manera, el reemplazo de un componente por otro que realice la misma labor o la actualización del mismo es completamente transparente al resto del sistema haciendo que el mismo sea muy adaptable.

Otro aspecto fundamental para realizar la correspondencia entre las tuplas depositadas y los componentes que la gestionan es la denominada política de *matching* o emparejamiento. En las operaciones de lectura realizadas por los componentes adicionales y por el sistema de gestión de *workflows* científicos utilizamos la política de emparejamiento débil ya que es más flexible que la política de emparejamiento fuerte. Por contra, no se puede utilizar este tipo de política en las operaciones de lectura realizadas por los mediadores.

Cuando se define un trabajo, puede indicarse el *grid* en el que se quiere ejecutar el mismo. Si no se indica este aspecto, el sistema decide en qué infraestructura se va a ejecutar el trabajo. Es en este segundo caso cuando la política de emparejamiento débil no funciona. En la figura 5 se muestra un ejemplo del problema. Los mediadores esperan tuplas que vayan dirigidas a los mismos, sin embargo, cuando no se especifica un *grid*, ni el emparejamiento débil ni el emparejamiento fuerte consiguen que exista una correspondencia entre las tuplas.

```
Tupla del trabajo:              ["DEPLOYED",[...],[...],[...],[usuario, "null"]]
Patrón del mediador Hermes:     ["DEPLOYED",?,?,?, [?, "hermes.cps.unizar.es" ]]
Patrón del mediador Aragrid:    ["DEPLOYED",?,?,?, [?, "ui-ara.bifi.unizar.es"]]
Patrón del mediador Piregrid:   ["DEPLOYED",?,?,?, [?, "ui-prg.bifi.unizar.es"]]
Patrón del mediador de Amazon: ["DEPLOYED",?,?,?, [?, "AmazonEC2"]]
```

Figura 5: Ejemplo de tuplas utilizadas para el despliegue de trabajos

Para solucionar este aspecto se define una nueva política de emparejamiento competitivo utilizada solamente con tuplas que describen trabajos sin especificar el *grid* de ejecución. En este caso, el sistema permite el emparejamiento con los mediadores que sean capaces de ejecutar la aplicación indicada. Si existen varios mediadores capaces de ejecutar dicha aplicación, el *broker* selecciona uno de ellos de forma no determinista. El algoritmo en pseudocódigo utilizado para la política de emparejamiento competitivo aparece en la figura 6.

```
/* Devuelve cierto si se pueden emparejar tupla y patrón */
boolean emparejamientoCompetitivo(tupla, patrón) {
   // Obtenemos la palabra clave
   id_tupla = obtenerPalabraClave(tupla);
   id_patrón = obtenerPalabraClave(patrón);

   // Comprobamos si es una tupla de trabajo y el patrón puede emparejarse
   Si ( id_tupla == "DEPLOYED" AND id_patrón == "DEPLOYED") {
      // Obtenemos el entorno de ejecución
      grid = obtenerGrid(tupla);

      // Comprobamos si la tupla es guiada hacia algún Grid
      Si (grid == null) {
         // Si no es guiada obtenemos el grid del patrón y la aplicación
         // a ejecutar y el usuario que lo solicita de la tupla
         grid = obtenerGrid(patrón);
         aplicación = obtenerAplicacion(tupla);
         usuario = obtenerUsuario(tupla);

         // Hay emparejamiento si existe el grid definido en el patrón,
         // el usuario tiene permiso para ejecutar en el grid y el grid es
```

```
            // es capaz de ejecutar la aplicación
            devuelve existeGrid(grid) AND existeUsuario(usuario, grid) AND
                     existeAplicacion(aplicación, grid);
         }
         si no {
            // Si la tupla es guiada utilizamos el emparejamiento débil
            devuelve emparejamientoDebil(tupla, patrón);
         }
      }
      si no {
         // Para el resto de lecturas utilizamos el emparejamiento débil
         devuelve emparejamientoDebil(tupla, patrón);
      }
}
```

Figura 6: Algoritmo para la política de emparejamiento competitivo.

Otro aspecto fundamental del que se encarga el *broker* tiene que ver con la identificación de los trabajos entre los diferentes componentes del sistema. Los trabajos se identifican externamente utilizando los ficheros de salida del mismo, pero éste identificador no es suficiente como ya se ha comentado. Internamente, es necesario asegurar que los identificadores de los trabajos son únicos porque un trabajo puede pasar por diferentes componentes y su tratamiento no debe mezclarse con el de otros trabajos.

Para solucionar esta limitación, el *broker* asigna un identificador único a cada trabajo cuando este es desplegado. La razón de que el *broker* sea el encargado de asignar los identificadores se debe a que es el componente central del sistema, el que es compartido por todos los componentes del sistema. El identificador utilizado consiste en un número entero que se va incrementando sucesivamente junto con la fecha en la que se despliega la tupla. Con esto conseguimos que el identificador sea único, aunque el *broker* sea reiniciado.

El identificador de la tupla se añade al final de la misma y es recuperado por el componente que lee la tupla de forma completamente transparente al usuario. Una vez en el componente, se almacena dicho identificador y éste se utiliza para identificar el resto de tuplas relacionadas con el trabajo. Esta característica es importante en ciertos componentes, por ejemplo, en el de gestión de fallos para conocer cuántas veces ha fallado un mismo trabajo.

La gestión de estos identificadores se realiza de forma completamente transparente para el usuario. Para ello, el identificador se almacena internamente en el tipo de dato *Tupla* utilizado en el sistema. Sin embargo, debido a la implementación del *broker* de mensajes, no es posible almacenar el identificador en el mismo. Para poder almacenar dicho identificador ha sido necesario modificar la API que permite interactuar con el *broker*.

Los cambios realizados afectan tanto a las operaciones de escritura como a las operaciones de lectura. En las operaciones de escritura se añade al final de la tupla el identificador del trabajo. Este identificador se obtiene internamente de la tupla que se quiere añadir o es generado por el *grid* si la tupla no tiene identificador. En las operaciones de lectura se añade un comodín al final de la tupla patrón para poder recuperar ese identificador. Una vez recuperado se almacena en la tupla obtenida y se elimina el último campo de la misma, que contiene el identificador, para que la gestión del mismo sea completamente transparente.

En el *Anexo V - Broker de coordinación* se puede consultar más información sobre el broker de coordinación.

4.5 - Componente de gestión de fallos

En la figura 7, puede observarse el diseño del componente de gestión de fallos y su interacción con el *broker* de mensajes para recuperar las tuplas de error y escribir las tuplas correspondientes a la decisión tomada.

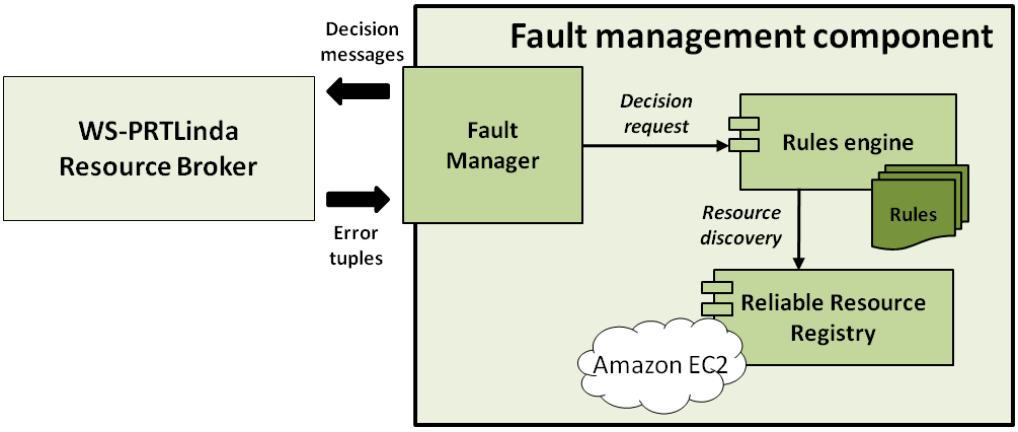

Figura 7: Arquitectura del componente de gestión de fallos

En primer lugar, de forma totalmente transparente a este componente, cuando la ejecución de un trabajo falla, el mediador correspondiente captura el fallo e introduce en el *broker* una tupla de error. Esta tupla se identifica por la palabra clave `ERROR` y contiene tanto la información relativa al trabajo que falló como la causa del fallo. El gestor de fallos (*Fault Manager*) es el encargado de recuperar dicha tupla, procesarla, creando para ello un nuevo hilo de ejecución, y solicitar al motor de reglas que tome una decisión sobre el trabajo.

Por su parte, el motor de reglas (*Rules engine*) toma una decisión teniendo en cuenta una serie de reglas básicas (base de conocimiento) y si el trabajo había fallado ya antes. Para definir estas reglas, hemos utilizado RuleML [W21] como lenguaje de reglas ya que se trata del lenguaje estándar para reglas en entornos Web. Las decisiones que puede tomar pueden ser: abortar la ejecución del trabajo, reiniciar el trabajo en el mismo *grid* o en otro diferente, reiniciar el mismo utilizando un *grid* confiable o continuar la ejecución de forma normal.

Este último caso contempla la ejecución del trabajo en un tipo especial de recurso que es catalogado como fiable. No hay que olvidar, que nuestro objetivo es ejecutar de forma satisfactoria los diferentes trabajos solicitados, por lo que la utilización de estos recursos ayudan a lograr dicho objetivo. Para ello se incluye un Registro de Recursos Fiable (*Reliable Resource Registry*) en el que se guarda una lista de *grids* fiables. Es aquí donde proponemos la utilización de alguna infraestructura de computación externa fiable como el servicio de computación *Cloud* de Amazon, *Amazon Elastic Compute Cloud* (Amazon EC2) [W9]. En cualquier caso, el motor de reglas es el encargado de decidir en cuál de los recursos fiables debe ser ejecutado el trabajo, devolviendo esta información al gestor de fallos.

Independientemente, de si se reinicia el trabajo en un *grid* fiable o se toma otra decisión, el gestor de fallos recupera esa decisión e introduce una nueva tupla en el *broker* de mensajes indicando la misma. Esta tupla será capturada por el mediador correspondiente si la decisión es reiniciar el trabajo o por el motor encargado de la ejecución del *workflow* si la acción tomado es la de abortar el trabajo o continuar con la ejecución.

Para ilustrar el funcionamiento del componente, se incluye la figura 8. En dicha figura, se muestra la ejecución del componente en el caso de que sea necesario reiniciar la ejecución del trabajo en un *grid* confiable. En caso de que la decisión tomada por el motor de reglas sea otra diferente, el comportamiento es análogo eliminando la interacción con el servicio de descubrimiento de *grids* fiables.

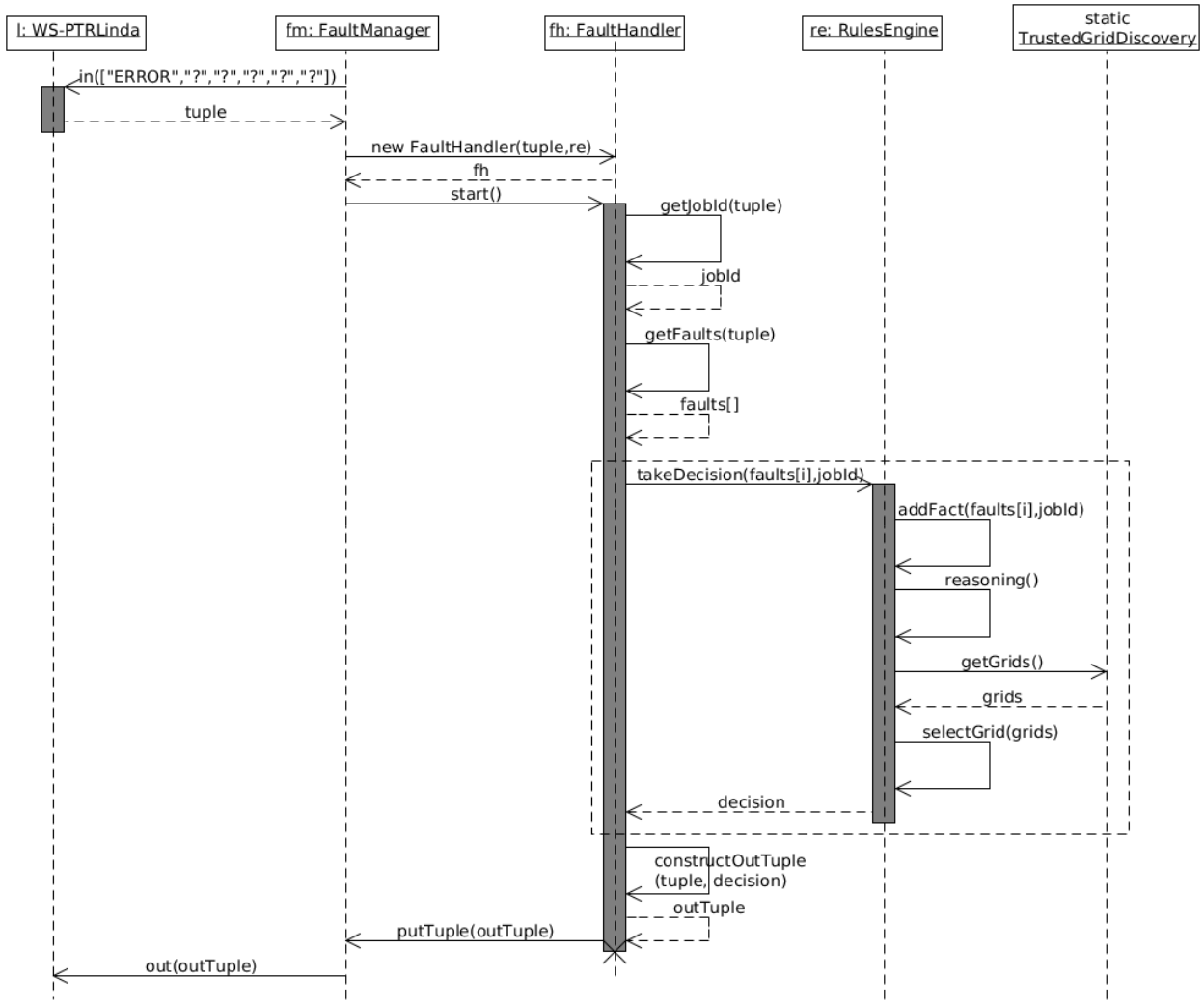

Figura 8: Traza de eventos del componente de gestión de fallos

En primer lugar, el *Fault Manager* obtiene una tupla del *broker* WS-PTRLinda y crea un nuevo hilo para gestionar la tupla (denominado *Fault Handler*). A continuación se obtiene el identificador del trabajo y los fallos que se han producido, solicitándose al motor de reglas que tome una decisión para cada fallo. Para ello, el motor añade el fallo al sistema y razona utilizando las reglas definidas para tomar una decisión. En caso de que la decisión sea reiniciar el trabajo en un *grid* fiable, se solicita al servicio de descubrimiento todos los *grids* fiables. Una vez obtenida la lista, el motor indica la decisión al gestor de fallos seleccionando uno de los *grids* de forma no determinista. Cuando se han tratado todos los fallos se construye la tupla asociada a la decisión generada y se coloca la misma en el *broker.*

La aparición de fallos puede deberse a diferentes causas, la aparición de un fallo durante la ejecución de la tarea, un fallo durante el movimiento de información necesario (a la entrada

o a la salida) de la tarea o un fallo en la definición del trabajo. En la figura 9, se puede observar una relación de todos los fallos que pueden aparecer al ejecutar las diferentes tareas que componen un *workflow* científico.

```
ERROR(1) : Malformed tuple.
ERROR(2) : Malformed scp URI $(uri). Impossible to move the file
ERROR(3) : The string $(file) can not be converted to an URI
ERROR(4) : Application can not be null.
ERROR(5) : Impossible to replace wildcards in $(path)
ERROR(6) : Execution failed. $(Execution_error)
ERROR(7) : Download/Upload from $(uri) failed. Connection error. $(error)
ERROR(8) : Download/Upload from $(uri) failed. Data movement error. $(error)
ERROR(9) : Job deployment failed. $(error)
ERROR(10): Undefined email for user $(user)
ERROR(11): User for $(global_user) undefinied on grid $(host)
ERROR(12): Grid $(host) undefinied
ERROR(13): Application $(appName) undefined on $(host)
ERROR(14): Output data retrieval failed. $(error)
ERROR(15): Movement from $(source) to $(destination) failed. $(Error)
ERROR(16): Job execution error. $(status)
ERROR(17): Invalid URI $(uri)
ERROR(18): Impossible to retrieve log. $(error)
```

Figura 9: Tipos de errores identificados en el sistema

Las acciones correctoras propuestas para los errores anteriores aparecen en la tabla 3. Se han definido cuatro tipos de acciones correctoras, dependiendo del tipo de fallo:

- La acción más restrictiva consiste en abortar la ejecución del trabajo. Este tipo de acción se toma cuando se está seguro de que el reinicio del trabajo provocará un nuevo fallo, por ejemplo, si la tarea ha sido mal definida. En este caso, se notifica al usuario del fallo para que corrija el error cometido.
- Otra posibilidad es reiniciar el trabajo. Este tipo de acción se realiza cuando existe la posibilidad de que reiniciar el trabajo puede solucionar el problema. Esta política es frecuentemente utilizada en sistemas distribuidos y permite solventar problemas de disponibilidad de los recursos, principalmente. Dentro de esta acción correctora se establecen tres posibilidades:
  1. Si es la primera vez que falla esa tarea, se reinicia la misma en los recursos habituales.
  2. Si se produce por segunda vez el mismo fallo, se reinicia el trabajo en un *grid* fiable para aumentar las posibilidades de que el mismo se ejecute con éxito.
  3. Si el trabajo falla por tercera vez, se considera que existe un error grave y se aborta el trabajo en lugar de reiniciarlo.
- En caso de que el fallo se haya producido a la hora de recuperar los ficheros de salida, se permite continuar normalmente la ejecución indicando en el *log* que se ha producido dicho fallo. A pesar de que es posible que este fallo provoque un fallo en sucesivas tareas, un error al mover la salida no justifica volver a ejecutar el trabajo de forma completa, ya que éste puede ser muy costoso, por lo que es más sencillo y útil que el usuario realice el movimiento de dicha salida manualmente.
- La última acción correctora propuesta es similar a la anterior y se utiliza en el caso de que no se pueda recuperar el *log* de la tarea ejecutada. En este caso, se decide continuar con la

ejecución normal del *workflow* indicando en el *log* que no se ha podido recuperar el mismo.

Número de fallo	Acción correctora
1, 2, 3, 4, 10, 11, 12, 13, 17	Abort
5, 6, 7, 8, 9, 16	Restart
14, 15	Continue without output
18	Continue without log

Tabla 3: Acciones correctoras para los diferentes tipos de errores.

Se puede consultar más información sobre este componente en el *Anexo VI – Componente de gestión de fallos*.

4.6 - Componente de movimiento de datos

La inclusión de diferentes *grids* bajo un mismo sistema plantea el problema del movimiento de datos entre los mismos, el cual se dificulta porque el usuario no conoce a priori en qué *grid* va a ser ejecutado su trabajo. Además, no se puede conocer la infraestructura de ejecución hasta el momento exacto en el que se va a ejecutar el trabajo, complicando enormemente el movimiento de los datos.

La primera cuestión que era necesaria resolver antes de definir cómo se iba a realizar el movimiento de datos era la identificación de los mismos. Se valoraron tres alternativas:

- Identificar los datos como si fueran datos locales. Esta primera alternativa tenía como objetivo identificar los datos sin tener en cuenta los recursos, permitiendo que los mismos se ubicaran en cualquiera de los *grids* definidos. El principal beneficio que presenta esta alternativa es que abstrae al usuario de definir el *grid* en el que se encuentra un dato. Por contra, esta alternativa implica que el sistema tenga que buscar el dato en todos los *grids* usados, con el aumento de complejidad y la pérdida de eficiencia consecuente, y presenta el problema de que no es posible saber qué fichero es el correcto si existen ficheros con el mismo nombre en diferentes *grids*. Este último aspecto hizo que se descartara la idea.
- La segunda alternativa consistía en especificar, con algún tipo de lenguaje propio, tanto el propio dato como el recurso en el que se encuentra el mismo. Sin embargo, esta alternativa fue descartada porque uno de los objetivos del proyecto es facilitar la compartición de *workflows* científicos por lo que la utilización de un lenguaje propio no era una alternativa viable.
- La tercera alternativa planteada, y que fue elegida finalmente, fue la utilización de URIs [W23] para identificar de forma única los datos. El uso de URIs, es una forma estándar de identificar datos en entornos distribuidos y heterogéneos por lo que representa la mejor alternativa para nuestro propósito. Además, esta opción nos da la posibilidad de utilizar datos que se encuentren en servidores externos de cualquier tipo con lo que no limitamos los datos que pueden ser utilizados a datos que se encuentren en los Grids definidos.

La utilización de URIs planteaba el problema de que el usuario debe ser consciente del protocolo de transferencia que debe ser usado para transferir el fichero. Para facilitar este aspecto se introducen dos características. En primer lugar, para el caso de que el usuario esté seguro de que los ficheros se encuentran en el *grid* en el que se va a ejecutar una tarea, se permite utilizar simplemente la ruta del fichero. En segundo lugar, se permite identificar un fichero con el esquema o protocolo `file`. La utilización de este formato permite al usuario no indicar el usuario del *grid* y el protocolo de transferencia, siendo estos recuperados del registro de usuarios y del registro de *grids* del sistema. Un ejemplo de cómo identificar un fichero con estos dos mecanismos puede observarse en la figura 10.

```
a) file://hermes.cps.unizar.es/~/ejemplo.txt
b) sftp://shernandez@hermes.cps.unizar.es/~/ejemplo.txt
```

Figura 10: Identificación de un fichero.

a) Identificación con URI 'file'. **b)** Identificación con URI 'sftp'.

Asimismo, se permite la utilización de comodines para definir la ruta de un fichero o grupo de ficheros dentro de un *grid*. Por ejemplo, en la figura 1 se ha utilizado el carácter '~' para definir que el fichero 'ejemplo.txt' se encuentra en el *home* del usuario. Esta característica permite definir de una forma sencilla y potente un grupo de ficheros utilizando solamente una URI. Finalmente, en la figura 11 se incluye el árbol de decisión utilizado para definir el comportamiento que debe ser seguido ante las diferentes descripciones posibles.

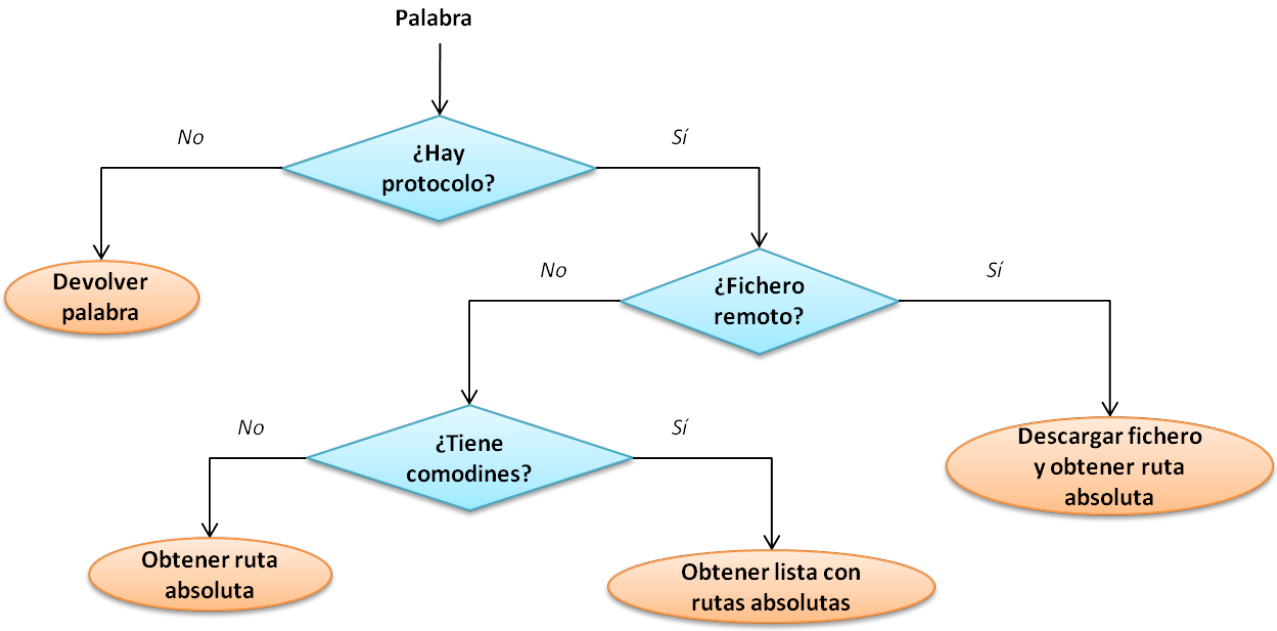

Figura 11: Árbol de decisión para las URIs

Cuando se parsea una palabra, se comprueba en primer lugar que la palabra corresponde efectivamente a una URI. Si no es así, la palabra puede hacer referencia a un argumento o parámetro de la aplicación o a una ruta relativa y no se realiza ninguna modificación sobre la misma al no tener certeza de dicha circunstancia. Si la palabra se trata de una URI, se detecta si la misma corresponde a un fichero local o a un fichero remoto. En el caso de que se trate de un fichero remoto, se descarga el mismo y se obtiene la ruta absoluta del fichero dentro del *grid* en el que se ha descargado el fichero. Si por contra se trata de un fichero local, se obtiene

simplemente la ruta absoluta del mismo. Sin embargo, en este último caso, la palabra podría tener algún comodín y referirse a una lista de ficheros por lo que hay que detectar esta situación y, en el caso de que exista algún comodín, reemplazar el mismo por la lista de ficheros correspondiente.

En el *Anexo VII - Componente de movimiento de datos* puede consultarse más información acerca del componente de movimiento de datos.

4.7 - Mediadores

En la figura 12, se muestra el diseño arquitectural realizado para los mediadores, independientemente del *middleware* concreto con el que interactúa el mediador. En la figura puede observarse como el mediador está dividido en una serie de componentes. El primero de ellos es el que se conoce como *Job Manager*. Este componente se encarga de gestionar y coordinar la ejecución de los diferentes trabajos que son encargados al mediador. Las principales tareas de las que se encarga este componente son las siguientes:

- Interactuar con el *broker* de mensajes Linda, para obtener los trabajos que hay que ejecutar en el *grid* y para escribir los resultados de los mismos en el *broker*.
- Mover los datos generados por el trabajo a su ubicación final correspondiente, ya sea dentro del mismo *grid* o entre *grids*, utilizando para ello el componente de movimiento de datos.
- Mantener una lista de los trabajos que se encuentran en ejecución. Esta lista indica la correspondencia entre los identificadores asignados por el *grid* concreto en el que se ejecuta un trabajo y el propio trabajo. Sirve saber qué trabajo ha finalizado en cada momento.

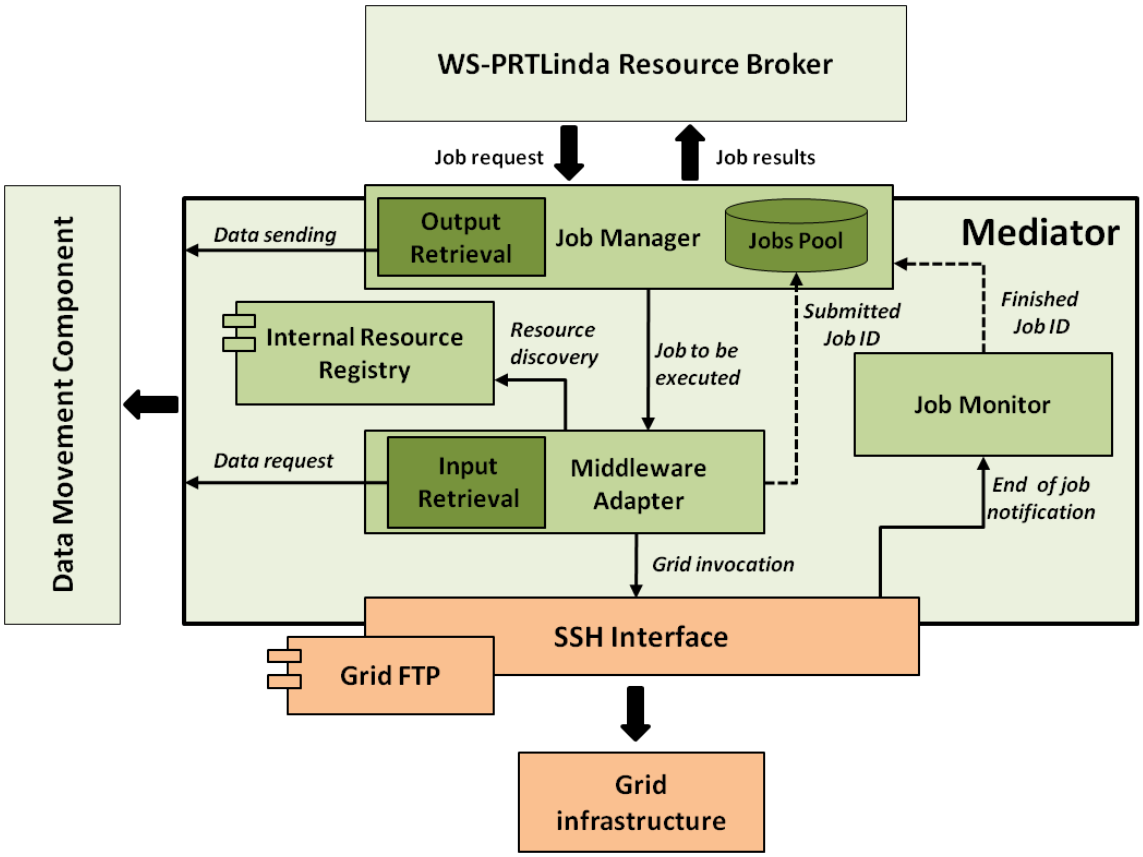

Figura 12: Arquitectura general de un mediador

Por su parte, el *Middleware Adapter* se encarga de transformar la descripción del trabajo obtenida del *broker*, la cual es independiente del entorno de ejecución, en una descripción que pueda ser ejecutada por el *middleware* de la infraestructura *grid*. Asimismo, prepara los datos necesarios para poder ejecutar el trabajo y realiza el despliegue del mismo. Para recuperar los datos necesarios, se utiliza el componente de movimiento de datos, que es común a todos los mediadores. Mientras que, para realizar el despliegue del trabajo se utiliza una interfaz SSH que nos permite establecer una conexión segura con la infraestructura. Una vez desplegado el trabajo, el *middleware* utilizará internamente protocolos como GridFTP para mover los datos a su ubicación final dentro del *grid*.

El *Internal Resource Registry* se utiliza a la hora de realizar la transformación comentada anteriormente. Este componente, obtiene información referente a la aplicación que se desea ejecutar y el usuario que va a ejecutar dicha aplicación utilizando la misma para definir el trabajo a ejecutar.

El último componentes del mediador es el *Job Monitor*. La necesidad del mismo viene determinada por el comportamiento asíncrono del sistema, en el cual, un trabajo es lanzado pero no se sabe cuándo va a terminar el mismo. Para detectar el fin del trabajo se introduce este componente. La detección de la finalización del trabajo puede realizarse de diferentes formas dependiendo del *middleware* utilizado.

Un mecanismo habitual que permite detectar la finalización de un trabajo es utilizar el correo electrónico. Cuando el trabajo finaliza, el *middleware* se encarga de enviar un correo electrónico a la dirección especificada al encargar la ejecución del trabajo. En otros *middlewares*, este mecanismo no está soportado por lo que es necesario utilizar un mecanismo de encuesta para detectar si un trabajo determinado ha finalizado.

De esta forma, cuando un trabajo finaliza, se detecta esta situación con alguno de los mecanismos anteriores y se notifica de este hecho al *Job Manager* indicando el identificador utilizado por el *grid* para el trabajo que ha finalizado. Este identificador, permite recuperar al *Job Manager* los datos del trabajo, realizar el movimiento de los datos de salida, si es necesario, y notificar al *broker* de dicho evento.

Otro aspecto importante que debe ser controlado por el mediador es todo lo relativo a la aparición de errores. Los errores pueden aparecer tanto antes de ejecutar el trabajo, debido a fallos en la definición del trabajo o en el movimiento de los datos necesarios, por ejemplo, como después de ejecutar el mismo, es el caso de errores de ejecución del trabajo o de movimiento de los datos de salida. Si se produce un fallo, el mediador debe detectar esta circunstancia y notificar al *broker* de mensajes del mismo para que el fallo sea gestionado por el componente de gestión de fallos.

Se puede consultar más información sobre los mediadores en el *Anexo VIII - Mediadores*.

Capítulo 5 -Evaluación del sistema

La evaluación del rendimiento del sistema desarrollado es un aspecto fundamental para comprobar que se cumplen algunos de los objetivos marcados al inicio del proyecto. En nuestro caso, esta evaluación consiste en el análisis del rendimiento y la escalabilidad del sistema desarrollado y su comparación con otro sistema, como Taverna.

5.1 - Métricas de evaluación

Los dos aspectos que vamos a evaluar son el rendimiento y la escalabilidad del sistema.

En lo referente al rendimiento del sistema queremos medir la sobrecarga que introduce el sistema en referencia a la ejecución manual de los trabajos. En general, este no es un aspecto crítico en un sistema de gestión de *workflows* científicos ya que los trabajos desplegados son computacionalmente muy costosos y la introducción de una pequeña sobrecarga no es significativa. En cualquier caso, creemos que es importante conocer el rendimiento del sistema en cualquier situación y garantizar que la sobrecarga introducida por el sistema no es un aspecto crítico. Para ello realizamos dos tipos de experimentos:

1. En un primer experimento, queremos comprobar el rendimiento del sistema al ejecutar un trabajo de forma aislada. El objetivo de este experimento es comprobar la sobrecarga natural que introduce el sistema.
2. El segundo experimento busca comprobar el rendimiento del sistema cuando se ejecuta un trabajo en una situación en la que existe una gran carga en el sistema, en cuanto a número de trabajos pendientes de ejecución. Este experimento tiene el objetivo de comprobar si la existencia de una gran carga en el sistema afecta a la ejecución aislada de un trabajo por lo que también es importante para la escalabilidad del sistema.

Por su parte, en lo que respecta a la escalabilidad del sistema, vamos a medir el rendimiento del *broker* cuando se somete al mismo a una gran carga en cuanto a número de trabajos que se ejecutan a la vez. Este aspecto es fundamental a la hora de determinar el número de trabajos que pueden ejecutarse al mismo tiempo sin que se reduzcan gravemente las prestaciones. Para ello, vamos a desplegar la misma tarea un gran número de veces de forma que haya un gran número de peticiones de lectura y peticiones de escritura a la vez.

Tanto para medir el rendimiento del sistema como para medir la escalabilidad del mismo, realizaremos los experimentos con el objetivo de comparar el sistema desarrollado con Taverna, otro sistema de gestión de *workflows* científicos. Esta comparación aborda tanto las capas de modelado como la capa de ejecución del sistema por lo que se contemplan tres posibilidades: la utilización de Taverna de forma aislada, la integración de Taverna con el sistema desarrollado y la utilización de Renew como entorno de modelado del sistema.

5.2 - Definición de experimentos

Para evaluar el sistema se han definido una serie de experimentos dedicados a medir los aspectos relatados anteriormente. En este apartado ofrecemos una breve descripción de los mismos, una descripción más completa aparece en el *Anexo IX – Evaluación del sistema*.

El primer experimento consiste en la ejecución en serie de una tarea variando el número de repeticiones, midiendo el tiempo al inicio y al final del experimento y calculando la sobrecarga introducida por el sistema. Con este experimento buscamos medir el rendimiento de una tarea al ejecutarse de forma aislada en el sistema. El segundo experimento mide la sobrecarga introducida al ejecutar el sistema de forma aislada. Para ello se introducen un elevado número de trabajos en el *broker* y se procede de la misma forma que en el experimento anterior. El tercer experimento se utiliza para evaluar el sistema en cuanto a escalabilidad. Para ello se lanzan un elevado número de tareas en paralelo de forma que las mismas comienzan y terminan prácticamente a la vez, midiéndose el tiempo al inicio y al final del experimento y calculando la sobrecarga introducida.

5.3 - Resultados

En este apartado se presentan los resultados correspondientes al experimento de escalabilidad al ser el más representativo de los realizados. El resultado, análisis y conclusiones del resto de experimentos puede consultarse en el *Anexo IX – Evaluación del sistema*.

Los experimentos se han desarrollado desplegando el *middleware* desarrollado sobre un AMD Athlon 3500+ de 64 bits, 2.2GHz, 2GB de RAM y un subsistema de E/S SATA. El sistema operativo utilizado ha sido GNU/Linux Ubuntu, con un núcleo 2.6.35 optimizado para el hardware utilizado. Renew 2.2 se ha ejecutado sobre la versión 1.6.0r26 del Java SE Runtime Environment. Los experimentos se han ejecutado sobre el sistema con un único usuario, y los procesos innecesarios se han cerrado.

La figura 13 muestra los datos de la sobrecarga introducida por el sistema en función del número de tareas que se ejecutan en paralelo. En la misma sólo se proporcionan los resultados utilizando el *framework* con Renew como entorno de modelado ya que con Taverna no es posible ejecutar de forma paralela tantos trabajos. Podemos observar como la sobrecarga introducida en el sistema al incrementar el número de trabajos que se ejecutan en paralelo crece rápidamente. El incremento de la sobrecarga sigue una tendencia polinómica de tercer grado, casi exponencial. La sobrecarga es aceptable hasta un número de 125 trabajos ejecutándose en paralelo y se dispara si intentamos ejecutar un mayor número de trabajos.

Para comparar en cuanto a escalabilidad las herramientas de modelado Renew y Taverna realizamos un análisis más detallado de los resultados obtenidos hasta un máximo de 25 trabajos ejecutándose en paralelo. Los resultados de este análisis se muestran en la figura 14. Como se observa en la gráfica, el incremento de la sobrecarga limitando el número de tareas a 25 es lineal en Renew mientras que es polinómica de tercer grado en Taverna. Esto se debe a que los flujos paralelos en Taverna se han desarrollado de forma explícita, al no proporcionar un mecanismo nativo para definir tareas en paralelo de forma implícita, a través de bucles o similar.

De esta forma, al incrementar el número de tareas el *workflow* se vuelve muy pesado en términos de memoria necesaria para gestionar y ejecutar el mismo provocando que se disparen las prestaciones. Asimismo, este aspecto también provoca que el modelado del *workflow* se ralentice enormemente y que la máquina se colapse. Por esta razón no se han podido realizar experimentos con un mayor número de tareas en paralelo utilizando Taverna.

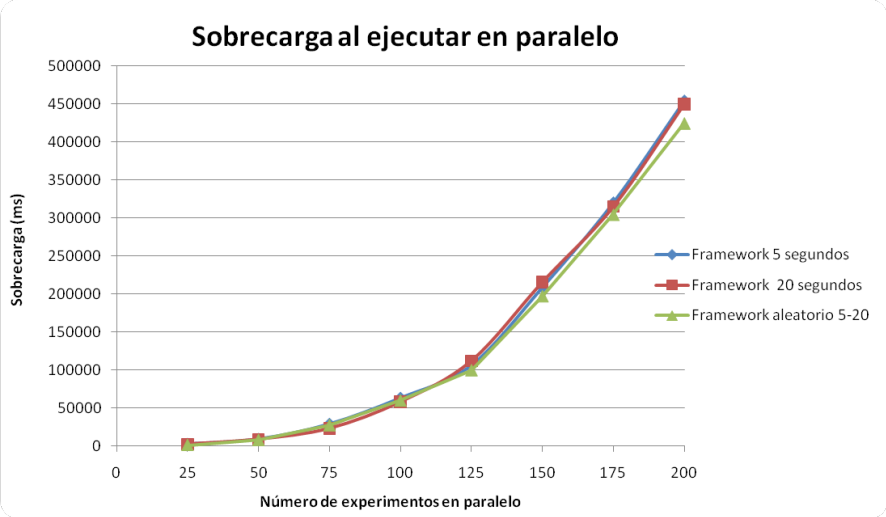

Figura 13: Gráfica que muestra la sobrecarga del *framework* al aumentar el número de tareas en paralelo.

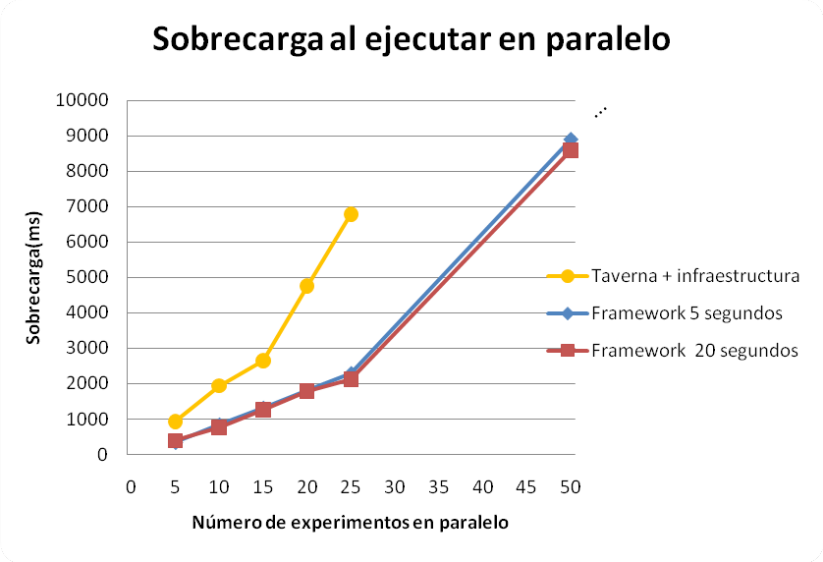

Figura 14: Gráfica que compara Taverna y el *framework* en términos de escalabilidad.

5.4 - Conclusiones

Como era de esperar, la sobrecarga del sistema crece enormemente al ejecutar un elevado número de tareas en paralelo. Esto se debe a que el *broker* se convierte en el cuello de botella del sistema al tener que atender un gran número de peticiones a la vez, siendo especialmente crítica la operación de lectura por la gran cantidad de tuplas que hay que comprobar. Las prestaciones se mantienen dentro de un límite aceptable hasta 125 tareas.

En cualquier caso, la utilización de las redes de referencia permite introducir un gran nivel de paralelismo de forma natural mientras que los sistemas de gestión de *workflows* científicos actuales no suelen proporcionar mecanismos para proporcionar ese nivel de paralelismo. En el caso de Taverna, hay que incluir el paralelismo de forma explícita, no se pueden realizar construcciones genéricas que nos permitan llegar a altos niveles de paralelismo. En lo que se refiere a las prestaciones, esto implica que el rendimiento de Taverna al paralelizar un elevado número de tareas disminuye muy rápidamente.

Capítulo 6 - Caso de estudio: First Provenance Challenge

Como caso de estudio se utilizó el *workflow* científico propuesto en el *First Provenance Challenge*[W24]. La elección del mismo se fundamenta en que es un *workflow* de referencia en la comunidad científica, por lo que ha sido muy estudiado y utilizado para la verificación de diferentes sistemas de gestión de *workflows*. Además, resulta un problema fácilmente escalable y que requiere del desplegado y ejecución de diversas tareas tanto de forma secuencial como paralela, por lo que resulta muy adecuado para demostrar la viabilidad de la solución propuesta.

6.1 - Objetivos

Nuestro objetivo principal consiste en verificar el correcto funcionamiento de la herramienta desarrollada utilizando un *workflow* científico real.

También queremos ilustrar la flexibilidad del *framework* desarrollado tanto en lo que respecta a la posibilidad de utilizar diferentes entornos de modelado como a la posibilidad de utilizar diferentes infraestructuras *Grid* de forma transparente. Con este propósito presentamos dos implementaciones diferentes una realizada con redes de referencia en Renew y otra realizada con Taverna. En estas implementaciones se utilizarán los recursos *Grid* a los que tiene acceso el grupo de investigación GIDHE: Aragrid, Piregrid y Hermes.

6.2 - Definición del problema

Se plantea un *workflow* para crear "atlas" cerebrales de la población en base a imágenes de alta resolución del Centro de Datos de imágenes de resonancias magnéticas funcionales (fMRI) [W25]. Este *workflow* está compuesto de varios procedimientos, mostrados como óvalos naranjas, y datos que fluyen entre ellos, mostrados como rectángulos. Pueden distinguirse 5 fases o etapas en la ejecución del mismo. Cada una de las etapas se muestra en una misma franja horizontal. Los procedimientos utilizan la suite AIR[W26], para crear una imagen cerebral promedio a partir de una colección de imágenes tridimensionales de alta resolución, y la suite FSL [W27] para crear las imágenes bidimensionales de cada dimensión del cerebro que constituyen la salida del *workflow*. Las entradas del *workflow* son un conjunto de imágenes cerebrales (*anatomy images*) y una única imagen de referencia (*reference image*). Todas las imágenes anatómicas de entrada, son escáneres tridimensionales de un cerebro con diferente resolución. Para cada imagen se proporciona el fichero con la imagen y un fichero de cabecera con metadatos sobre la misma.

Las etapas del *workflow* se describen a continuación:

1. Para cada imagen de entrada, el procedimiento **align_warp** compara la imagen de entrada y la imagen de referencia para determinar como debe alinearse la imagen de entrada con respecto a la imagen de referencia. La salida de cada proceso define una serie de parámetros que indican la transformación espacial a realizar sobre cada imagen.
2. Los procedimientos **reslice** realizan de forma efectiva la transformación indica en el paso anterior, utilizando los parámetros obtenidos como salida de align_warp. Como resultado se obtiene la imagen modificada.

3. El proceso **softmean** junta todas las imágenes anteriores en una sola imagen promedio de las mismas y obteniendo una imagen tridimensional de mayor calidad.
4. Cada uno de los procesos **slicer** separa la imagen tridimensional obtenida mediante softmean, en imágenes individuales de cada una de sus dimensiones (x, y, z). Estas imágenes en 2D se obtienen a partir del centro de la imagen 3D.
5. Finalmente, el proceso **convert** convierte la imagen de entrada obtenido con el proceso slicer a formato gif para un tratamiento más sencillo de la misma.

El resultado de la ejecución del *workflow* con los datos originales de entrada se puede observar en la figura 15.

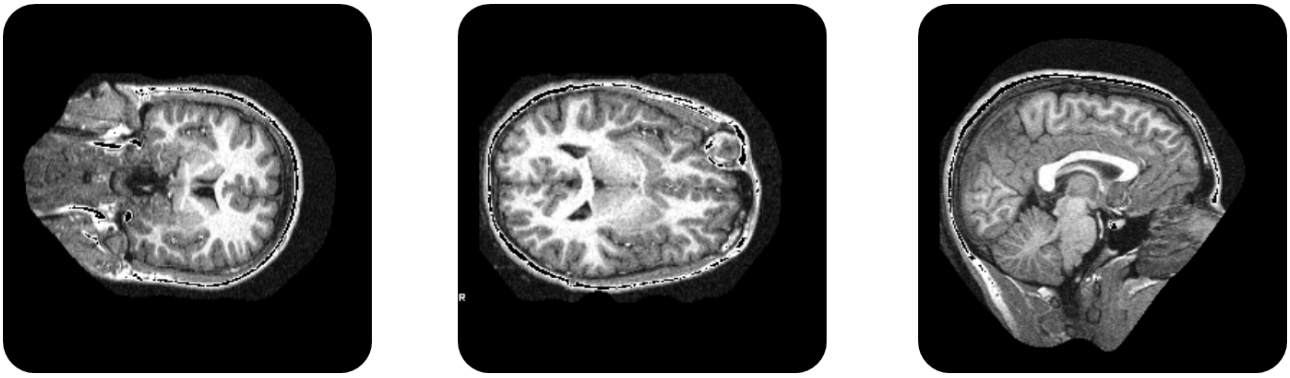

Figura 15: Imágenes cerebrales resultado de la ejecución del First Provenance Challenge.

6.3 - Modelado del problema

Para ilustrar el funcionamiento del sistema se ha modelado el *workflow* anterior utilizando el entorno de modelado propuesto, Renew, y un sistema de gestión de *workflows* científicos actual como es el caso de Taverna. De esta forma queremos demostrar la posibilidad de utilizar la infraestructura independientemente de la herramienta de modelado utilizada.

Modelado del problema con Renew

En la figura 16, se muestra el modelo del *workflow* científico correspondiente al *First Provenance Challenge* utilizando la herramienta Renew. Para dotar a la figura de una mayor claridad, en las etapas 1 y 2 sólo se incluye de forma completa la descripción de los procesos para la primera imagen. La representación es análoga para el resto de imágenes.

En lo que se refiere a las entradas, para demostrar la flexibilidad del sistema en lo que a la representación de los datos se refiere, se describen las entradas de diversas formas. Las imágenes 3 y 4 se describen sin indicar el protocolo concreto de comunicación utilizado para su descarga, que es obtenido del registro del *Grid* concreto, mientras que en las imágenes 1 y 2 sí que se especifica el protocolo de comunicación . Por su parte, la imagen de referencia que es utilizada en todos los procesos de la primera etapa se obtiene del servidor Web del grupo de investigación GIDHE, mostrando que no todos los ficheros deben estar en un *Grid*.

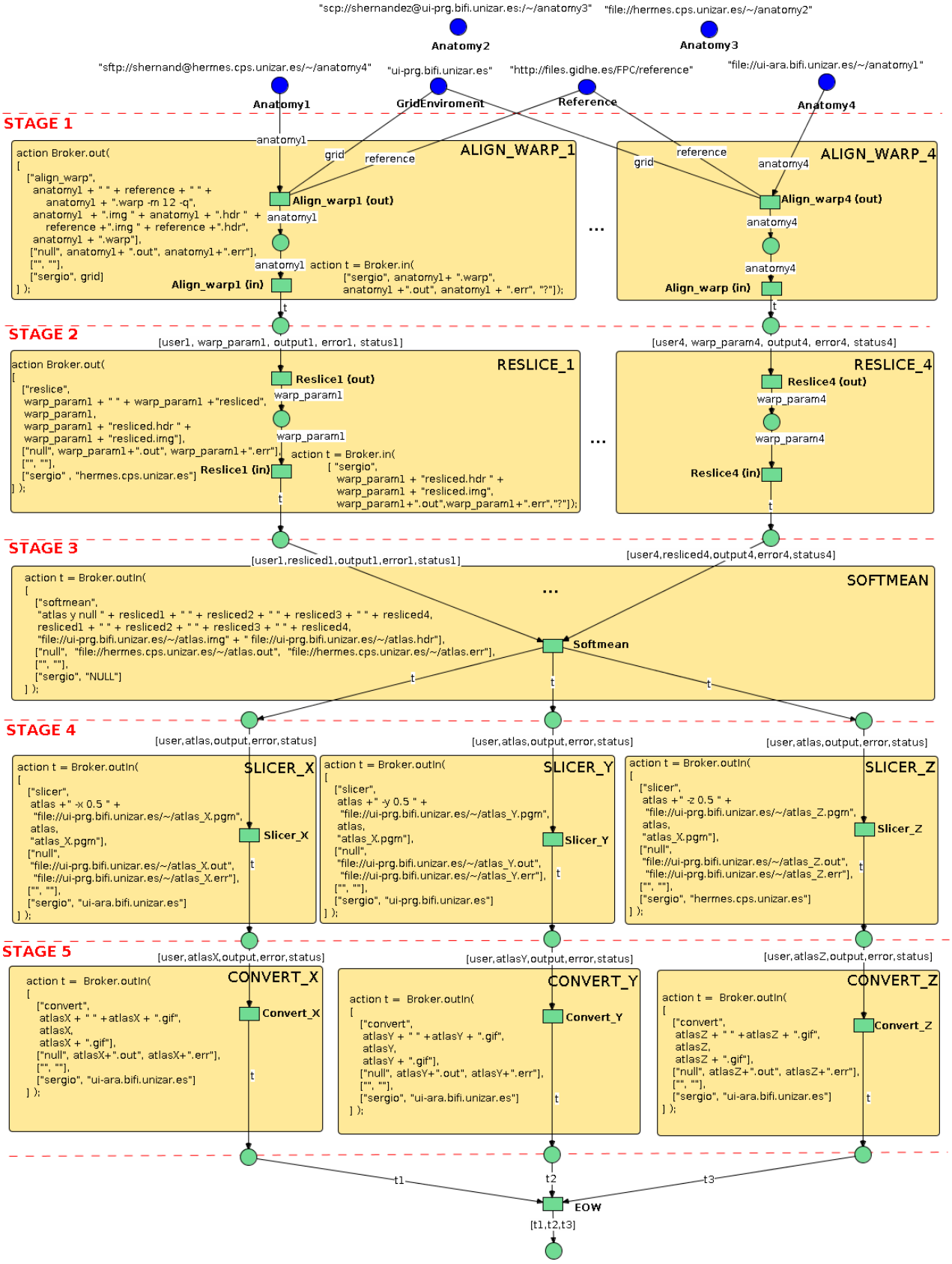

Figura 16: Modelo del *workflow* científico del *First Provenance Challenge* con Renew.

Las tareas de las etapas 1, 2 y 5 del *workflow* son ejecutadas por un sólo Grid. En la etapa 4, cada tarea es ejecutada por una de las infraestructuras disponibles, balanceando la carga. Finalmente, para la tercera etapa no se indica el *Grid* que se debe utilizar. En este caso, el sistema seleccionará de forma no determinista una de las infraestructuras comprobando que la aplicación se pueda ejecutar en la misma y que el usuario tenga permiso de ejecución en ese *Grid*. Al final del *workflow* se añade una transición con nombre `EOW` (*End Of Workflow*) que sirve para identificar el fin del *workflow*. Esta transición no es obligatoria pero se recomienda su utilización ya que permite detectar fácilmente que se ha llegado al final del problema.

Modelado del problema con Taverna

El modelo del *workflow* científico utilizando Taverna se muestra en la figura 17 a). En la parte superior del *workflow* aparecen las entradas del mismo, que son las mismas que en el caso anterior aunque la representación de Taverna sólo nos indica el nombre de la entrada, siendo el valor de la misma interno y no visible en el modelo.

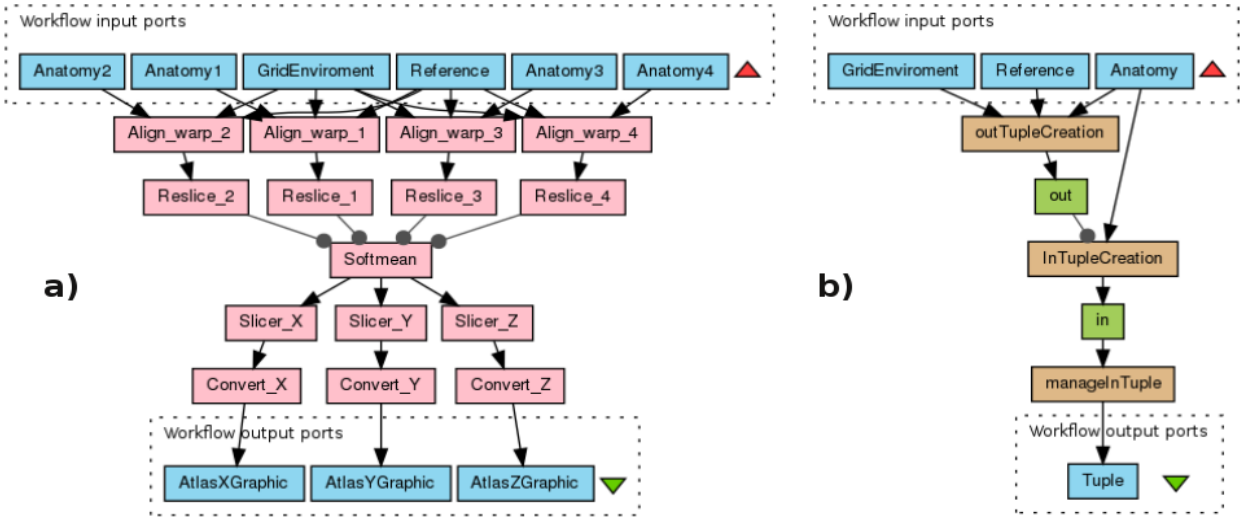

Figura 17: Modelo del *workflow* científico del *First Provenance Challenge* con Taverna. **a)** Modelo general del *workflow*. **b)** Modelo concreto para el *subworkflow* `align_warp`

En el modelo cada caja rosas representa la ejecución de un trabajo. Estas cajas son a su vez otros *workflows* más sencillos compuestos por las mismas operaciones que en el caso del modelo con Renew. La implementación de los diferentes *subworkflows* es análoga a la realizada en Renew y un ejemplo para el *subworkflow* `align_warp` puede observarse en la figura 17 b). En este caso, la implementación se ha realizado utilizando los componentes estándares de Taverna y no utilizando la clase `Broker` como en el caso anterior.

En primer lugar, se crea la tupla que se escribirá en el *broker* a partir de los parámetros de entrada en el componente `createOutTuple`. Utilizando dicha tupla se realiza la operación de escritura `out` sobre WS-PTRLinda. El siguiente paso consiste en crear el patrón utilizado para recuperar la tupla que indica el fin del trabajo . El mismo es realizado por el componente `createInTuple`. Después se realiza la operación de lectura `in` que nos devuelve como resultado una tupla, la cual es gestionada por el componente `manageInTuple` para obtener el *log* de ejecución y proporcionar el resultado. Finalmente, en la parte inferior del modelo se localizan las salidas del *workflow* que nos permiten recoger los resultados.

6.4 - Conclusiones

En este capítulo se ha mostrado cómo podemos modelar el mismo problema con dos técnicas diferentes, generando dos modelos equivalentes que se integran con la infraestructura desarrollada en el PFC. En el caso del modelado mediante la herramienta Renew, las *reference nets* nos ofrecen varias ventajas inherentes a su propia naturaleza, principalmente en lo relacionado con la gestión de concurrencia y paralelismo. En cualquier caso, lo que queda patente es que la infraestructura se puede integrar fácilmente con cualquiera de las tecnologías. Es más, en el caso de Taverna las modificaciones para adaptar el modelo basado en invocaciones directas al esquema de interacción requerido por la infraestructura son mínimas. Esto no hace sino favorecer la viabilidad y fácil adopción de la solución propuesta. Finalmente, cabe destacar cómo la posibilidad de balancear la computación del *workflow* en las diferentes infraestructuras disponibles mejora ciertos parámetros de calidad de servicio, como los tiempos de respuesta, las prioridades en cada *Grid* y los costes computacionales asociados.

Capítulo 7 - Conclusiones y trabajo futuro

En este proyecto se ha desarrollado el prototipo de un sistema de gestión de *workflows* científicos que permite integrar diferentes entornos *Grid* de una forma completamente transparente para el usuario. Esta integración se ha llevado a cabo utilizando un *broker* de mensajes basado en el modelo de coordinación Linda [B4, B5] y un conjunto de mediadores que gestionan las infraestructuras de computación. Además, se ha incluido un componente que gestiona los fallos que puedan aparecer en el sistema.

Gracias a que las capas de modelado y ejecución se encuentran desacopladas y a la utilización de un lenguaje independiente del entorno de ejecución para describir los trabajos a ejecutar, el sistema puede ser integrado en otros sistemas de gestión de *workflows* haciendo que el mismo sea muy adaptable. Asimismo, se ha propuesto la utilización de las redes de referencia como lenguaje de modelado para solventar algunos de los problemas de los lenguajes actuales como la falta de mecanismos para representar el paralelismo.

Como resultado se ha obtenido un sistema muy flexible ya que puede integrar gran cantidad de componentes de forma sencilla, dar servicio a usuarios con diferentes características, integrar entornos de ejecución muy diversos y ejecutar *workflow* realizados con diferentes lenguajes de modelado. Asimismo, el sistema es muy adaptable a los cambios del entorno al permitir que se modifique la configuración durante la ejecución del propio sistema.

La evaluación del sistema y el caso de estudio mostrados han demostrado la viabilidad del mismo. En el primer caso, se ha demostrado que el sistema es escalable al permitir un gran número de tareas ejecutándose a la vez en paralelo, mientras que, en el segundo caso, se ha demostrado la posibilidad de ejecutar un *workflow* real integrando diferentes grids.

Desde el punto de vista de la contribución de este proyecto a nivel científico, se ha desarrollado un sistema que abre la solución a uno de los principales problemas a los que se enfrentaba la comunidad científica, la integración de diferentes entornos de computación de forma transparente. Esto va a permitir que científicos de diferentes áreas puedan disponer de una mayor potencia de cálculo y avanzar más rápidamente en sus investigaciones. Asimismo, la interoperabilidad del sistema va a fomentar la compartición de recursos y experimentos entre científicos y la colaboración de los mismos, dos aspectos que van a ser clave en la forma de investigar en los próximos años. Finalmente, cabe destacar la posibilidad de mejorar la calidad de servicio y de aplicar los conceptos desarrollados a diferentes ámbitos que han ganado una gran importancia en los últimos años, como es el caso de la computación evolutiva y adaptativa y el *Green computing*.

7.1 - Trabajo futuro

Como trabajo futuro, se contempla de forma prioritaria la modificación del sistema para permitir que cada componente esté distribuido en diferentes máquinas. Para ello hay que modificar la gestión de los recursos del sistema ya que estos deben ser consultados por diferentes componentes. En este aspecto, se estudiarán dos posibilidades: integrar la configuración dentro del *broker* WS-PTRLinda [B9] y añadir una interfaz de servicio Web a los registros para que estos sean accesibles desde cualquier máquina.

Asimismo otras líneas de trabajo futuro son las siguientes:

- Incorporación de requisitos de calidad de servicio (QoS) a los trabajos y gestión de dichos requisitos. Para este punto se contempla la posibilidad de añadir nuevas políticas de emparejamiento que contemplen y satisfagan los requisitos de QoS. En este aspecto, también se propone la definición de políticas de emparejamiento dirigidas a *Green computing*.
- Adición de nuevos componentes como un sistema de monitorización que permita conocer el estado de cada trabajo en cualquier momento y un componente de *scheduling* avanzado que permita realizar el despliegue de los trabajos de forma más potente a lo permitido en las políticas de emparejamiento.
- Adición de nuevos mediadores al sistema para permitir la integración de un mayor número de infraestructuras de computación al sistema.
- Obtención y gestión de la información de *provenance* o procedencia, es decir, de toda la información respecto de las tareas ejecutadas, las relaciones entre las mismas, los resultados generados, etc.
- Realización de un *plugin* que permita traducir de forma automática *workflows* modelados con diferentes lenguajes a redes de referencia para permitir la integración directa de los mismos en Renew.
- Utilización de *frameworks* de gestión específicos que faciliten las labores de despliegue de los trabajos en Amazon Elastic Compute Cloud (Amazon EC2) [W9] y añadan funcionalidades como monitorización completa de los mismos o balanceo de carga eficiente.

7.2 - Conclusiones a nivel personal

Durante la realización de la carrera de Ingeniería Informática he recibido formación en los diferentes campos de interés de este área. Sin embargo, una de las carencias que he detectado durante estos cinco años ha sido la falta de asignaturas en las que se aborde el mundo de la investigación. Por ello, decidí realizar un proyecto que tuviera una alta carga de labor investigadora para conocer ese mundo y completar mejor mi formación.

El proceso de búsqueda de información, lectura de artículos y análisis de las propuestas de otros autores son fases fundamentales del proceso de investigación. En un primer momento, estas fases pueden ser muy costosas llegando a dar la sensación de que no se está avanzando en el desarrollo del proyecto. Sin embargo, a medida que avanza con la investigación me he dado cuenta de que esta fase es muy enriquecedora ya que permite conocer la opinión y forma de afrontar un mismo problema por parte de otras personas. Además, analizar el trabajo de otros autores me ha permitido darme cuenta de la importancia de realizar un análisis crítico de los mismos. En un artículo el autor muestra su opinión y propuesta en relación a un aspecto determinado, sin embargo, ésta no tiene porque ser la mejor o puede estar equivocada por lo que es importante realizar un análisis propio.

Otro aspecto muy importante ha sido la oportunidad de enfrentarme con un proyecto de magnitud y aplicación real lo que ha significado una gran oportunidad para evaluar mi

capacidad como ingeniero. La magnitud del proyecto, me ha permitido darme cuenta de la importancia de la planificación y la gestión del trabajo. Antes de empezar el mismo no consideraba esto un aspecto realmente importante, sin embargo, a lo largo del mismo me he dado cuenta de que es muy importante llevar el mejor control posible sobre todos los aspectos que involucran el desarrollo del proyecto. De la misma manera, la realización en solitario del proyecto me ha permitido mejorar en mi capacidad auto organizativa, mejorando mi la metodología de trabajo en las últimas semanas de desarrollo, lo que ha supuesto un incremento de la productividad a lo largo del mismo.

En lo referente a la implementación, el proyecto me ha servido para mejorar mis aptitudes como programador y darme cuenta de la importancia de las fases de análisis y diseño. Al principio del proyecto me lanzaba a implementar ciertas cosas sin analizarlas y pensar en ellas demasiado lo que a la postre ha implicado tener que adaptar o rehacer parte del código desarrollado. Al final del proyecto, he conseguido analizar del problema de una manera más concienzuda, lo que me ha permitido a su vez diseñar la mejor solución para el mismo teniendo en cuenta todas las opciones posibles. Como resultado a la hora de implementar, esta fase ha sido mucho más sencilla.

En definitiva, la experiencia ha sido muy enriquecedora y satisfactoria. La realización de este proyecto me ha permitido iniciarme en el mundo de la investigación, descubriendo además que se trata de un mundo apasionante y animándome a realizar el doctorado. Asimismo, creo que el proyecto me ha permitido desarrollarme profesionalmente mejorando mis aptitudes como ingeniero y completando mi formación.