Universidad Zaragoza
1542

# Proyecto Fin de Carrera

Título del trabajo:
DESARROLLO DE UN SISTEMA DE WORKFLOW CIENTÍFICO QUE PERMITA LA EJECUCIÓN FLEXIBLE DE TAREAS EN UN CLÚSTER DE COMPUTADORES

English tittle:
DEVELOPMENT OF A SCIENTIFIC WORKFLOW SYSTEM THAT ALLOWS THE FLEXIBLE EXECUTION OF TASKS IN A COMPUTER CLUSTER

Autor/es

Ester Burges Blanco

Director/es

Rafael Tolosana Calasanz

ESCUELA DE INGENIERÍA Y ARQUITECTURA
2017

## Resumen

Los workflows científicos han surgido como una tecnología que da soporte computacional en la realización de experimentos científicos. Por una parte, un workflow puede verse como una especificación abstracta de un conjunto de tareas y sus dependencias entre sí. Estas dependencias establecen qué pasos deben realizarse para acometer un experimento científico. Por otro lado, un workflow puede verse como un programa, y un sistema de gestión de workflows como un entorno de programación especializado cuyo objetivo es la simplificación de las tareas de programación necesarias que tienen que realizar los científicos. Los sistemas de gestión de workflows científicos deben hacer una gestión eficiente de los recursos computacionales, la gestión de fallos, y así como la supervisión de los resultados intermedios y finales y la reproducibilidad total del experimento. La aproximación habitual para ejecutar un workflow en entornos de ejecución diferentes como clusters, grid o cloud consiste en la traducción de una especificación abstracta del workflow en una especificación concreta teniendo en cuenta los datos y los recursos. Sin embargo, estas aproximaciones suelen estar ligadas a soluciones concretas como la generación de un DAG (Grado dirigido acíclico) y su ejecución en un entorno High Throughput computing como HTCondor. Estas aproximaciones hacen que la monitorización, tratamientos de los fallos y gestión de recursos estén ligadas al entorno de ejecución y no a los aspectos de la especificación abstracta original. El objetivo de este proyecto es desarrollar un prototipo de sistema de workflow científico que permita definir políticas de gestión de recursos y de recuperación de fallos a nivel de aplicación. Por este motivo, se proporciona una especificación de workflows que es independiente del entorno de ejecución y que proporciona mecanismos de tolerancia a fallos para tratar fallos de aplicación (v. gr. gestión de excepciones). La especificación se realiza mediante Redes de Petri y Renew, y su diseño tendrá en cuenta que pueda ejecutarse en cualquier infraestructura: clúster Condor, contendores o incluso microservicios.

# Índice

# 1. Introducción

A partir de la década de los 90's, la tecnología workflow o flujo de trabajo se aplicó en el contexto de Sistemas de Información, como una nueva solución para la administración o gestión de los procesos que se realizan dentro de una organización. Con el auge de Internet y de las Tecnologías de Información, los workflows han evolucionado de manera rápida y su uso se ha incrementado en las actividades de negocio de diversas industrias. Los sistemas workflow se aplican en variedad de campos, extendiéndose a la coordinación y gerencia de procesos, manejo y control empresarial, y otros.

Los workflows científicos se utilizan para orquestar cálculos a gran escala en una amplia variedad de dominios científicos, incluyendo la astronomía, la bioinformática, la sismología y la física. Los workflows permiten la automatización de procesos de simulación y análisis de datos, asegurando que los cálculos se ejecutan de forma fiable, eficiente y correcta. El uso de tecnologías de workflow trae consigo una serie de beneficios sobre los enfoques tradicionales (como el *shell scripting*), entre los que se incluyen los siguientes [1]:

- *Facilidad de expresión computacional.* Los workflows proporcionan interfaces de alto nivel que dependen de formatos de fichero, lenguajes de programación específicos del workflow y herramientas de composición visual.
- *Facilidad de reutilización.* Como resultado de la naturaleza declarativa de la especificación de workflow, se puede reutilizar parte o todo el workflow, o modificarse para ajustarse a nuevos requisitos.
- *Gestión de fallos.* En caso de error por un hardware defectuoso y/o fallos puntuales durante la ejecución, cada paso en un workflow se puede reintentar automáticamente, y todo el flujo de trabajo puede ser comprobado y reanudado.
- *Computación paralela.* Los workflows permiten que los cálculos y los datos se procesen en paralelo en recursos distribuidos, disminuyendo considerablemente el tiempo de ejecución.
- *Seguimiento de la procedencia (*provenance *en inglés).* Esta es una característica fundamental para garantizar el método científico y la reproducibilidad de los experimentos. A medida que se ejecuta un workflow, la información sobre los datos procesados, los cálculos realizados y los recursos utilizados se guarda, para una posterior inspección, verificación y validación.

Las aplicaciones de flujo de trabajo se ejecutan en variedad de entornos, incluyendo portátiles, clusters de campus, cloud (como Chameleon o Amazon Web Services [AWS]), recursos distribuidos de alto rendimiento (como los gestionados por HTCondor) y computación de alto rendimiento (HPC) (como los que gestionan XSEDE y los laboratorios nacionales del Departamento de Energía de los Estados Unidos).

Las diferentes aplicaciones, según sus necesidades computacionales, cuentan con diferentes recursos informáticos y con diferentes aspectos de los workflows. Por ejemplo, las aplicaciones de bioinformática que realizan procesamiento genómico podrían ejecutarse en clústeres "cercanos" a la máquina de secuenciación. Los investigadores individuales pueden utilizar flujos de trabajo de astronomía y ejecutar estos flujos en su portátil u ordenador personal, mientras que los grandes recursos de datos pueden generarse en clouds comerciales. También hay aplicaciones, como las de la ciencia de terremotos, que necesitan recursos HPC para ejecutar trabajos paralelos dentro de un flujo de trabajo.

## 1.1. Motivación

A diferencia de los procesos de negocio computacionales, que se fundamentan frecuentemente en operaciones transaccionales, un proceso científico computacional está constituido por un conjunto de operaciones cuyo objetivo es el análisis o el procesamiento de datos. Muchas veces dichas operaciones conllevan un carácter inherentemente exploratorio, o bien se realizan complejas simulaciones. Habitualmente, los workflows científicos utilizan infraestructuras distribuidas muy complejas de terceros y que, por tanto, su ejecución está sujeta a una disponibilidad variable o incluso a la aparición de fallos inesperados. La estrategia habitual de ejecución de un workflow científico consiste en realizar transformaciones sobre la especificación original hasta adaptarla a las particularidades de la infraestructura destino. Hasta la fecha, pocas han sido las propuestas que permiten hacer el camino inverso. Dos son las razones, por un lado, se perseguía la máxima eficiencia, las transformaciones realizadas van encaminadas muchas veces a modificar la granularidad de los cómputos para así reducir el tiempo de procesamiento global. Por otro lado, las infraestructuras computacionales distribuidas habitualmente ofrecen mecanismos de tolerancia a fallos. No obstante, puede haber determinados fallos cuya acción automatizada requiera de la especificación original del workflow abstracto. Las herramientas disponibles no ofrecen dicha flexibilidad, por ello necesitamos una herramienta propia que nos permita de forma rápida hacer prototipos, con patrones, y monitorizar de forma visual las ejecuciones.

Queremos representar un proceso de ejecución de forma gráfica y obtener una herramienta con la que poder experimentar, que disponga de ciertas características de los workflows científicos y nos permita abstraernos de la infraestructura. Nuestra herramienta permitirá además al científico realizar modificaciones sin tener un elevado conocimiento de programación. Las marcas nos permiten ejecutar una parte del workflow, utilizando los datos ya obtenidos, sin necesidad de reejecutar el proceso entero. También queremos poder tomar decisiones a nivel de workflow en caso de una excepción o fallo.

## 1.2. Propósito y Alcance del Proyecto

Utilizando una interfaz en Java proporcionada, que permite el lanzamiento de trabajos a Condor y la monitorización de forma asíncrona de su terminación, hemos modelado un workflow científico muy habitual en el dominio, Montage, utilizando redes de Petri y Renew. Esto nos permite ver gráficamente su funcionamiento y realizar una especificación sin ambigüedad alguna. No queremos competir con otras herramientas existentes, ni tampoco pretendemos realizar una herramienta para uso comercial. El objetivo es realizar una herramienta que nos de flexibilidad para la experimentación.

## 1.3. Estructura del Documento

El **capítulo 2** ofrecerá una introducción a los workflows científicos, se verán algunos de los distintos entornos de ejecución como los clústeres HTC y concretamente HTCondor. Además, se describirá el concepto de Redes en Redes y la herramienta Renew.

El **capítulo 3** ofrecerá una descripción más detallada de los lenguajes de Workflow, su representación y patrones, así como su aplicación en el ejemplo utilizado, Montage.

En el **capítulo 4** se explicarán las clases utilizadas para las conexiones ssh y Condor, su relación e integración con Renew.

En el **capítulo 5** se mostrarán las conclusiones y se propondrán posibles mejoras futuras.

Por último, en los **anexos** se incluirán los patrones Condor, patrones de Montage, recursos, clases auxiliares utilizadas y algunos de los test realizados en Java.

# 2. Conceptos Subyacentes

Antes de entrar en detalle, veremos el concepto de workflow, más concretamente workflows científicos y algunos entornos de ejecución.

## 2.1. Workflows Científicos

El término "workflow" se ha utilizado tradicionalmente en el contexto de flujos de trabajo de negocio. La *Workflow Management Coalition* (WfMC), una organización internacional centrada en el flujo de trabajo empresarial, define un flujo de trabajo, o workflow, como [2]: "la facilitación o automatización de un proceso de negocio, en su totalidad o en parte". De manera similar, un workflow científico se ve como una descripción de los procesos que un científico necesita ejecutar para crear un "resultado experimental científico" [3]. En términos generales, un workflow implica varias tareas que se realizan usando ciertas herramientas y recursos. Además, a veces, las tareas pueden necesitar recursos que tienen que ser producidos previamente por otras tareas. Otras veces, se requiere una cierta sincronización entre las tareas. Dos cuestiones clave en esta descripción son el flujo de datos (es decir, la transferencia de los "contenidos" entre estas tareas) y el flujo de control (es decir, la transferencia del "orden de ejecución" entre las tareas). Además, cada tarea requiere algunos "recursos activos" para realizarse. Igualmente, estos conceptos pueden aplicarse en un contexto científico.

La idea principal en el enfoque de flujo de trabajo es separar la descripción del proceso de flujo de trabajo o especificación del sistema responsable de su ejecución. Los workflows se modelan y se representan gráficamente como "Nodos" (representando tareas) interconectados por "Enlaces" dirigidos (que representan flujos/dependencias de control). Entonces, las especificaciones del workflow científico pueden ser ejecutadas (o interpretadas) por un software apropiado de ejecución científica, denominado como Workflow Management System (WfMS). De acuerdo con WfMC, un WfMS se define como [2]: "un sistema que define, gestiona y ejecuta completamente los flujos de trabajo a través de la ejecución de software cuyo orden o ejecución se impulsa por una representación computarizada de la lógica de workflow". Una arquitectura básica de un WfMS científico típico que implica cuatro elementos principales se describe en [4]:

- Una especificación de *Workflow* que contiene la descripción del proceso con tareas, flujos y dependencias, cuya descripción veremos en el capítulo 3.
- Los *Recursos* computacionales disponibles. En el dominio de los workflow científicos se suele distinguir entre servicios y recursos. Nosotros trabajaremos principalmente en un clúster HTCondor.

- Un *Motor* para organizar el comportamiento del sistema. Este componente es responsable de interpretar la descripción del workflow y "darle vida". En nuestro caso la herramienta Renew, de la que se hablará en el punto 2.3.
- Un *Middleware* que realiza la comunicación y vinculación entre los Recursos y el Motor (detallado en el punto 4). Particularmente, debe apoyar la transferencia de datos y la invocación de operaciones.

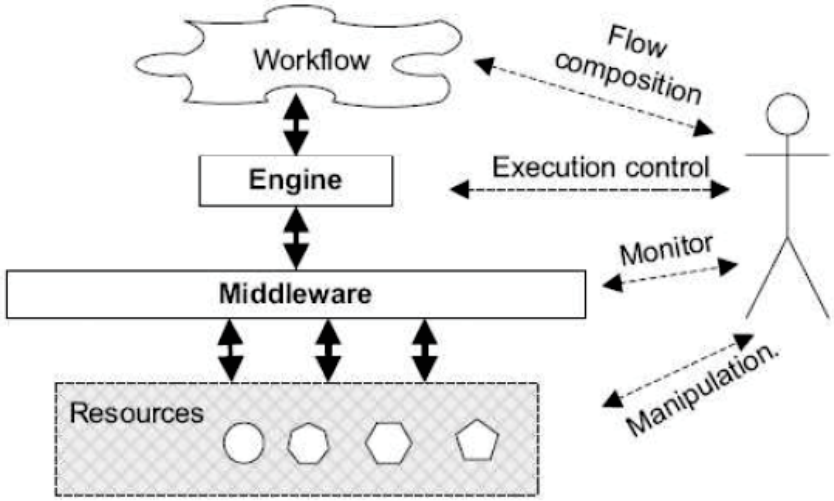



Figura 1: Principales componentes de un sistema de Grid workflow

## 2.2. Entornos de Ejecución

En esta sección veremos el clúster HTC, y más concretamente HTCondor, utilizado en nuestro ejemplo.

### 2.2.1. Clúster HTC – Condor

*High-throughput computing* (HTC), es un término informático para describir el uso de muchos recursos computacionales durante largos periodos de tiempo para llevar a cabo una tarea computacional [5].

Aunque tanto a high-throughput computing como a high-performance computing (HPC) los denominamos Computación de alto rendimiento, existen diferencias entre ellos. Las tareas de HPC se caracterizan por necesitar grandes cantidades de potencia de cálculo durante cortos períodos de tiempo, mientras que las tareas de HTC también requieren grandes cantidades de computación, pero durante mucho más tiempo (meses y años, en lugar de horas y días).

Los entornos de HPC se miden a menudo en términos de FLOPS. La comunidad HTC, sin embargo, no se preocupa por las operaciones por segundo, sino más bien por las operaciones por

mes o año. Por lo tanto, el campo HTC está más interesado en la cantidad de trabajos que se pueden completar durante un largo período de tiempo en lugar de la rapidez.

HTCondor es un sistema de software que crea un entorno de computación de alto rendimiento HTC. Sigue una filosofía de código abierto. Utiliza efectivamente la potencia de cálculo de las estaciones de trabajo que se comunican a través de una red. Se ejecuta en Linux, Unix, Mac OS X, FreeBSD y los sistemas operativos Windows contemporáneos.

Al igual que otros sistemas de batch, HTCondor proporciona un mecanismo de colas de trabajo, una política de programación, un esquema de prioridad y una monitorización y administración de recursos. Puede utilizarse para administrar la carga de trabajo en un clúster dedicado de computadoras, y/o para extender el trabajo a los equipos de escritorio inactivos.

Los usuarios envían sus trabajos a HTCondor, HTCondor los coloca en una cola, elige cuándo y dónde ejecutarlos basándose en una política, monitoriza su progreso y, en última instancia, informa al usuario al finalizar. Además, tiene la capacidad de detectar que una máquina que está ejecutando un trabajo de HTCondor ya no está disponible (el demonio detecta una pulsación de tecla, un movimiento del mouse o un uso alto de CPU por un proceso que no es de HTCondor). En ese momento, HTCondor comprobará el punto en que se encuentra el trabajo y podrá moverlo a una máquina diferente que de lo contrario estaría inactiva, continuando el trabajo en la nueva máquina desde donde lo dejó. [6, 7]

## 2.3. Redes en Redes y Renew

Una red de Petri ordinaria puede definirse informalmente como un grafo bipartito dirigido que consiste en lugares, transiciones, arcos y tokens (ver [8] para una definición formal y una introducción general al formalismo). Hay muchas extensiones a las redes de Petri ordinarias, tales como redes de Petri de alto nivel y redes de Petri cronometradas, que proporcionan muchas características adicionales, tales como mayores niveles de abstracción. Las redes de Petri ordinarias y sus extensiones han sido utilizadas para la especificación, análisis e implementación de workflows [9].

En este trabajo, utilizamos un tipo específico de Redes de Petri de Alto Nivel, Redes dentro de Redes [10] (*Nets-within-Nets* en inglés). En particular, utilizamos *Reference nets*, cuyo nombre se debe al hecho de que sus tokens también pueden ser redes. Por lo tanto, las redes pueden contener otras redes, formando una estructura anidada. Sin embargo, el distintivo principal de las redes de referencia es su naturaleza dinámica: en las redes de referencia, una red en sí puede expresar explícitamente la creación de nuevas instancias de red, una característica que permite a una red auto-modificar su estructura de forma dinámica y que, en este sentido, es similar al concepto de reflexión de los lenguajes de programación.

Además, una red padre (conocida como red del sistema) y sus redes de token hijas (conocidas como redes de objetos) pueden interactuar entre sí a través de un mecanismo de comunicación denominado canales síncronos. Intuitivamente, este tipo de comunicación se basa en la sincronización entre una transición en una red del sistema con una transición en una red de objetos. Además, un canal también puede albergar variables cuyo enlace se basa en la unificación, proporcionando un mecanismo de comunicación flexible y bidireccional. El concepto de unificación es una de las principales ideas detrás de la programación lógica y representa el mecanismo de unión de los contenidos de las variables [11].

En canales síncronos, una de las redes de comunicación actúa como un invocador, tratando de sincronizar con otra transición, mientras que el otro interlocutor actúa como invocado. Por ejemplo, la expresión de canal síncrono en el lado de red invocador `netexpr:channelname(expr, expr, . . .)` indica que `netexpr` es una expresión que debe evaluar a una red que debe tener una transición con un canal síncrono llamado `channelname`. En el lado invocado, la expresión debe ser de la forma `:channelname(expr, expr, . . .)`, ignorando completamente quién es el invocador. Esto se puede ver en la Figura 2, en la que `channelname` se ha denominado como `ch`. Posteriormente, sólo cuando la unificación de las variables de canal es posible y las transiciones correspondientes e implícitas están habilitadas, estas transiciones se pueden disparar (el disparo es simultáneo) y se puede establecer una comunicación bidireccional. Estos aspectos, junto con el dinamismo inherente del formalismo, constituyen un mecanismo de comunicación clave para apoyar la adaptabilidad ambiental de los workflows.

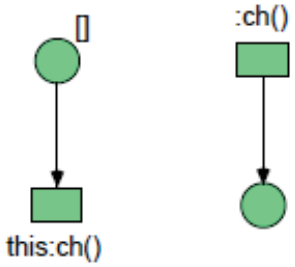



Figura 2: Ejemplo en Renew de red simple con canal síncrono

Otra ventaja del formalismo de las Redes de Referencia es que la herramienta Renew [12, 13] se puede utilizar para interpretarlas, de modo que las redes de Referencia pueden convertirse en un formalismo modelado ejecutable. Renew, herramienta utilizada para este trabajo, es un editor y simulador basado en Java que proporciona un modelado gráfico flexible de redes de referencia. La primera versión oficial de Renew fue lanzada en 1999 y desde entonces se ha desarrollado continuamente como un entorno de edición y simulación de red de Petri. Además de usar Renew principalmente para modelar redes de Petri, los complementos proporcionan soporte para diferentes técnicas de modelado, es decir, diagramas de UML o BPMN. Renew también utiliza tuplas y expresiones Java como lenguaje de inscripción primario. Además de las redes de objetos, las activaciones de transición también pueden transportar objetos Java.

Cada elemento de red puede llevar inscripciones semánticas. Los lugares pueden tener opcionalmente un tipo de lugar y un número arbitrario de expresiones de inicialización. Las expresiones de inicialización se evalúan y los valores resultantes sirven como marcas iniciales de los lugares. En una expresión, [] denota un token negro simple. Por defecto, un lugar no tiene marca inicialmente. Los arcos pueden tener opcionalmente una inscripción de arco. Cuando se produce una transición, se evalúan sus expresiones de arco y se mueven los tokens según el resultado. Las transiciones pueden estar equipadas con una variedad de inscripciones. Las inscripciones de expresión son expresiones ordinarias que se evalúan mientras que el simulador de red busca un enlace de la transición. El resultado de esta evaluación se descarta, pero se puede usar el operador de igualdad = para unir variables que se utilizan en otra parte. Las inscripciones de transición que están prefijadas con la palabra reservada `guard` tienen restricciones en sus modos de disparo, una transición sólo se puede disparar si todas sus inscripciones de guardia se evalúan como verdaderas. Además, las inscripciones de transición que están prefijadas con la palabra reservada `action` se pueden ejecutar después del disparo de la transición.

En los siguientes puntos veremos con más detalle estos usos de Renew aplicados en nuestro ejemplo.

# 3. Lenguaje de Workflow

La representación de workflows y patrones se ha realizado mediante la herramienta Renew, mediante la cual hemos desarrollado el ejemplo Montage, que veremos a continuación.

## 3.1. Caso de Estudio: Montage

Montage (nombre completo Montage Astronomical Image Mosaic Engine) es un conjunto de herramientas de software utilizado en astrofotografía para ensamblar imágenes astronómicas en formato Flexible Image Transport System (FITS) en imágenes compuestas, llamadas mosaicos, que preservan la calibración y la fidelidad de posición de las imágenes de entrada originales. [14]

La astronomía tiene un rico patrimonio de descubrimiento a partir de colecciones de datos de imágenes que cubren esencialmente toda la gama del espectro electromagnético. Las colecciones de imágenes en un rango de frecuencia se han estudiado a menudo aisladamente de las de otros rangos de frecuencias. Por lo tanto, la astronomía necesita un software de mosaico de imágenes que proporcione mosaicos de grado científico a partir de conjuntos de datos de imágenes múltiples como si fueran imágenes individuales con un sistema de coordenadas común, proyección de mapas, etc. Es decir, el software debe preservar la integridad astrométrica y fotométrica de los datos originales y rectificar la emisión de fondo desde el cielo o desde el instrumento utilizando modelos basados en la física [15].

El proyecto Montage [16] proporciona al astrónomo las herramientas necesarias para construir mosaicos en formato FITS [17], incluyendo soporte para todos los sistemas de coordenadas astronómicas comunes, todas las proyecciones del mapa del Sistema Mundial de Coordinación (WCS) [18], tamaños de imagen arbitrarios (incluyendo imágenes de cielo completo) y rotaciones, y muestreo espacial especificado por el usuario. El soporte computacional de Montage toma como entrada un conjunto de imágenes constituyentes de un mosaico y, como salida, las compone para formar una imagen de mayor tamaño, resultado de las operaciones de composición.

Montage se ha diseñado como un kit de herramientas portátil escalable que puede ser utilizado por astrónomos en sus escritorios para el análisis científico. Montage proporciona varias herramientas que pueden utilizarse de forma separada, muy frecuentemente se utilizan de forma secuencial. No obstante, se han integrado también en sistemas de workflow científico como Montage para reconstruir mosaicos de gran tamaño. Por lo tanto, puede considerarse una tecnología habilitadora, ya que los mosaicos que genera amplían las vías de investigación astronómica y es una herramienta valiosa.

Montage utiliza los siguientes cuatro pasos, aplicados secuencialmente, para calcular un mosaico:

- Re-proyección de imágenes de entrada a una escala espacial común, sistema de coordenadas y proyección WCS.
- Modelación de la radiación de fondo en imágenes para lograr escalas de flujo y niveles de fondo comunes minimizando las diferencias entre imágenes.

- Rectificación de imágenes a una escala de flujo y nivel de fondo comunes.
- Ensamblado de imágenes re-proyectadas con fondos corregidos en un mosaico final.

Algunas de las etapas de Montage, utilizadas para este proyecto, son:

- **mAdd** reúne las imágenes reproyectadas utilizando la misma plantilla de cabecera template.hdr y trabajando desde la misma lista mImgtbl que se usó para reproyectar el conjunto de imágenes.
- **mBackground** elimina un plano de fondo de una imagen FITS.
- **mBgModel** es un programa de modelado/adaptación. Utiliza la tabla de parámetros de diferencia de imagen a imagen para determinar interactivamente un conjunto de correcciones que se aplicarán a cada imagen para lograr un "mejor" ajuste global.
- **mDiffFit** crea, ejecutando mDiff y mFitplane, una tabla de parámetros de diferencia de imagen a imagen.
  - **mDiff** calcula una diferencia simple entre un solo par de imágenes superpuestas. Está destinado para su uso en imágenes reproyectadas donde los píxeles ya se alinean exactamente.
  - **mFitplane** ajusta un plano (excluyendo píxeles secundarios) a una imagen. Destinado para usar en las diferentes imágenes generadas tras usar mDiff.
- **mImgtbl** extrae la información de cabecera FITS de un conjunto de archivos y crea una tabla de metadatos de imagen ASCII que puede ser utilizada por otros de los programas.
- **mOverlaps** analiza una tabla de metadatos de imagen para determinar una lista de imágenes superpuestas.
- **mProjectPP** reproyecta una sola imagen a la escala definida en un archivo de plantilla de encabezado FITS.
- **mJPEG** crea una imagen JPEG de una o más imágenes FITS de entrada.

Una forma de mejorar el tiempo de ejecución de Montage es ejecutarlo en sistemas multiprocesador o un clúster de computadores, puesto que varias de las etapas de Montage se pueden ejecutar en paralelo. Esto se puede ver en la Figura 3, que muestra un flujo de procesamiento paralelizado para el caso simple de procesar cuatro imágenes de entrada.

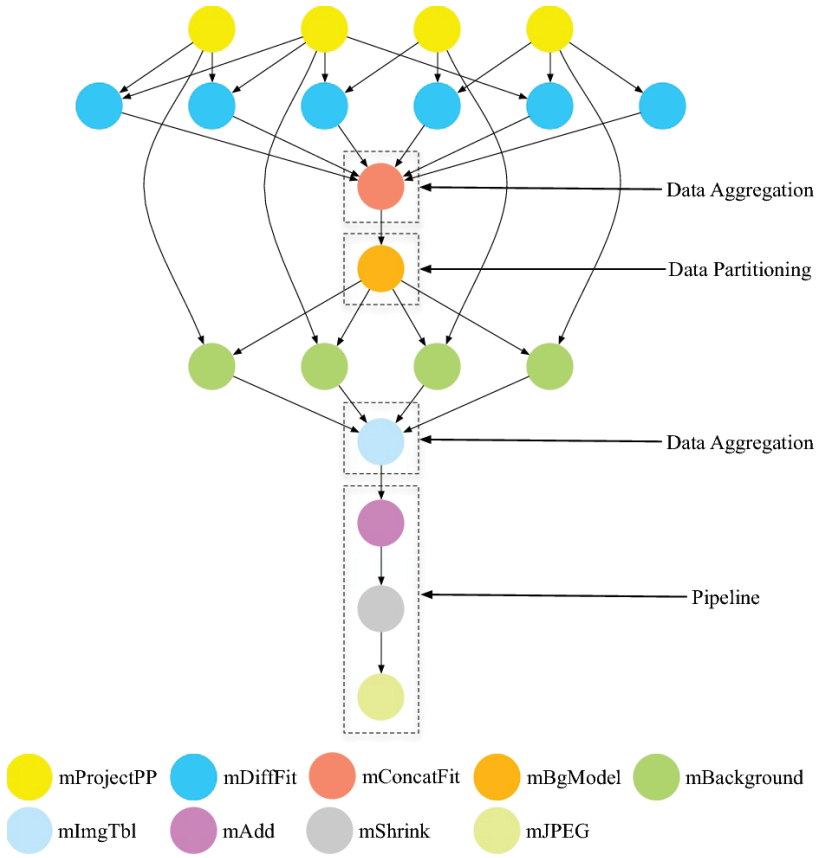



Figura 3: Ejemplo de flujo de trabajo de proceso paralelo en Montage, para un caso en el que cuatro imágenes se convierten en un mosaico. La reproyección, el cálculo de las diferencias entre imágenes y la aplicación de la rectificación de fondo pueden realizarse en paralelo.

Habitualmente, se utiliza la tecnología de workflows científicos para ejecutar Montage en un clúster. No obstante, en el diseño tradicional, la estructura del workflow varía en función del número de imágenes que constituyen la entrada.

Partiendo de un workflow abstracto como el de de la Figura 3, el sistema de workflow Pegasus [19] lo transforma en un workflow concreto que se puede ejecutar en un clúster de computadores, como el de la Figura 4.

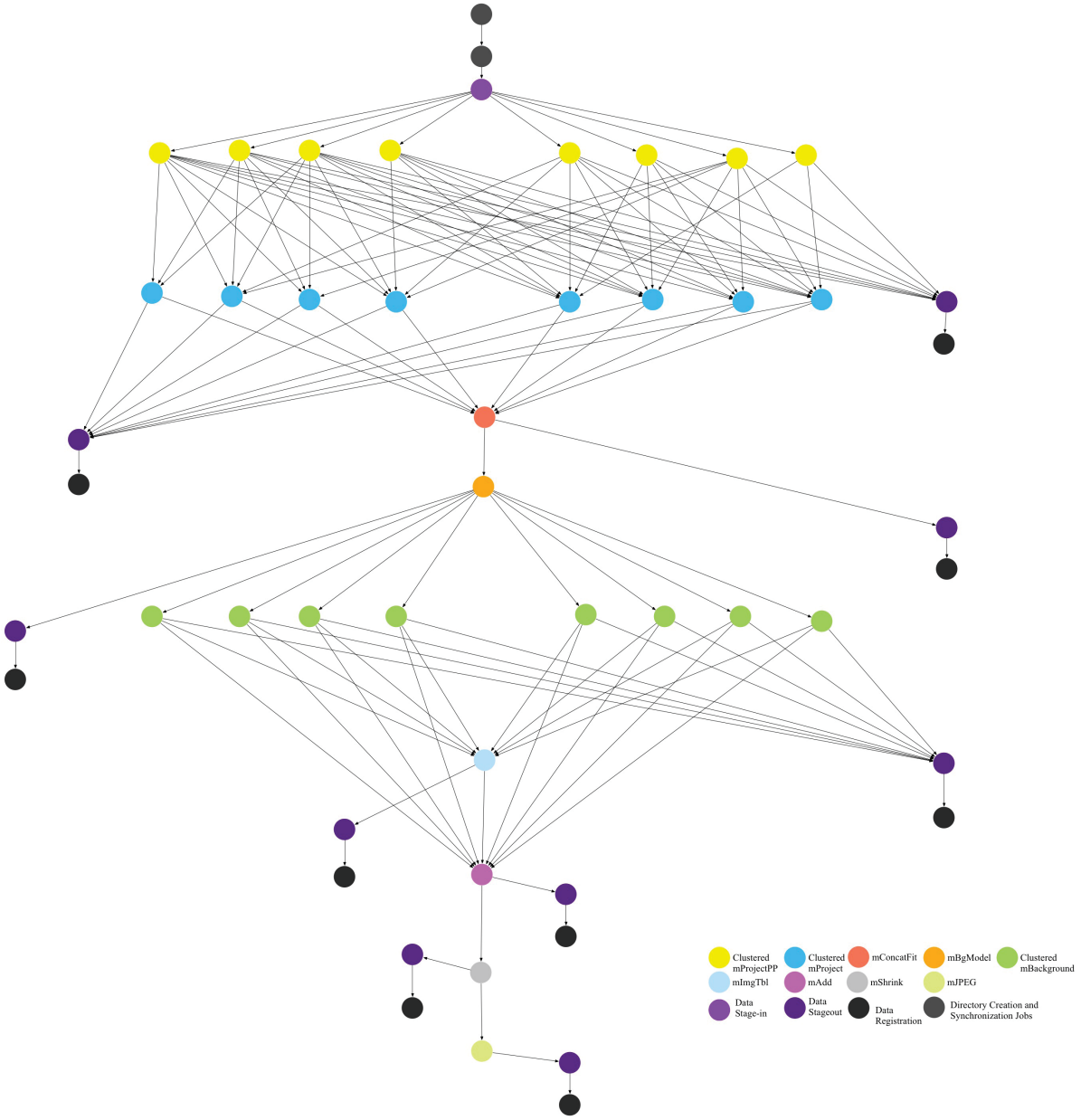



Figura 4: Workflow ejecutable generado a partir de un workflow abstracto con cuatro imágenes de entrada.

Cuando empezamos a trabajar con un mayor número de imágenes, como hemos comentado, la estructura del workflow ejecutable varía. La Figura 3 muestra una instantánea de un workflow de Montage que consta de 1200 trabajos ejecutables y que corresponde a un conjunto de datos distinto.

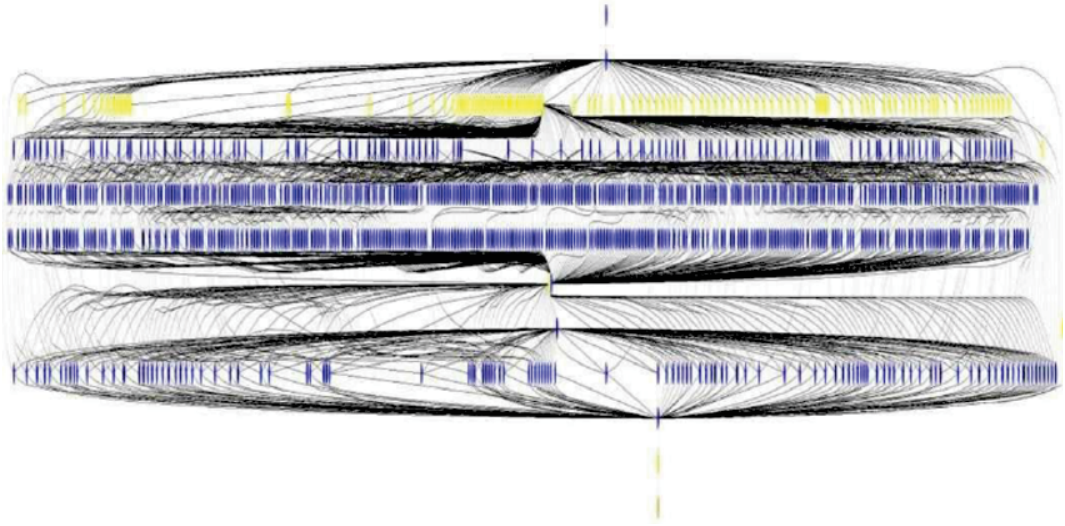


Figura 5: Flujo de trabajo de Montage que consta de 1200 trabajos ejecutables.

Como veremos en las siguientes secciones, la estructura de nuestro workflow, construida a partir de distintos patrones, no varía en función del número de imágenes a tratar.

### 3.2. Representación de Patrones

Algunas de las etapas de Montage que hemos comentado, y como se puede ver en la Figura 3, se realizan de forma secuencial como mImgtbl, mOverlaps, mBgModel, mAdd y mJPEG, para ello hemos creado la red SeqStep que lanzará un único job para la operación como podemos ver en la Figura 6. Se utiliza para las operaciones secuenciales, crea un JobSubmission (red que podemos ver en el anexo) que lanzará un job a Condor y cuando termina devuelve el resultado.

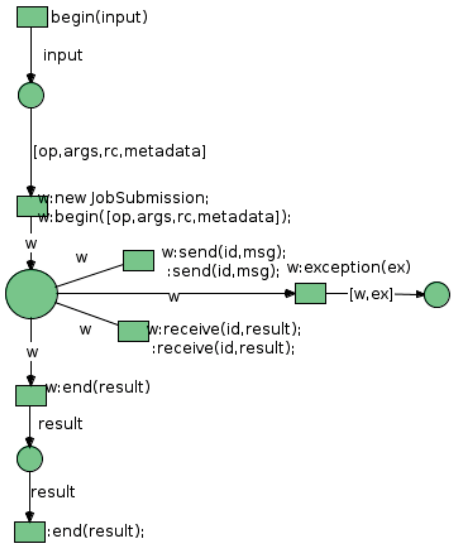



Figura 6: Red de Renew SeqStep utilizada para lanzar jobs simples.

Otras como mProjectPP, mDiffFit y mBackground se pueden realizar en paralelo, por lo que los patrones de workflow empleados serán diferentes, en este caso se utilizará EndAllwithReasoner que vemos en la Figura 20, a la que se le pasa un MontageReasoner para el tratamiento de errores. Esta red crea un JobSubmission que, en este caso, se dispara varias veces, según el tamaño de los datos que le pasamos, y lanzará varios jobs que realizarán las operaciones en pararelo. Esperará a que terminen e irá añadiendo el resultado en un ArrayList para su posterior uso.

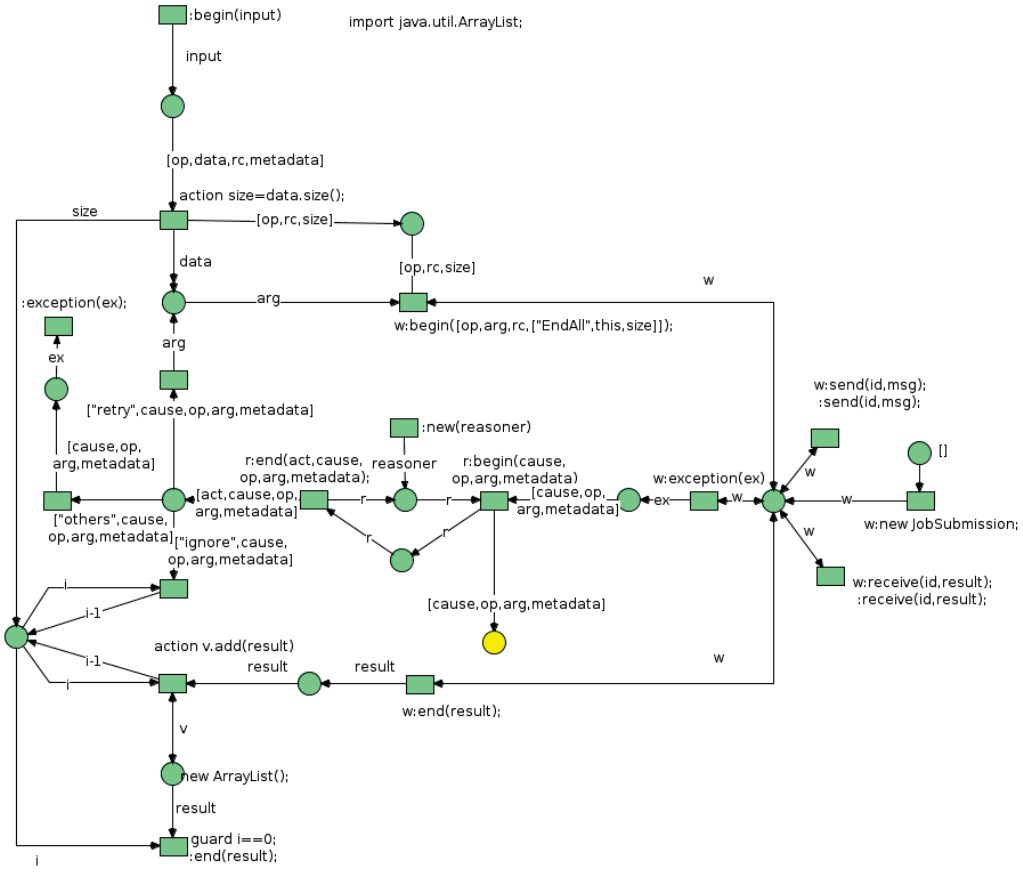



Figura 7: Red de Renew EndAllwithReasoner utilizada para lanzar jobs compuestos que lanzan varios jobs en paralelo.

También disponemos de la variante EndAllwithDemmonds, creada para la operación mDiffFit, ya que para preparar los datos necesitaremos ejecutar varias acciones de nuestra clase Montage, antes y después. Así pues, esta red, se diferencia en que lanzará varios JobSubmissionwithDemmonds a los que se les pasa dos workflows para ejecutar, antes y después de la operación.

Dentro de la red JobSubmissionWithDemmonds, que podemos ver completa en el anexo, comentaremos algo más en detalle la parte de :new(b,a) que podemos ver en la Figura 8. Al dispararse desde EndAllWithDemmonds *w:new JobSubmissionWithDemmonds(before,after)* se crearán ambas redes. Como para los dos casos el funcionamiento es el mismo, la diferencia está en el punto de la red en que se comunica con begin y end, comentaremos el ejemplo con b. Así pues, en el caso en el que la red que le pasamos no sea vacía (*guard !b.equals("")*), se creará mediante *action net=de.renew.net.Net.forName(b); action wB=net.buildInstance()*, se lanzará con *this:bBegin(args,i)* y terminará, una vez realizada la acción en la red correspondiente, con

*this:bEnd(out,i).*

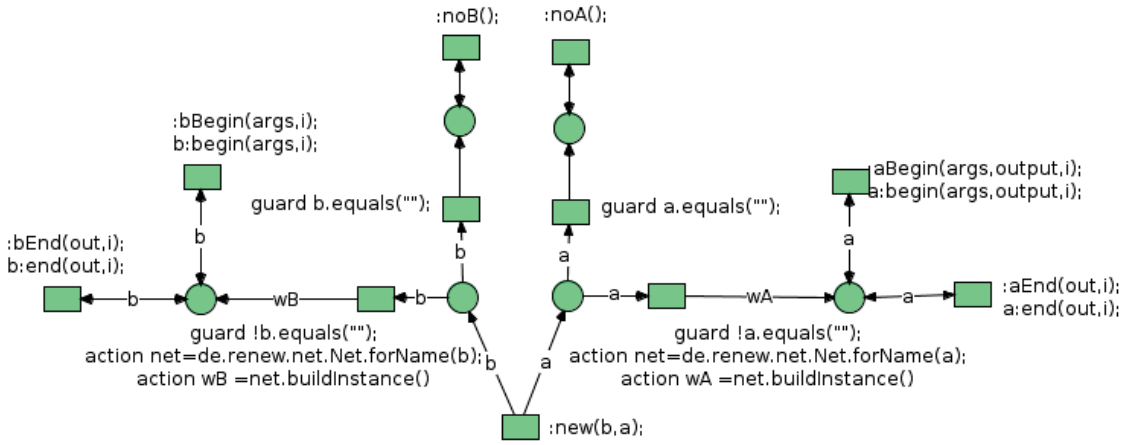



Figura 8: Parte de JobSubmissionWithDemmonds en la que se crean las redes mDiffFitBefore y mDiffFitAfter.

### 3.3. Modelado de Montage utilizando Patrones

La utilización de las Reference nets nos permite mantener el pipeline original de Montage, una secuencia de etapas, al mismo tiempo que internamente en cada etapa modelar distintos modos de ejecución –secuencial o paralelo–, así como distintos tipos de sincronización.

Hemos desarrollado mediante la herramienta Renew un conjunto de patrones que pueden utilizarse para especificar Montage, el cual podemos ver completo en el anexo. En concreto, se basa en la especificación de Montage que puede encontrarse en el tutorial: m101 Mosaic, disponible en la web [16].

En el disparo de comienzo del workflow se recibirán como entrada los datos necesarios para la ejecución del trabajo, como son el directorio donde se encuentra la carpeta con los archivos del ejemplo:

```
m101/
  |--rawdir/
  |   |--10 imágenes 2MASS Atlas, SIN proyección
  |--projdir/
  |--diffdir/
  |--corrdir/
  |--final/
  |--template.hdr
```

rawdir, que contiene las imágenes 2MASS originales, así como otros directorios de Montage donde se irán guardando los resultados, la librería donde encontraremos los ejecutables de Montage y el PATH actualizado con los ejecutables y el archivo de configuración de Condor.

Para cada una de las operaciones encapsuladas, como vemos en la Figura 9, introducimos los argumentos necesarios para la operación, obtenidos del canal *Montage*, se definen los canales *send*, *receive* y *exception* para la comunicación con la red de Engine y damos la opción de visualizar la salida del mensaje correspondiente al resultado. Como vemos, *m* será el mensaje y al sepa-

rar en *[a,b,c]*, c corresponderá a *result* o *output*, ya que los mensajes de resultado son del tipo *["Reply",result,metadata]* o *["Reply",output,metadata]*. De esta forma es fácil ver si las salidas de los pasos son las esperadas.

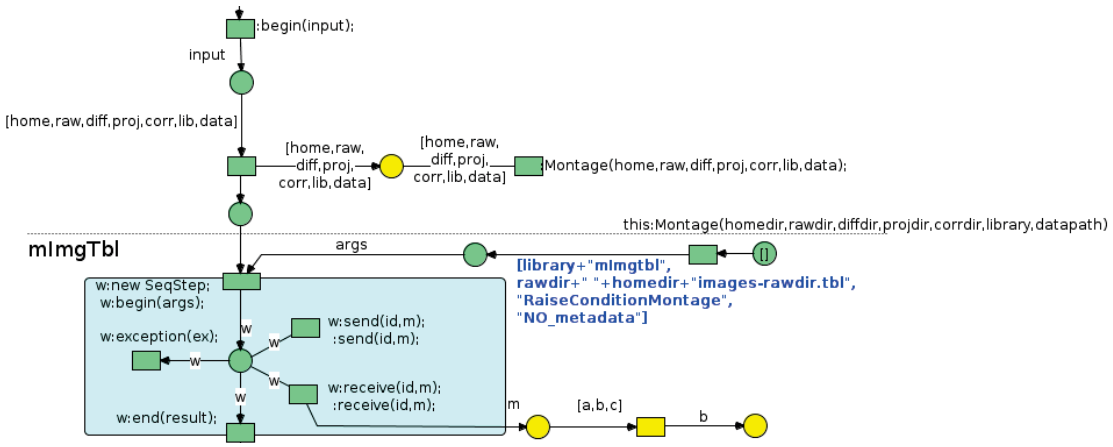



Figura 9: Comienzo de nuestro workflow de Montage en el que se puede ver dónde se recogen los datos y como se definen los canales para la comunicación.

Con mImgtbl generamos una tabla de metadatos de imagen que describe el contenido del directorio rawdir. Esta operación se realizará utilizando SeqStep.

La operación mProjectPP utiliza los metadatos de la tabla generada anteriormente y reproyecta cada imagen. Estas reproyecciones se pueden realizar en paralelo, para ello, necesitamos separar los datos y colocarlos con el formato adecuado. Hacemos "Retrieve" de la tabla, utilizando la red FileSubmission y una vez tenemos los resultados, usamos *action* con la función ls de la clase Montage que nos devolverá un ArrayList con los parámetros para mProjectPP. Para que se ejecute en paralelo utilizaremos la red EndAllWithReasoner, guardará el resultado en un ArrayList para su posterior uso y se creará un set de imágenes reproyectadas en el directorio projdir. Con las imágenes de esta carpeta generaremos una nueva tabla de metadatos, disparando una nueva operación mImgtbl con SeqStep pero con diferentes argumentos.

Montage necesita saber qué imágenes se superponen, ésto se determina usando mOverlaps. En este caso realizamos la operación también utilizando SeqStep. Esto genera el archivo diffs.tbl, que será utilizado por la operación mDiffFit para restar cada par de imágenes superpuestas, creando un conjunto de imágenes de diferencia en el subdirectorio especificado diffdir. La operación mDiffFit puede hacer el cálculo de diferencias de distintas imágenes en paralelo, para ello, utilizamos un workflow similar al comentado anteriormente EndAllWithReasoner. Sin embargo, necesitamos preparar esos datos y darles el formato necesario para lanzar la operación, por ello hemos creado EndAllWithDemmonds al que le pasaremos como argumento dos workflows (mDiffFitBefore y mDiffFitAfter), que se ejecutarán antes y después de la operación. Esto se hará lanzando varios JobSubmissionWithDemmonds(before,after), dependiendo del número de datos que tengamos a tratar, según el contenido de diffs.tbl. La red JobSubmissionWithDemmonds, se distinguirá de JobSubmission en que no dispara la operación, en este caso mDiffFit, a través de los canales send y received que se comunican con Engine, hasta que no haya terminado mDiffFitBefore. Esta red ejecuta la acción de Montage mDiffArgs para preparar los parámetros necesarios para la operación. Al finalizar la operación, lanzará mDiffFitAfter, que ejecuta la acción Montage.mFitLine(Montage.mFitFiles(args),result) y prepara en la salida el formato correcto para después con la función mConcatFit del resultado, formar la

estructura de fits.tbl, tabla de salida que contiene coeficientes de ajuste de plano y que enviaremos al servidor con la operación "Send" de FileSubmission.

Ahora que Montage ha calculado la mejor manera de suavizar las regiones de superposición usando las imágenes de diferencia, debe aplicar la eliminación de fondo a las imágenes reproyectadas originales. El primer paso es crear una tabla de correcciones que deben aplicarse a cada imagen usando mBgModel. Se realizará utilizando SeqStep y generará la tabla de correcciones globales corrections.tbl.

Después, y tras recuperar los datos de image.tbl y corrections.tbl y darles el formato que necesitamos, lanzaremos la operación mBackground utilizando EndAllWithReasoner, de esta forma se aplicará realmente la coincidencia de fondo a cada imagen reproyectada, y se creará un conjunto de imágenes reproyectadas con coincidencias de fondo en el directorio corrdir.

Utilizaremos mAdd para unir todas las imágenes reproyectadas y suavizadas en un mosaico final. Y mJPEG para crear un JPEG en escala de grises de la salida. Ambas mediante SeqStep. Por último, haremos una copia de la imagen resultante en local y la abriremos, mostrándola por pantalla, con la función openJPEG creada en Montage.

En el workflow, las operaciones se irán disparando a medida que los recursos necesarios estén disponibles. En la mayoría de los casos, estos datos son generados en la operación anterior, sin embargo, una vez que existen esos datos podemos relanzar las operaciones que nos interesen, poniendo una transición desde un token negro simple, que disparará la transición que le indiquemos, y retirando la dependencia de la anterior, además pondremos un token en blanco para que no se disparen las operaciones que no queramos realizar.

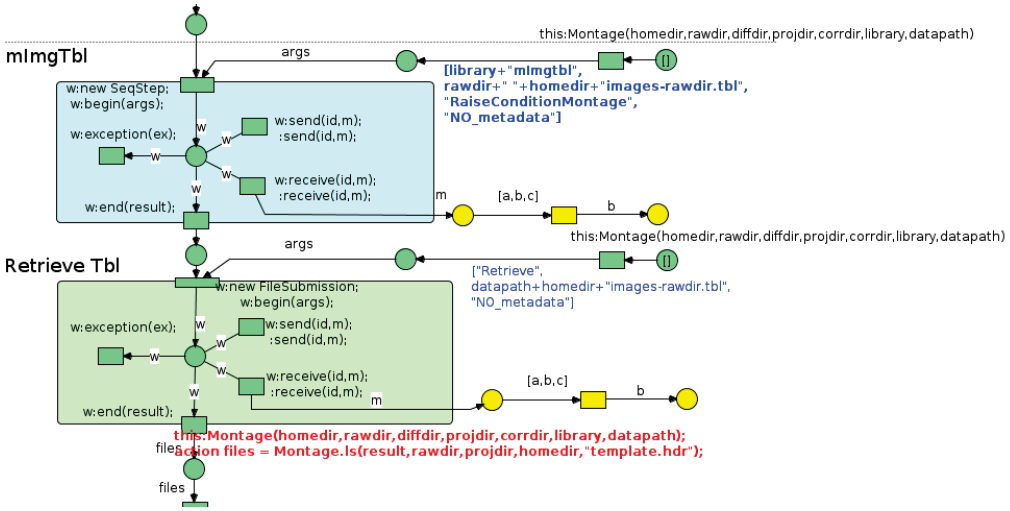
Por ejemplo, una vez tengamos los datos generados por la primera operación, podemos evitar repetirla en las siguientes pruebas haciendo la modificación como hemos comentado, y que vemos en la Figura 10.

### 3.4. Tratamiento de Excepciones

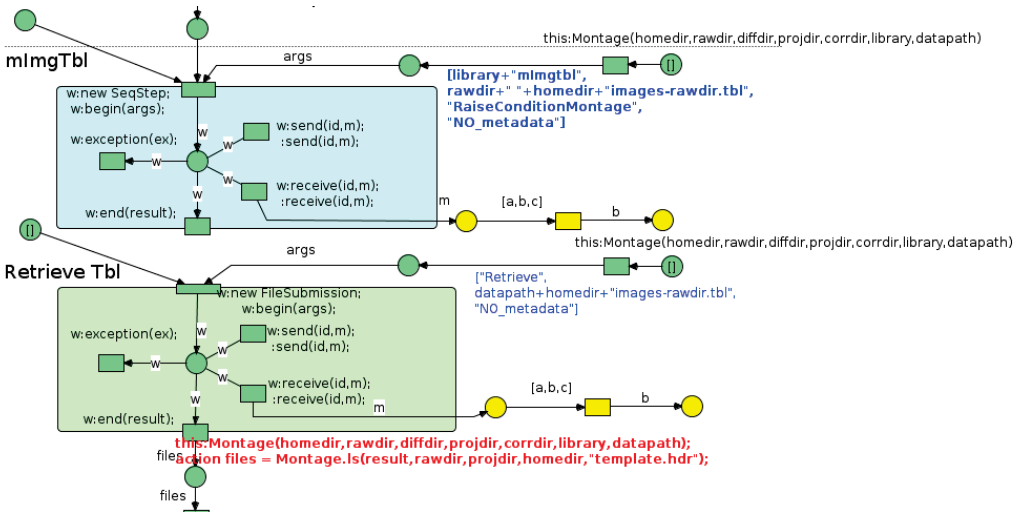
Una excepción puede verse como un evento inusual que se detecta bien por software o por hardware y que requiere de cierto tratamiento especial. En este proyecto, consideramos un modelo de excepciones habitual en el que una avería física o bien algún evento detectado por software generan un fallo y dicho fallo produce un evento excepcional en el sistema. De esta manera, el evento hace que una acción o un conjunto de acciones tengan que ejecutarse. Nuestros mecanismos de detección de fallos son los proporcionados por la terminación del código de Montage y el sistema de detección de fallos de Condor.

Todas los patrones desarrollados cuentan con los canales síncronos *receive* y *send* que sirven para comunicar mensajes entre los componentes de la arquitectura. Sin embargo, los eventos excepcionales, se tratan mediante el canal *exception*. Cuando en una red aparece *w:exception(ex)*, se está capturando un evento exceptional de un componente interno (*w* en ese caso). Con ese evento, los patrones podrían desarrollar cualquier acción. Se proponen las siguientes: (i) ignorar la excepción si es simplemente un warning, (ii) re-intentar la ejecución, si el problema es severo, (iii) buscar un componente alternativo, si es que existe, o bien (iv) solicitar intervención humana.

La red de Montage de la Figura 11 es la que, utilizando las funciones de la clase MontageMsg,




(a) Realizando la operación mImgTbl




(b) Saltando la operación mImgTbl

Figura 10: Cambios necesarios en caso de querer saltarnos una operación

analiza las excepciones y las comunica a los niveles superiores.

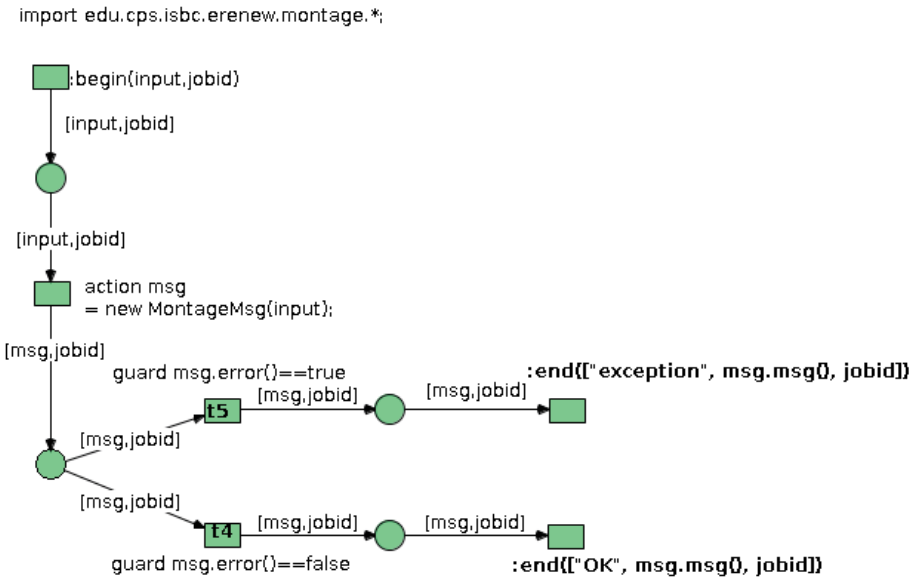



Figura 11: Red RaiseConditionMontage en Renew para el tratamiento de excepciones.

# 4. Diseño del Workflow Engine y Ejecución de Montage

El workflow engine desarrollado está inspirado en DVega [20, 21], cuyo modelo se muestra en la Figura 12. La red objeto Montage se ejecutará en el worflow engine (en la izquierda). El workflow engine interpretará la red y sus tareas las enviará a recursos computacionales de un clúster Condor. Para permitir una mayor flexibilidad en el sistema y posibilitar cualquier tipo de infraestructura distribuida en el futuro, se ha utilizado un componente arquitectural, basado en Linda, para comunicar el workflow engine con la infraestructura. Dicho componente actúa como *message broker* y, por simplicidad, es una versión reducida de Linda.

Los tres componentes arquitecturales, Workflow Engine, Tuple Space y Resources Manager se comunican mediante los canales síncronos denominados *receive* y *send*. En Engine, *w* es definido como *w:new Montage_ DVega7withReasonerJPEG*, lo que hará que cuando se dispare *w:receive(id,mensaje)* de la parte de Workflow, se dispare *:receive(id,m)* en nuestro Workflow de Montage. También de forma similar, *res* está definido como *res: new condorJobAndCnMngt*, lo que hace que cuando se dispare *res:receive(id,mensaje)* lo haga también *:receive(jobid,mensaje)* en la red de condorJobAndCnMngt. De este mismo modo funcionará la comunicación con send.

Además, para la comunicación entre ellos, *this:send(id,message)* se comunica con *:send(id,message)* del que denominamos espacio de tuplas. Por cada send que llega al espacio de tuplas, genera un *:receive(id,mensaje)*, con el mismo id de job y el mismo mensaje, que se sincronizará con otros puntos de nuestra red.

Para comenzar la ejecución, desde eclipse cargaremos el entorno de Renew con las redes que necesitaremos, ejecutando la clase MontageNetsLoader. Una vez en Renew, la ejecución de nuestro ejemplo comienza en la red Engine. En ella, además de la parte en la que definimos los valores

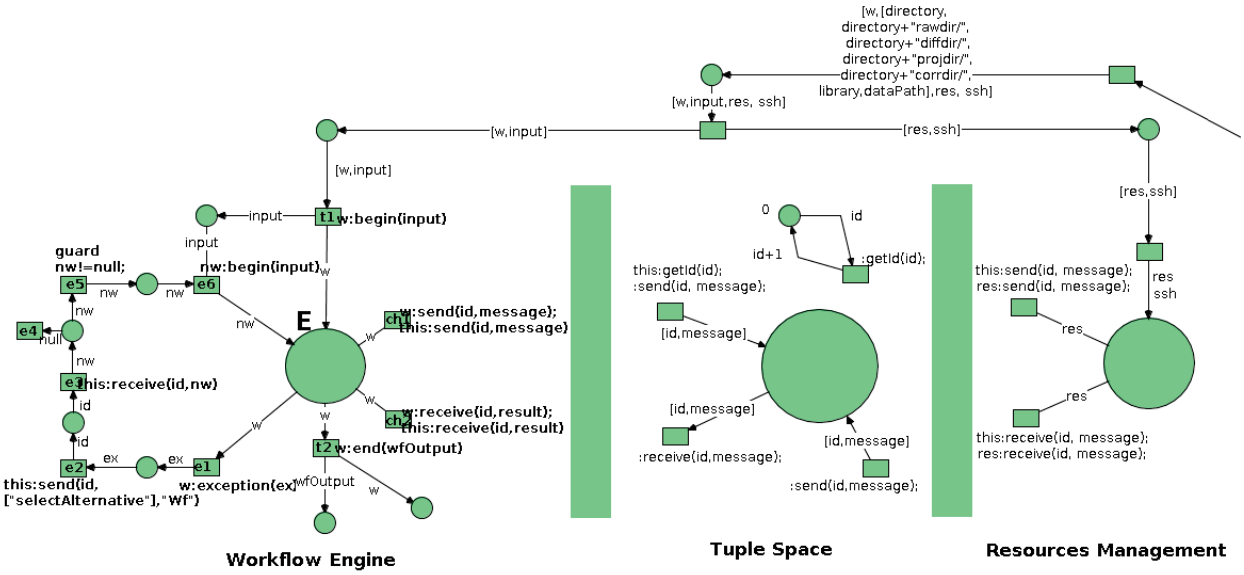



Figura 12: DVega workflow engine

de las variables necesarias para la ejecución, se distinguen 3 divisiones, como podemos ver en la Figura 12. Workflow Engine encapsula el workflow de Montage comentado anteriormente, Resources Management la parte de conexiones e interacción con Condor, que comentaremos a continuación, y Tuple Space que, basado en el modelo Linda, se encarga de la comunicación entre los otros dos.

## 4.1. Acceso al Clúster vía SSH

En Renew, para permitir ejecutar las tareas en un clúster hemos utilizado el protocolo SSH. Para ello, hemos desarrollado las redes SSH y sshJob, que se ejecutarán dentro de la red Resources manager de la Figura 12. La red sshJob, internamente, utilizará las operaciones disponibles en Java para realizar las conexiones necesarias para interactuar con el clúster de Condor.

La interfaz que se encarga de las conexiones SSH y proporciona operaciones de ejecución de comandos y copia de ficheros desde y hacia el servidor remoto está compuesta por dos clases:

- *SSHHandler* es la clase principal a instanciar y la que posee las operaciones.
- *SSHUserInfo* es una clase auxiliar utilizada para almacenar toda la información de conexión.

Como hemos comentado anteriormente, en Renew, las comunicaciones entre las distintas redes se van a realizar mediante canales síncronos. Así pues, desde Engine con *ssh:new SSH(host,user,passwd)*, se dispara *:new(host,user,passwd)* en la red SSH, Figura 13, con los datos del host, usuario y contraseña.

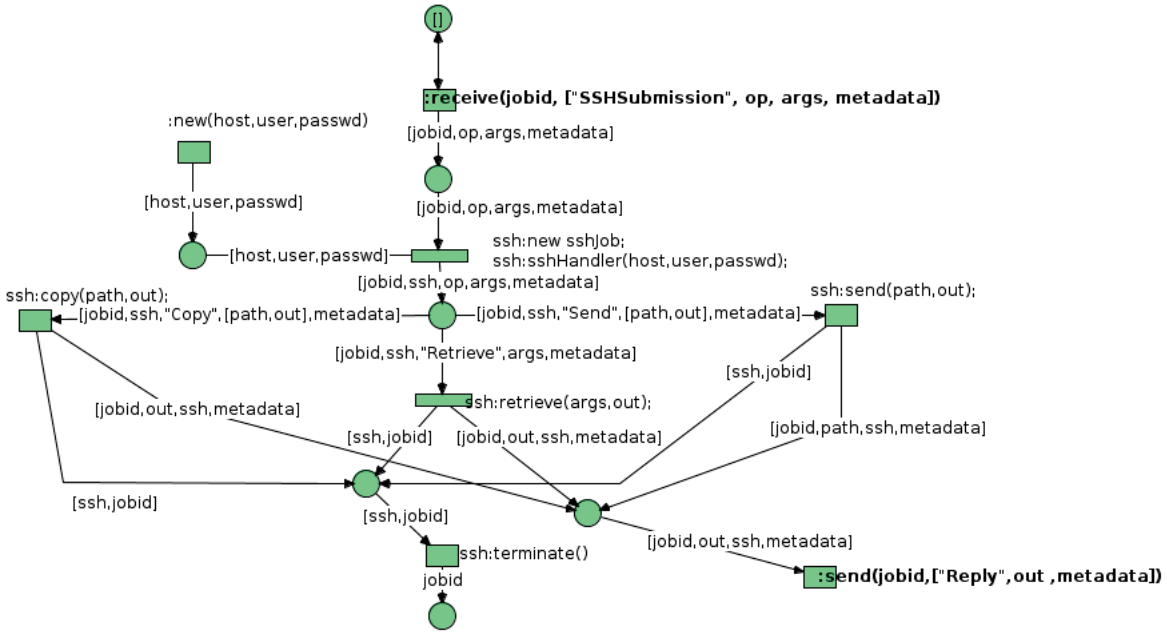



Figura 13: Red SSH en Renew.

Además, en algunos pasos de nuestro workflow, se disparará la red FileSubmission con *w:new FileSubmission; w:begin(args)*. Esta red, que podemos ver en el anexo, lanza *:send(jobid, ["SSHSubmission", op,args,metadata]*. El canal send se comunicará con el espacio de tuplas y al dispararse generará *:receive(jobid,["SSHSubmission",op,args,metadata]*, utilizado para disparar nuestra red SSH. Después, definimos ssh con *ssh:new sshJob*, para crear la referencia a la red sshJob, y se disparará al lanzar *ssh:Handler(host,user,passwd)*. Según la *op* recibida en la tupla, usamos el canal *ssh:send()*, *ssh:copy()* o *ssh:retrieve()*, para desde sshJob utilizar las funciones definidas en la clase Java SSHHandler, para enviar (send) y recibir datos del servidor como archivos (copy) o texto contenido en un archivo (retrieve), como se puede ver en la Figura 14. Para finalizar, al lanzar *ssh:terminate()*, en sshJob realizaremos la acción de desconectar y al mismo tiempo, con *:send(jobid,["Reply",out,metadata]* nos comunicamos de vuelta con FileSubmission dejando que termine.

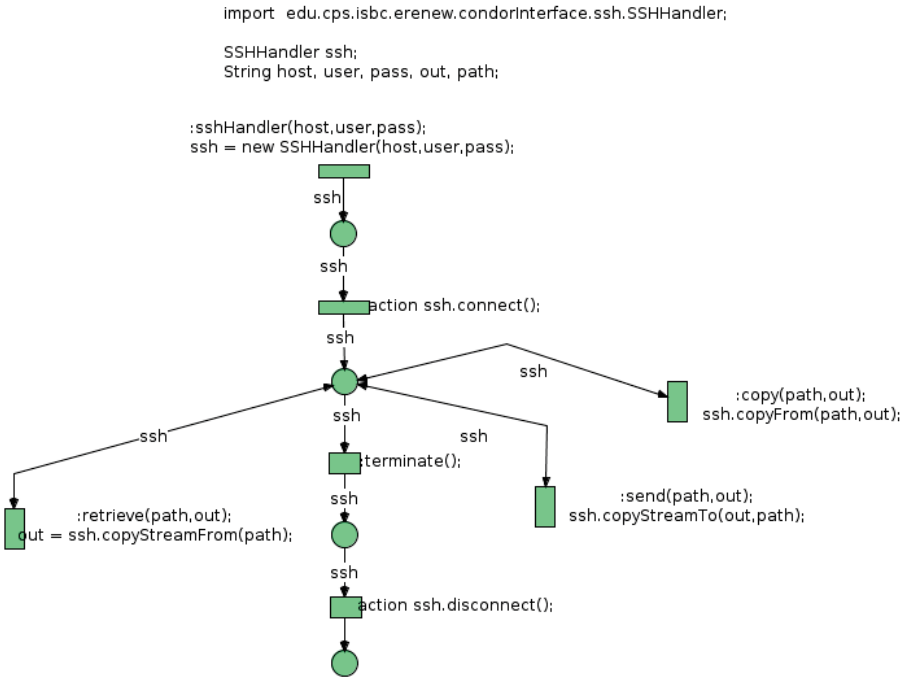



Figura 14: Red sshJob en Renew

## 4.2. Condor

Para el acceso de Condor, disponemos de cuatro clases:

- *CondorFacade* proporciona métodos para interactuar con un servidor Condor vía SSH. A través de su constructor se configura la información de conexión, tendremos que indicar el servidor Condor al que queremos conectarnos, el usuario y la contraseña, además podremos indicar nuestro directorio home remoto y el directorio donde se encuentran los ejecutables de condor. Es el encargado de enviar los trabajos y recibir la salida. Proporciona métodos de espera síncrona y asíncrona de la finalización del trabajo.

  Además de monitorizar un job, contiene otro método que comprueba si todos los jobs enviados con el mismo ID han finalizado, ya que una ejecución se puede realizar en paralelo mediante varios jobs, pero necesitaremos que todos ellos hayan terminado para tener el archivo resultante disponible para el siguiente paso. Permite también obtener el contenido de un fichero, esto nos servirá para obtener las salidas o valores de ficheros generados en las ejecuciones.

- *CondorJob* es una clase abstracta que define una interfaz común para toda implementación de CondorJob que la satisfaga. Además implementa métodos que interactúan con el CondorFacade y el CondorTerminationListener.

- *CondorJobHandler* es una implementación de CondorJob que provee métodos para configurar los parámetros básicos de un trabajo de Condor así como para poder gestionarlo. Por ejemplo con él se configura el ejecutable y los ficheros de entrada y de salida a utilizar.

- *CondorTerminationListener* es la clase hilo que utilizamos para monitorizar la terminación del trabajo enviado. Para ello se queda bloqueada leyendo el fichero de log creado por Condor

hasta detectar un registro de finalización.

En Renew, para acceder a Condor, disponemos de condorJobAndCnMngt, condorJob y McondorJob_3. Como ya hemos comentado, desde Engine definimos res como un condorJobAndCnMngt, Figura 15. En esta red se distinguen 3 partes. Al definir un nuevo condorJobAndCnMngt, se dispara la parte de arriba, en ella se genera un pool de CondorFacade conectados al clúster, que serán utilizadas en las otras dos secciones. Posteriormente, a través del espacio de tuplas, se disparará una de las otras dos secciones, dependiendo de si se trata de un job simple (abajo derecha) o un job compuesto (abajo izquierda), dependiendo del mensaje que contenga *res:receive(id,mensaje)*.

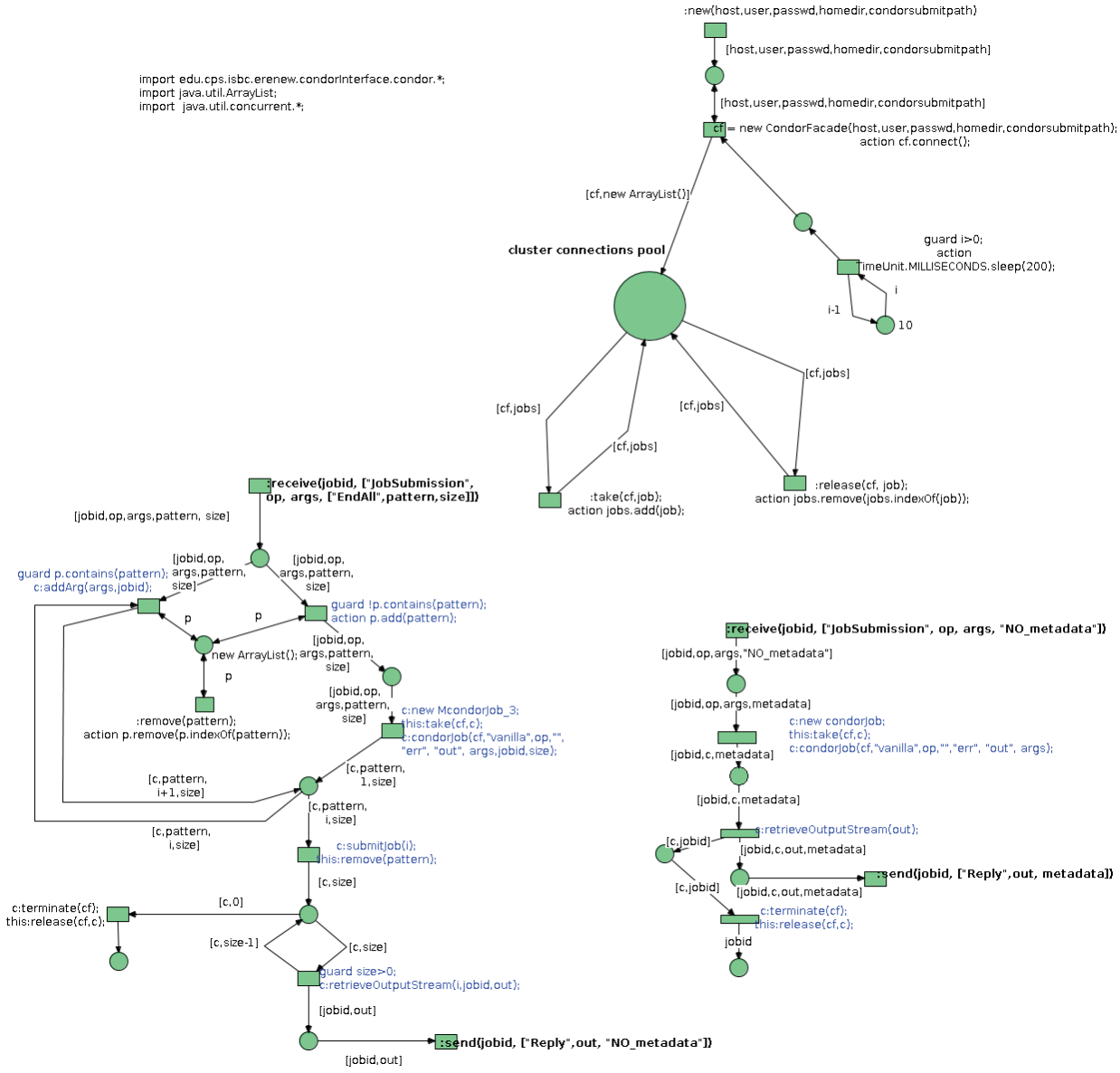



Figura 15: Red condorJobAndCnMngt en Renew

Para el caso de un job simple, se creará un condorJob y para jobs compuestos, un McondorJob_3. En condorJob crearemos un CondorJobHandler (clase Java) y utilizaremos sus funciones

para definir un entorno de trabajo e instalar el listener para que, tras lanzar el job en el clúster, seamos informados una vez que termine. Al finalizar recuperaremos la salida del job, el contenido de la salida y el contenido del fichero obtenido, para usar o mostrar según nos pueda interesar. Una vez finalizada la ejecución, se hace la limpieza del entorno de trabajo creado y se desconecta del servidor. McondorJob_3 es similar a condorJob aunque está preparado para jobs que lanzan varios jobs en paralelo, por lo que en este caso se le indicará el tamaño y esperará a que todos estén completados para finalizar.

En el anexo encontraremos las redes de condorJob y McondorJob_3, además de los diagramas de clase y de secuencia, en los que se puede ver la interacción entre las clases comentadas.

# 5. Conclusiones

Durante los últimos años, los workflows científicos han surgido como una tecnología que proporciona soporte computacional en la realización de experimentos científicos. En este proyecto se ha desarrollado una arquitectura orientada a servicio que permite desacoplar las especificaciones de los workflows científicos de los entornos de ejecución, y permite definir políticas de gestión de recursos y de recuperación de fallos a nivel de aplicación. La especificación de workflows se realiza mediante Redes de Petri y Renew, y como entorno de ejecución se emplea clúster Condor.

Con este proyecto hemos conseguido una especificación de workflow, basada en patrones, que hace que su estructura no sea dependiente de los datos de entrada, lo cual permite tener mayor control sobre el mismo, para realizar pruebas y visualizar resultados. Se ha podido desarrollar en Renew un workflow con cierto nivel de recuperación de fallos y tratamiento de excepciones. En particular, se ha desarrollado en el lenguaje de especificación de los workflows, un mecanismo de gestión de excepciones que permite tratar los fallos de aplicación y realizar distintas acciones, tales como ignorar el fallo, reintentar, intentar con otro componente alternativo o solicitar la intervención humana. Con todas estas características nuestros workflows son más independientes del entorno de ejecución.

Para líneas futuras, sería interesante hacer pruebas en otros entornos de trabajo, como infraestructuras Cloud y Kubernetes, ya que las pruebas sólo se han realizado en un clúster Condor. También se podría trabajar con un mayor número de imágenes iniciales, ya que no se ha llegado a comprobar el correcto funcionamiento de nuestro workflow con un gran número de procesos.

# Bibliografía


[1] Ewa Deelman, Karan Vahi, Mats Rynge, Gideon Juve, Rajiv Mayani, and Rafael Ferreira da Silva. Pegasus in the cloud: Science automation through workflow technologies. University of Southern California. IEEE Internet Computing.

[2] David Hollingsworth. *The workflow reference model*. Technical Report TC00-1003, The Workflow Management Coalition, Hampshire, UK, January 1995.

[3] Bertram Ludaescher, Shawn Bowers, Timothy McPhillips, Norbert Podhorszki, and Norbert Podhorszki. Scientic workflows: More e-science mileage from cyberinfrastructure. *In E-SCIENCE '06: Proceedings of the Second IEEE International Conference on e-Science and Grid Computing, page 145, Washington, DC, USA, 2006. IEEE Computer Society.*

[4] Zhiming Zhao, Adam Belloum, Hakan Yakali, Peter Sloot, and Bob Hertzberger. Dynamic workflow in a grid enabled problem solving environment. *In CIT '05: Proceedings of the The Fifth International Conference on Computer and Information Technology, pages 39.345,Washington, DC, USA, 2005. IEEE Computer Society.*

[5] High-throughput computing. `https://en.wikipedia.org/wiki/High-throughput_computing`.

[6] HTCondor. `https://en.wikipedia.org/wiki/HTCondor`.

[7] Computing with HTCondor. `https://research.cs.wisc.edu/htcondor/index.html`.

[8] Murata T. Petri nets: Properties, analysis and applications. *Proceedings of IEEE 1989.*

[9] Van der Aalst W. and Van Hee K. *Workflow Management: Models, Methods, and Systems.* MIT Press: Cambridge, MA, U.S.A., 2004.

[10] Valk R. Petri nets as token objects: An introduction to elementary object nets. *Nineteenth International Conference on Application and Theory of Petri Nets (Lecture Notes in Computer Science, vol. 1420), Lisbon, Portugal, 1998.*

[11] Russell S and Norvig P. *Artificial Intelligence: A Modern Approach (2nd edn).* Prentice-Hall: Englewood Cliffs, NJ, 2003.

[12] Kummer O., Wienberg F., Duvigneau M., Schumacher J., Köhler M., Moldt D., Rölke H., and Valk R. An extensible editor and simulation engine for petri nets: Renew. *Applications and Theory of Petri Nets 2004, 25th International Conference,ICATPN'04, 2004.*

[13] Olaf Kummer, Frank Wienberg, and Michael Duvigneau. Renew – The Reference Net Workshop. `http://www.renew.de`.

[14] Montage Image Mosaic Software, `https://github.com/Caltech-IPAC/Montage/wiki`.

[15] G. B. Berriman, Ewa Deelman, John C. Good, Joseph C. Jacob, Daniel S. Katz, Carl Kesselman, Anastasia C. Laity, Thomas A. Prince, Gurmeet Singh, and Mei-Hu Su. Montage: a grid-enabled engine for delivering custom science-grade mosaics on demand. *Optimizing Scientific Return for Astronomy through Information Technologies. Edited by Quinn, Peter J.; Bridger, Alan. Proceedings of the SPIE, Volume 5493, pp. 221-232 (2004).*

[16] The Montage project web page, `http://montage.ipac.caltech.edu/`.

[17] The Flexible Image Transport System (FITS), http://fits.gsfc.nasa.gov, http://www.cv.nrao.edu/fits.

[18] E.W. Greisen and M. Calabretta. Representation of Celestial Coordinates In FITS, http://www.atnf.csiro.au/people/mcalabre/WCS.htm.

[19] Ewa Deelman, Karan Vahi, Gideon Juve, Mats Rynge, Scott Callaghan, Philip J Maechling, Rajiv Mayani, Weiwei Chen, Rafael Ferreira da Silva, Miron Livny, et al. Pegasus, a workflow management system for science automation. *Future Generation Computer Systems*, 46:17–35, 2015.

[20] Rafael Tolosana-Calasanz, José A. Bañares, Pedro Álvarez, and Joaquin Ezpeleta. Vega: A service-oriented grid workflow management system. In *On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS, OTM Confederated International Conferences CoopIS, DOA, ODBASE, GADA, and IS 2007, Vilamoura, Portugal, November 25-30, 2007, Proceedings, Part II*, pages 1516–1523, 2007.

[21] Rafael Tolosana-Calasanz, José Ángel Bañares, Pedro Álvarez, Joaquín Ezpeleta, and Omer F. Rana. An uncoordinated asynchronous checkpointing model for hierarchical scientific workflows. *J. Comput. Syst. Sci.*, 76(6):403–415, 2010.

[22] What is Cloud Computing? Amazon Web Services. https://aws.amazon.com/what-is-cloud-computing/.

[23] Cloud computing. https://en.wikipedia.org/wiki/Cloud_computing. http://www.investopedia.com/terms/c/cloud-computing.asp.

[24] MAAS documentation. https://docs.ubuntu.com/maas/2.1/en/.

[25] Introducing Metal as a Service: provisioning for the hyperscale era. http://www.markshuttleworth.com/.

[26] Containerization (container-based virtualization). http://searchservervirtualization.techtarget.com/definition/container-based-virtualization-operating-system-level-virtualizati

# Anexos

## A. Anexo 1: Redes de Renew

Aquí veremos las redes desarrolladas en Renew con una descripción de cada una de ellas.

### A.1. Engine

Engine es nuestro workflow inicial. Recibe los datos con los que trabajaremos, como el nombre del servidor, el usuario y la contraseña para realizar la conexión; las carpetas en las que encontraremos los datos con los que trabajar, así como la ruta en la que se encuentran los ejecutables de Condor y Montage. Como hemos comentado anteriormente se distinguen 3 divisiones: "Workflow Engine" encapsula el workflow de Montage, "Resources Management" la parte de conexiones e interacción con Condor, y "Tuple Space" que, basado en el modelo Linda, se encarga de la comunicación entre los otros dos.

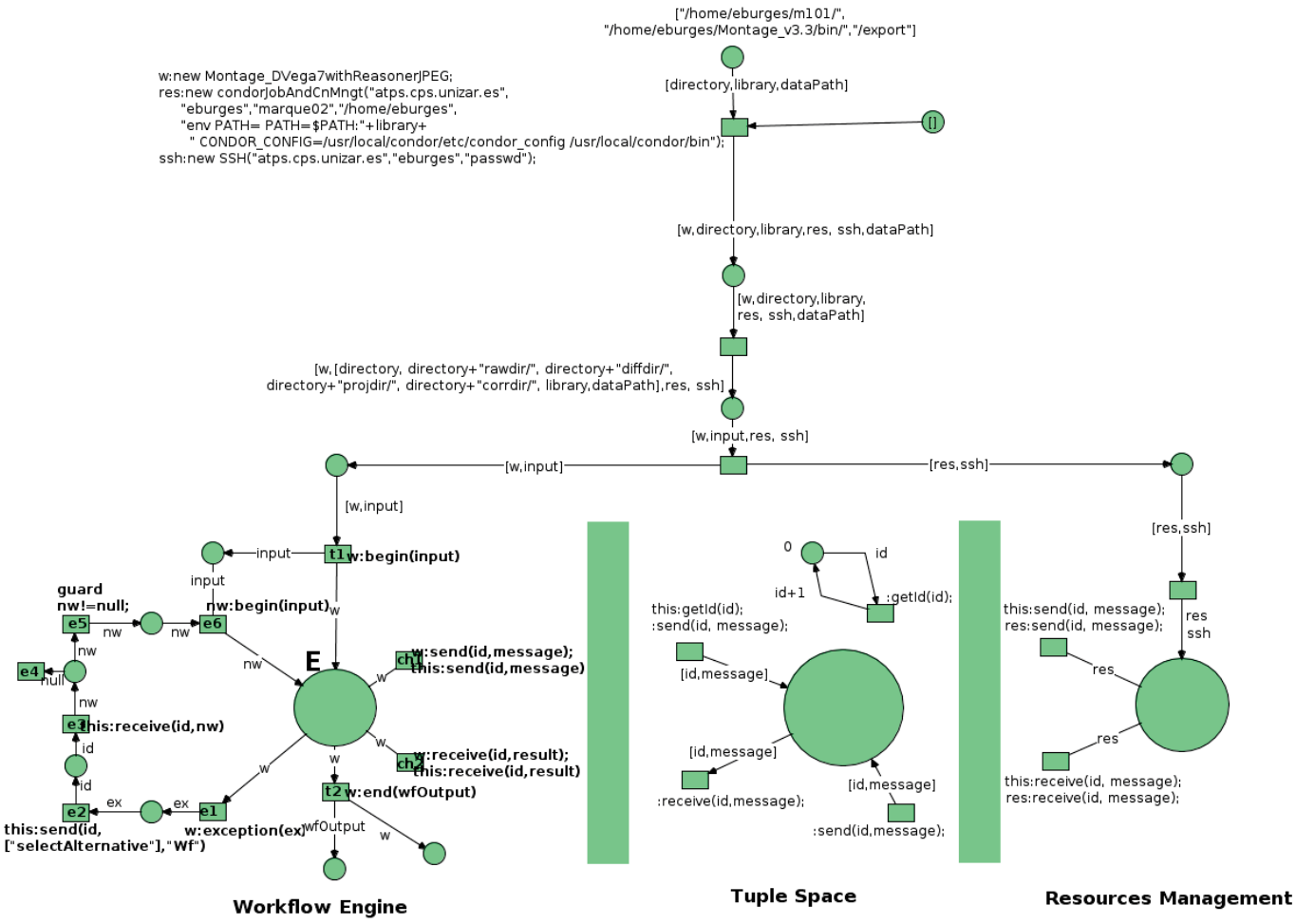


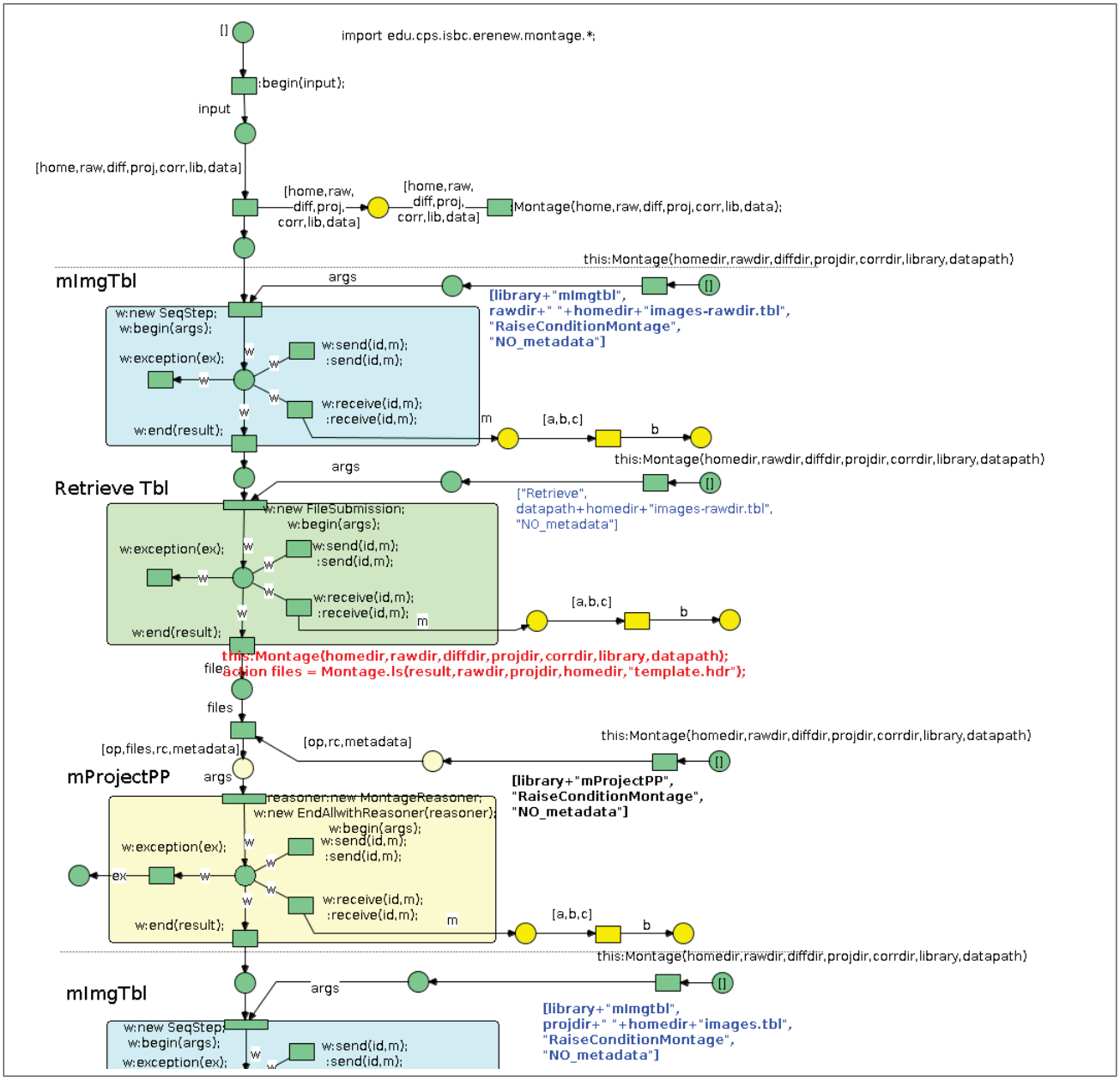

Figura 16: Engine

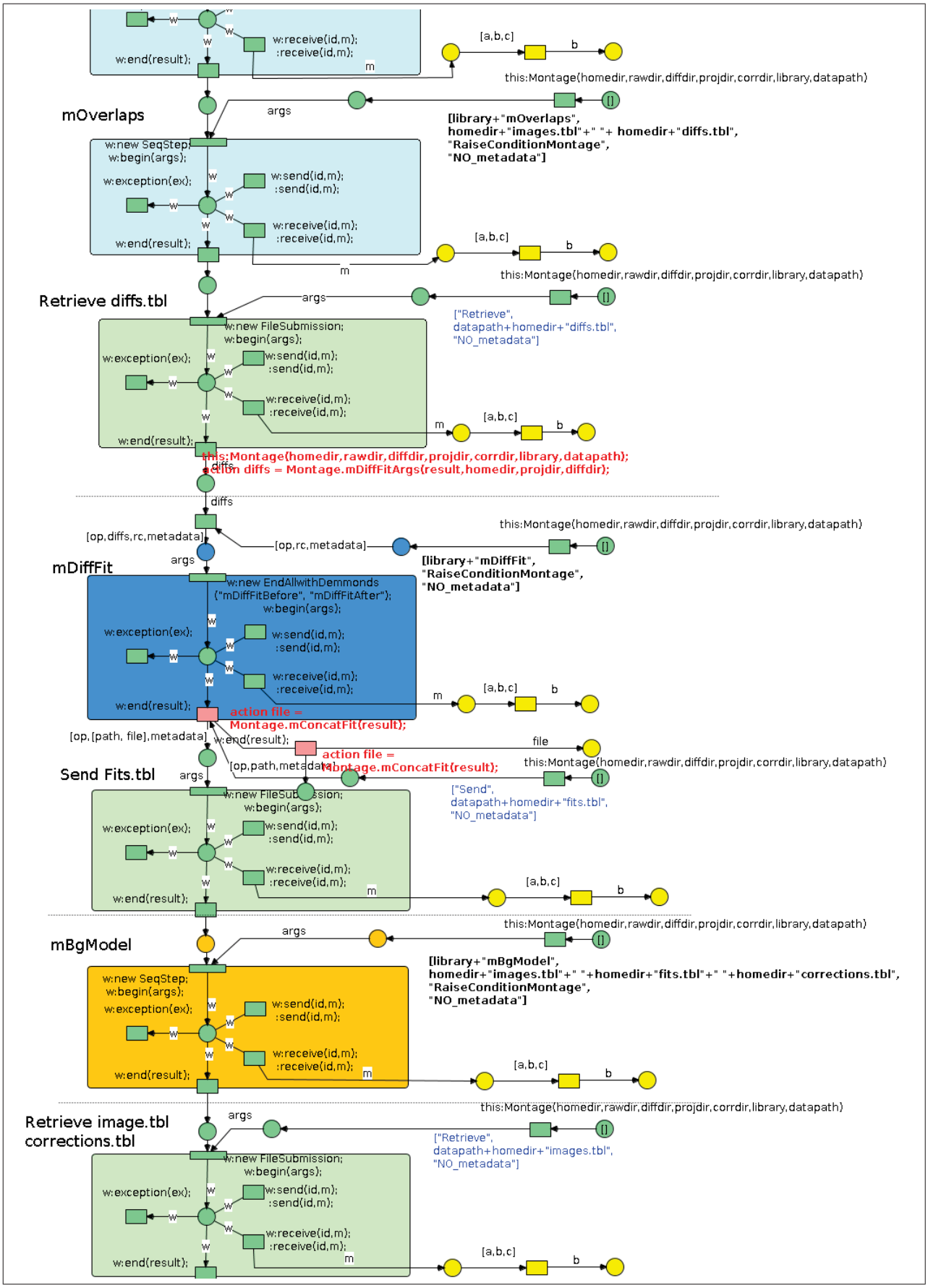
## A.2. Patrones Montage

Patrones relacionados con el workflow del ejemplo Montage.

### A.2.1. Montage_DVega7WithReasonerJPEG

Se trata del workflow principal de nuestro ejemplo, en él se encuentran encapsuladas las distintas operaciones de Montage del ejemplo desarrollado. En cada uno de los pasos se crea la red necesaria para la ejecución de la operación. Además, utiliza operaciones de la clase Montage de Java. También se mostrará, para cada uno de los pasos, la salida del job o el contenido del fichero obtenido para facilitar la visualización y la comprobación de cada paso.




w:end{result};
w:receive{id,m};
:receive{id,m};
m
[a,b,c]
b
this:Montage{homedir,rawdir,diffdir,projdir,corrdir,library,datapath}
[]
mOverlaps
args
[library+"mOverlaps",
homedir+"images.tbl"+" "+ homedir+"diffs.tbl",
"RaiseConditionMontage",
"NO_metadata"]
w:new SeqStep;
w:begin{args};
w:exception{ex};
w:send{id,m};
:send{id,m};
Retrieve diffs.tbl
["Retrieve",
datapath+homedir+"diffs.tbl",
"NO_metadata"]
w:new FileSubmission;
w:begin{args};
this:Montage{homedir,rawdir,diffdir,projdir,corrdir,library,datapath};
action diffs = Montage.mDiffFitArgs{result,homedir,projdir,diffdir};
diffs
[op,diffs,rc,metadata]
[op,rc,metadata]
mDiffFit
[library+"mDiffFit",
"RaiseConditionMontage",
"NO_metadata"]
w:new EndAllwithDemmonds
("mDiffFitBefore", "mDiffFitAfter");
w:begin{args};
action file =
Montage.mConcatFit{result};
[op,[path, file],metadata]
file
Send Fits.tbl
["Send",
datapath+homedir+"fits.tbl",
"NO_metadata"]
mBgModel
[library+"mBgModel",
homedir+"images.tbl"+" "+homedir+"fits.tbl"+" "+homedir+"corrections.tbl",
"RaiseConditionMontage",
"NO_metadata"]
Retrieve image.tbl
corrections.tbl
["Retrieve",
datapath+homedir+"images.tbl",
"NO_metadata"]

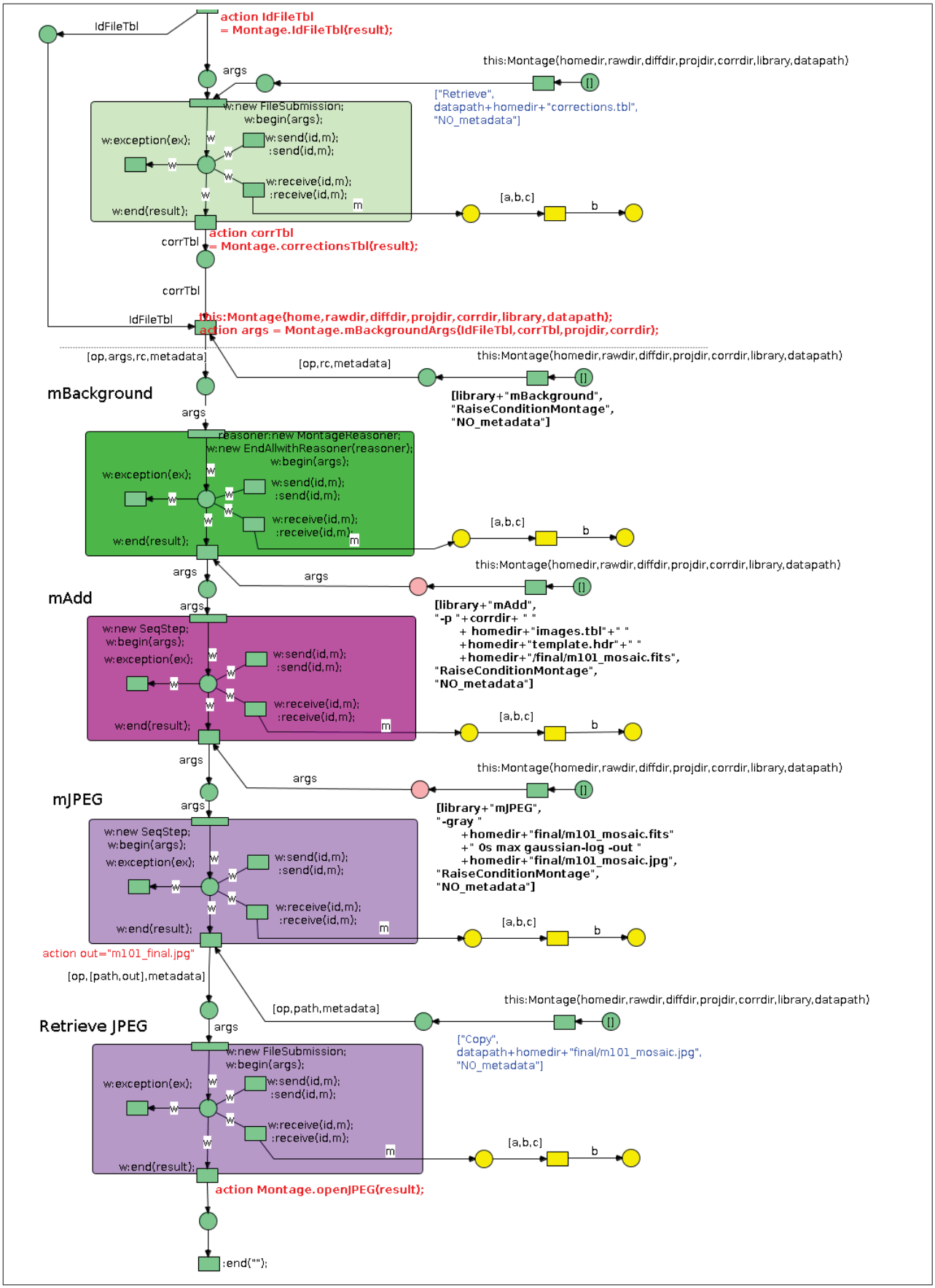



Figura 17: Montage_DVega7withReasonerJPEG

### A.2.2. SeqStep

SeqStep se utiliza para las operaciones secuenciales, crea un JobSubmission que lanzará un job a Condor y cuando termina devuelve el resultado.

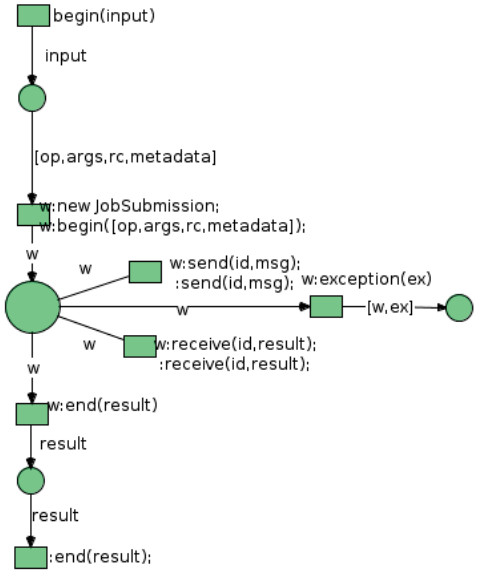



Figura 18: SeqStep

### A.2.3. EndAll

Red que crea, según el tamaño de los datos que le pasamos, un JobSubmission que lanzará a su vez varios jobs que realizarán operaciones en pararelo. Esperará a que terminen e irá añadiendo el resultado en un ArrayList para su posterior uso.

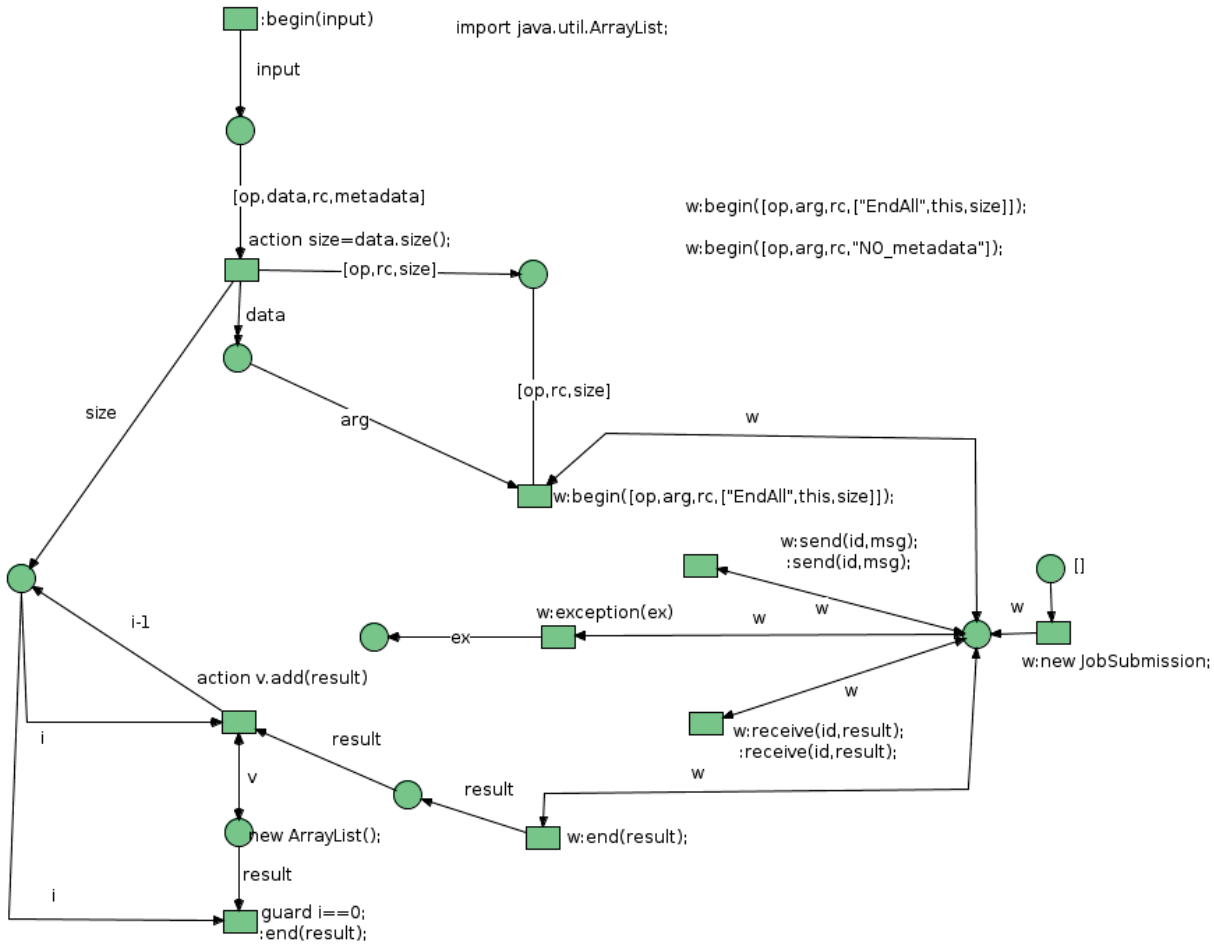



Figura 19: EndAll

### A.2.4. EndAllwithReasoner

Similar a EndAll pero con un tratamiento de errores incorporado, por lo que es la que utilizamos ahora. Nos permite definir según el tipo de excepción si queremos ignorarla y continuar, reintentar o fallar.

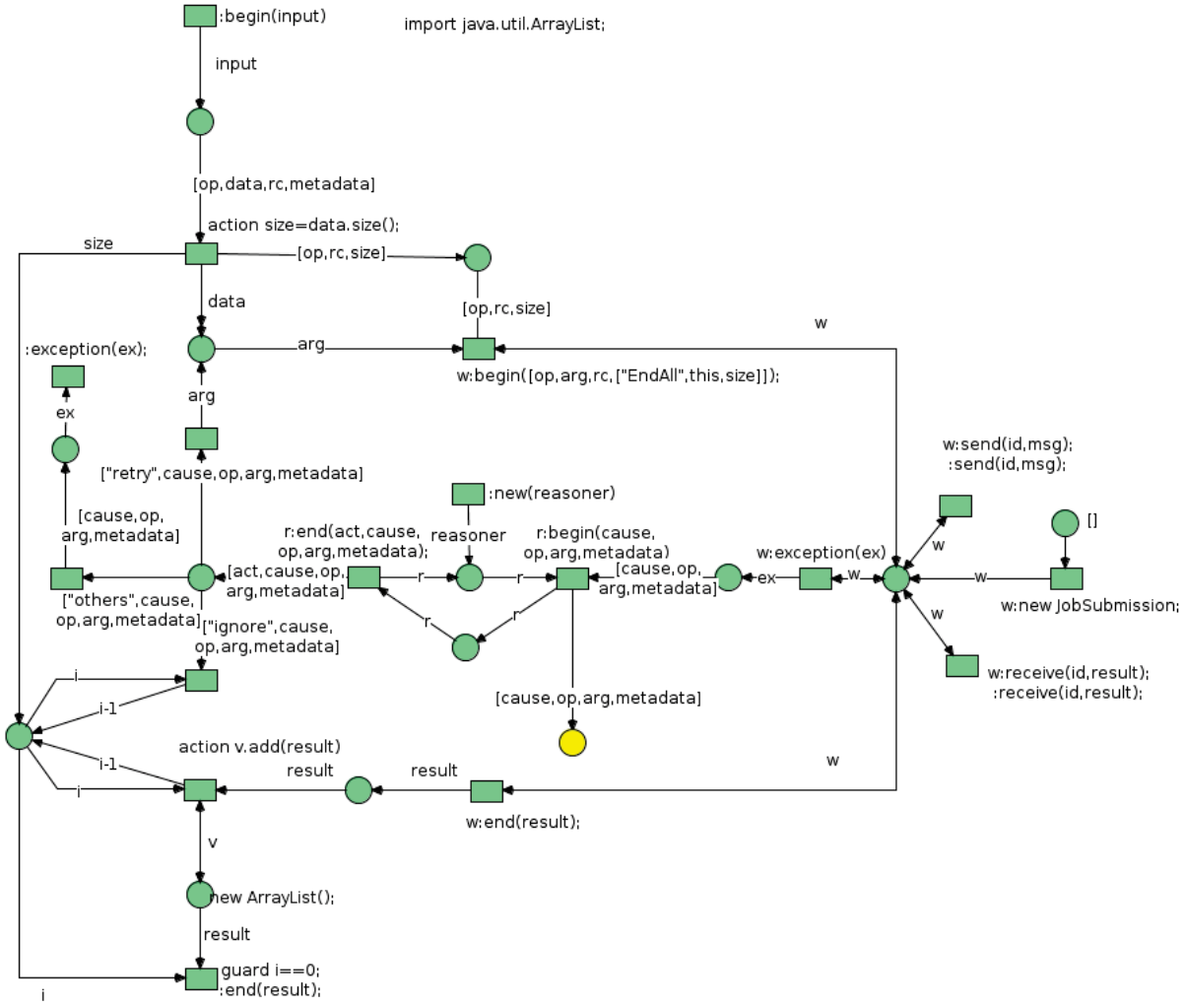



Figura 20: EndAllwithReasoner

### A.2.5. EndAllwithDemmonds

Similar a EndAll aunque en este caso creamos un JobSubmissionWithDemmonds al que le pasamos before y after, que se ejecutarán antes y después como veremos al hablar de esta red.

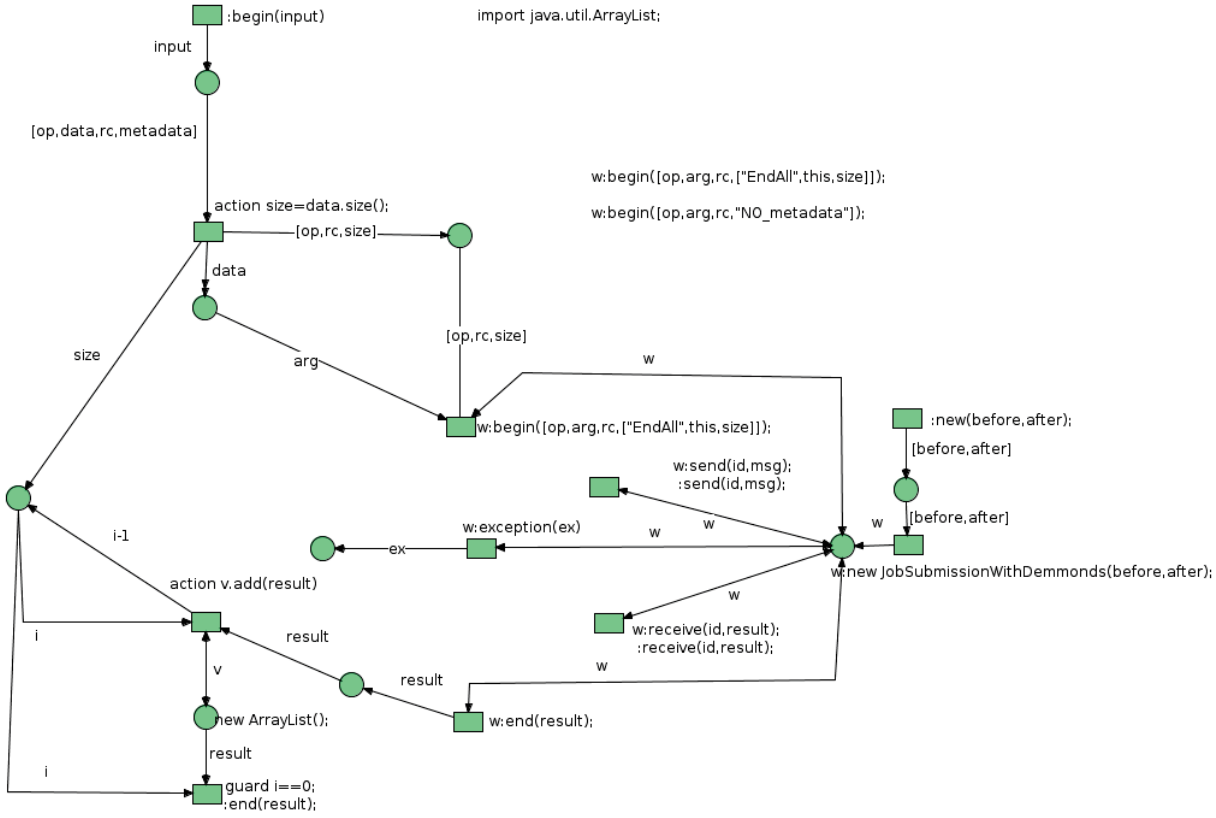



Figura 21: EndAllwithDemmonds

### A.2.6. MontageReasoner

Esta red se utiliza para filtrar excepciones.

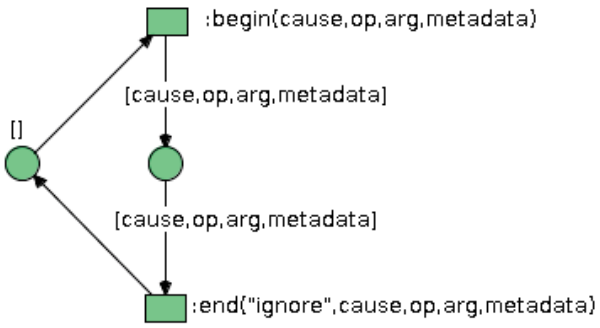



Figura 22: MontageReasoner

### A.2.7. RaiseConditionMontage

Esta red utiliza la clase MontageMsg de Java para analizar los resultados y genera un mensaje de excepción cuando ha detectado algún error.

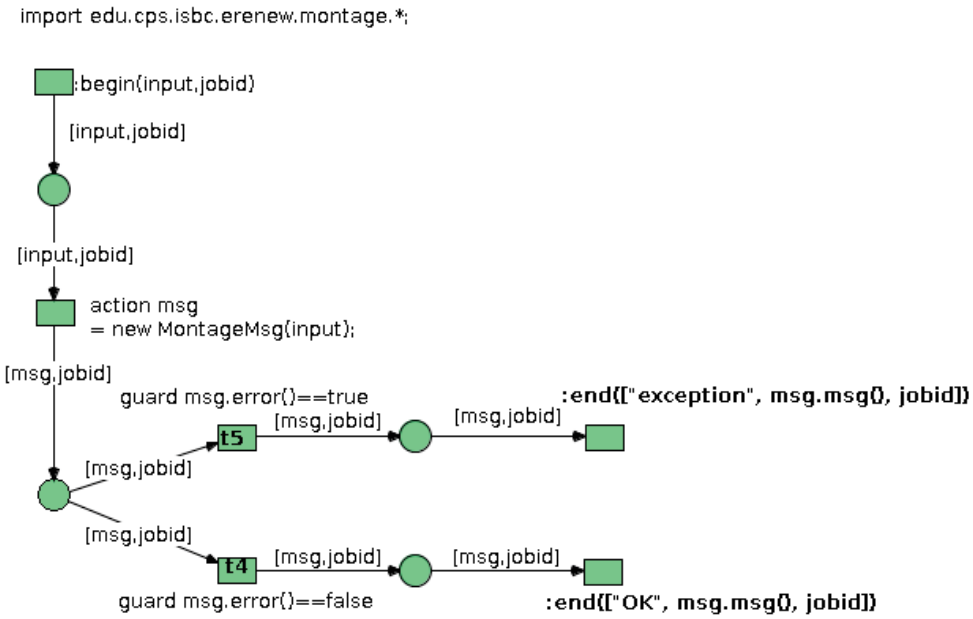



Figura 23: RaiseConditionMontage

### A.2.8. FileSubmission

Esta red se crea en algunos de los pasos de nuestro workflow Montage_DVega6WithReasoner, y se le pasa el tipo de operación a realizar en el servidor: Retrieve, Send o Copy. En él creamos una tupla que será tratada desde Engine para realizar la operación corresponda desde la red SSH.

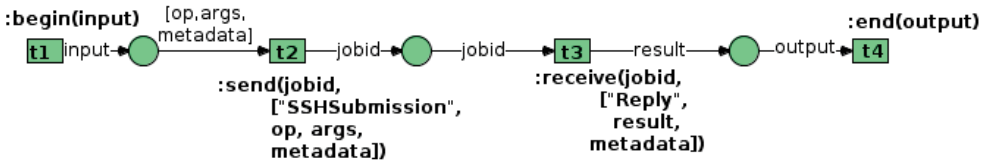



Figura 24: FileSubmission

### A.2.9. mDiffFitAfter

Se ejecuta después de mDiffFit y se utiliza para ajustar el resultado obtenido al formato de salida de la operación de Montage.

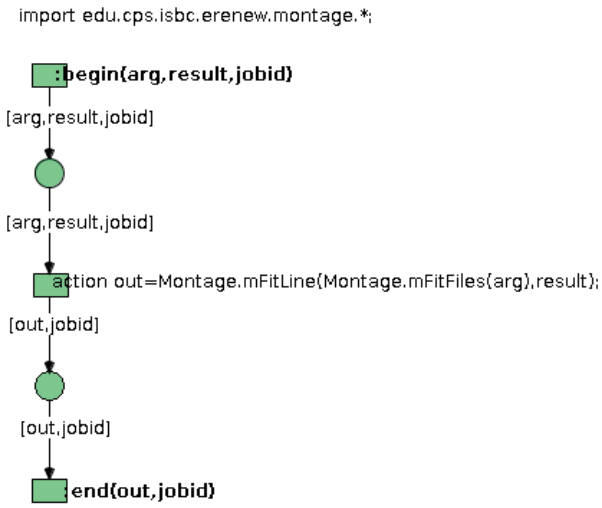



Figura 25: mDiffFitAfter

### A.2.10. mDiffFitBefore

Se ejecuta antes de mDiffFit y se utiliza para adaptar los argumentos recibidos en el formato de entrada de la operación de Montage.

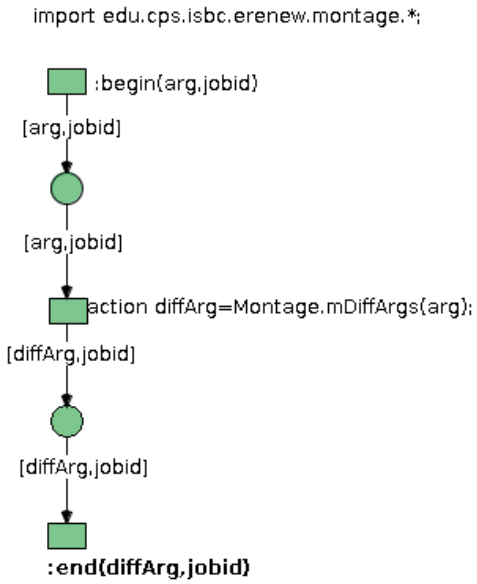



Figura 26: mDiffFitBefore

## A.3. Patrones SSH

En esta subsección encontraremos las redes empleadas para la comunicación con el servidor.

### A.3.1. SSH

La red SSH, creará una red sshJob desde la que se enviarán (send) y recibirán datos del servidor como archivos (copy) y texto contenido en un archivo (retrieve), como salidas de jobs o contenido de los ficheros creados, según nos interese.

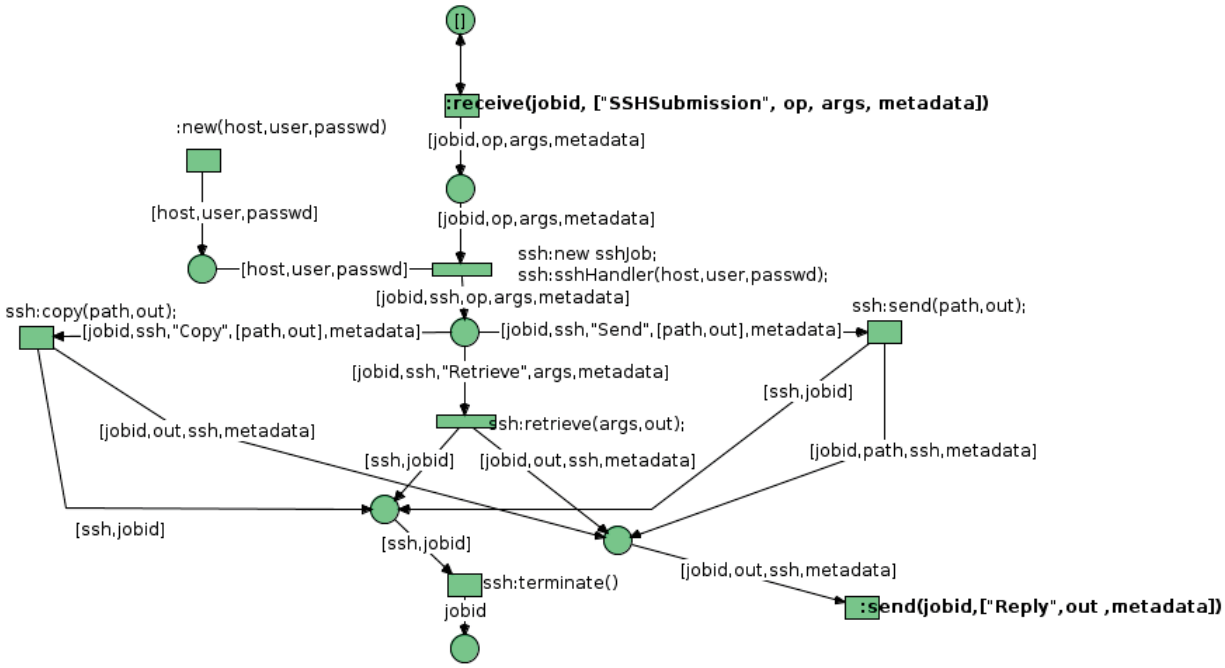



Figura 27: SSH

### A.3.2. sshJob

Utiliza las funciones de la clase sshHandler. Realizará la conexión (connect) con el servidor proporcionado, y las operaciones indicadas desde SSH para enviar y recibir texto y/o archivos, devolviendo en cada caso la salida que más tarde utilizaremos. Una vez realizada la operación, se desconectará del servidor (disconnect).

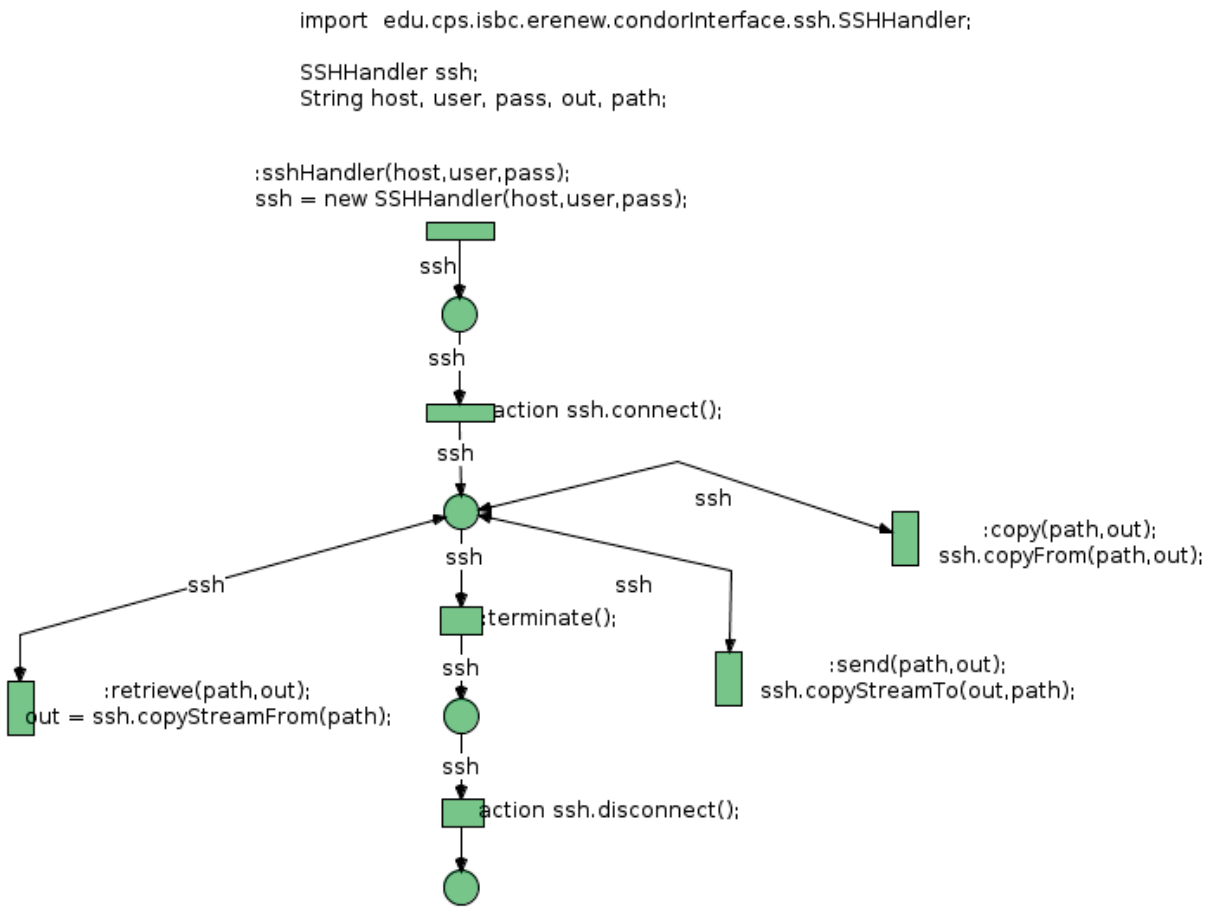



Figura 28: sshJob

## A.4. Patrones Condor

Aquí veremos los patrones relacionados con la submisión de jobs al clúster de Condor.

### A.4.1. condorJobAndCnMngt

En esta red podemos distinguir tres secciones, la parte de arriba es la que se dispara al ejecutar new condorJobAndCnMngt desde Engine, genera un pool de CondorFacade conectados al clúster, que serán utilizadas en las otras dos secciones. Se disparará una de las otras dos secciones dependiendo del tipo de tupla recibida. La de abajo a la derecha sería para el caso de un job simple, se creará un condorJob y se recuperará la salida de la ejecución. La parte de la izquierda, es para jobs que ejecutan varios jobs con el mismo id, como hemos comentado en EndAll. En este caso dispararemos McondorJob_3 e iremos recuperando la salida de cada uno de los jobs, esperando a que terminen todos.

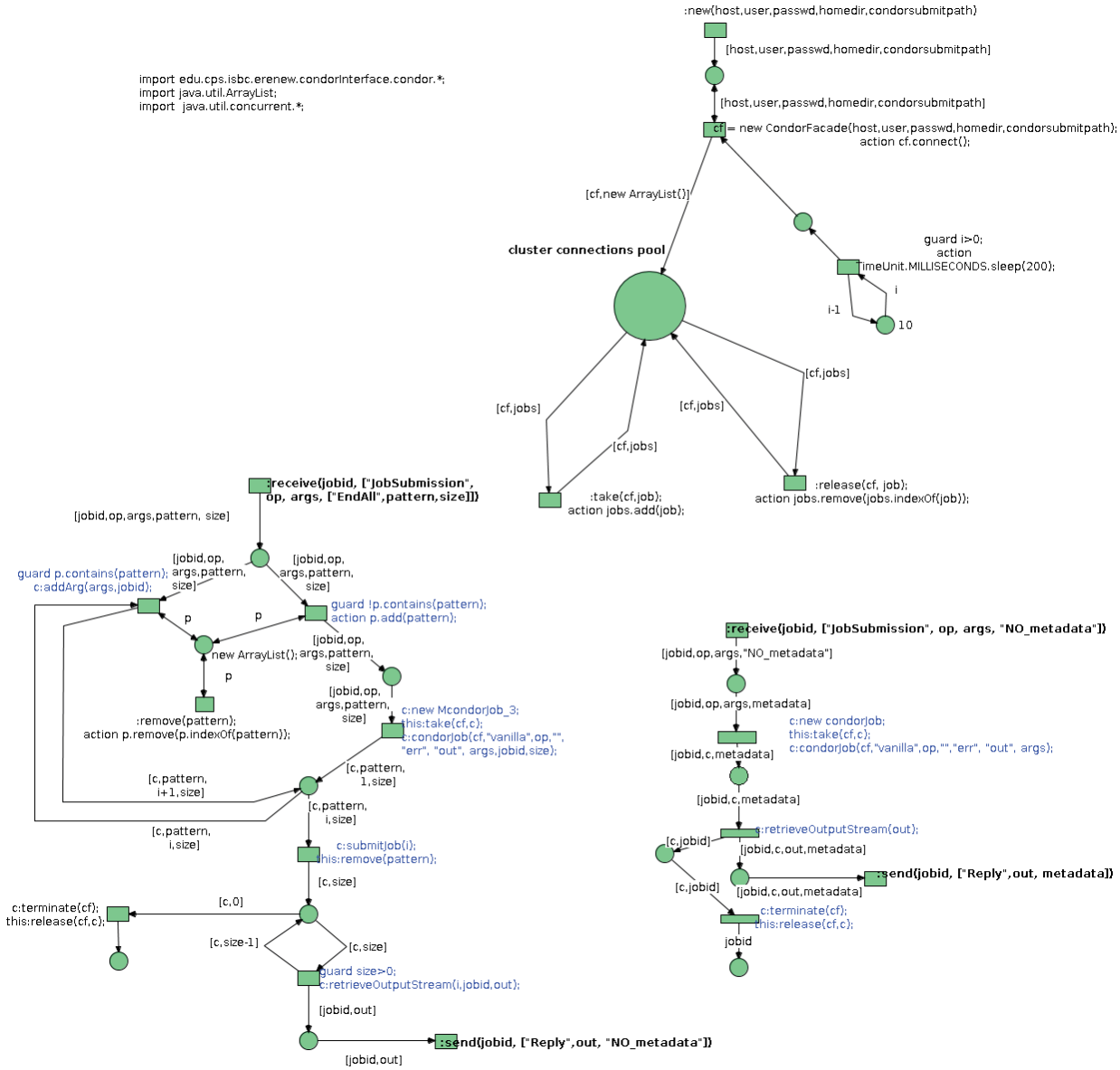



Figura 29: condorJobAndCnMngt

### A.4.2. condorJob

Se puede disparar de varias formas, según los parámetros de condorJob, según le pasemos el cf (CondorFacade) u otros argumentos. En cualquiera de los casos, con el CondorFacade proporcionado o creado, construiremos un CondorJobHandler (clase Java) y utilizaremos sus funciones para definir un entorno de trabajo e instalar el listener para que tras lanzar el job en el clúster, seamos informados una vez que termine. Al finalizar recuperaremos la salida del job, el contenido de la salida y el contenido del fichero obtenido, para usar o mostrar según nos pueda interesar. Una vez finalizada la ejecución, se hace la limpieza del entorno de trabajo creado y se desconecta del servidor.

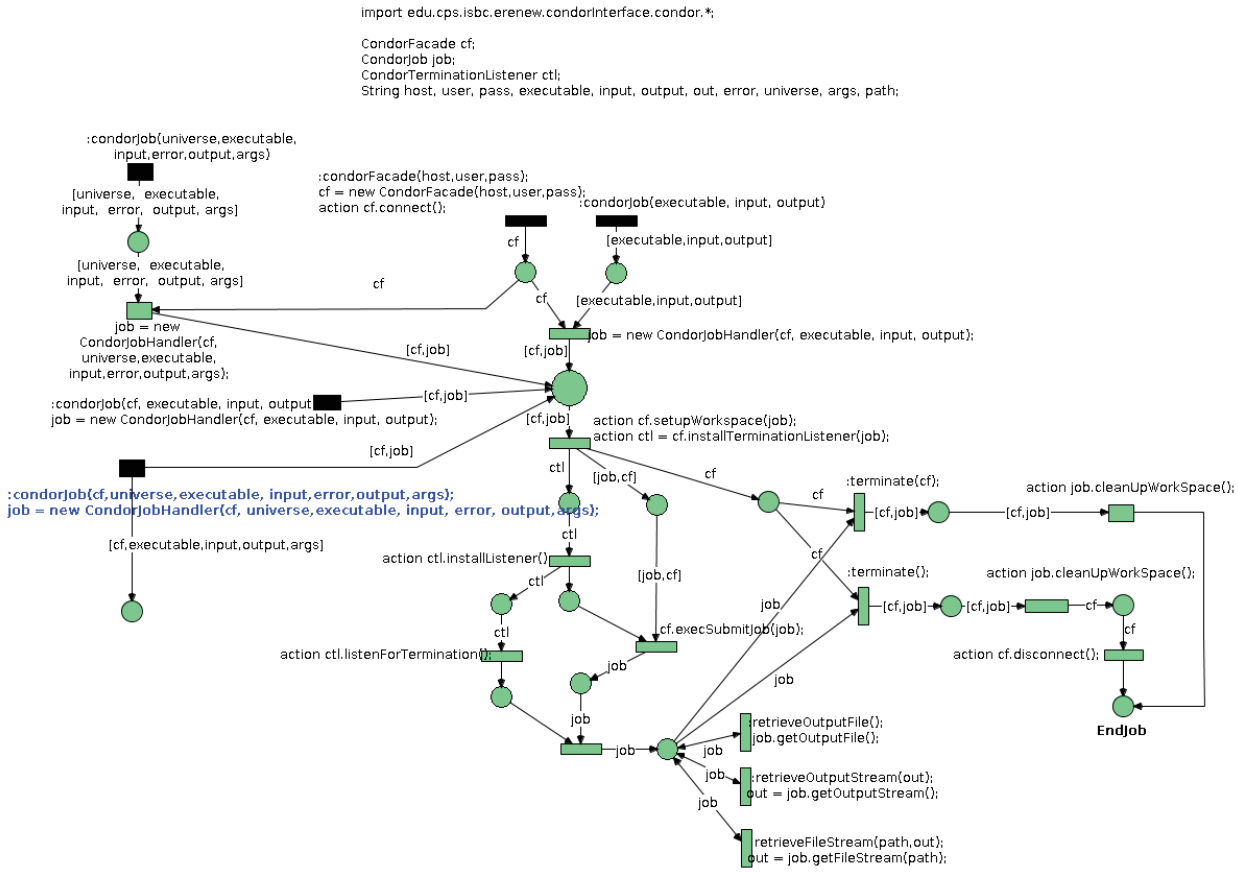



Figura 30: condorJob

### A.4.3. McondorJob_3

Similar a condorJob aunque preparado para jobs que lanzan varios jobs en paralelo. Se le indica el tamaño y espera a que todos estén completados para finalizar.

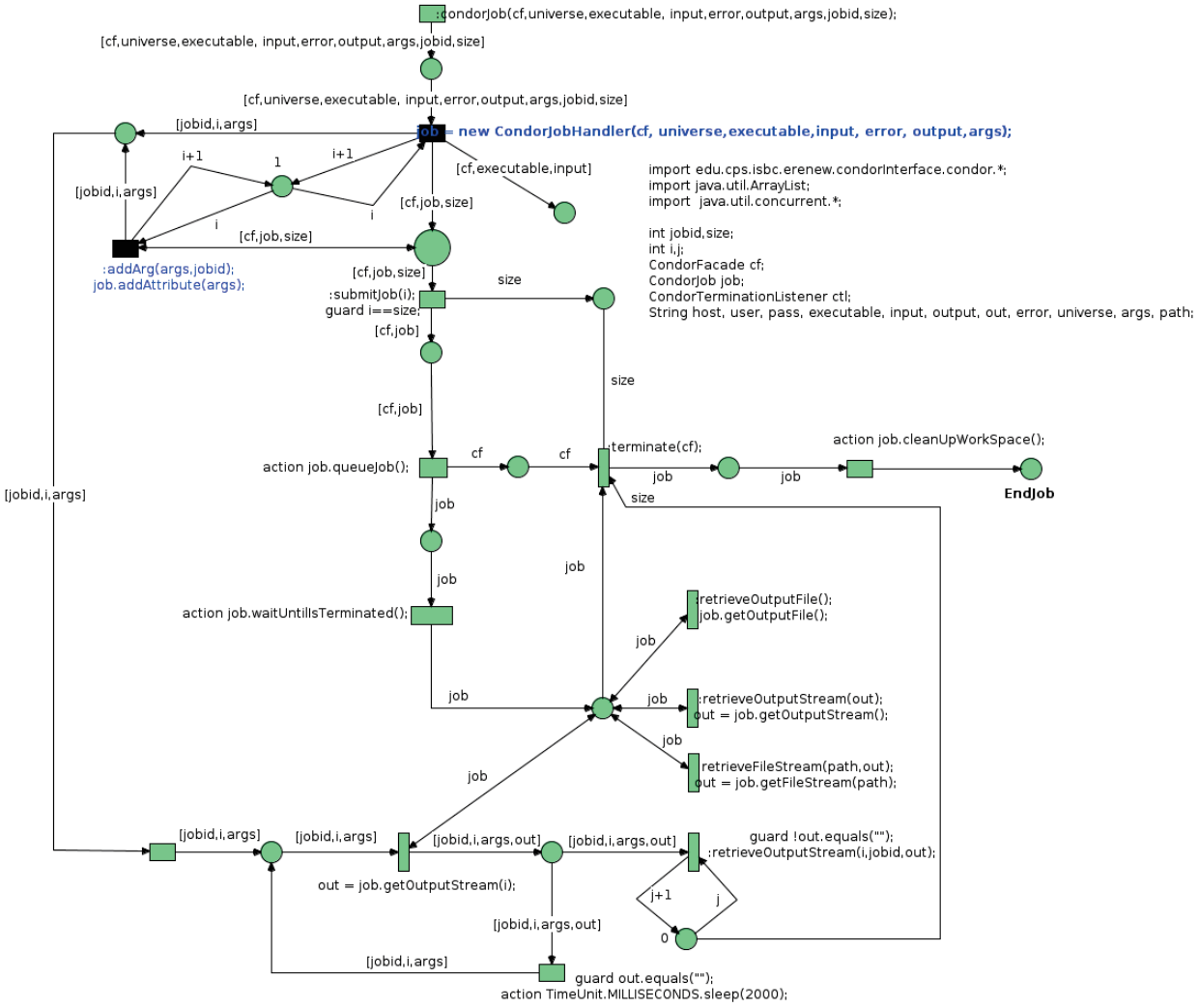



Figura 31: McondorJob_3

### A.4.4. JobSubmission

Es disparado desde EndAll y EndAllWithReasoner, se encarga de crear un RaiseConditionMontage, comentado anteriormente, en caso de habérselo pasado como parámetro y crea las tuplas que a través de Engine, dispararán condorJobAndCnMngt para la ejecución del job.

### A.4.5. JobSubmissionWithDemmonds

Esta red es como JobSubmission pero le añadimos la ejecución de otras dos redes, una antes de crear la tupla para lanzar el job y otra tras finalizar.

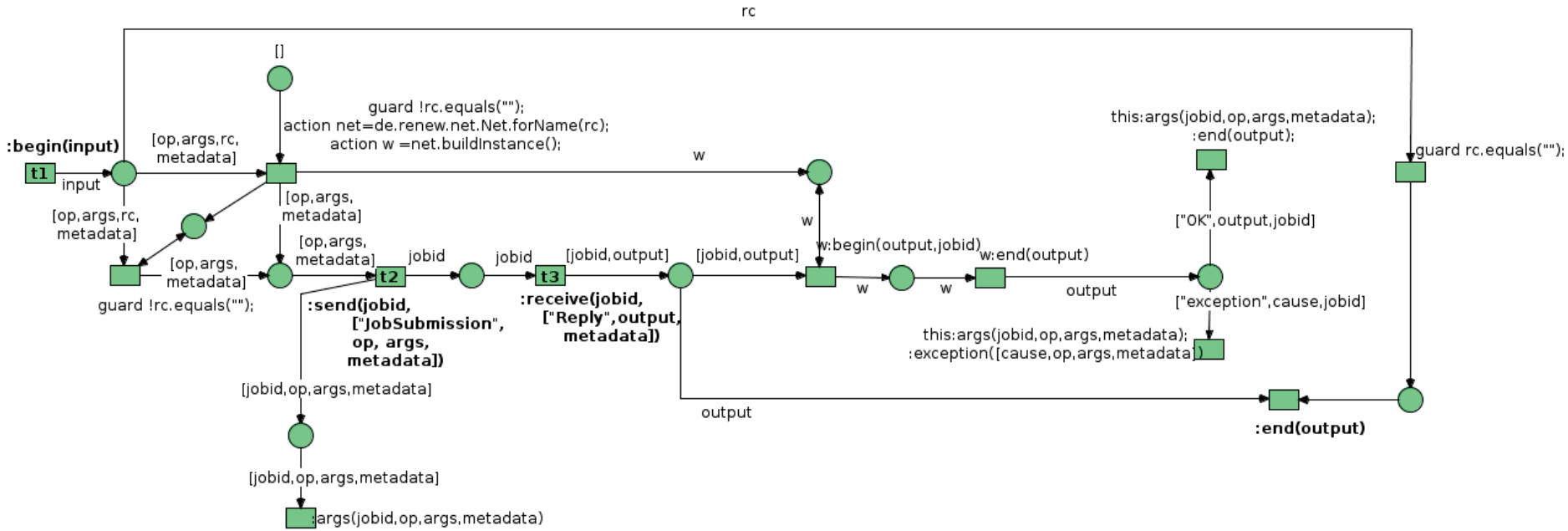



Figura 32: JobSubmission

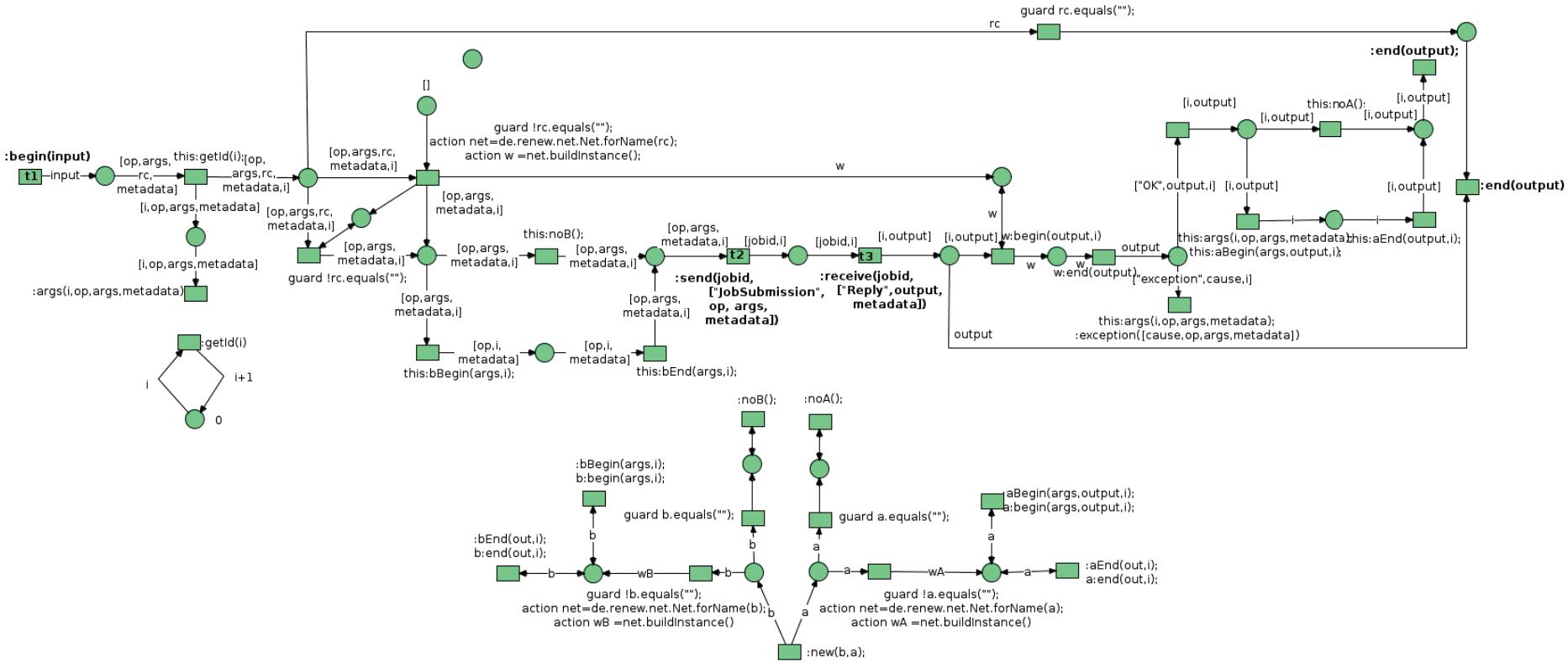
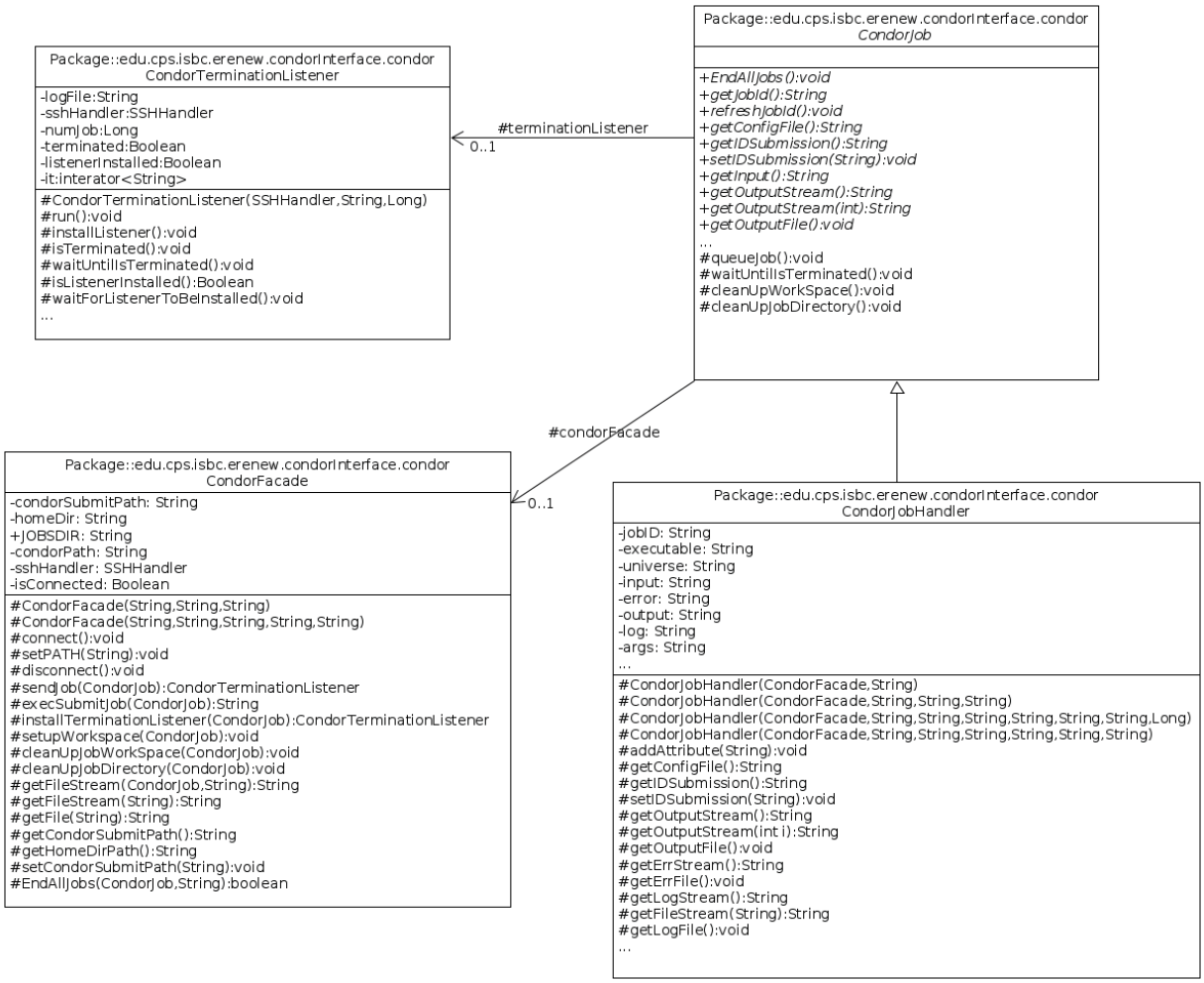



Figura 33: JobSubmissionWithDemmonds

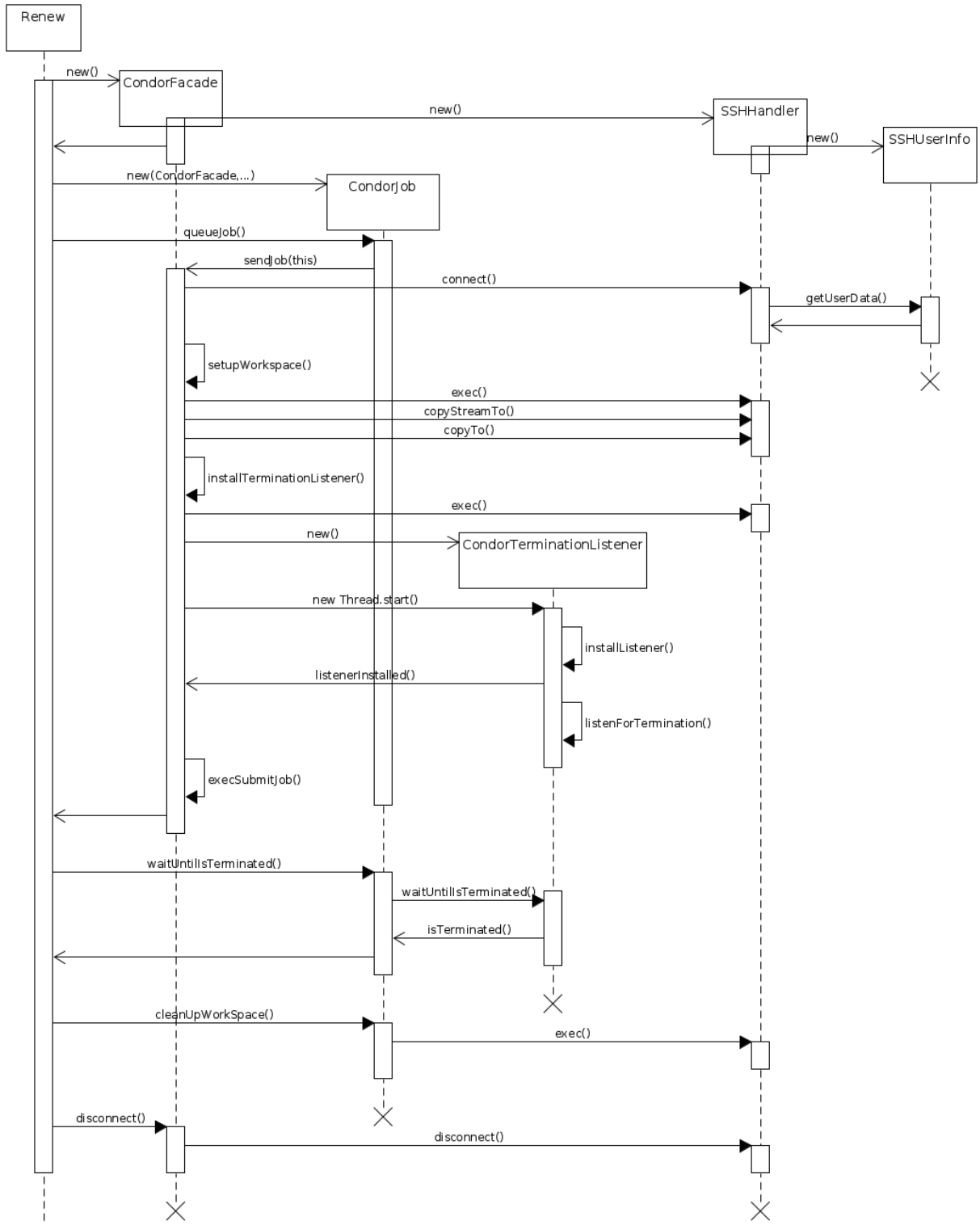
# B. Anexo 2: Diagramas de clase y secuencia

Para ver cómo se relacionan las clases de Condor y SSH, principales para la comunicación, veremos a continuación el diagrama de secuencia y de clase.

## B.1. Diagrama de clase



## B.2. Diagrama de secuencia



# C. Anexo 3: Clases Java

## C.1. Clases Montage

### C.1.1. Montage.java

```
package edu.cps.isbc.erenew.montage;

import java.awt.Desktop;
import java.io.File;
import java.util.ArrayList;

import org.junit.Test;

import edu.cps.isbc.erenew.condorInterface.ssh.SSHHandler;
import junit.framework.Assert;

public class Montage implements MontageCodes {

  /* Takes the diffargs content and return the parameters to use in mDiffFit */
  static public ArrayList<String> mDiffFitArgs(String diffFileString, String homedir, String
      projdir, String diffdir){
    String[] lista=null;
    String[] parameters=null;
    ArrayList<String> arg=new ArrayList<String>();

    try {
      lista= diffFileString.split("\n");

      for (int i=2, j=0; i< lista.length;i++,j++){
        parameters = lista[i].split("\\s+");
        arg.add(parameters[1]+" "+parameters[2]+" "+projdir+parameters[3]+" "+
            projdir+parameters[4]+" "+  diffdir+parameters[5]+ " "+ homedir+"template.hdr");
      }
    }catch (Exception e) {
      e.printStackTrace();
    }
    return arg;
  }

  /* Takes the diffargs content and return the parameters to use in mDiff */
  static public String mDiffArgs (String diffargs){

    String[]  parameters = diffargs.split("\\s+");

    return parameters[2]+" "+parameters[3]+" "+parameters[4]+" "+parameters[5];

  }

  /* Takes the diffargs content and return the parameter to use in mFit */
  static public String mFitArgs (String diffargs){

    String[]  parameters = diffargs.split("\\s+");

    return parameters[4];

  }

  /* Takes the diffargs content and return the int[] with the images with differences */
  static public int[] mFitFiles (String diffargs){

    int[] result=new int[2];
    String[]  parameters = diffargs.split("\\s+");

    result[0]=Integer.parseInt(parameters[0]);
    result[1]=Integer.parseInt(parameters[1]);
```

```
  return result;
}

/* Takes the int[] files and String fitResult content and return the format to use in
    mConcatFit */
static public String mFitLine (int[] files,String fitResult){
  String[]  parameters = fitResult.split(",");

  for (int i=0; i<parameters.length;i++){
    parameters[i]=parameters[i].substring(parameters[i].indexOf("=")+1,parameters[i].length());
  }

  //    "| plus|minus|       a    |      b     |      c     | crpix1  | crpix2  | xmin | xmax
      | ymin | ymax | xcenter | ycenter |  npixel |    rms     |    boxx    |    boxy    |
      boxwidth  | boxheight  |   boxang   |"+"\n";
  return String.format(" %5d %5d %11.5e %11.5e %11.5e %9.2f %9.2f %6d %6d %6d %6d %9.2f %9.2f
      %9d %11.5e %12.1f %12.1f %12.1f %12.1f %12.1f",
    files[0],files[1],
    Float.parseFloat(parameters[0]),Float.parseFloat(parameters[1]),Float.parseFloat(parameters[2]),
    Float.parseFloat(parameters[3]),Float.parseFloat(parameters[4]),
    Integer.parseInt(parameters[5]),Integer.parseInt(parameters[6]),Integer.parseInt(parameters[7]),Integer.
    Float.parseFloat(parameters[9]),Float.parseFloat(parameters[10]),
    Integer.parseInt(parameters[11]),
    Float.parseFloat(parameters[12]),
    Float.parseFloat(parameters[13]),Float.parseFloat(parameters[14]),Float.parseFloat(parameters[15]),Float

}


/*mConcatFit takes an ArrayList<String> where each element of array is the result of mFitplane
    and
 * return the string to copy in the file fits.tbl
 */
static public String mConcatFit (ArrayList<String> fitsArray){
  String file="| plus|minus|       a    |      b     |      c     | crpix1  | crpix2  | xmin |
      xmax | ymin | ymax | xcenter | ycenter |  npixel |    rms     |    boxx    |    boxy
      |  boxwidth  | boxheight  |   boxang   |"+"\n";

  for (int i=0; i < fitsArray.size(); i++){
    file=file+fitsArray.get(i)+"\n";
  }
  return file;
}

/*Takes a tbl file and return an ArrayList<String> with the last parameter content */
static public ArrayList<String> ls (String tbl){
  String[] lista=null;
  String[] parameters=null;
  ArrayList<String> arg =new ArrayList<String>();


  try {
    lista= tbl.split("\n");

    for (int i=3, j=0; i< lista.length;i++,j++){
      parameters = lista[i].split("\\s+");
      arg.add(parameters[parameters.length-1]);
    }
  }catch (Exception e) {
    e.printStackTrace();
  }
  return arg;

}

/*Takes tbl, orgDir, DestDir, homeDir and args parameters and return an ArrayList<String>
 * with the format to be used by mProjectPP
 */
```

```
static public  ArrayList<String>  ls (String tbl, String orgDir, String DestDir, String
    homeDir, String args){
  String[] lista=null;
  String[] parameters=null;
  ArrayList<String> out =new ArrayList<String>();

  try {
  lista= tbl.split("\n");

  for (int i=3, j=0; i< lista.length;i++,j++){
    parameters = lista[i].split("\\s+");
    out.add(parameters[parameters.length-1]+"
        "+DestDir+parameters[parameters.length-1].replace(orgDir, "")+" "+homeDir+args);
  }
  }catch (Exception e) {
    e.printStackTrace();
  }
  return out;
}

/* Used to print a String[] */
static public void lsPrint(String[] lista){
  for (int i=0; i< lista.length;i++){
    System.out.println(lista[i]);
  }
}

/* Used to print an ArrayList<String> */
static public void lsPrint(ArrayList<String> lista){
  for (int i=0; i< lista.size();i++){
    System.out.println(lista.get(i));
  }
}

/* Takes the content of imageTbl and return an ArrayList<String> with the parameters needed
    for mBackgroundArgs*/
public static ArrayList<String> IdFileTbl(String imageTbl){
  String[] lista=null;
  String[] parameters=null;
  ArrayList<String> arg =new ArrayList<String>();


  try {
    lista= imageTbl.split("\n");

    for (int i=3; i< lista.length;i++){
      parameters = lista[i].split("\\s+");
      arg.add(parameters[1]+" "+parameters[parameters.length-1]);
    }
  }catch (Exception e) {
    e.printStackTrace();
  }
  return arg;
}

/* Takes the content of corrTbl and return an ArrayList<String> with the parameters needed for
    mBackgroundArgs*/
public static ArrayList<String> correctionsTbl(String corrTbl){
  String[] lista=null;
  String[] parameters=null;
  ArrayList<String> arg =new ArrayList<String>();


  try {
    lista= corrTbl.split("\n");

    for (int i=1; i< lista.length;i++){
      parameters = lista[i].split("\\s+");
      arg.add(parameters[1]+" "+parameters[2]+" "+parameters[3]+" "+parameters[4]);
```

```
      }
    }catch (Exception e) {
      e.printStackTrace();
    }
    return arg;
  }

  /* Get the arguments and returns the format for mBackground */
  public static ArrayList<String> mBackgroundArgs(ArrayList<String> IdNameTbl, ArrayList<String>
      IdABCTbl,String projdir, String corrdir){
    String[] parameters=null;
    String[] IdName=null;
    ArrayList<String> arg =new ArrayList<String>();

    try {
      for (int i=0; i< IdABCTbl.size();i++){
        parameters = IdABCTbl.get(i).split("\\s+");
        IdName=IdNameTbl.get(Integer.parseInt(parameters[0])).split("\\s+");
        arg.add(projdir+IdName[1].replace(projdir,"")+"
            "+corrdir+IdName[1].replace(projdir,"")+" "+parameters[1]+" "+parameters[2]+"
            "+parameters[3]);
        System.out.println(projdir+IdName[1].replace(projdir,"")+"
            "+corrdir+IdName[1].replace(projdir,"")+" "+parameters[1]+" "+parameters[2]+"
            "+parameters[3]);
      }
    }catch (Exception e) {
      e.printStackTrace();
    }
    return arg;
  }

  /* Open a JPEG file */
  public static void openJPEG(String imageFile) {
    try {
      File f = new File(imageFile);
      Desktop d = Desktop.getDesktop();
      d.open(f);
      System.out.println("Abriendo imagen resultado");
    } catch (Exception e) {
      e.printStackTrace();
      Assert.fail();
    }
  }
}
```

### C.1.2. MontageNetsLoader.java

```
package edu.cps.isbc.erenew.montage;

public class MontageNetsLoader {
  public static void main(String[] args) {
    de.renew.plugin.Loader.main((new String[] {"gui",
      "nets/condor-nets/workflows/mDiffFitAfter.rnw",        // To execute before mDiffFit
          Operation and EndAllWithDemmonds
      "nets/condor-nets/workflows/mDiffFitBefore.rnw",       // To execute after mDiffFit
          Operation and EndAllWithDemmonds
      "nets/condor-nets/workflows/RaiseConditionMontage.rnw",
      "nets/condor-nets/patterns/EndAll.rnw",
      "nets/condor-nets/patterns/EndAllwithDemmonds.rnw",
      "nets/condor-nets/patterns/EndAllwithReasoner.rnw",
      "nets/condor-nets/patterns/SeqStep.rnw",
      "nets/condor-nets/patterns/JobSubmission.rnw",
      "nets/condor-nets/patterns/JobSubmissionWithDemmonds.rnw",  //Alone instance to submit in
          EndAllwithDemmonds
      "nets/condor-nets/patterns/FileSubmission.rnw",
      "nets/condor-nets/resources/SSH.rnw",
      "nets/condor-nets/resources/condorJobAndCnMngt.rnw",
```

```
      "nets/condor-nets/condorJob.rnw",
      "nets/condor-nets/McondorJob_3.rnw",
      "nets/condor-nets/resources/sshJob.rnw",
      "nets/condor-nets/workflows/Montage_DVega7withReasonerJPEG.rnw", //wf with the creation
          and opening of the created image
      "nets/condor-nets/workflows/MontageReasoner.rnw",
      "nets/condor-nets/Engine.rnw",
      }));
  }
}
```

### C.1.3. MontageMsg.java

```
package edu.cps.isbc.erenew.montage;

public class MontageMsg implements MontageCodes {

  private String status;
  private String msg;
  private boolean error;
  private String homedir="/home/eburges/m101/";
  private String diffdir="diffdir/";
  private String projdir =" projdir/";

  public MontageMsg(){
  }

  public MontageMsg(String out) {

    try {
      if (out.contains("OK"))
        {
        error = false;
        status="OK";
        }
      else{
        if (out.contains("ERROR"))
        {
          status="ERROR";
          error = true;
        }
        else {
            status="WARNING";
            error = false;
        }
      }
      if (out==null) error = true;

      if (out.contains("msg="))
        msg=out.substring(out.indexOf("msg=")+4,out.length()-2);

        msg=out.substring(out.indexOf(",")+2,out.length()-2);

    }catch (Exception e) {
      e.printStackTrace();
    }
  }

  public boolean error (){
    return error;
  }

  public String msg(){
    if (msg==null) msg="Empty message";
    return msg;
  }
```

```
public String toString(){
  return "status:"+status+","+ "msg:" + msg;
}

public String[] diffArgs (String diffFileString){
  String[] lista=null;
  String[] parameters=null;
  String[] arg=null;


  try {
    lista= diffFileString.split("\n");
    arg = new String[lista.length-2];

    for (int i=2, j=0; i< lista.length;i++,j++){
      parameters = lista[i].split("\\s+");
      arg[j] = homedir+projdir+parameters[3]+" "+ homedir+projdir+parameters[4]+" "+
          homedir+diffdir+parameters[5]+ " "+ homedir+"template.hdr";
      //System.out.println(parameters[3]+" "+ parameters[4]+" "+ parameters[5]);
    }
  }catch (Exception e) {
    e.printStackTrace();
  }
  return arg;

}

public void diffArgsPrint(String[] lista){
  String[] parameters;
  for (int i=2; i< lista.length;i++){
    parameters = lista[i].split("\\s+");
    System.out.println(parameters[3]+" "+ parameters[4]+" "+ parameters[5]);
  }
}

static public String[] ls (String tbl){
  String[] lista=null;
  String[] parameters=null;
  String[] arg=null;


  try {
    lista= tbl.split("\n");
    arg = new String[lista.length-3];

    for (int i=3, j=0; i< lista.length;i++,j++){
      parameters = lista[i].split("\\s+");
      arg[j] =  parameters[parameters.length-1];
      //System.out.println(parameters[3]+" "+ parameters[4]+" "+ parameters[5]);
    }
  }catch (Exception e) {
    e.printStackTrace();
  }
  return arg;

}

static public String [] ls (String tbl, String homeDir, String DestDir, String args){
  String[] lista=null;
  String[] parameters=null;
  String[] out=null;


  try {
    lista= tbl.split("\n");
    out = new String[lista.length-3];

    for (int i=3, j=0; i< lista.length;i++,j++){
      parameters = lista[i].split("\\s+");
```

```
        out[j]=homeDir+parameters[parameters.length-1]+"
            "+DestDir+parameters[parameters.length-1]+" "+args;
      }
    }catch (Exception e) {
      e.printStackTrace();
    }
    return out;
  }

  static public void lsPrint(String[] lista){
    for (int i=0; i< lista.length;i++){
      System.out.println(lista[i]);
    }
  }
}
```

## C.2. Ejemplo de test en Java

Algunas de las funciones de Montage testeadas en Java:

```
package edu.cps.isbc.erenew.test.condor;

import junit.framework.Assert;
import junit.framework.TestCase;

import java.awt.Desktop;
import java.util.ArrayList;
import java.util.Date;

import org.junit.Test;
import java.io.File;

import edu.cps.isbc.erenew.condorInterface.condor.CondorFacade;
import edu.cps.isbc.erenew.condorInterface.condor.CondorJob;
import edu.cps.isbc.erenew.condorInterface.condor.CondorJobHandler;
import edu.cps.isbc.erenew.condorInterface.ssh.SSHHandler;
import edu.cps.isbc.erenew.montage.*;

public class CondorTestAtps extends TestCase {
  static ArrayList<String> ls,diffs,args,resultFits;
  static String fits,result,file;

  @Test
  public void testCondorLocal() {
    CondorFacade cf = new CondorFacade("atps.cps.unizar.es", "eburges","passwd",
            "/home/eburges",
          "env CONDOR_CONFIG=/usr/local/condor/etc/condor_config /usr/local/condor/bin");

    CondorJob job = new CondorJobHandler(cf, "/bin/cat", "test.txt", "test.out");

    String out = null;
    try {

      job.queueJob();

      job.waitUntilIsTerminated();

      out = job.getOutputStream();

      job.cleanUpWorkSpace();

      cf.disconnect();
    } catch (Exception e) {

      e.printStackTrace();
      Assert.fail();
    }
```

```
  Assert.assertEquals(out, "testOK");
}

@Test
public void testCondormImgTbl() {
//Generate an image metadata table describing the contents of rawdir

  CondorFacade cf = new CondorFacade("atps.cps.unizar.es", "eburges","passwd",
        "/home/eburges",
      "env CONDOR_CONFIG=/usr/local/condor/etc/condor_config /usr/local/condor/bin");

  CondorJob job = new CondorJobHandler(cf, "vanilla",
      "/home/eburges/Montage_v3.3/bin/mImgtbl",
      "",
      "err",
      "out",
      "/home/eburges/m101/rawdir /home/eburges/m101/images-rawdir.tbl",
      1L);
  String out = null;
  try {
    System.out.println(job.getConfigFile() + "Trabajos:" + job.getQueueTimes());

    job.queueJob();

    job.waitUntilIsTerminated();

    String input = job.getInput();
    System.out.println("getInput():"+input);

    out = job.getOutputStream().replace("\"", "").trim();

    System.out.println("Salida getOutputStream():"+out);

    job.cleanUpWorkSpace();
    cf.disconnect();
  } catch (Exception e) {
    e.printStackTrace();
    Assert.fail();
  }

  Assert.assertEquals(out, "[struct stat=OK, count=10, badfits=0, badwcs=0]");
}


@Test
public void  testRetrieveTbl() {
  //return array list to submit to the testCondormProjectPP. Static global variable.

  String returnString = null;

  SSHHandler ssh = new SSHHandler("atps.cps.unizar.es","eburges","passwd");
  try {
    ssh.connect();
    returnString = ssh.copyStreamFrom("/home/eburges/m101/images-rawdir.tbl");

    ssh.disconnect();
  } catch (Exception e) {
    e.printStackTrace();
    Assert.fail();
  }
  // Remember, Condor takes the paths from /export
  //            text  file    org path                      dest path
      /home                 args
  ls=Montage.ls(returnString,"/home/eburges/m101/rawdir/", "/home/eburges/m101/projdir/",
      "/home/eburges/m101/","template.hdr");
}


@Test
```

```
public void testCondormProjectPP() {
  //Creates a set of reprojected images in the directory projdir,
  //and a table showing processing times for each image: stats.tbl

  int results = 0;
  MontageMsg msg;

  CondorFacade cf = new CondorFacade("atps.cps.unizar.es", "eburges","passwd",
        "/home/eburges",
      "env CONDOR_CONFIG=/usr/local/condor/etc/condor_config /usr/local/condor/bin");

  CondorJob job = new CondorJobHandler(cf, "vanilla",
      "/home/eburges/Montage_v3.3/bin/mProjectPP", "",
      "err",
      "out",
      ls.get(0),
      1L);
  try {
    //System.out.println(job.getConfigFile());
    for(int i=1; i<ls.size();i++) job.addAttribute(ls.get(i));

    System.out.println(job.getConfigFile() + "Trabajos:" + job.getQueueTimes());

    job.queueJob();


    System.out.println("Waiting");
    job.waitUntilIsTerminated();
    System.out.println("Terminated");
    for(int i=1; i<=ls.size();i++){
      msg = new MontageMsg(job.getOutputStream(i));
      if ( msg.error()!=true) results++;
      System.out.println((i)+")"+job.getOutputStream(i));
    }

    job.cleanUpWorkSpace();
    cf.disconnect();
  } catch (Exception e) {
    e.printStackTrace();
    Assert.fail();
  }

  Assert.assertEquals(results, 10);

}

@Test
public void testCondormImgtblProjdir() {
  //After the images have been reprojected, generate a new metadata table
  //that contains the new header information: images.tbl

  CondorFacade cf = new CondorFacade("atps.cps.unizar.es", "eburges","passwd",
        "/home/eburges",
      "env CONDOR_CONFIG=/usr/local/condor/etc/condor_config /usr/local/condor/bin");

  CondorJob job = new CondorJobHandler(cf, "vanilla",
    "/home/eburges/Montage_v3.3/bin/mImgtbl",
    "",
    "err",
    "out",
    "/home/eburges/m101/projdir /home/eburges/m101/images.tbl",
    1L);
  String out = null;
  try {
    System.out.println(job.getConfigFile() + "Trabajos:" + job.getQueueTimes());

    job.queueJob();

    job.waitUntilIsTerminated();
```

```
    String input = job.getInput();
    System.out.println("getInput():"+input);

    out = job.getOutputStream().replace("\"", "").trim();

    System.out.println("Salida getOutputStream():"+out);

    job.cleanUpWorkSpace();
    cf.disconnect();
    } catch (Exception e) {
      e.printStackTrace();
      Assert.fail();
  }

  Assert.assertEquals(out, "[struct stat=OK, count=10, badfits=0, badwcs=0]");
}

@Test
public void testCondormOverlaps() {
  //Determine, using mOverlaps, which images overlap

  CondorFacade cf = new CondorFacade("atps.cps.unizar.es", "eburges","passwd",
        "/home/eburges",
      "env CONDOR_CONFIG=/usr/local/condor/etc/condor_config /usr/local/condor/bin");

  CondorJob job = new CondorJobHandler(cf, "vanilla",
    "/home/eburges/Montage_v3.3/bin/mOverlaps",
    "",
    "err",
    "out",
    "/home/eburges/m101/images.tbl /home/eburges/m101/diffs.tbl",
    1L);
  String out = null;
  try {
    System.out.println(job.getConfigFile() + "Trabajos:" + job.getQueueTimes());

    job.queueJob();

    job.waitUntilIsTerminated();

    String input = job.getInput();
    System.out.println("getInput():"+input);

    out = job.getOutputStream().replace("\"", "").trim();

    System.out.println("Salida getOutputStream():"+out);

    job.cleanUpWorkSpace();
    cf.disconnect();
    } catch (Exception e) {
      e.printStackTrace();
      Assert.fail();
  };

  Assert.assertEquals(out, "[struct stat=OK, count=17]");
}

@Test
public void  testRetrieveDiffsTbl() {
  //return array list to submit to the testCondormDiffFit. Static global variable.
  String returnString = null;

  SSHHandler ssh = new SSHHandler("atps.cps.unizar.es","eburges","passwd");
  try {
    ssh.connect();
    returnString = ssh.copyStreamFrom("/home/eburges/m101/diffs.tbl");

    ssh.disconnect();
```

```
  } catch (Exception e) {
    e.printStackTrace();
    Assert.fail();
  }
  // Remember, Condor takes the paths from /export
  //                    text  file,   homedir,        projdir,                    diffdir
  diffs=Montage.mDiffFitArgs(returnString,"/home/eburges/m101/",
      "/home/eburges/m101/projdir/", "/home/eburges/m101/diffdir/");
}


@Test
public void testCondormDiffFit() {

  int results = 0;
  MontageMsg msg;
  String diffArg;

  CondorFacade cf = new CondorFacade("atps.cps.unizar.es", "eburges","passwd",
        "/home/eburges",
      "env CONDOR_CONFIG=/usr/local/condor/etc/condor_config /usr/local/condor/bin");

  CondorJob job = new CondorJobHandler(cf, "vanilla",
      "/home/eburges/Montage_v3.3/bin/mDiffFit", "",
      "err",
      "out",
      Montage.mDiffArgs(diffs.get(0)),
      1L);
  try {

    for(int i=1; i<diffs.size();i++){
      diffArg=Montage.mDiffArgs(diffs.get(i));
      job.addAttribute(diffArg);
    }
    System.out.println(job.getConfigFile() + "Trabajos:" + job.getQueueTimes());

    job.queueJob();


    System.out.println("Waiting");
    job.waitUntilIsTerminated();
    System.out.println("Terminated");
    for(int i=1; i<=diffs.size();i++){
      msg = new MontageMsg(job.getOutputStream(i));
      if ( msg.error()!=true) results++;
      System.out.println((i)+")"+job.getOutputStream(i));
      resultFits.add(Montage.mFitLine(Montage.mFitFiles(diffs.get(i)),job.getOutputStream(i)));
    }

    job.cleanUpWorkSpace();
    cf.disconnect();
  } catch (Exception e) {
    e.printStackTrace();
    Assert.fail();
  }

  Assert.assertEquals(results, 10);
}


@Test
public void testCondormDiff() {
  //runs mDiff for each record.

  int results = 0;
  MontageMsg msg;

  CondorFacade cf = new CondorFacade("atps.cps.unizar.es", "eburges","passwd",
        "/home/eburges",
```

```
        "env CONDOR_CONFIG=/usr/local/condor/etc/condor_config /usr/local/condor/bin");

    CondorJob job = new CondorJobHandler(cf, "vanilla",
        "/home/eburges/Montage_v3.3/bin/mDiff", "",
        "err",
        "out",
        Montage.mDiffArgs(diffs.get(0)),
        1L);
    try {
      //System.out.println(job.getConfigFile());
      for(int i=1; i<diffs.size();i++) job.addAttribute(Montage.mDiffArgs(diffs.get(i)));

      System.out.println(job.getConfigFile() + "Trabajos:" + job.getQueueTimes());

      job.queueJob();


      System.out.println("Waiting");
      job.waitUntilIsTerminated();
      System.out.println("Terminated");
      for(int i=1; i<=diffs.size();i++){
        msg = new MontageMsg(job.getOutputStream(i));
        if ( msg.error()!=true) results++;
        System.out.println((i)+")"+job.getOutputStream(i));
      }

      job.cleanUpWorkSpace();
      cf.disconnect();
    } catch (Exception e) {
      e.printStackTrace();
      Assert.fail();
    }

    Assert.assertEquals(results, 17);
  }

  @Test
  public void testCondormFitPlane() {
    //runs mFitplane for each record.

    int results = 0;
    MontageMsg msg;

    CondorFacade cf = new CondorFacade("atps.cps.unizar.es", "eburges","passwd",
          "/home/eburges",
        "env CONDOR_CONFIG=/usr/local/condor/etc/condor_config /usr/local/condor/bin");

    CondorJob job = new CondorJobHandler(cf, "vanilla",
        "/home/eburges/Montage_v3.3/bin/mFitPlane", "",
        "err",
        "out",
        Montage.mFitArgs(diffs.get(0)),
        1L);
    try {
      //System.out.println(job.getConfigFile());
      for(int i=1; i<diffs.size();i++) job.addAttribute(Montage.mFitArgs(diffs.get(i)));

      System.out.println(job.getConfigFile() + "Trabajos:" + job.getQueueTimes());

      job.queueJob();


      System.out.println("Waiting");
      job.waitUntilIsTerminated();
      System.out.println("Terminated");
      for(int i=1; i<=diffs.size();i++){
        msg = new MontageMsg(job.getOutputStream(i));
        if ( msg.error()!=true) results++;
        System.out.println((i)+")"+job.getOutputStream(i));
```

```
    }

    //mFitLine(Montage.mFitFiles(args),result);
    // PONER la salida en un Array List para usar luego en concatFit

    //job.cleanUpWorkSpace();
    cf.disconnect();
  } catch (Exception e) {
    e.printStackTrace();
    Assert.fail();
  }

  Assert.assertEquals(results, 17);
}


@Test
public void testSendFitsTbl(){
  //The fitting parameters resulted in testCondormDiffFit are written to a file to be used by
      mBgModel.

  file=Montage.mConcatFit(resultFits);

  SSHHandler ssh = new SSHHandler("atps.cps.unizar.es","eburges","passwd");
  try {
    ssh.connect();
    ssh.copyStreamTo(file,"/home/eburges/m101/fits.tbl");

    ssh.disconnect();
  } catch (Exception e) {
    e.printStackTrace();
    Assert.fail();
  }

}

@Test
public void testCondormDiffFitExec() {
  //Using the table of overlaps found by mOverlaps, mDiffFitExec runs both mDiff and mFitplane
      for each record.
  //mDiffExec uses this file to subtract each pair of overlapping images,
  //creating a set of difference images in the specified diffdir subdirectory.
  //mFitExec calculates plane-fitting coefficients for each difference image.
  //The fitting parameters are written to a file to be used by mBgModel.

  CondorFacade cf = new CondorFacade("atps.cps.unizar.es", "eburges","passwd",
        "/home/eburges",
      "env CONDOR_CONFIG=/usr/local/condor/etc/condor_config /usr/local/condor/bin");

  CondorJob job = new CondorJobHandler(cf, "vanilla",
    "/home/eburges/Montage_v3.3/bin/mDiffFitExec",
    "",
    "err",
    "out",
    "-p /home/eburges/m101/projdir /home/eburges/m101/diffs.tbl
        /home/eburges/m101/template.hdr /home/eburges/m101/diffdir
        /home/eburges/m101/fits.tbl",
    1L);
  String out = null;
  try {
    System.out.println(job.getConfigFile() + "Trabajos:" + job.getQueueTimes());

    job.queueJob();

    job.waitUntilIsTerminated();

    String input = job.getInput();
    System.out.println("getInput():"+input);
```

```
    out = job.getOutputStream().replace("\"", "").trim();

    System.out.println("Salida getOutputStream():"+out);

    job.cleanUpWorkSpace();
    cf.disconnect();
    } catch (Exception e) {
      e.printStackTrace();
      Assert.fail();
  }

  Assert.assertEquals(out, "[struct stat=OK, count=17, diff_failed=0, fit_failed=0,
      warning=0]");
}

@Test
public void testCondormDiffExec() {


  CondorFacade cf = new CondorFacade("atps.cps.unizar.es", "eburges","passwd",
        "/home/eburges",
      "env CONDOR_CONFIG=/usr/local/condor/etc/condor_config /usr/local/condor/bin");


  CondorJob job = new CondorJobHandler(cf, "vanilla",
    "/home/eburges/Montage_v3.3/bin/mDiffExec",
    "",
    "err",
    "out",
    "-p /home/eburges/m101/projdir /home/eburges/m101/diffs.tbl
        /home/eburges/m101/template.hdr /home/eburges/m101/diffdir",
    1L);
  String out = null;
  try {
    System.out.println(job.getConfigFile() + "Trabajos:" + job.getQueueTimes());

    job.queueJob();

    job.waitUntilIsTerminated();

    String input = job.getInput();
    System.out.println("getInput():"+input);

    out = job.getOutputStream().replace("\"", "").trim();

    System.out.println("Salida getOutputStream():"+out);

    job.cleanUpWorkSpace();
    cf.disconnect();
    } catch (Exception e) {
      e.printStackTrace();
      Assert.fail();
  }

  Assert.assertEquals(out, "[struct stat=OK, count=17, failed=0]");

}


@Test
public void testCondormFitExec() {


  CondorFacade cf = new CondorFacade("atps.cps.unizar.es", "eburges","passwd",
        "/home/eburges",
      "env CONDOR_CONFIG=/usr/local/condor/etc/condor_config /usr/local/condor/bin");

  CondorJob job = new CondorJobHandler(cf, "vanilla",
    "/home/eburges/Montage_v3.3/bin/mFitExec",
```

```
    "",
    "err",
    "out",
    "/home/eburges/m101/diffs.tbl /home/eburges/m101/fits.tbl /home/eburges/m101/diffdir",
    1L);
  String out = null;
  try {
    System.out.println(job.getConfigFile() + "Trabajos:" + job.getQueueTimes());

    job.queueJob();

    job.waitUntilIsTerminated();

    String input = job.getInput();
    System.out.println("getInput():"+input);

    out = job.getOutputStream().replace("\"", "").trim();

    System.out.println("Salida getOutputStream():"+out);

    job.cleanUpWorkSpace();
    cf.disconnect();
    } catch (Exception e) {
      e.printStackTrace();
      Assert.fail();
  }

  Assert.assertEquals(out, "[struct stat=OK, count=17, failed=0, warning=0, missing=0]");

}



@Test
public void testCondormBgModel() {
  //Create a table of corrections that need to be applied to each image

  CondorFacade cf = new CondorFacade("atps.cps.unizar.es", "eburges","passwd",
        "/home/eburges",
      "env CONDOR_CONFIG=/usr/local/condor/etc/condor_config /usr/local/condor/bin");

  CondorJob job = new CondorJobHandler(cf, "vanilla",
    "/home/eburges/Montage_v3.3/bin/mBgModel",
    "",
    "err",
    "out",
    "/home/eburges/m101/images.tbl /home/eburges/m101/fits.tbl
        /home/eburges/m101/corrections.tbl",
    1L);
  String out = null;
  try {
    System.out.println(job.getConfigFile() + "Trabajos:" + job.getQueueTimes());

    job.queueJob();

    job.waitUntilIsTerminated();

    String input = job.getInput();
    System.out.println("getInput():"+input);

    out = job.getOutputStream().replace("\"", "").trim();

    System.out.println("Salida getOutputStream():"+out);

    job.cleanUpWorkSpace();
    cf.disconnect();
    } catch (Exception e) {
      e.printStackTrace();
```

```
      Assert.fail();
  }

  Assert.assertEquals(out, "[struct stat=OK]");
}



@Test
public void  testRetrieveArgsBg() {
  // Retrieve image.tbl, corrections.tbl and backgroud arguments

  String returnStringImages = null,returnStringCorrections = null;
  ArrayList<String> IdFileTbl,corrTbl;

  SSHHandler ssh = new SSHHandler("atps.cps.unizar.es","eburges","passwd");
  try {
    ssh.connect();
    returnStringImages = ssh.copyStreamFrom("/home/eburges/m101/images.tbl");
    returnStringCorrections = ssh.copyStreamFrom("/home/eburges/m101/corrections.tbl");
    ssh.disconnect();
  } catch (Exception e) {
    e.printStackTrace();
    Assert.fail();
  }
  IdFileTbl=Montage.IdFileTbl(returnStringImages);
  corrTbl=Montage.correctionsTbl(returnStringCorrections);
  //return array list to submit to the testCondormDiffFit. Static global variable.
  // Remember, Condor takes the paths from /export
  //                        text file, text file  projdir,                corrdir
  args=Montage.mBackgroundArgs(IdFileTbl, corrTbl, "/home/eburges/m101/projdir/",
      "/home/eburges/m101/corrdir/");
}

@Test
public void testCondormBackground() {
  int results = 0;
  MontageMsg msg;

  CondorFacade cf = new CondorFacade("atps.cps.unizar.es", "eburges","passwd",
        "/home/eburges",
      "env CONDOR_CONFIG=/usr/local/condor/etc/condor_config /usr/local/condor/bin");

  CondorJob job = new CondorJobHandler(cf, "vanilla",
      "/home/eburges/Montage_v3.3/bin/mBackground", "",
      "err",
      "out",
      args.get(0),
      1L);
  try {
    //System.out.println(job.getConfigFile());
    for(int i=1; i<args.size();i++) job.addAttribute(args.get(i));

    System.out.println(job.getConfigFile() + "Trabajos:" + job.getQueueTimes());

    job.queueJob();


    System.out.println("Waiting");
    job.waitUntilIsTerminated();
    System.out.println("Terminated");
    for(int i=1; i<=args.size();i++){
      msg = new MontageMsg(job.getOutputStream(i));
      if ( msg.error()!=true) results++;
      System.out.println((i)+")"+job.getOutputStream(i));
    }

    job.cleanUpWorkSpace();
    cf.disconnect();
```

```
  } catch (Exception e) {
    e.printStackTrace();
    Assert.fail();
  }

  Assert.assertEquals(results, 10);
}

@Test
public void testCondormBgExec() {
  //Apply the background matching to each reprojected image

  CondorFacade cf = new CondorFacade("atps.cps.unizar.es", "eburges","passwd",
        "/home/eburges",
      "env CONDOR_CONFIG=/usr/local/condor/etc/condor_config /usr/local/condor/bin");

  CondorJob job = new CondorJobHandler(cf, "vanilla",
    "/home/eburges/Montage_v3.3/bin/mBgExec",
    "",
    "err",
    "out",
    "-p /home/eburges/m101/projdir /home/eburges/m101/images.tbl
        /home/eburges/m101/corrections.tbl /home/eburges/m101/corrdir",
    1L);
  String out = null;
  try {
    System.out.println(job.getConfigFile() + "Trabajos:" + job.getQueueTimes());

    job.queueJob();

    job.waitUntilIsTerminated();

    String input = job.getInput();
    System.out.println("getInput():"+input);

    out = job.getOutputStream().replace("\"", "").trim();

    System.out.println("Salida getOutputStream():"+out);

    job.cleanUpWorkSpace();
    cf.disconnect();
    } catch (Exception e) {
      e.printStackTrace();
      Assert.fail();
  }

  Assert.assertEquals(out, "[struct stat=OK, count=10, nocorrection=0, failed=0]");
}



@Test
public void testCondormAdd() {
  //Apply the background matching to each reprojected image

  CondorFacade cf = new CondorFacade("atps.cps.unizar.es", "eburges","passwd",
        "/home/eburges",
      "env CONDOR_CONFIG=/usr/local/condor/etc/condor_config /usr/local/condor/bin");

  CondorJob job = new CondorJobHandler(cf, "vanilla",
    "/home/eburges/Montage_v3.3/bin/mAdd",
    "",
    "err",
    "out",
    "-p /home/eburges/m101/corrdir /home/eburges/m101/images.tbl
        /home/eburges/m101/template.hdr /home/eburges/m101/final/m101_mosaic.fits",
    1L);
  String out = null;
  try {
```

```
    System.out.println(job.getConfigFile() + "Trabajos:" + job.getQueueTimes());

    job.queueJob();

    job.waitUntilIsTerminated();

    String input = job.getInput();
    System.out.println("getInput():"+input);

    out = job.getOutputStream().replace("\"", "").trim();

    System.out.println("Salida getOutputStream():"+out);

    job.cleanUpWorkSpace();
    cf.disconnect();
    } catch (Exception e) {
      e.printStackTrace();
      Assert.fail();
  }

  Assert.assertTrue(out.contains("[struct stat=OK, time="));
}

@Test
public void testCondormJPEG() {
  //Apply the background matching to each reprojected image

  CondorFacade cf = new CondorFacade("atps.cps.unizar.es", "eburges","passwd",
        "/home/eburges",
      "env CONDOR_CONFIG=/usr/local/condor/etc/condor_config /usr/local/condor/bin");

  CondorJob job = new CondorJobHandler(cf, "vanilla",
    "/home/eburges/Montage_v3.3/bin/mJPEG",
    "",
    "err",
    "out",
    "-gray /home/eburges/m101/final/m101_mosaic.fits 0s max gaussian-log -out
        /home/eburges/m101/final/m101_mosaic.jpg",
    1L);
  String out = null;
  try {
    System.out.println(job.getConfigFile() + "Trabajos:" + job.getQueueTimes());

    job.queueJob();

    job.waitUntilIsTerminated();

    String input = job.getInput();
    System.out.println("getInput():"+input);

    out = job.getOutputStream().replace("\"", "").trim();

    System.out.println("Salida getOutputStream():"+out);

    job.cleanUpWorkSpace();
    cf.disconnect();
    } catch (Exception e) {
      e.printStackTrace();
      Assert.fail();
  }

  Assert.assertTrue(out.contains("[struct stat=OK"));
}


@Test
public void testgetJPEG() {
  SSHHandler ssh = new SSHHandler("atps.cps.unizar.es","eburges","passwd");
  try {
```

```
      ssh.connect();
      ssh.copyFrom("/home/eburges/m101/final/m101_mosaic.jpg","m101_mosaic.jpg");
      ssh.disconnect();

      Montage.openJPEG("m101_mosaic.jpg");

    } catch (Exception e) {
      e.printStackTrace();
      Assert.fail();
    }
  }
}
```