|  |  |
| --- | --- |
| Universidad Nacional del Centro  de la Provincia de Buenos Aires  Facultad de Ciencias Exactas |  |

**Introducción a Grid Computing**

Integrantes:

Pérez Fuentes, Joaquín Alejandro [joaquinpf@gmail.com](mailto:rodrigocavo@gmail.com)

Steimbach, Marcos Maximiliano msteimba@alumnos.exa.unicen.edu.ar

## Introduccion

Se plantea como objetivo la inclusion de la plataforma Condor en el proyecto JGRIM.

La meta consiste en realizar una solucion acorde al diseño actual de JGRIM, proveyendo una manera simple y univoca de invocar ejecutables en un cluster Condor, obtener su respuesta y brindar mecanismos de configuracion al usuario.

## Desarrollo de la solucion

### Diseño de la solución

Partiendo de la premisa de obtener una solucion simple y que respete los estandares de diseño actuales de JGRIM, se comenzo haciendo un analisis de la plataforma para identificar partes reutilizables del diseño del modulo *parallelization*.

Esto resulto en que el esquema actual (propuesto por IbisServer, IbisClient, IbisMethods, IbisExecutionRequest, IbisResult e IbisInterceptor) podia adaptarse sin mayor problemas a Condor.

En este esquema, el usuario debe generar primero la instanciacion particular de IbisMethods (tanto interface como implementacion), junto con su correspondiente XML para la ejecucion de Spring. El usuario entonces corre uno de los metodos definidos en IbisMethods. Estos son capturados por el interceptor, que genera el cliente pertinente y envia la solicitud de ejecucion al servidor, para luego aguardar por la respuesta de este y devolversela al usuario.

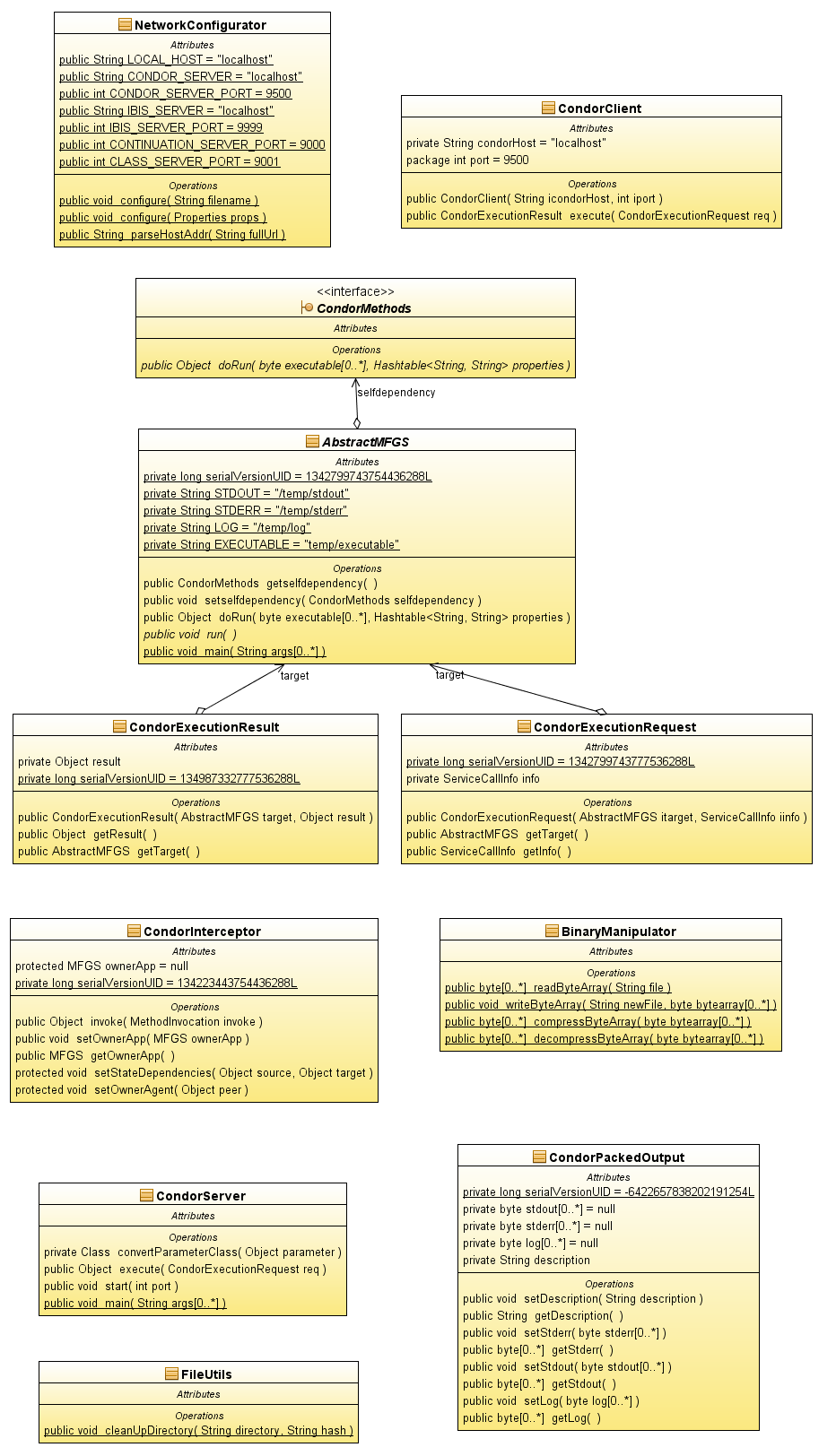
A simple vista las diferencias con el esquema actual son:

* Necesidad de Condor de serializar y enviar el ejecutable en el pedido de invocacion.
* Menor cantidad de puntos de configuracion dado que Condor no esta atado al lenguaje y simplemente corre “ejecutables”en un universo dado. No existe necesidad de generar la interface “IbisMethods”para cada aplicacion en particular ya que siempre se terminara corriendo el mismo metodo. De la misma manera, ya no es necesario generar un XML particular para Spring, este sera provisto por el framework. De este modo se le deja al usuario unicamente la carga del ejecutable y sus opciones de invocacion.

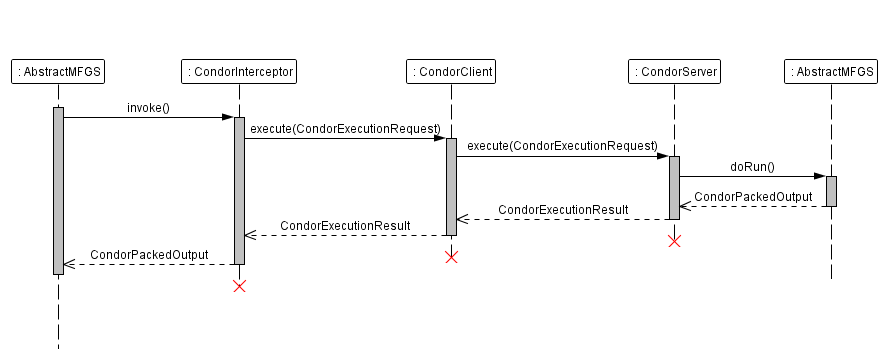
Una linea de ejecución de la aplicación deberia ser:

*El usuario carga el ejecutable y opciones de ejecucion de condor en memoria e invoca el metodo de ejecución de Condor. Este ultimo sera capturado por Spring y enviado como solicitud al servidor, que realizara la ejecución real segun la configuración de Condor que posea y las opciones enviadas por el usuario. Luego de ejecutarse, se deben obtener los archivos resultantes, serializarlos y empaquetarlos, para entonces devolver normalmente la respuesta al usuario.*

El diseño que soporta todo esto puede verse reflejado en el siguiente diagrama de clases.



Debe notarse que el metodo “run()”de la clase AbstractMFGS es el unico hook que provee la implementacion de Condor sobre JGRIM. En este metodo se deberan cargar el ejecutable (utilizando la clase BinaryManipulator) y las opciones de ejecucion de Condor, definidas como pares “clave-valor”validos para los “submit description” de Condor.

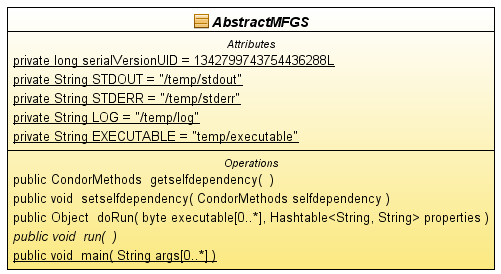
La interacción entre las clases para un curso tipico de la aplicación puede apreciarse en el siguiente diagrama de secuencia:

Cabe destacar que el diagrama anterior representa una versión simplificada de la linea de ejecucion real, mostrando de manera simple la presencia de Spring en CondorInterceptor, pero no su modo de trabajo real (no es llamado explicitamente).

La descripción especifica de cada clase sera mostrada a continuacion.

### Detalles de las clases de la solución

#### AbstractMFGS



La clase AbstractMFGS es el corazon de la ejecucion en Condor e implementa el metodo main que será llamado en toda ejecucion Condor.

**Su metodo “doRun(byte[], Hashtable<String,String>)”**se encarga de implementar el pedido de ejecucion a Condor. Esto involucra la creacion de un Job Description, objeto que contiene todas las opciones necesarias para la ejecucion de Condor, generar el pedido, esperar y devolver el resultado.

Para trabajar de manera generica los destinos del ejecutable y salidas de Condor se generan de manera opaca al usuario, razon por la cual las opciones “output”, “error”, “log” y “log\_xml”seran ignoradas y generadas por el metodo como la concatenacion de un metodo predeterminado y el hash del objeto invocador. De esta manera podemos obtener un mejor control de la ejecucion y evitar problemas en ejecuciones concurrentes que puedan llegar a sobreescribir los mismos archivos. Se debe tener en cuenta todo esto para luego eliminar los archivos temporales del servidor.

La espera de resultado se realiza de manera no bloqueante verificando el estado de la ejecucion cada cierto periodo de tiempo.

Los valores de retorno esperables dentro de la descripcion en CondorPackedOutput son:

***"Execution failed with (Exception)"*** : Indica el fallo de la ejecucion con el nombre de Exception dado.

***"Execution complete"*:** Indica que la ejecución fue completada con éxito.

***"Null parameters recieved in AbstractMFGS.doRun(...) (either executable or properties)"*** : Indica que no se llego a ejecutar Condor por errores en los parametros recibido.

De manera simplificada podemos ver al metodo como:

**byte**[] toRun = BinaryManipulator.*decompressByteArray*(executable);

BinaryManipulator.*writeByteArray*(localExecutable, toRun);

// Condor init

Condor.*setDebug*(**true**);

Condor condor = **new** Condor();

JobDescription jd = **new** JobDescription();

Enumeration<String> auxEnum = properties.keys();

// Add executable, stderr, stdout and logging to job description

jd.addAttribute("executable", localExecutable);

jd.addAttribute("output", localStdout);

jd.addAttribute("error", localStderr);

jd.addAttribute("log\_xml", "True");

jd.addAttribute("log", localLog);

// Si el universo no fue enviado como parametro se utiliza vanilla

**if**(properties.get("universe") == **null**){

jd.addAttribute("universe", "vanilla");

}

// Agregar los parametros especificos del usuario

**while** (auxEnum.hasMoreElements()) {

String key = auxEnum.nextElement();

**if** (key != "executable" && key != "output"

&& key != "error" && key != "log"

&& key != "log\_xml")

jd.addAttribute(key, properties.get(key));

}

jd.addQueue();

// Enviar el trabajo y esperar a que condor finalice

Cluster cluster = condor.submit(jd);

Job j = cluster.getJob(0);

**while** (!j.isCompleted()) {

**try** {

Thread.*sleep*(1000);

} **catch** (InterruptedException e1) {

e1.printStackTrace();

}

}

String result = "Execution complete";

CondorPackedOutput c = **new** CondorPackedOutput();

c.setDescription(result);

c.setStderr(BinaryManipulator.*readByteArray*(*STDERR*));

c.setStdout(BinaryManipulator.*readByteArray*(*STDOUT*));

c.setLog(BinaryManipulator.*readByteArray*(*LOG*));

FileUtils.*cleanUpDirectory*("temp/", hash);

**return** c;

**El metodo “run()”**de la clase AbstractMFGS es el unico hook que provee la implementacion de Condor sobre JGRIM. En este metodo se deberan cargar el ejecutable (utilizando la clase BinaryManipulator) y las opciones de ejecucion de Condor, definidas como pares “clave-valor”validos para los “submit description” de Condor.

Valores importantes a tener en cuenta son:

* **universe-standard/vanilla/java**: Define el universo de ejecucion de Condor. Si se quiere correr un binario Java (class) se deberia definir “java”como el universo. De manera similar, si se desea correr un ejecutable standard de consola de Windows/Dos se puede utilizar el universo “vanilla”. Universe debe pasarse como parametro, caso contrario se utilizara vanilla.

Para mantener la generacion de trabajos sencilla, no se tendran en cuenta los valores (seran utilizados valores por defecto):

* **output**: Define la ubicacion del archivo que contendra la salida estandar de la ejecución de Condor.
* **error**: Define la ubicacion del archivo que contendra la salida estandar de errores de la ejecución de Condor.
* **log**: Define la ubicacion del archivo de log de la ejecución de Condor.
* **log\_xml**: Define si se utilizan logs con formato XML.

Un pequeño ejemplo de la utilizacion de run() se ve a continuacion:

//Carga y comprime el ejecutable en memoria

**byte**[] b = BinaryManipulator.*readByteArray*("EJECUTABLE");

b = BinaryManipulator.*compressByteArray*(b);

CondorPackedOutput result;

//Carga las opciones particulares para setearle a condor

Hashtable<String, String> h = **new** Hashtable<String, String>();

h.put("universe", "vanilla");

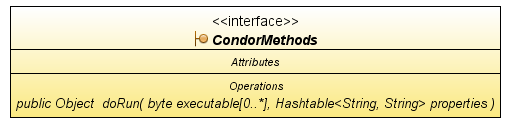
CondorMethods selfDep = getselfdependency();

**if** (selfDep != **null**) {

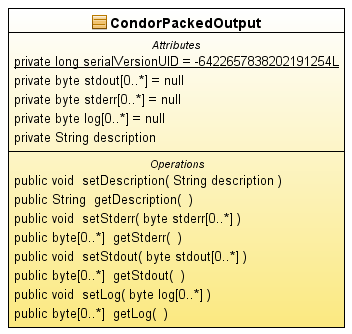
CondorPackedOutput result = (CondorPackedOutput) selfDep.doRun(b, h)

}

#### CondorMethods



#### CondorPackedOutput



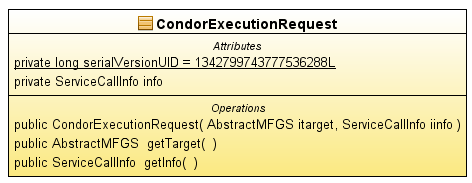
La clase CondorPackedOutput actua de paquete para los diversos archivos generados en la ejecucion de un programa sobre Condor. Estos archivos son:

* Salida de stdout
* Salida de stderr
* Log

Cada uno de estos archivos se serializa con la ayuda de BinaryManipulator y se setea como propiedad a la clase.

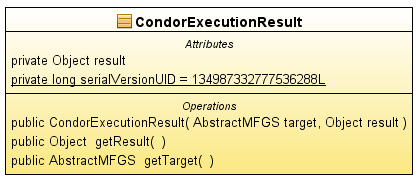
Existe ademas un String que contiene la descripcion pertinente al resultado de la operacion, tanto si fue exitosa como si fallo.

#### CondorExecutionRequest



La clase CondorExecutionRequest encapsula todo lo necesario para realizar un pedido de ejecucion a un CondorServer. Su funcionamiento es muy similar al de IbisExecutionRequest de JGRIM con la unica salvedad de que la aplicacion serializada es en este caso una instancia de la implementacion de AbstractMFGS. Ademas de esto lleva los datos del metodo a ejecutar, junto con sus parametros para la invocacion en el servidor.

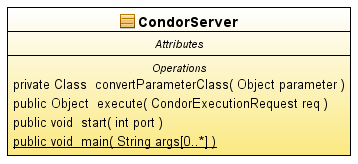
#### CondorExecutionResult



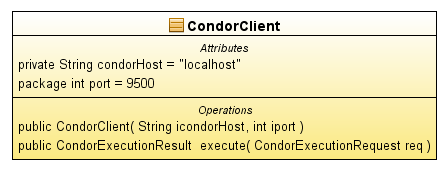
De manera analoga a CondorExecutionRequest, esta clase realiza el camino inverso siendo la encargada de encapsular una respuesta derivada de un pedido de ejecucion en condor.

El objeto resultado contenido en la clase es una instancia de CondorPackedOutput.

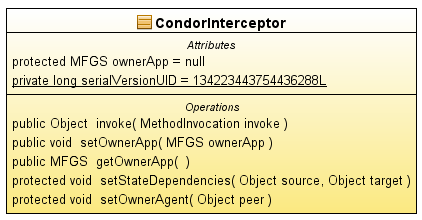
#### CondorServer



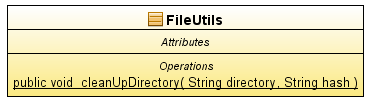
#### CondorClient



#### CondorInterceptor



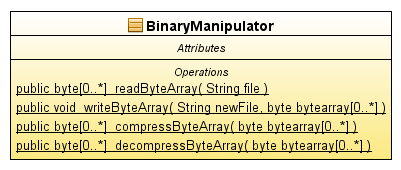
#### FileUtils



La clase FileUtils provee funciones para el manejo de archivos en disco.

El metodo “cleanUpDirectory” sera el encargado de limpiar un directorio dado, teniendo en cuenta un hash en el nombre de archivos, luego de finalizada una ejecucion de condor. Recordemos que tendremos archivos guardados en disco como salida de stdout, stderr, log y ejecutable, los cuales luego de la ejecucion ya no son necesarios.

#### BinaryManipulator



La clase BinaryManipulator se encarga de proveer los metodos necesarios para convertir archivos a byte[], comprimirlos, descomprimirlos o escribir un byte[] a un archivo dado del disco.

Esta clase se utiliza para levantar y comprimir en memoria los archivos necesarios para un pedido de ejecucion de Condor, o su respuesta.

#### NetworkConfigurator

