|  |  |
| --- | --- |
| 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 sencillos 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.

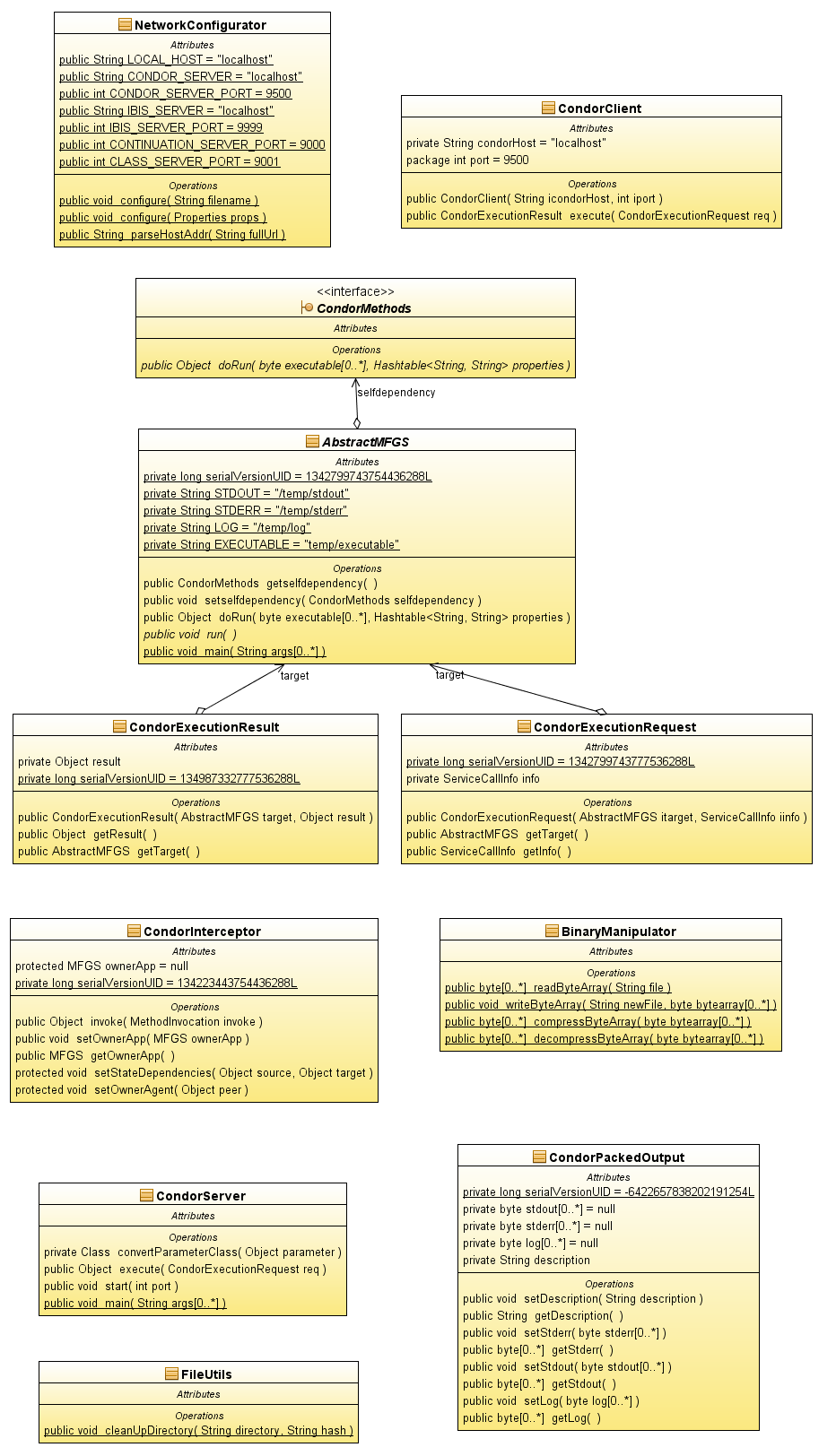
A simple vista las diferencias de Condor con el esquema actual son:

* 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 (aunque puede requerirse) 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.

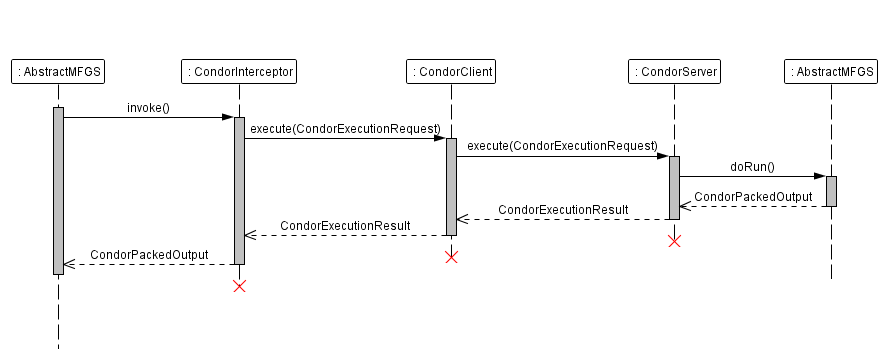
Una linea de ejecución de la aplicación deberia ser:

*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.

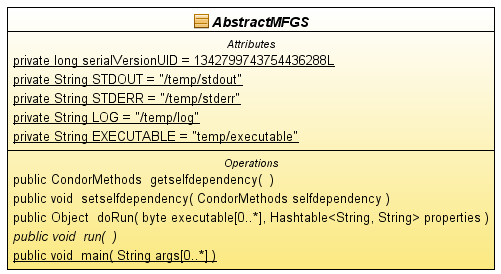
La interacción entre las clases para un curso tipico de la aplicación puede apreciarse en el siguiente diagrama de secuencia:

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 y archivos de configuracion 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. Cabe destacar que este metodo hace referencia al archivo simple-condor.xml y esta configurado para instanciar la clase CondorTest. Para cambiar esto y ejecutar cualquier clase implementada por el usuario, se debe editar la linea siguiente del archivo simple-condor.xml:

<bean id="service" class="test.parallelization.condor.CondorTest">

**Para configuraciones avanzadas, o utilizacion de otro XML o varios XML, debe generarse un nuevo metodo main de la misma manera que el actual, referenciando al XML especifico.**

Un ejemplo de esto es:

**public** **static** **void** main(String[] args) **throws** Exception {

NetworkConfigurator.*configure*("configuration/network.properties");

PropertyConfigurator.*configure*("configuration/log4j.properties");

ApplicationContext appContext = **new** FileSystemXmlApplicationContext(

"configuration/simple-condor.xml");

Runnable service = (Runnable) appContext.getBean("service");

service.run();

}

**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");

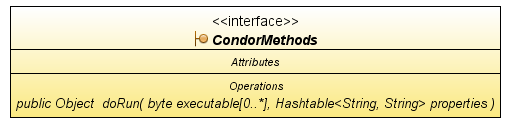
CondorMethods selfDep = getselfdependency();

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

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

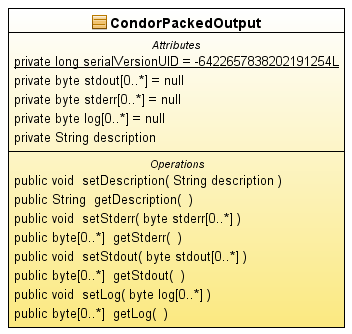
}

#### CondorMethods



La interface CondorMethods define la signatura de los metodos que pueden ejecutarse mediante Spring. En nuestro caso, el unico metodo disponible es doRun(byte[], Hashtable<String,String>) que genera la ejecucion Condor del ejecutable pasado como bytearray. Este metodo esta implementado en AbstractMFGS.

#### CondorPackedOutput



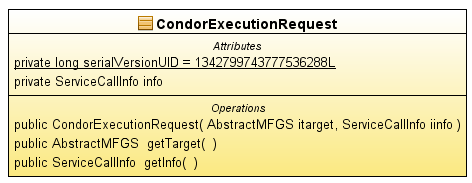
La clase CondorPackedOutput actua de paquete para los diversos archivos generados en la ejecucion de un programa sobre Condor. Estos archivos son:

* 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.

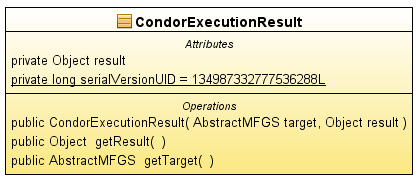
Existe ademas un String que contiene la descripcion pertinente al resultado de la operacion, tanto si fue exitosa como si fallo.

#### 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.

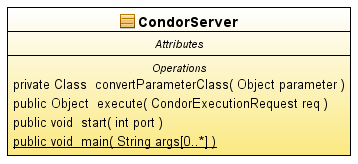
#### CondorExecutionResult



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



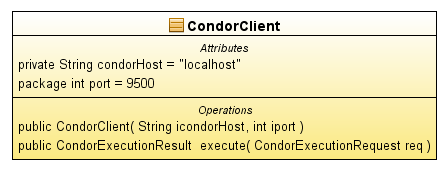
Esta clase tiene por objetivo implementar el lado servidor de una ejecucion Condor.

Al arrancar se abre un ServerSocket segun la configuracion especificada por NetworkConfigurator, para luego esperar hasta recibir un CondorExecutionRequest.

El metodo **execute(CondorExecutionRequest)** se encarga de, dado el CondorExecutionRequest recibido, desempaquetar las opciones y el ejecutable, para luego invocar el metodo adecuado, que llega como un ServiceCallInfo incluido en el pedido de ejecucion.

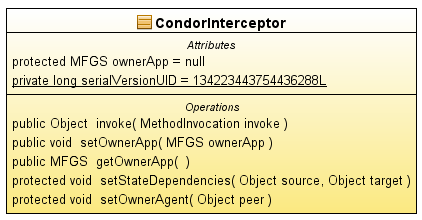
Luego de esto debe simplemente esperar por la respuesta del metodo, empaquetarla en un CondorExecutionResult y devolver el resultado.

#### CondorClient



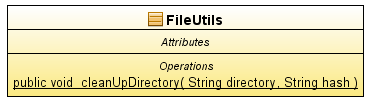
El proposito de la clase CondorClient es la de manejar el lado cliente de una ejecucion Condor. Esto significa que es la encargada de, dado un CondorExecutionRequest, enviar el pedido de ejecucion al servidor especificado en NetworkConfigurator y aguardar la respuesta

#### CondorInterceptor



Esta clase se encarga de interceptar la ejecucion de los metodos definidos en CondorMethods, ejecutados desde la implementacion de AbstractMFGS como “selfdependency.method()”. Sus funciones principal son las de generar el pedido de ejecucion de Condor (CondorExecutionRequest) e instanciar el un CondorClient que sera alimentado con dicha solicitud.

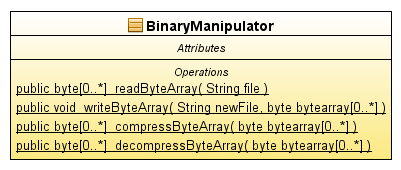
#### FileUtils



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.

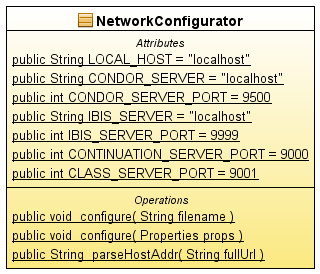
#### BinaryManipulator



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



La clase NetworkConfigurator maneja la configuracion de red para las diferentes plataformas habilitadas y trabaja como punto de acceso unico para su utilizacion.

Hace uso de los diferentes parametros seteados en el archivo network.properties, como pueden ser la direccion del server Condor, o el puerto a utilizar.

Los parametros a tener en cuenta de dicho archivo son:

* condor.server: El server al que se debe delegar la ejecucion de Condor (ej: localhost, 192.168.1.123).
* condor.server.port: El puerto al que se debe conectar el cliente para delegar (ej: 9500).

#### xml de configuracion para spring

El siguiente XML define las propiedades necesarias para Spring. Estas son: el interceptor, los metodos que maneja (por la interface CondorMethods) y la clase que se ejecutara con este. Este ultimo deberia ser el unico punto que el usuario necesitaria cambiar, como se explico en la seccion de AbstractMFGS.

Un ejemplo completo del XML se muestra a continuacion:

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:util="http://www.springframework.org/schema/util"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.0.xsd http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-2.0.xsd">

<bean id="service" class="test.parallelization.condor.CondorTest">

<property name="selfdependency"><ref local="selfdependency"/></property>

</bean>

<!-- Parallelism -->

<bean id="selfdependency" class="org.springframework.aop.framework.ProxyFactoryBean">

<property name="proxyInterfaces"><value>core.parallelization.condor.CondorMethods</value></property>

<property name="interceptorNames">

<list>

<value>condorInterceptor</value>

</list>

</property>

</bean>

<bean class="core.parallelization.condor.CondorInterceptor" id="condorInterceptor">

<property name="ownerApp"><ref bean="service"/></property>

</bean>

</beans>

## Ejemplo de Caso de prueba

En el paquete “test.parallelization.condor” se presenta un caso de prueba de la aplicacion, la clase “CondorTest”. Veamos como se forma esta clase.

En primer lugar, cualquier aplicacion que desee utilizar Condor debe extender AbstractMFGS.

**public** **class** CondorTest **extends** AbstractMFGS

Luego de hacer eso hay dos puntos que se deben tomar en cuenta. El primero y mas sencillo consiste en generar la constante serialVersionUID. Esto puede realizarse de forma automatica con Eclipse y es mandatorio para la serializacion de la clase.

Un ejemplo de esto es:

/\*\* The Constant serialVersionUID. \*/

**private** **static** **final** **long** *serialVersionUID* = 13498730928756288L;

Por ultimo, debemos implementar la corrida de Condor. Para realizar esto debemos implementar el metodo abstracto run() de AbstractMFGS. En este metodo se deben cargar el ejecutable y las opciones de invocacion que deben ser pasadas a Condor. Un ejemplo sencillo puede verse a continuacion:

@Override

**public** **void** run() {

//Carga del ejecutable en memoria

**byte**[] b = BinaryManipulator.*readByteArray*("x264.exe");

//Comprime el ejecutable

b = BinaryManipulator.*compressByteArray*(b);

CondorMethods selfDep = getselfdependency();

CondorPackedOutput result;

//Setea las propiedades de ejecucion para Condor

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

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

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

//Ejecucion y captura de resultados

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

System.*out*.println(result.getDescription());

}

}

En este ejemplo, se carga el ejecutable “x264.exe” y se setea la propiedad “universe” con el valor “vanilla”.

El resultado de la ejecucion produce un CondorPackedOutput. En este caso se obtiene su descripcion y se la imprime por la salida standard.

Vale destacar que Condor, de estar bien configurado, puede correr class Java con el universo “java”, u obj de C, entre tantas opciones.

En el ejemplo anterior se utilizo el XML por defecto, sin cambios, corriendo el metodo “main” provisto por AbstractMFGS, tambien sin modificaciones.

## Conclusion

Se logro el objetivo de incluir Condor en la plataforma JGRIM, manejado de forma sencilla y sin romper los esquemas ya sentados por JGRIM. El usuario puede, por medio de la implementacion de un unico metodo, invocar distintos ejecutables de distinto origen sobre distintas configuraciones particulares de Condor.

Existen puntos que probablemente debieran ser modificados para realizar de manera mas generica la inclusion de otras plataformas. Un ejemplo claro de esto es la clase NetworkConfigurator, que pese a funcionar correctamente, esta muy atada a Ibis y Condor.