

Índice

Configurar Cobertura en el NetBeans.....	1
Instrumentalizar el código.....	2
Ejecutar los tests.....	3
Crear los informes de cobertura.....	4
Configurar el Hudson.....	7
Instalar Cobertura.....	7
Instalar el plugin.....	7
Configurar el job.....	7

Una vez tenemos montado un [entorno TDD](#), y si efectivamente estamos trabajando con esta orientación, nuestro proyecto empezará a acumular tests unitarios con [JUnit](#) o cualquier otro framework lo que permitirá a nuestro servidor de integración (Hudson) utilizarlos como [tests de regresión](#).

Si estamos siendo estrictos en la aplicación de una [metodología TDD](#), en principio el grado de cobertura del código (el porcentaje de líneas de código que son evaluadas por un test) debería estar cerca del 100% por aquello del [“escribe primero el test y luego el código que lo supera”](#). Sin embargo bien porque seamos principiantes aplicando la metodología o bien porque escribamos los tests como parte del proceso de calidad en una metodología diferente, puede resultar muy complicado calcular el grado de cobertura mediante técnicas manuales. Incluso en un entorno TDD maduro, la propia complejidad del software puede complicar extraer esta información. Es por ello que se han desarrollado una serie de frameworks y herramientas que permiten automatizar este proceso.

Como decía, no estamos solos ante el peligro y tenemos herramientas diversas y con todo tipo de licencias. Por citar algunas: [Clover](#), [EMMA](#), [VectorCAST](#) o [Cobertura](#). Mi elección en este caso será la última por varios motivos: es un producto open source (por los que tengo debilidad, especialmente cuando uno es autónomo), ofrece *tasks* para usarlo desde Ant y tiene plugins para Hudson. Es una lástima que no tenga un plugin nativo para NetBeans, aunque como el IDE nos permite ejecutar *targets* de Ant fácilmente, se lo podemos perdonar.

Por cierto, existe un [plugin](#) de análisis de cobertura de código para NetBeans. No he jugado mucho con él, pero parece que es un servicio que sólo puede activarse para proyectos de JSE y no está disponible, por ejemplo, para un proyecto web. Si alguien tiene experiencia con el mismo, se agradecerá el feedback.

Gran parte de la información necesaria que he usado para escribir esta entrada la he obtenido de [este post](#).

Configurar Cobertura en el NetBeans

El proceso es sencillo:

1. descargar el software,
2. instalarlo,
3. modificar el *build.xml* del NetBeans.

Así pues, lo primero que tenemos que hacer es [descargarnos](#) el software desde SourceForge. Nosotros vamos a contentarnos con la versión binaria y dejaremos el hacking para otro momento. A continuación procederemos a la instalación que no es más que descomprimirlo en una carpeta de la máquina de



desarrollo. Yo tengo la costumbre de instalar estas cosas en el directorio `/srv` y además suelo crear un enlace simbólico (que será el que referencie desde los proyectos) que no incluya el número de versión para que cada vez que haga una actualización no tenga que recorrer todos los scripts para corregir la referencia.

```
root@hargon:/srv# tar -zxf cobertura-1.9.1-bin.tar.gz
root@hargon:/srv# ln -s cobertura-1.9.1 cobertura
root@hargon:/srv# ls -alh
total 684K
drwxr-xr-x  7 root      root      4,0K 2009-02-07 13:42 .
drwxr-xr-x 21 root      root      4,0K 2009-02-07 11:23 ..
lrwxrwxrwx  1 root      root        15 2009-02-07 13:42 cobertura -> cobertura-1.9.1
drwxr-xr-x  4 root      root      4,0K 2009-02-04 21:23 cobertura-1.9.1
-rw-r--r--  1 root      root     649K 2009-02-04 21:40 cobertura-1.9.1-bin.tar.gz
root@hargon:/srv# rm cobertura-1.9.1-bin.tar.gz
```

Como [ya he explicado](#) en alguna ocasión, NetBeans usa scripts de Ant para realizar todas sus tareas y están modularizados de manera que podamos introducir nuestros propios *targets* y personalizaciones. En particular el fichero que podemos “tocar” tranquilamente es el *build.xml* que está en la raíz del proyecto así que, mientras no diga lo contrario, todos los fragmentos de código deberán picarse en este fichero.

Lo primero que tendremos que hacer proporcionar a Ant los diferentes *tasks* proporcionados por Cobertura.

```
<path id="cobertura.classpath">
  <fileset dir="${cobertura.dir}">
    <include name="cobertura.jar" />
    <include name="lib/**/*.jar" />
  </fileset>
</path>

<taskdef classpathref="cobertura.classpath" resource="tasks.properties" />
```

Esto es uso estándar de Ant. La ruta a la raíz del frameworks lo indicamos en la propiedad *cobertura.dir* que a su vez cargamos del fichero de propiedades privadas. Usamos el fichero de propiedades privadas (*nbproject/private/private.properties*) porque la ruta del framework tiene sentido que pueda ser diferente en las máquinas de desarrollo y en el servidor de integración y, recordemos, si tenemos bien montado el repositorio de código el directorio *nbproject/private* no suele incluirse en el mismo. Para que todo esto funcione deberemos cargar desde el *build.xml* directamente dicho fichero de propiedades:

```
<property file="nbproject/private/private.properties"/>
```

y efectivamente añadir la propiedad en *nbproject/private/private.properties*:

```
cobertura.dir=/srv/cobertura
```

Los pasos para usar Cobertura son los siguientes:

1. instrumentalizar el código bajo testing,
2. ejecutar los tests unitarios con JUnit que genera los datos de cobertura y
3. parsear los datos de cobertura para generar informes.

Instrumentalizar el código

Cobertura (y la mayoría de de frameworks de este tipo) funciona de manera que al ejecutar los tests unitarios se analiza qué parte de las clases bajo testing están siendo accedidas por las pruebas y qué partes no y se vuelca esta información en un archivo. Para que el framework de testing (en nuestro caso JUnit) pueda obtener esta información, Cobertura habrá tenido que instrumentalizar previamente las clases bajo test. Es decir, antes de ejecutar los tests, habremos tenido que permitir a Cobertura que genere un

bytecode modificado (instrumentalizado) para cada una de las clases bajo testing. Es un proceso similar a como funcionan algunos *profilers*. Para obtener las clases instrumentizadas lo haremos de la siguiente manera:

```
<target name="cobertura-instrument" depends="compile">
  <cobertura-instrument todir="${cobertura.classes.dir}"
    datafile="${cobertura.ser.file}" >
    <fileset dir="${build.classes.dir}">
      <include name="**/*.class"/>
    </fileset>
  </cobertura-instrument>
</target>
```

Creo que el *target* se entiende bastante fácilmente: básicamente estamos especificando donde guardaremos las las clases instrumentizadas (*cobertura.classes.dir*), en qué fichero se escribirán los datos de cobertura (*cobertura.ser.file*) y dónde residen las clases a instrumentizar (*build.classes.dir*). Esta última propiedad es una de las propiedades estándar que residen en el *nbproject/properties*; las otras dos deberemos introducirlas en dicho fichero.

```
cobertura.ser.file=${build.dir}/cobertura.ser
cobertura.classes.dir=${build.dir}/cobertura/classes
```

Como nuestro *target* requiere los *bytecode* de las clases a instrumentizar tiene como dependencia el *target* estándar *compile* que es el que usa NetBeans para compilar el proyecto. Entre las dependencias del propio *compile* se encuentra el *target* *init* que, entre otras cosas, es el encargado de cargar el fichero de propiedades (y por eso nosotros no tenemos que hacer de manera explícita la carga del mismo).

Ejecutar los tests

El siguiente paso consiste en ejecutar los tests sobre las clases instrumentizadas para así extraer la información de cobertura. Aquí también podemos aprovecharnos de los scripts generados por NetBeans. Nuestro nuevo *target*, que llamaremos *test-cobertura* básicamente será una copia del *target* *test* (que es el que normalmente usa NetBeans para ejecutar las diferentes tests unitarios) con pequeñas modificaciones. Esta es la pinta que debería tener:

```
<target name="cobertura-test"
  depends="set-cobertura-file, init, compile-test,
  -pre-test-run, cobertura-instrument,
  -do-test-run, test-report, -post-test-run, -test-browse">
</target>
```

Con respecto al original, hemos añadido dos nuevas dependencias. Por un lado *cobertura-instrument* que es una referencia al *target* anterior para asegurarnos que existen las clases instrumentizadas. Por otro lado hemos definido otra dependencia a otro *target* que también debemos construir y que muestro a continuación:

```
<target name="set-cobertura-file" depends="init">
  <property name="test-sys-prop.net.sourceforge.cobertura.datafile"
    value="${cobertura.ser.file}"/>
</target>
```

Este *target* es necesario para establecer un parámetro de sistema que debemos pasar a todos los tasks de JUnit usados en los diferentes *targets* de los ficheros de scripting incluidos desde el *build.xml* (en particular el *build-impl.xml*). Necesitamos pasar este parámetro para que los *tasks* del JUnit sepan dónde escribir la información de cobertura mientras ejecuta los tests.

El nombre de la propiedad de sistema a pasar es *net.sourceforge.cobertura.datafile* el porqué se pasa el parámetro de sistema en una propiedad con un nombre algo diferente, lo tengo explicado en [este otro post](#). El valor de la propiedad, como ya hemos visto (pues es un propiedad que ya he necesitado en

targets anteriores), está dentro del fichero *nbproject/project.properties* y por eso nuestro *target* tiene una dependencia del *target init* que es el que se encargaba de cargar las propiedades definidas en ese fichero.

Falta un detalle adicional: especificar al JUnit que ejecute las clases instrumentalizadas y no las generadas directamente en la fase de compilación. Para hacer esto sólo tenemos que introducir unos pequeños cambios en el *classpath* utilizado por los *targets* de testing lo que implica cambiar la propiedad *run.test.classpath* definida en el *nbproject/project.properties* para que quede así:

```
run.test.classpath=\
    ${cobertura.dir}/cobertura.jar:\
    ${cobertura.classes.dir}:\
    ${javac.test.classpath}:\
    ${build.test.classes.dir}
```

Si hemos hecho todo correctamente al ejecutar el *target cobertura-test* deberían ejecutarse los tests unitarios como siempre pero además debería haberse construido el fichero *build/cobertura.ser* que contiene la información de cobertura.

Crear los informes de cobertura

El fichero *build/cobertura.ser*, como decíamos, contiene la información de cobertura pero está en formato binario porque ha tenido que generarse en tiempo de ejecución de los tests de manera eficiente. Para extraer información legible hay que parsear dicho fichero. Para ello Cobertura nos proporciona el *task cobertura-report*.

Este *task* permite generar los informes en dos formatos: en *XML* o en *HTML*. Nosotros generaremos ambos. El primero lo necesitaremos para pasárselo al plugin correspondiente en el Hudson. El segundo nos proporcionará una vista agradable y legible. Veamos el *target* que tenemos que definir:

```
<target name="cobertura-report" depends="cobertura-test">
  <cobertura-report
    datafile="${cobertura.ser.file}"
    format="xml"
    destdir="${cobertura.report.dir}"
    srcdir="${src.dir}" />

  <cobertura-report
    datafile="${cobertura.ser.file}"
    format="html"
    destdir="${cobertura.report.dir}"
    srcdir="${src.dir}" />
</target>
```

Creo que aquí no hay mucho que explicar. Defino dónde está el fichero que contiene la información de cobertura, dónde quiero generar los informes y el formato de los mismos. Como siempre habremos de definir la propiedad *cobertura.report.dir* en el *nbproject/project.properties*:

```
reports.dir=${build.dir}/reports
cobertura.report.dir=${reports.dir}/cobertura-report
```

Si ahora ejecutamos este *target* deberían generarse los informes en el directorio especificado. Si le echamos un ojo a los informes de tipo *HTML* vemos que básicamente ha generado unas páginas al estilo de la documentación de las API pero con la información de cobertura. Una manera muy útil e intuitiva de mostrar la información.

Coverage Report - Mozilla Firefox

Archivo Editar Ver Historial Delicioso Marcadores Herramientas Ayuda

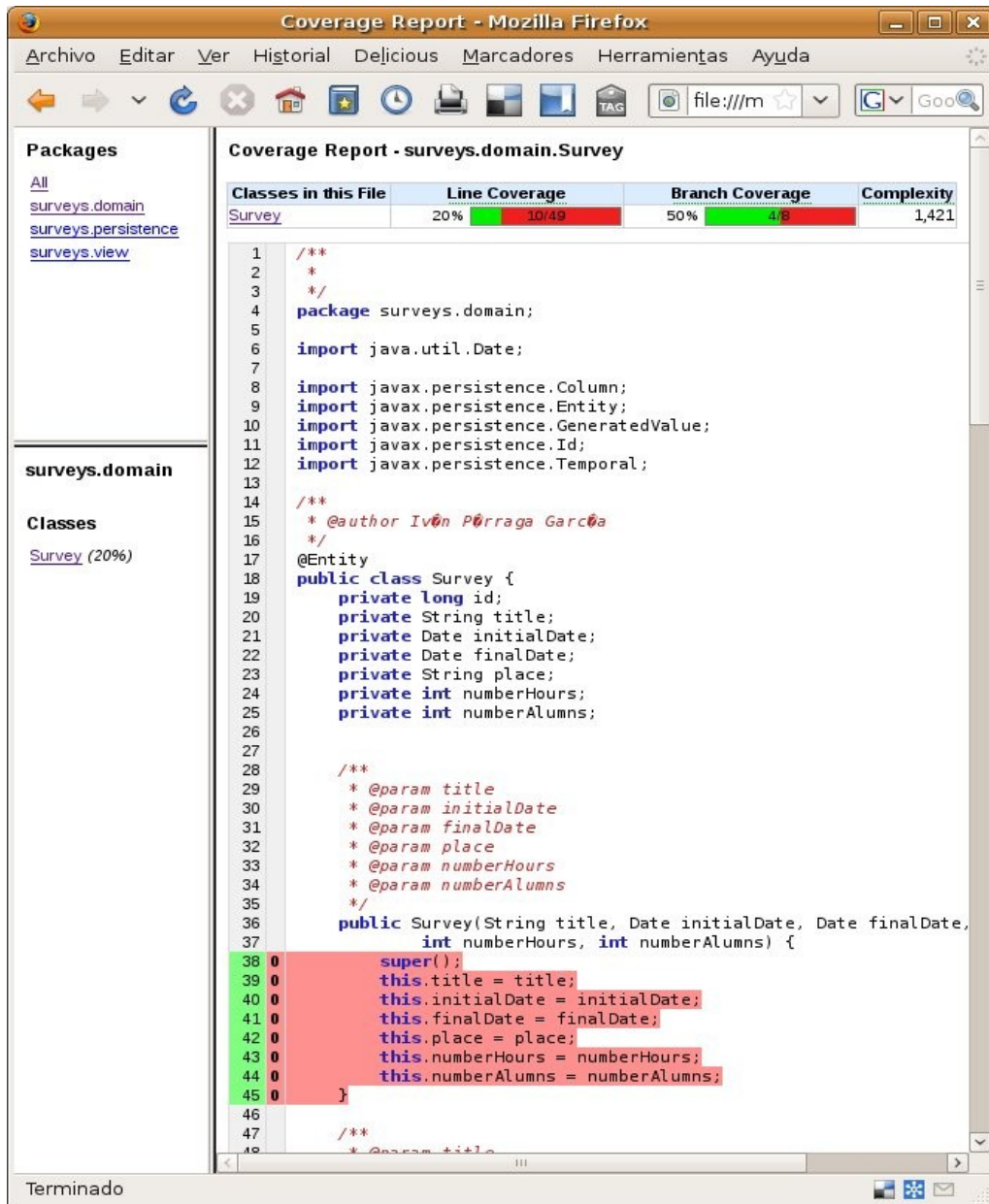
file:///m

Coverage Report - All Packages

Package	# Classes	Line Coverage	Branch Coverage	Complexity
All Packages	4	12% 10/82	16% 4/24	1,655
surveys.domain	1	20% 10/49	50% 4/8	1,421
surveys.persistence	2	0% 0/17	0% 0/8	2,167
surveys.view	1	0% 0/16	0% 0/8	2

Report generated by Cobertura 1.9.1 on 7/02/09 16:29.

file:///mnt/Datos/Documentos/NetBeansProjects/Encuestas/build/reports/cobertura-r...



Coverage Report - surveys.domain.Survey

Classes in this File	Line Coverage	Branch Coverage	Complexity
Survey	20% 10/49	50% 4/8	1,421

```

1  /**
2  *
3  */
4  package surveys.domain;
5
6  import java.util.Date;
7
8  import javax.persistence.Column;
9  import javax.persistence.Entity;
10 import javax.persistence.GeneratedValue;
11 import javax.persistence.Id;
12 import javax.persistence.Temporal;
13
14 /**
15  * @author Iván Párraga García
16  */
17 @Entity
18 public class Survey {
19     private long id;
20     private String title;
21     private Date initialDate;
22     private Date finalDate;
23     private String place;
24     private int numberHours;
25     private int numberAlumns;
26
27
28     /**
29     * @param title
30     * @param initialDate
31     * @param finalDate
32     * @param place
33     * @param numberHours
34     * @param numberAlumns
35     */
36     public Survey(String title, Date initialDate, Date finalDate,
37         int numberHours, int numberAlumns) {
38         super();
39         this.title = title;
40         this.initialDate = initialDate;
41         this.finalDate = finalDate;
42         this.place = place;
43         this.numberHours = numberHours;
44         this.numberAlumns = numberAlumns;
45     }
46
47     /**
48     * @param title

```

Configurar el Hudson

Una vez hemos configurado todo lo anterior, ya tenemos hecho el trabajo duro. La configuración del Hudson, afortunadamente, es bastante sencilla. Estos son los pasos:

1. instalar Cobertura,
2. instalar el plugin de Cobertura y
3. configurar el *job*:
 1. configurar el workspace para indicar dónde está Cobertura,
 2. añadir el *cobertura-report* en la lista de *targets* que tiene que ejecutar el Ant y
 3. configurar el directorio donde están los informes en XML

Instalar Cobertura

Para que el Ant lanzado por Hudson al construir el proyecto sea capaz de instanciar los *tasks* del

Cobertura, éste tendrá que ser instalado en la máquina donde reside Hudson. El proceso de instalación será exactamente análogo al que hemos hecho en el entorno de desarrollo: descargar el paquete, descomprimirlo en el directorio `/srv` y establecer un enlace simbólico.

Instalar el plugin

Una vez tenemos Hudson instalado, añadir plugins es trivial. Desde la interfaz web vamos a **Hudson** -> **Manage Hudson** -> **Manage Plugins** -> **Available** y entonces seleccionamos **Hudson Cobertura plugin** y le damos a **Install**. El asistente entonces instala el plugin tras lo cual habrá que reiniciar Hudson.

Configurar el job

Aquí estamos presuponiendo que ya existe un *job* previo correctamente configurado y al que simplemente vamos a añadir el soporte de cobertura. Si no fuera así, echadle un ojo a este [otro post](#) que ya publiqué en su momento.

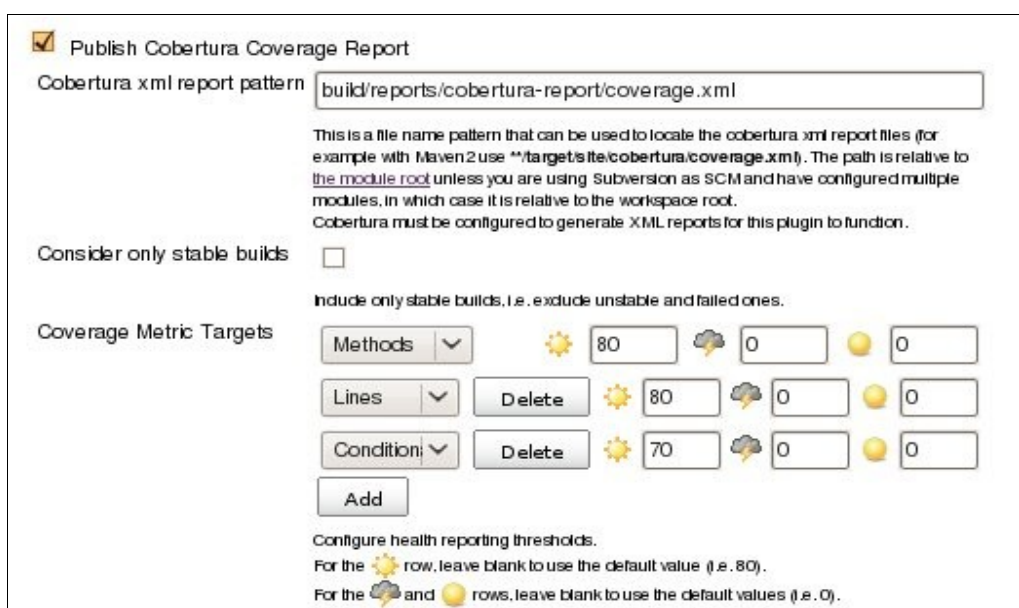
Lo primero que haremos es configurar el *workspace* para que sea capaz de encontrar Cobertura durante la fase de construcción del proyecto. Para ello, nos vamos al directorio `nbproject/private/private.properties` del *job* correspondiente y añadimos la propiedad que indica dicha ruta:

```
cobertura.dir=/srv/cobertura
```

A continuación modificaremos los *targets* de Ant que usa Hudson para construir el *job*. Si tuviéramos el *target test* lo eliminaríamos y a continuación añadiremos el *target cobertura-report*.

Lo siguiente es configurar el plugin para que utilice el informe XML generado:

- **Publish Cobertura Coverage Report:** activamos la opción para que Hudson muestre la información de cobertura.
 - **Cobertura xml report pattern:** `build/reports/cobertura-report/coverage.xml`
Aquí ponemos la ruta al fichero donde hemos generado el informe en formato XML.
 - **Coverage Metric Targets:** este apartado nos permite definir las métricas a partir de las cuales la construcción del *job* se considera estable o no.



Publish Cobertura Coverage Report

Cobertura xml report pattern




This is a file name pattern that can be used to locate the cobertura xml report files (for example with Maven2 use `**/target/site/cobertura/coverage.xml`). The path is relative to the module root unless you are using Subversion as SCM and have configured multiple modules, in which case it is relative to the workspace root. Cobertura must be configured to generate XML reports for this plugin to function.

Consider only stable builds

Include only stable builds, i.e. exclude unstable and failed ones.

Coverage Metric Targets

Methods	<input type="button" value="Delete"/>	<input type="text" value="80"/>	<input type="text" value="0"/>	<input type="text" value="0"/>
Lines	<input type="button" value="Delete"/>	<input type="text" value="80"/>	<input type="text" value="0"/>	<input type="text" value="0"/>
Condition	<input type="button" value="Delete"/>	<input type="text" value="70"/>	<input type="text" value="0"/>	<input type="text" value="0"/>

Configure health reporting thresholds.
For the  row, leave blank to use the default value (i.e. 80).
For the  and  rows, leave blank to use the default values (i.e. 0).

En principio ya está todo. Ahora podríamos lanzar una construcción para comprobarlo. Si todo va bien, Hudson mostrará los informes de cobertura bajo el enlace *Coverage Report* en la raíz del *job*.



Hudson - Mozilla Firefox

Archivo Editar Ver Historial Delicioso Marcadores Herramientas Ayuda

http://192.168.1 netbe

Hudson

Hudson » Encuestas-2 » #49 ENABLE AUTO REFRESH

- [Back to Project](#)
- [Status](#)
- [Changes](#)
- [Console Output](#)
- [Tag this build](#)
- [Test Result](#)
- [Coverage Report](#)**
- [Previous Build](#)

Code Coverage

Coertura Coverage Report

Trend

Name	Packages	Files	Classes	Methods	Lines	Conditionals
Coertura Coverage Report	33% (1/3)	25% (1/4)	25% (1/4)	10% (3/31)	12% (10/82)	17% (4/24)

Coverage Breakdown by Package

Name	Files	Classes	Methods	Lines	Conditionals
surveys.view	0% (0/1)	0% (0/1)	0% (0/5)	0% (0/16)	0% (0/8)
surveys.persistence	0% (0/2)	0% (0/2)	0% (0/7)	0% (0/17)	0% (0/8)
surveys.domain	100% (1/1)	100% (1/1)	16% (3/19)	20% (10/49)	50% (4/8)

Hudson ver. 1.279

Encontrar: [Anterior](#) [Siguiente](#) [Resaltar todo](#) [Coincidencia de may](#)