Trabajo Fin de Grado: Generación automática de tests en DomJudge

Trabajo Fin de Grado Curso 2015/2016

Jaime Boixadós Martínez
Benito Álvaro Cifuentes de la Torre
Yaofang Zhang

Facultad de Informática
Universidad Complutense de Madrid

Junio 2016

Documento maquetado con TEXIS v.1.0+.

Trabajo Fin de Grado: Generación automática de tests en DomJudge

Memoria presentada por
Jaime Boixadós Martínez
Benito Álvaro Cifuentes de la Torre
Yaofang Zhang

Dirigida por el tutor
Miguel Gómez-Zamalloa Gil

Facultad de Informática
Universidad Complutense de Madrid

Junio 2016

Agradecimientos

A todos los que la presente vieren y entendieren.

Inicio de las Leyes Orgánicas. Juan Carlos I

Solo al acabar un proyecto uno es realmente consciente de todo el trabajo que ha costado y quienes te han ayudado a sacarlo adelante. Por eso queríamos dedicar esta sección a todos aquellas personas que nos han acompañado y ayudado, cada uno a su manera, durante el transcurso de este trabajo. A nuestro tutor, Miguel Gómez-Zamalloa, por proporcionarnos acceso y mantenimiento a la herramienta JPET así como guiarnos durante todo el año. A todos nuestros familiares por apoyarnos incondicionalmente y mantener nuestra moral alta. A todos nuestros compañeros de clase que durante tantos años se han preocupado por nosotros y sin los que no habríamos llegado hasta este día. O al menos no habríamos disfrutado tanto de la carrera. Y finalmente queríamos darle las gracias a Marco Antonio Gómez Martín y a Pedro Pablo Gómez Martín, por distribuir la plantilla de una tesis en LaTeX. Que nos ha ayudado mucho a la hora de elaborar nuestra memoria. Muchas gracias a todos por todo. Sin vosotros este proyecto no habría sido posible.

AUTORIZACIÓN PARA LA DIFUSIÓN DEL TRABAJO FIN DE GRADO Y SU DEPÓSITO EN EL REPOSITORIO INSTITUCIONAL E-PRINTS COMPLUTENSE

Los abajo firmantes, alumno/s y tutor/es del Trabajo Fin de Grado (TFG) en el Grado en .....de la Facultad de ....., autorizan a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el Trabajo Fin de Grado (TF) cuyos datos se detallan a continuación. Así mismo autorizan a la Universidad Complutense de Madrid a que sea depositado en acceso abierto en el repositorio institucional con el objeto de incrementar la difusión, uso e impacto del TFG en Internet y garantizar su preservación y acceso a largo plazo.

Periodo de embargo (opcional):

- ☐ 6 meses
- ☐ 12meses

TÍTULO del TFG: .....
.....
Curso académico: 20..... / 20.....

Nombre del Alumno/s:
.....
.....

Tutor/es del TFG y departamento al que pertenece:
.....
.....
.....

Firma del alumno/s Firma del tutor/es

Abstract

The online judges are becoming increasingly important, especially in the field of education. Its operation is simple, a judge / teacher uploads a programming problem with a statement and a test-cases to the online judge. The student / contestant must upload the code they think is right. If the student code returns the same outputs which are in the test-cases for the same inputs, the code works properly.
At the Faculty of computer science of the Complutense University of Madrid has appeared a virtual judge that is being more used by teachers as a complement to the assessment. This online judge is DomJudge. DomJudge is an online judge developed at the University of Utrecht in order to be a judge of virtual programming contests. His code is free and you can download and modify, which makes it ideal if you want to adapt its operation.
It is well known that generate quality test cases is a very complex task. There are several techniques that help the automatic generation of tests. For example, the symbolic execution can generate tests ensuring that all program execution paths are traversed to a certain depth. The aim of this project is to use these testing techniques to generate test cases automatically. For the self-assessment by these judges to be effective, teachers must provide quality test cases that are automatically executed when students upload their solutions. Write these test cases is costly and complex, and it is at this point where the use of Jpet could be very useful. Test cases generated by Jpet could serve as a starting point in generating a set of quality test cases. A very interesting aspect in this regard is that it is not necessary that the student's programs use the Java language. All that would be needed is that the teacher provide a solution coded in Java. From this code jPET can generate initial test cases. However if the solutions provided by the students are written in Java, it could pose more interesting approaches in which the tests are formed from both solutions. The student, to generate input data, and the teacher, to check that the outputs for those entries are correct.
That is why we have carried out not only an automatic translation of the xml files generated in JPET into in and out files. But a full integration with the DomJudge system to generate test cases without intermediaries. With this integration we can facilitate the work of teachers when it comes to create problems in online judges and we achieve an automatic generation of tests

in DomJudge.

Keywords

- JPet
- Domjudge
- SoftTest
- Testing
- Online judge
- Test cases
- XML
- Automatic generation

Resumen

Los jueces online están cobrando cada día más importancia, especialmente en el ámbito de la enseñanza. Su funcionamiento es simple, un juez/profesor sube un problema de programación con un enunciado y unos casos de prueba (entradas y salidas esperadas) al juez online. El alumno/concursante deberá subir el código que considera como solución al problema. Si el código del alumno devuelve las mismas salidas que las que se encuentran en los casos de prueba para las correspondientes entradas en los test-cases dada las mismas entradas, el código se considera correcto.
En la facultad de informática de la Universidad Complutense de Madrid ha aparecido un juez virtual que cada vez está siendo más usado por los docentes como complemento a la hora de evaluar. Este juez online es DomJudge. DomJudge es un juez online desarrollado en la universidad de Utrecht con el fin de ser un juez virtual de concursos de programación. Su código es libre y se puede descargar y modificar, lo que lo hace ideal si se quiere adaptar su funcionamiento.
Es bien sabido que generar casos de prueba de calidad es una tarea muy compleja. Existen diversas técnicas que ayudan a la generación automática de tests. Por ejemplo, la ejecución simbólica permite generar tests garantizando que todos los caminos de ejecución del programa son ejercitados hasta una cierta profundidad. El objetivo de este proyecto es hacer uso de estas técnicas de testing para generar casos de prueba de forma automática.
Para que la autoevaluación realizada por estos jueces sea efectiva, los profesores deben proporcionar casos de prueba de calidad que son ejecutados automáticamente cuando los alumnos suben sus soluciones. Escribir estos casos de prueba resulta costoso y complejo, y es en este punto, donde el uso de jPET podría resultar muy útil. Los casos de prueba generados por jPET podrían servir como punto de partida a la hora de generar un conjunto de casos de prueba de calidad. Un aspecto muy interesante en este sentido, es que no es necesario que los programas de los alumnos utilicen el lenguaje Java. Lo único que sería necesario es que el profesor proporcione una solución escrita en Java. A partir de esta se podrían generar los casos de prueba iniciales. Si se diese el caso de que las soluciones de los alumnos viniesen escritas en Java, se podrían plantear enfoques más interesantes en los cuales los tests se forman a partir de ambas soluciones. La del alumno, para generar

los datos de entrada, y la del profesor, para chequear que las salidas para esas entradas son las correctas.
Por eso hemos llevado a cabo no solo una traducción automática del xml generado por JPET en ficheros in y out. Sino una total integración con el sistema DomJudge para poder generar los casos de prueba sin intermediarios. Gracias a esta integración podemos facilitar el trabajo de los profesores a la hora de crear problemas en jueces online y conseguimos una generación automática de tests en DomJudge.

Palabras Claves

- JPet
- Domjudge
- SoftTest
- Testing
- Juez online
- Casos de prueba
- XML
- Generación automática

Índice

Agradecimientos v

Abstract vii

Resumen ix

1. Introducción 1
- 1.1. Introducción . . . 1
- 1.2. Introduction . . . 4
- En el próximo capítulo . . . 6

2. Testing 7
- 2.1. Introducción al Testing . . . 7
- 2.2. Tipos de Testing . . . 9
  - 2.2.1. Testing de Caja Blanca . . . 10
  - 2.2.2. Testing de Caja Negra . . . 16
- En el próximo capítulo . . . 20

3. Domjudge 21
- 3.1. Introducción . . . 21
- 3.2. Instalación . . . 22
  - 3.2.1. Prerrequisitos . . . 22
  - 3.2.2. Instalación de DomJudge . . . 23
  - 3.2.3. Instalación de las bases de datos . . . 24
  - 3.2.4. Configuración del servidor web . . . 25
  - 3.2.5. Instalación de un host . . . 25
- 3.3. Interfaz web . . . 26
  - 3.3.1. Pantalla principal . . . 26
  - 3.3.2. Panel del administrador . . . 27
  - 3.3.3. Panel del jurado . . . 33
  - 3.3.4. Panel del equipo concursante . . . 35
- 3.4. Guía de utilización de DomJudge . . . 35

3.5. Resultados de las entregas . . . 39
3.6. Conclusión . . . 41
Notas bibliográficas . . . 41
En el próximo capítulo . . . 42

4. jPET **43**
4.1. Introducción . . . 43
4.2. PET . . . 43
4.3. jPET . . . 44
4.4. Conclusión . . . 45
Notas bibliográficas . . . 45
En el próximo capítulo . . . 46

5. Integración de jPET **47**
5.1. Introducción . . . 47
5.2. ¿Cómo funciona? . . . 48
5.3. Las distintas fases de desarrollo . . . 49
5.3.1. Primera fase: pruebas y definición . . . 49
5.3.2. Segunda fase: Traducción xml y ficheros de salida . . . 49
5.3.3. Tercera fase: arrays de int . . . 49
5.3.4. Cuarta fase: String . . . 49
5.3.5. Quinta fase: Depuración de la traducción y vuelta a los arrays . . . 50
5.3.6. Fase final: Preparando la herramienta para Domjudge 50
5.3.7. Parametros jPET . . . 50
5.3.8. Adaptación ficheros in y out . . . 51
5.4. Uso de la herramienta SoftTest . . . 54
5.4.1. Uso de la interfaz Gráfica . . . 54
5.5. Uso de la herramienta a través de la terminal . . . 57
5.6. Recapitulación . . . 58
5.7. Integración . . . 59
5.8. Dificultades encontradas . . . 62
5.9. Conclusión . . . 64
En el próximo capítulo . . . 64

6. Estadísticas **65**
6.1. FP Carreras . . . 65
6.1.1. Explicación . . . 65
6.1.2. Razón de interés . . . 66
6.1.3. Estadística . . . 66
6.1.4. Comentarios . . . 66
6.2. EDA Cima . . . 66

6.2.1. Explicación . . . 66
6.2.2. Razón de interés . . . 67
6.2.3. Estadística . . . 67
6.2.4. Comentarios . . . 67
6.3. EDA SumaBuen . . . 67
6.3.1. Explicación . . . 67
6.3.2. Razón de interés . . . 68
6.3.3. Estadística . . . 68
6.3.4. Comentarios . . . 68
6.4. EDA Paréntesis balanceados . . . 68
6.4.1. Explicación . . . 68
6.4.2. Razón de interés . . . 69
6.4.3. Estadística . . . 69
6.4.4. Comentarios . . . 69
6.5. EDA Vector divisible en pares impares . . . 70
6.5.1. Explicación . . . 70
6.5.2. Razón de interés . . . 70
6.5.3. Estadística . . . 70
6.5.4. Comentarios . . . 70
6.6. EDA Sucesión de naturales . . . 70
6.6.1. Explicación . . . 70
6.6.2. Razón de interés . . . 71
6.6.3. Estadística . . . 71
6.6.4. Comentarios . . . 71
6.7. EDA Complementario . . . 71
6.7.1. Explicación . . . 71
6.7.2. Razón de interés . . . 72
6.7.3. Estadística . . . 72
6.7.4. Comentarios . . . 72
6.8. EDA El hotel Pico . . . 72
6.8.1. Explicación . . . 72
6.8.2. Razón de interés . . . 72
6.8.3. Estadística . . . 73
6.8.4. Comentarios . . . 73
En el próximo capítulo . . . 73

7. Conclusiones y Líneas futuras **75**
7.1. Conclusiones . . . 75
7.2. Conclusions . . . 77
7.3. Líneas futuras . . . 79

8. Contribuciones personales **81**
8.1. Trabajo realizado por Jaime Boixadós Martínez 81
8.2. Trabajo realizado por Benito Álvaro Cifuentes de la Torre . . 83
8.2.1. Testing 83
8.2.2. SoftTest 83
8.2.3. Memoria 85
8.3. Trabajo realizado por Yaofang Zhang 85
8.3.1. DomJudge 86
8.3.2. Estadísticas 86
8.3.3. Memoria 87

Capítulo 1

Introducción

La inteligencia es útil para todo,
suficiente para nada

Amiel, Henri Frèderic

1.1. Introducción

En la actualidad el uso de jueces en línea esta cobrando una mayor relevancia, no tan solo centrándose en el ámbito académico sino también en el competitivo, llegándose a celebrar concursos de programación en estas plataformas.

Un juez en línea consiste en una plataforma online, donde se plantean problemas de programación, cada problema cuenta con un enunciado y tienden a ir acompañados de ejemplos significativos de entrada y salida para que el alumno o el usuario sea consciente de que se esta pidiendo y entienda el problema con una mayor facilidad. Los problemas planteados en estos entornos, suelen contar una complejidad muy reducida a nivel de patrones arquitectónicos y de programación. Desacoplando la dificultad en estas tareas se consigue poder plantear problemas algorítmicos complejos sin necesitar de un solido entorno que haga uso de sus resultados. Una vez se ha planteado un problema, los usuarios suben su código fuente, el cual al ejecutarse lee por la entrada estándar los parámetros de cada problema, y muestra las soluciones del algoritmo siguiendo el patrón indicado por los ejemplos. Una solución subida por el alumno, puede tomar cinco principales resultados, la recomendable que es correcta, la errónea porque devuelve una salida que no coincide con lo que se esperaba que devolviera el algoritmo, compilation error que indica que no se ha podido ejecutar el código correctamente y ha habido un error, y por último time-limit que se da en el caso de que una solución haya sobrepasado un tiempo fijado, ideal para obligar a las soluciones a ser realmente eficientes.

Para este trabajo de fin de grado, se ha optado por la elección del juez online DomJudge. Su uso se ha ido extendiendo debido a versatilidad que tiene para no depender de un único lenguaje y a las facilidades de uso que presenta llegando a ser usado en la Facultad de Informática de la Universidad Complutense de Madrid como acompañamiento y sistema de evaluación de asignaturas fuertemente orientadas a la algoritmia. Los roles para los usuarios permiten una diferenciación de funcionalidades restringiendo la creación de problemas, asi como su revisión. Se hablará con mucho mas detalle del sistema de DomJudge en el Capítulo 3.

A pesar de la relevancia y la importancia del Testing en el mundo de la informática, resulta ser una de las ramas mas abandonadas, contando con escasas menciones durante la carrera, y con casi nulo uso práctico a lo largo de esta. El testing en cualquier entorno resulta una tarea crucial. La capacidad de asegurar que tu sistema cumple correctamente con su funcionalidad, de que además es capaz de soportar entradas no esperadas y mantiene una robustez ante situaciones no esperadas resulta un aspecto de vital importancia, lamentablemente siempre va a acompañado de un alto coste siendo una de las principales tareas a las que se destinan los recursos.

Existen diferentes clases de testing, y muchas de ellas adaptables al entorno de DomJudge, en el próximo capítulo se explicarán las estrategias mas significativas, estableciendo una base en conocimientos de testing que permita entender la solución propuesta.

Una de las dificultades que surgen con el uso de los jueces en línea es la posibilidad de garantizar que todos aquellos problemas que se plantean tengan unos casos de prueba realmente significativos, que verifiquen completamente su corrección. Esta tarea no resulta nada sencilla, y muy difícilmente se podrá asegurar una verificación completa con los casos de prueba.

Para tratar dicho problema, se ha tendido a la generación de casos de prueba manuales,haciendo de esto una tarea muy costosa, y complementando dichos tests manuales con otros generados de manera aleatoria, lo que facilita una creación masiva de entradas para el problema, pero no por ello garantiza que se planteen todas las posibilidades de dicho problema, esta es la principal motivación que surge para la integración de herramientas autogeneradoras de casos de prueba no basadas en la aleatoriedad.

Como se comentará en el capítulo de testing, usando de ejemplo un algoritmo que dado un elemento n calcule el resultado de 1/n, entendemos que fácilmente, para cuando n tome el valor n=0 vamos a encontrarnos ante un comportamiento especial. Haciendo uso de herramientas de generación aleatoria, pudiendo elegir entre todo dominio de posibilidades, muy díficilmente seremos capaces de conseguir este valor. No por ello se descarta el método de generación aleatoria ni se menosprecia, ya que en muchos contextos suele resultar una estrategia realmente efectiva además de servir como soporte y

garantizar por mas de un medio la corrección de los problemas.

Otras soluciones planteadas para la generación de casos de prueba ha sido la de la aplicación de técnicas de testing, consciente o inconscientemente, donde se ha estudiado la naturaleza del problema a resolver, y se han elaborado entradas que apuntan directamente a los puntos conflictivos de cada problema. A pesar de ser una medida mas costosa en relación al esfuerzo invertido, se trata de una solución realmente recomendable si de verdad se quiere asegurar la corrección de la solución planteada.

La solución que planteamos para solventar el problema de la generación de casos de prueba, es la obtencioón de estos de forma automática, basando nuestra metodología en técnicas de caja blanca. Para ello usamos una estrategia de ejecución simbólica, la ejecución simbólica de manera simplificada se puede decir que analiza las condiciones que hacen tomar un flujo u otro dentro del programa y ver el estado de las variables para cada uno de esos caminos. Permitiéndonos generar casos de prueba que intentan recorrer o cubrir cada una de todas las posibles ramas que pueda tomar el algoritmo. En la figura 2.2 se puede apreciar el resultado de la transformación de una función en java a un grafo de flujo de control (normalmente denominado por las siglas CFG Control Flow Graph). El criterio de Path Coverage que es el aplicado a la obtención del grafo obtenido mediante ejecución simbólica, se encarga de generar entradas para ser capaces de recorrer todas y cada una de las ramas de este grafo, es un criterio con una exigencia altísima cuyo coste crece exponencialmente con el tamaño del problema, es por esto que se cree que la mayoría de problemas planteados en estos entornos, resultan ideales para esta aproximación ya que su complejidad tiende a ser pequeña. Para ello hacemos uso de la herramienta jPET, que ha sido proporcionada por el tutor del trabajo de fin de grado Miguel Gomez-Zamalloa y que cuenta con un capítulo destinado a ella donde se explicaán su forma de uso asi como sus limitaciones.

El objetivo del trabajo consiste en la aplicación de testing sobre un entorno de jueces en línea, en este caso DomJudge, con el fin de permitir la creación de casos de prueba de manera automática y que estos tengan un impacto realmente relevante en los resultados sobre los ejercicios propuestos por los usuarios. Para ello integraremos la herramienta jPET en el entorno de DomJudge, permitiendo que dado un código fuente que resuelva el algoritmo, generar automáticamente casos de entrada y salida para el problema dentro de la propia aplicación de DomJudge.

Para ello se ha realizado un trabajo de investigación acerca de diferentes materias, que dividirá el contenido del documento. Comenzando los antecedentes y el estado del arte se destinará un capítulo a explicar las principales características del testing, no entrando a gran detalle, pero aportando los conocimientos necesarios para la justificación de la propuesta dada, a continuación se hablará del juez online elegido, explicando todas sus funciona-

lidades y opciones que aporta, pudiendo entender el porque se eligió para este trabajo, seguidamente se dedicará una sección a jPET para explicar su funcionamiento y sus limitaciones, a continuación, terminando ya el estado del arte, se dedicará la totalidad de un capítulo a como ha sido paso por paso la integración de jPET sobre DomJudge, así como todos los inconvenientes y problemas que se han ido presentando. Para finalizar el documento, se incluirán unas estadísticas realizadas haciendo uso de esta integración sobre problemas que están siendo parte de la evaluación de diferentes asignaturas permitiendo obtener un resultado sobre problemas reales que se plantean, finalizando dicha sección con un apartado para las conclusiones y las posibles líneas futuras.

1.2. Introduction

Nowadays the use of online judges is gaining greater importance, not only focusing on the academic field but also in the competitive one, even reaching to celebrate programming contests on these platforms.
An Online judge is an online platform where programming problems are proposed. Every problem has a statement and tend to be accompanied by significant examples of input and output. Thanks to that, the student or the user is aware of what he is being asked and can understand the problem with greater ease. The issues raised in these environments, often have a very low level architectural patterns and programming complexity. By decoupling the difficulty in getting these tasks we can pose complex algorithmic problems without requiring a robust environment that makes use of their results. Once a problem has been uploaded, users upload their source code, which when executed reads from standard input the parameters of each problem, and shows the algorithm solutions following the pattern indicated by the examples. A solution uploaded by a student can take five major results. The recommended which is the correct one. Wrong answer, because it returns an output that does not match what was expected to return the algorithm. Compilation error indicates that the code couldnÂ´t be correctly executed and there has been an error. And finally time-limit, given in the event that a solution has exceeded a fixed time, ideal to force solutions to be really efficient.
For this final degree project, we have chosen the online judge DomJudge. Its use has been extended because of the versatility that is not dependent on a single language and also to the ease of use that it represents. Even arriving to be used in the School of Computing at the Universidad Complutense de Madrid as a complement and evaluation system strongly oriented to subjects of algorithmic. The roles for users allow differentiation of functions restricting the creation of problems and their reviews. The DomJudge system will be discussed in much more detail in Chapter 3.

Despite the relevance and importance of Testing in the computer world, it turns out to be one of the most neglected branches, with little mention during the degree, and with almost no practical use over during its duration. The testing in any environment is a crucial task. The capacity to ensure that your system complies properly with its functionality, that it is also able to withstand unexpected input and maintains robustness in situations not expected is an aspect of vital importance. Unfortunately it will always come along with a high cost making it of the main tasks to which resources are allocated.
There are different kinds of testing, and many of them are adaptable to the environment of DomJudge. The next chapter will explain the most significant strategies, establishing a knowledge base in testing that allows understanding the proposed solution.
One of the main problems that arise with the use of online judges is the impossibility to guarantee that all the posed problems have really significant test cases that completely verify their correctness. This task is not simple at all, and can very hardly ensure a complete verification with these test cases.

To deal with this problem, it has been opted for the generation of test cases randomly. Which facilitates a massive creation of input for the problem, but don′t guarantee that all the possibilities of such a problem arise. This is the main motivation that comes to the integration of self-generating tools test cases not based on randomness.
As we will see in the chapter on testing, using for example an algorithm that given an element n calculate the result of 1 / n, when n takes the value 0 we will find ourselves in a special behavior. Using only random generation tools, that chooses between any domain of possibilities, it will be hard to get this value. Therefore the method of random generation and belittles it is not discarded, since in many environments is often a really effective strategy in addition to serving as a support and guarantee more than one way to correct the problems.
Other solutions proposed for the generation of test cases has been the application of testing techniques. Consciously or unconsciously, where the nature of the problem to be solved has been studied, and have been made entries pointing directly to the conflictive points of each problem. Despite being a more expensive relative to the effort involved, it is a really recommended solution if we really want to ensure the correctness of the proposed solution.
The solution we propose to solve the problem of the generation of test cases is to obtain these automatically, basing our methodology on white box techniques. For this we use a strategy of symbolic execution. One can say in a simplified way that the symbolic execution analyzes the conditions that make taking a stream or another within the program and view the status of the variables for each of those paths. Allowing us to generate test cases that try to cover each of all possible branches that can take the algorithm. In

Figure 2.2 we can see the result of the transformation of a function in java to a control flow graph (commonly referred to by the acronym CFG Control Flow Graph). Path Coverage criterion, which is applied to obtain the graph obtained by symbolic execution, is responsible for generating inputs that are able to cover each and every one of the branches of this graph. It is a criterion with a high requirement whose cost grows exponentially with the problem size. That is why it is believed that most problems in these environments are ideal for this approach because its complexity tends to be small. For this we use the tool JPET, which has been provided by our final degree project tutor Miguel Gomez-Zamalloa and has a chapter dedicated to it where its usage is explained as well as its limitations.
The aim of this work is the application of testing on an environment of online judges, in this case DomJudge. In order to allow the creation of test cases automatically and that they have a really significant impact on the results of the exercises by users. To achieve it we will integrate the Jpet tool into the DomJudge′s environment. Allowing than given a source code that solves the algorithm automatically, JPet generate input and output cases for the problem within DomJudge.
To achieve this a research on different subjects has been carried out, which will divide the contents of the document. Starting with the background and state of the art a chapter will be used to explain the main features of testing, without going into much detail, but providing the necessary knowledge for the justification of the proposal given. Then we will talk about the online judge elected, explaining all its features and the options it provides, understanding why it was chosen for this work. Then a section will be dedicated to Jpet explaining its operation and limitations. Then and ending the state of the art, an entire chapter will be dedicated to the integration of Jpet on DomJudge step by step, and all the inconveniences and problems that have emerged. Finally some statistics will be included which have been obtained by using this integration into the problems that have been part of the evaluation of different subjects. Thus allowing obtaining an outcome on real problems that are set out in different subjects, ending this section with a section for conclusions.

En el próximo capítulo...

Se hablarán de las bases del testing y de las metodologías existentes y que se usaran.

Capítulo 2

Testing

Software testing is a sport like hunting,
it's bughunting.

Amit Kalantri

Resumen: Se explicarán los conceptos necesarios de testing para el entendimiento y la justificación de la solución propuesta en el entorno del juez online utilizado.

2.1. Introducción al Testing

El testing es una tarea que desde que se empezó a practicar en las empresas, se ha estimado que no solo consume mas de la mitad del tiempo invertido, sino que además se lleva más de la mitad de los recursos necesarios. Y a pesar de esto, sigue siendo una ciencia inexacta, convirtiéndola en una de las ramas de las que menos conocimiento del desarrollo de Software se tiene.

En este apartado no se pretende explicar toda la teoría del testing. Se hablarán de los principales aspectos de este, haciendo especial énfasis a aquellos que pueden o podrían haberse integrado en el desarrollo del entorno junto a los conocimientos básicos que nos permitirán entender el funcionamiento de los tests.

El testing en este trabajo de fin de grado cobra su relevancia debido a la necesidad que se tiene de demostrar que las soluciones propuestas por los alumnos a la hora de entregar un ejercicio son correctas. Más adelante se repasarán las muchas formas que existen de testear software, y su aplicación al entorno de los jueces online.

El testing puede tomar diferentes definiciones según en que contexto se estén dando. La extensa definición que nos aporta Wikipedia dice asi:

> Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test. Software testing can also provide an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation. Test techniques include the process of executing a program or application with the intent of finding software bugs (errors or other defects). Software testing involves the execution of a software component or system component to evaluate one or more properties of interest. In general, these properties indicate the extent to which the component or system under test:
> -Meets the requirements that guided its design and development.
> -Responds correctly to all kinds of inputs.
> -Performs its functions within an acceptable time.
> -Is sufficiently usable.
> -Can be installed and run in its intended environments.
> -Achieves the general result its stakeholders desire.
>
> Wikipedia- Articulo Software testing.

En menos palabras se podría decir que el testing se encarga de comprobar que el software cumple los requisitos, no es difícil de usar y además cuenta con una robustez que le permite funcionar correctamente sin importar las entradas que reciba. A pesar de ser una definición bastante amplia tiene un enfoque acertado. Otra definición, esta vez mas amigable es la ofrecida en el libro The Art of Software Testing cuya traducción literal es:

> Serie de procesos orientados a asegurar que el software hace lo que debe hacer y no hace lo que no debe.
>
> Glenford J. Myers Tom Badgett Corey Sandler- The Art of Software Testing.

Una definición acertada también, que prioriza la funcionalidad del Software, dejando de lado aspectos como la usabilidad, aunque teniendo en cuenta también la robustez que deben presentar los programas para no actuar de forma inesperada ante ciertas situaciones. A nosotros nos gusta entender que a diferencia de lo que dicen las definiciones mas formales, el proceso de testing esta mas destinado a buscar errores del software, mas que a asegurarse de que lo que se espera que haga bien lo haga bien. Es por esto que consideraremos tests exitosos aquellos que efectivamente demuestren que nuestro

Software presenta carencias o no estaba preparado para recibir cierto tipo de entradas y responde de una forma no esperada.

El testing como se puede esperar se presenta de muchas formas, teniendo en cuenta lo que se quiere testear, ya sea un sistema completo funcionando o simplemente una pequeña sección de código. Estas diferentes aproximaciones pueden clasificarse en los siguientes niveles.

- Unit testing
- Integration testing
- Component interface testing
- System testing
- Operational Acceptance testing

A pesar del interés que presentan todas las categorías, solo nos centraremos en el testing de unidad, que es aquel cuyo enfoque se centra en demostrar que pequeños fragmentos de código hacen lo que deben de hacer. Generalmente, los test de unidad suelen ir asociados a las funciones que se quieren verificar, lo que en el contexto de los jueces en línea nos resulta muy favorable.

Una vez elegido el tipo de aproximación queda decidir el tipo de testing que se hará, si se basará en el código fuente, o si por el contrario usará la especificación, estos puntos los trataremos con mas énfasis en los siguientes temas.

2.2. Tipos de Testing

Existen dos grandes opciones a la hora de plantearse realizar tests sobre un Software, estas opciones a veces son de elección forzada ya que no se disponen de las premisas necesarias. Estas dos opciones son test de Caja Blanca, basados en el conocimiento previo del código fuente, y tests de Caja Negra, centrados en una especificación previa dada del producto. Es común pensar que en caso de disponer del código fuente, lo ideal sería ajustarse a la metodología de Caja Blanca, este planteamiento esta muy lejano de la realidad, en el mundo empresarial, en caso de acceder al código fuente los primeros tests planteados hacia el producto son acorde a la especificación inicial, es decir de Caja Negra, y refinados con tests de Caja Blanca.

Es importante entender que el coste del testing tiende a ser muy elevado, no solo la generación de estos, sino también su corrección, y que es imposible o en muchos casos inviable asegurarse que un sistema es completamente

correcto. Debido a esto hay que tomar un acercamiento al problema mas cauto, y ya que resulta de gran coste asegurarse el correcto funcionamiento de todo el sistema, estos tests, deberían estar dirigidos a aquellas zonas de código mas conflictivas del producto y las que mas uso van a tener, ya que no tiene sentido invertir un gran capital en asegurarse que una tarea que no va a ser realizada con mucha frecuencia funciona correctamente dejando de lado aquellas que si son mas vitales para el sistema.

2.2.1. Testing de Caja Blanca

El testing de Caja Blanca es aquel encargado de revisar que la lógica de los programas es correcta, para ello se exploran las diferentes ramas de ejecución que puede presentar un programa. Se asume por lo tanto, que un test óptimo de Caja Blanca es aquel que explora todas y cada una de las ramas.
Como se ha comentado con anterioridad, esto no siempre resulta factible y para examinar las ramas de un programa se siguen diferentes estrategias. Para poder explicar estos criterios, se ha extraido un fragmento del libro The Art Of Software Testing referenciado en la bibliografía. Que se puede ver en la figura 2.1

2.2.1.1. Statement coverage

Una primera aproximación bastante simple es la de asegurarnos que los tests generados consiguen ejecutar todas las lineas de código de nuestro programa, más adelante veremos que esta opción a pesar de ser simple no es realmente ilustrativa en el sentido de determinar la corrección de una función. Basándonos en la figura 2.1 se puede ver que fácilmente con un único test se pueden recorrer todas las instrucciones, bastaría tan solo asignar A al valor 2, B al valor 0 y X a cualquier valor superior a 1 y su recorrido en el grafo de flujo seria de A $\rightarrow$ C$\rightarrow$ E.

Podemos ver que el criterio seguido realmente no nos asegura el correcto funcionamiento, ya que tenemos caminos como A $\rightarrow$ B$\rightarrow$ D. en los que la variable x no es siquiera afectada, y si esto se tratara de un error, pasaría desapercibido. Por no decir que para el test dado la segunda condición del or no es ni tenida en cuenta. Debido a estas razones, nos encontramos a que este criterio no es de real utilidad en muchas ocasiones, pero en cambio sirve para detectar fragmentos de código muerte (Lineas de código no accesibles durante la ejecución) a pesar de que esta tarea es muchas veces realizada por los propios entornos de desarrollo.

```
public void foo(int A, int B, int X) {
    if(A>1 && B==0) {
        X=X/A;
    }
    if(A==2 || X>1) {
        X=X+1;
    }
}
```

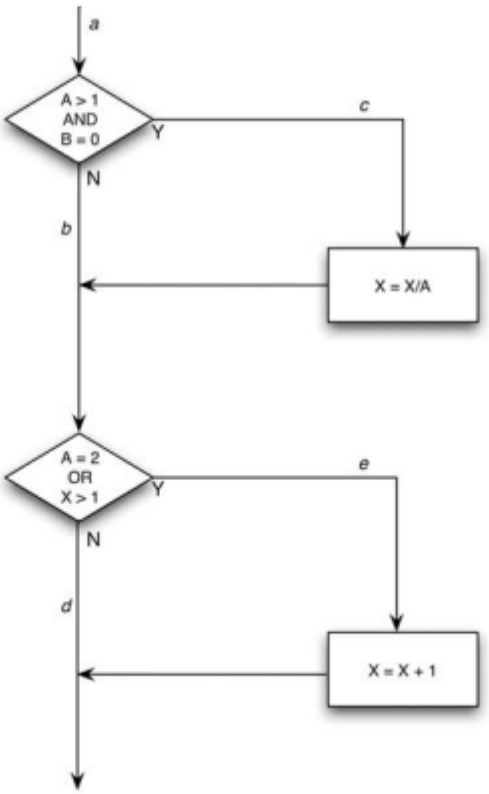

Figura 2.1: Ejemplo de representación en forma de grafo de una función

2.2.1.2. Decision coverage

Las carencias del anterior criterio conllevaron a la creación de uno nuevo evolucionando de la forma mas lógica posible, si antes nos conformabamos recorriendo todas las instrucciones del código, esta vez no nos será eso suficiente, sino que además deberemos recorrer todas las ramas del grafo de flujo generado. Por lo general se puede presuponer que Decision coverage satisface el criterio de Statement coverage, ya que al fin y al cabo, las instrucciones se encuentran en cada rama, pero existen unas pequeñas excepciones que se mencionarán a modo de curiosidad que generalmente no suelen presentarse en código de día a día. Estas son:

- Programas sin decisiones
- Programas con diferentes puntos de entrada, haciendo que algunas instrucciones sean ignoradas en función del punto de acceso

En este caso, si quisieramos realizar un suite de tests que satisfagan este criterio, nos bastaría el uso de dos tests, uno con el recorrido A $\rightarrow$ C$\rightarrow$ D y otro que vaya A $\rightarrow$ B$\rightarrow$ E, de esta forma conseguiríamos recorrer todas las posibles ramas. Un ejemplo de dos tests que cumplan esto serían A=3, B=0, X=3 y A=2, B=1 y X=1.

Aunque este criterio en perspectiva es mucho mas completo que el anterior, sigue sin ser realmente exhaustivo ya que solo verifica que los bucles condicionales sean true o false, sin importar si ambas condiciones estas correctamente expresadas, ignorando así en el caso de un or, la segunda condición si la primera se cumple. Esta ampliación del criterio se conoce como Condition coverage.

2.2.1.3. Condition coverage

Como se ha comentado en la anterior sección, Condition coverage es una ampliación directa de Decision coverage. En este criterio, no basta tan solo satisfacer que se recorren todas las ramas y que a su vez todas las condiciones son evaluadas a true o false al menos una vez, este incluye que cada condición necesita de al menos un test que cubra cada uno de todos sus posibles resultados. Siguiendo el ejemplo de la figura 2.1 podemos ver que en el algoritmo se encuentran cuatro condiciones, A>1, B=0, A=2 y X>1, por lo que el nuevo suite de tests tendrá que ser capaz de forzar situaciones en las que A>1, A<=1, B=0, B!=0 para el punto A, y A=2, A!=2, X>1 y X<=1 para el punto b, en este caso el problema al ser de gran sencillez y estar las condiciones ligadas de forma conveniente, el criterio podría satisfacerse con tan solo 2 tests:

A=2, B=0, X=4 A → C→ E
A=1, B=1, X=1 A → B→ D

A pesar de que el número de tests coincida para este criterio y el anterior, Condition coverage tiende a ser mucho mas amplio en ese aspecto, haciendo que el coste de la realización de tests que cubran el criterio comience a crecer fuertemente basándose en el número de condiciones que presente el algoritmo.

Como se ha mencionado con anterioridad, este criterio puede llegar a ser realmente efectivo, pero hay que tratarlo con mucho cuidado, ya que es posible que se den situaciones en las que no se cumpla ni el statement coverage, si nos fijamos en el ejemplo de código, y proponemos otro par de tests diferentes.
A=1, B=0, X=3 A → B→ E
A=2, B=1, X=1 A → B→ E
Si repasamos las condiciones A>1, A<=1, B=0, B!=0 para el punto A, y A=2, A!=2, X>1 y X<=1 para el punto b, podemos ver que efectivamente esta cubriendo todas y cada una de las posibilidades de cada condición pero a pesar de ello, es incapaz de ejecutar todas las instrucciones del código, en este caso concreto esto es debido a la naturaleza de las condiciónes AND y OR, en el que la AND del primer caso al ser True AND False, va a rechazar siempre la entrada a la condición y la OR ante la misma situación va a permitir siempre la entrada. La solución a este problema viene dada en el siguiente criterio.

2.2.1.4. Decision/condition coverage

El criterio Decision/condition coverage incluye los dos anteriores, de tal forma que para satisfacer este criterio se debe cumplir que al menos haya un test que haga que cada condición tome todos los valores posibles, que para cada decisión se asegure que al menos una vez se toman todos los caminos posibles y para evitar incumplimientos de criterio además es necesario que se cumpla que todos los puntos de inicio sean invocados al menos una vez.

De igual forma que todos los criterios anteriores, siguen encontrándose debilidades, aunque ya no son de forma tan evidente y no necesariamente se tienen que cumplir siempre. Para ser capaces de entender donde se encuentra el problema se ha usado otra imagen del libro The Art Of Software Testing, en la que se traduce a código máquina el algoritmo planteado en la figura 2.1.

En este caso se puede apreciar que las condiciones dentro de una misma decisión han sido divididas, y representadas segun su naturaleza OR o AND, de tal forma que en el caso de ser una condición AND si la primera parte se evalua a false, es irrelevante el contenido de la segunda, ya que será ignorado, asumiendo que la evaluación de las condiciones sea secuencial y no paralela.

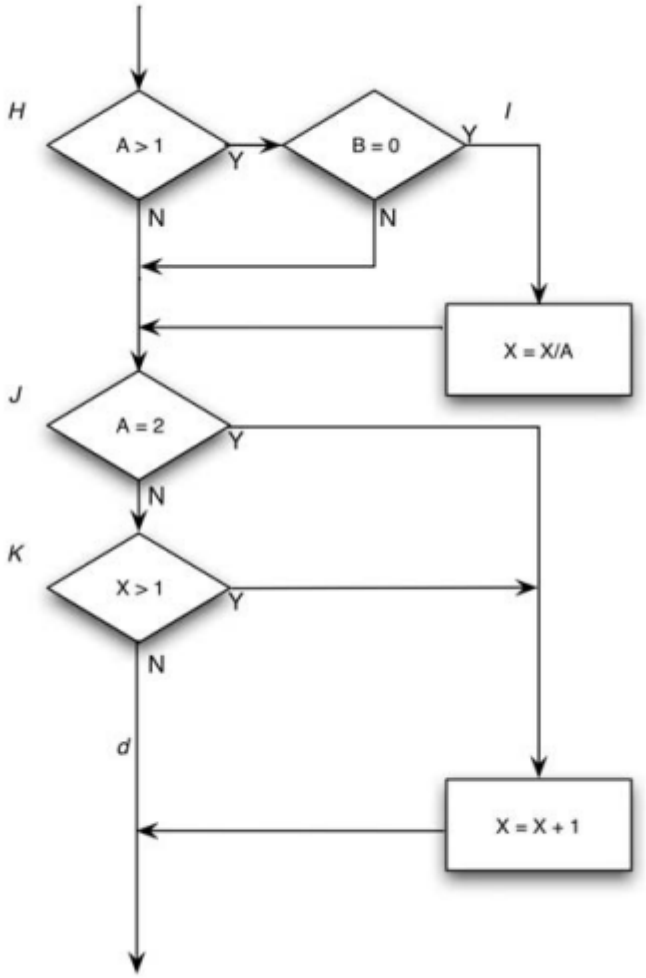

Figura 2.2: Ejemplo de código máquina generado a partir de la función representada por el grafo de la figura 2.1

Para ser capaces de cumplir este requerimiento se necesita de un conjunto de tests capaces de visitar cada rama al menos una vez, resulta trivial ver que las anteriores parejas presentadas no son capaces de cumplir dicho requisito. Este problema de enmascaramiento de condiciones es solucionado en el siguiente criterio donde generalmente un mayor número de tests será necesario.

2.2.1.5. Multiple-condition coverage

Multiple-condition coverage es un criterio que tiende a ser realmente exigente ya que para satisfacerlo, es necesario que haya suficientes tests tales que estos sean capaces de cubrir al menos una vez todas y cada una de las posibles combinaciones de condiciones tales que generen un resultado, añadiendo además la restricción que se comento con anterioridad que haga que se ejecuten al menos una vez todos los puntos de partida de ejecución.

Volviendo al ejemplo en el que hemos basado toda la explicación, tenemos que cubrir esta vez ocho posibles combinaciones para cumplir con este requisito, dichas combinaciones son:

1. A>1, B=0
2. A>1, B!=0
3. A<=1, B=0
4. A<=1, B!=0
5. A=2, X>1
6. A=2, X<=1
7. A!=2, X>1
8. A!=2, X<=1

A pesar de que se necesiten cumplir las ocho combinaciones, esto no implica que necesariamente se tengan que usar ocho test, ya que se puede observar que muchas condiciones están ligadas a una misma variable y no son incompatibles. Para este caso bastaría tan solo el uso de cuatro tests para realizar el recubrimiento completo.
A=2, B=0, X=4 Para las condiciones 1, 5,
A=2, B=1, X=1 Para las condiciones 2, 6,
A=1, B=0, X=2 Para las condiciones 3, 7 y
A=1, B=1, X=1 Para las condiciones 4, 8.

Como se puede comprobar el esfuerzo necesario para generar tests dirigidos a un problema de una complejidad minúscula como es la de este ejemplo

va en aumento en función del criterio que se quiera satisfacer. Para problemas tan simples como este puede no parecer una gran complicación, pero los algoritmos reales, nunca tienden a esta sencillez tan absoluta, suelen ser mas tergiversados, además de contar con el factor humano que es capaz de añadir condiciones innecesarias que estorban a los tests, o simplemente dejar un algoritmo poco optimizado que cuenta con muchas ramas sustituibles o inútiles.

La principal recomendación dada a la hora de realizar tests sobre una aplicación es ser cautelosos en cierta medida, es demasiado costoso realizar tests que verifiquen que todo el sistema funciona correctamente, por eso se toma como buena medida aquella que centraliza los tests a las partes críticas y a aquellas que van a ser usadas en mayor medida.

2.2.2. Testing de Caja Negra

El testing de Caja Negra es aquel en el que no se cuenta con el código fuente o bien no se quiere hacer uso de el, y se basa unicamente en una especificación previa que se haya tenido. También es conocido como testing de entrada/salida.

Partiendo de la premisa de que para cualquier algoritmo, la entrada tiende a ser infinita, se deben seguir algunas pautas para intentar filtrar el mayor número de entradas posibles, quedándonos con aquellas que son mas propensas a detectar el error.

2.2.2.1. Equivalence Partioning

Es una técnica de testeo de Software basada en subdividir todas las entradas posibles para un algoritmo en diferentes secciones de forma que se pueda encontrar un test que se amolde a cada una de estas secciones. Este particionamiento sigue dos premisas importantes

1. Cada partición deberá reducir a mas de uno el numero de tests que debán ser generados para llegar hasta el objetivo de demostrar que esta suficientemente bien testeado.

2. Abarca un gran número de otros posibles tests, de forma que al generar un test a partir de dicha partición haya muchos otros tests que sean irrelevantes y no nos aporten mayor información.

Juntando estas dos condiciones y haciendo uso de ellas, podemos ser capaz de teniendo el mínimo número de tests posibles abarcando todas las propiedades importantes del algoritmo.

A continuación se mostrará un ejemplo muy básico en donde se pueden apreciar bien estas propiedades.

Dado el siguiente código:

```
// Ejemplo basico de funcion tal que dado un float f, encuentre el
    resultado de 1/f

public float divide(float f) {
   return 1/f;
}
```

A pesar de ser una metodología de caja negra, y de mostrar el código, este podría ser ignorado y nos podríamos basar únicamente en la especificación previa. Al ser una división, sabemos que una propiedad interesante de estas, es que la división entre cero sigue unas reglas diferentes a las de otros números, y que a su vez, una división entre un número negativo, altera el signo del numerador y una con denominador positivo lo mantiene. Por lo que tenemos ya 3 grandes subgrupos, todos los números menores que cero, el propio cero y todos los números positivos. Una generación aleatoria de números, probablemente sería capaz de cubrir la variante de números positivos y negativos, pero difícilmente de todo el dominio de números podría seleccionar el 0. Aplicando las anteriores premisas, podemos deducir que si usamos como test f = -2000.0 estaríamos cubriendo todos los negativos, y nos serían necesarios tan solo 3 tests para satisfacer su corrección a un nivel razonable.

Este proceso se realiza en dos principales pasos. El primero consiste en la identificación de las clases de equivalencia, para esto es necesario un análisis de los requisitos esperados del Software, para cada uno de ellos se creará una clase de equivalencia que satisfaga el requisito y otra que no, por ejemplo, si los requisitos nos hablan acerca de que solo pueden recibir valores positivos, en la clase de equivalencia válida será incluida una partición con positivos, y en la no valida una con valores negativos, del mismo modo que cuando nos establecen un rango en los requisitos, generaremos 3 clases, una para valores menores del rango, otra para valores correctos que se adapten al rango y otra clase de equivalencia invalida para valores que queden por encima del rango. Por supuesto siempre es importante tener en consideración las imposiciones de los requisitos, es posible que un programa de análisis de cadenas de caracteres, necesite que el primer elemento sea una letra mayúscula para asegurar el correcto funcionamiento, de estas presunción generaremos dos nuevas clases de equivalencia, una que incluya una letra mayúscula y otra que no. Además de todo esto, también entra en juego la experiencia de la persona encargada de hacer los tests, ya que por lo general los errores suelen centrarse en los mismos puntos siguiendo patrones muy similares, por lo que es recomendable que en cierto modo dejar al testeador fiarse de sus instintos y dirigir un par de clases de equivalencia a errores comunes o que se espere que van a desarrollar un comportamiento diferente aunque no este realmente

especificado en los requisitos.

El segundo paso consiste en la identificación de los tests a generar, el cual sigue un procedimiento secuencial basado en los siguientes puntos.

1. Enumeración de todas las clases de equivalencia generadas.
2. Generar test cases que agrupen el mayor número de clases de equivalencias válidas hasta que todas hayan sido cubiertas.
3. Generar test cases que cubran una y solo una clase de equivalencia invalida hasta que hayan sido todas cubiertas.

El especial énfasis dado a las clases de equivalencia inválidas es debido a la proyección inicial de dicho criterio, en el que se priorizan aquellos tests que vayan más dirigidos a la detección de errores. Dejando los del comportamiento esperado cubiertos, pero no a tanto nivel de detalle, ya que los algoritmos y el Software en general, tiende a comportarse de forma no esperada ante entradas no esperadas o no válidas.

2.2.2.2. Boundary Value Analysis

La experiencia ha demostrado que aquellos tests centrados en las condiciones límite tienden por lo general a ser tests mas exitosos al encontrar mas errores en el código. Estas condiciones límite o Boundary conditions haciendo referencia al nombre del procedimiento, son aquellas que van enfocadas a los límites superiores e inferiores de las condiciones, es decir que en caso de que tengamos un algoritmo que limite el valor de un parámetro de entrada a cierto numero n, estos tests van a apuntar a los valores mas inmediatamente cercanos del n mencionado, n-1, n y n+1 en caso de tratarse de números enteros.

Las principales diferencias entre Boundary Value Analysis y el enfoque de las clases de equivalencia viene dado por la selección de los elementos, en Equivalence partioning, cualquier elemento dentro del intervalo es válido, mientras que en este criterio, Boundary Value Analysis, es necesario que haya un test o mas de uno que englobe los bordes mas de dicha clase. De igual modo, a diferencia del anterior criterio, al seleccionar estos elementos, desviamos nuestra atención mas a la salida en vez de a la entrada, considerando todo el espacio de soluciones que puede dar la salida del algoritmo. De la misma forma que el anterior criterio, existen ciertas pautas, que nos pueden ayudar a realizar un conjunto de tests aptos para este criterio:

1. En caso de tener una regla que especifique el dominio valido de una entrada, por ejemplo los numeros entre 10 y 20, seria correcto realizar tests para los valores 9,10,20 y 21.

2. Del mismo modo que en el anterior apartado, cuando se especifica un número determinado de valores para la entrada, escribir tests que contengan el número mínimo de elementos, el valor mínimo menos uno, el valor máximo y el valor máximo mas uno.

3. Examinar las posibles salidas, del programa, si sabemos que el valor va a diferir entre dos valores, sería recomendable crear tests que fuercen a que la salida tome dichos valores.

4. Del mismo modo que en el punto 2, si se espera que la salida vaya a ser de un número máximo de resultados, forzar al programa para que muestre el mínimo número de resultados, el máximo y a ser posible, usar intuición y conocimiento del sistema para intentar obligar al algoritmo a mostrar mas.

5. Aunque sea un concepto básico, si se espera una entrada ordenada, resulta interesante desordenarla teniendo en especial consideración el primer y último elemento.

6. Igual que en el resto de criterios de caja negra, si se espera que una entrada en concreto (entrada vacia, o de elementos nulos por ejemplo) va a hacer que el programa no se ejecute correctamente, es de vital importancia su adición.

Un ejemplo clásico que se suele tomar como modelo para la explicación de este criterio es la determinación de que dados 3 números que representan los lados de un triángulo determinar si es un triangulo válido y de que tipo es. Aunque no se vaya a realizar especial énfasis a dar una solución a dicho problema, sabemos que para que 3 lados formen un triángulo se deben cumplir las siguientes condiciones, tomando a,b y c como las longitudes de cada lado.

1. a + b >c

2. a + c >b

3. b + c >a

Este criterio va enfocado a esta clase de problemas, ya que esta demostrado, que el mayor número de errores en un algoritmo, suelen presentarse en asuntos triviales como intercambiar un mayor por un mayor o igual, considerando estas condiciones, y al dirigir los tests, a valores exactos, estaremos probando tanto el mayor como el mayor igual, detectándo el error potencial.

Existen mas metodologías para la creación de tests de caja negra, como Cause-effect graphing o Error guessing las cuales no se van a explicar en este

apartado, ya que se considera suficiente base teórica el conocimiento de los criterios explicados para ser capaces de entender las soluciones presentadas.

Notas bibliográficas

- Glenford J. Myers Tom Badgett Corey Sandler: The Art of Software Testing
- Cem Kaner, James Bach, Bret Pettichord: Lessons Learned in Software Testing
- Jerry Weinberg: Perfect Software: and other illusions about testing.
- James Whittaker: Exploratory Software Testing
- Articulo Software testing http://en.wikipedia.org/wiki/Software_testing
- Artículo informativo acerca de boundary value. http://istqbexamcertification.com/what-is-boundary-value-analysis-in-software-testing/

En el próximo capítulo...

En el próximo capítulo se hablará de la aplicación de los conceptos adquiridos a un entorno de juez online, analizando las posibles formas de tratarlo y justificando las decisiones tomadas.

Capítulo 3

Domjudge

El software es como el sexo: mejor si es libre y gratis

Linus Torvalds

Resumen: El capítulo contiene las guías de instalación y uso del juez en línea seleccionado para este proyecto.

3.1. Introducción

Domjudge es una aplicación web con jueces automatizados en línea que fue creada para organizar concursos de programación. La aplicación está diseñada para que los participantes puedan enviar las soluciones con algunas restricciones como el tiempo de ejecución. Posteriormente el juez en línea corrige de forma automática los programas subidos y proporciona los resultados a través de una interfaz web siendo visible para los equipos participantes, el jurado y el público general.

La aplicación hace mucho en énfasis en la seguridad y usabilidad. Pudiendo mantener varios "concursos" en línea simultáneamente.

El proyecto DomJudge comenzó en 2004 en la Universidad de Utrecht (Países Bajos) por un equipo de desarrolladores que tenían como objetivo imitar a la competición internacional universitaria ACM de programación (ACM-ICPC, por sus siglas en inglés).

Este proyecto está bajo la licencia GNU, esto permite a cualquiera que esté interesado descargarla, instalarlo, modificarlo y compartirlo con la comunidad de forma gratuita.

La aplicación pone a nuestra disposición tres tipos distintos de usuarios:

- Administrador: es el rol con más control en la aplicación y tiene permisos para modificar todas las configuraciones y datos.

- Juez: se encarga de crear concursos con la configuración que considere pertinente.
- Equipos: son los participantes del concurso.

3.2. Instalación

La instalación ha sido mucho mas difícil de lo que planeamos inicialmente, y conseguir que funcionara todo correctamente ha sido también una tarea muy laboriosa. Esto es debido a la gran magnitud de un proyecto como Domjudge y a la falta de experiencia que tenemos en proyectos de una escala tan grande.

Sin embargo gracias a la documentación presente en la página oficial de Domjudge hemos podido llevar a cabo su instalación. En esta se encuentran todos los detalles necesarios para la configuración del servidor y del host. Además, para llevar a cabo la instalación es necesario disponer de un equipo con un sistema operativo en base Linux, así como unos conocimientos básicos en Linux y servidores.

3.2.1. Prerrequisitos

Los requisitos de hardware no son difíciles de conseguir. Únicamente es necesario un ordenador donde instalar el servidor de DomJudge y que disponga de un sistema Linux con acceso a root.

Para el correcto funcionamiento de DomJudge, los siguientes requisitos son indispensables:

- Para cada lenguaje que acepte el juez on-line el equipo debe disponer de su correspondiente compilador para ser así capaz de generar los ejecutables necesarios.
- Servidor Apache que soporte PHP >= 5.4.
- MySQL o MariaDB >= 5.5.3
- PHP >= 5.4 con línea de comando y con extensión de curl y json.

También se puede instalar opcionalmente los siguientes programas:

- phpMyAdmin para el acceso a base de datos.
- Un demonio NTP para sincronizar la hora del juez y el equipo participante.
- Libcgroup para la seguridad del host.

- Libcurl y libJSONcpp para poder enviar resultados por línea de comandos.
- Libmagic para borrar los binarios generados por el anterior.

Gracias a la documentación facilitada por DomJudge en su página web, podemos realizar una instalación de todos los requisitos previos para el correcto funcionamiento del sistema. Todo ello únicamente con una instrucción a través de línea de comándos. La guía de instalación de DomJudge nos detalla cada paso a seguir para las distintas distribuciones de Linux como Ubuntu, Debian, RedHat, etc. En nuestro caso hemos utilizado Ubuntu como sistema operativo para la instalación tanto del servidor como del host.

En Ubuntu o en Debian se puede usar la si guiente línea para instalar prerrequisitos de servidor:

apt-get install gcc g++ make zip unzip mysql-server \
apache2 php5 php5-cli libapache2-mod-php5 \
php5-gd php5-curl php5-mysql php5-json \
bsdmainutils phpmyadmin ntp \
linuxdoc-tools linuxdoc-tools-text \
groff texlive-latex-recommended texlive-latex-extra \
texlive-fonts-recommended texlive-lang-dutch \
libcurl4-gnutls-dev libjsoncpp-dev libmagic-dev

Y la siguiente para instalar el host:

apt-get install make sudo debootstrap php5-cli php5-curl php5-json \
procps gcc g++ gcj-jre-headless gcj-jdk openjdk-7-jre-headless \
openjdk-7-jdk ghc fp-compiler

3.2.2. Instalación de DomJudge

Este paso de instalación del sistema DomJudge es tan sencillo como configurar los script y makefiles y ejecutarlos. Hay que tener en cuenta que tanto la instalación del servidor como del host son necesarios para el correcto funcionamiento de DomJudge.

DomJudge nos ofrece dos formas distintas de instalar estos servicios en nuestro sistema:

1. Árbol único de directorios: con éste método se generan todos los programas y ficheros relacionados de la instalación en una única carpeta en forma de árbol. De hecho hemos optado esta opción para nuestra instalación porque nos sería más fácil a la hora de localizar y editar los cambios. Para optar por esta opción simplemente se configura con el siguiente comando:

./configure –prefix=$HOME/domjudge

2. Estándar de jerarquía del sistema de archivos: éste método instala DomJudge en los directorios del sistema según el estándar de sistema del ficheros. Para configurarla sólo hay que copiar la siguiente línea:

   ./configure -enable-fhs

Una vez configurado todo correctamente. Los script se pueden instalar el sistema con los siguientes comandos:

- Para instalar el servidor:
  make domserver && sudo make install-domserver
- Para instalar el host:
  make judgehost && sudo make install-judgehost
- Para instar la documentación (opcional):
  make docs && make install-docs

Para realizar la instalación es necesario tener permiso de administrador ya que se necesita modificar tanto usuarios y grupos como ficheros de contraseña e incluso directorios. En el caso de no tener acceso root las modificaciones se deberán de hacer manualmente.

3.2.3. Instalación de las bases de datos

DomJudge puede usar MySQL o MariaDB como base de datos para guardar la información necesaria. En nuestro caso hemos utilizado MySQL ya que contamos con más experiencia en este sistema.

En nuestra instalación de la base de datos hemos optado por dejar el nombre por defecto de la base de datos y el de las tablas. La instalación se inicia ejecutando el script dj-setup-database incluido en DomJudge. Se le llama introduciendo el siguiente comando en la terminal:

dj-setup-database [-u <admin_ user>] [-p <password>] install

La base de datos de DomJudge guarda toda la información excepto las soluciones/programas enviados por los concursantes. Por lo tanto, al realizar una copia de seguridad es necesario guardar los envíos de los concursantes. Las soluciones se guardan en una carpeta del servidor con nombres específicos editados por el programa. El formato de los nombres es como se muestra en la figura 3.1 .

Seleccionamos como ejemplo el primer envío de resultados:

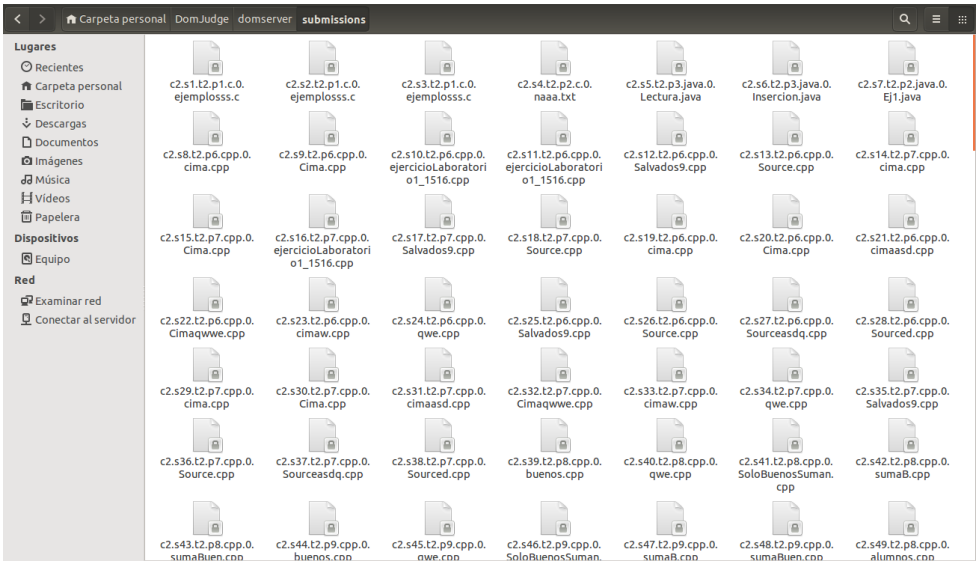

Figura 3.1: Carpeta con las soluciones guardadas en el servidor

- Número de concurso: c2 ->contest 2
- Número de solución: s1 ->solution 1
- Número de equipo: t2 ->team 2
- Número de problema: p1 ->problem 1
- Lenguaje que se ha usado: c
- Seguido del nombre que tenía.

3.2.4. Configuración del servidor web

Para la interfaz web es únicamente necesario un servidor web como por ejemplo Apache. Para su configuración, DomJudge nos proporciona el fichero apache.conf que contiene ejemplos de uso con distintas opciones que funcionan al descomentarlos.
En nuestro caso no hemos modificado ninguna línea ya que todo funciona con la configuración por defecto y tampoco necesitamos que funcione de manera más específica.

3.2.5. Instalación de un host

El equipo host es donde el juez crea el concurso y donde supervisa el correcto funcionamiento de todos los resultados obtenidos. En algunos casos el juez puede enviar clarificaciones sobre los resultados enviados.

Para la instalación, primero es necesario crear un usuario en el sistema con los permisos mínimos. Sin embargo no es necesario que tenga su propio directorio ni tampoco contraseña. La configuración por defecto de DomJudge asigna el nombre domjudge-run al usuario. En nuestro caso optamos por crear un nuevo usuario con el nombre de domjudge-run y no modificar la configuración de DomJudge.

El siguiente paso es configurar los permisos, ya que en algunos casos el host necesitaba permisos de root para realizar algunas operaciones. Para ello DomJudge nos proporciona un fichero con nombre sudoers-domjudge que contiene todas las reglas necesarias, para luego poder copiarlas al directorio /etc/sudoers.d/ del sistema.

Una vez realizadas todas las configuraciones, ya es posible ejecutar el juez online que corrige los envíos de los usuarios. Para ejecutarlo sólo es necesario ejecutar en un terminal el siguiente comando:

bin/judgedaemon

Tras ejecutarlo, el comando se queda a la espera de un envío. Al detectar un envío sin corregir, éste automáticamente corrige e imprime por pantalla la información sobre la compilación y corrección del mismo.

Llegado a este punto, DomJudge funciona perfectamente y se encuentra preparada integración del PET. DomJudge tiene más controles que pueden ser configurados, como por ejemplo configuraciones especiales para ejecutar y comparar programas, o configuraciones de otros lenguajes de programación, etc. Pero todo ello no influye a nuestro proyecto, por lo que hemos decidido obviarlos.

3.3. Interfaz web

Una vez instalado DomJudge, ya sea en un servidor local o en la nube, es recomendable comprobar que se puede acceder a él a través del navegador web y comprobar todas las funcionalidades de DomJudge. En nuestro caso, hemos optado por la instalación local.

3.3.1. Pantalla principal

Al acceder a la interfaz web vemos la siguiente figura 3.2.

Esta pantalla es donde el público general puede ver el marcador de los equipos participantes, en nuestro caso tenemos sólo un equipo de ejemplo que se llama Example teamname. Los equipos están ordenados por el ranking de puntuación que han obtenido en los concursos. El público puede hacer click en el enlace situado en la parte superior izquierda (problems) y acceder a los

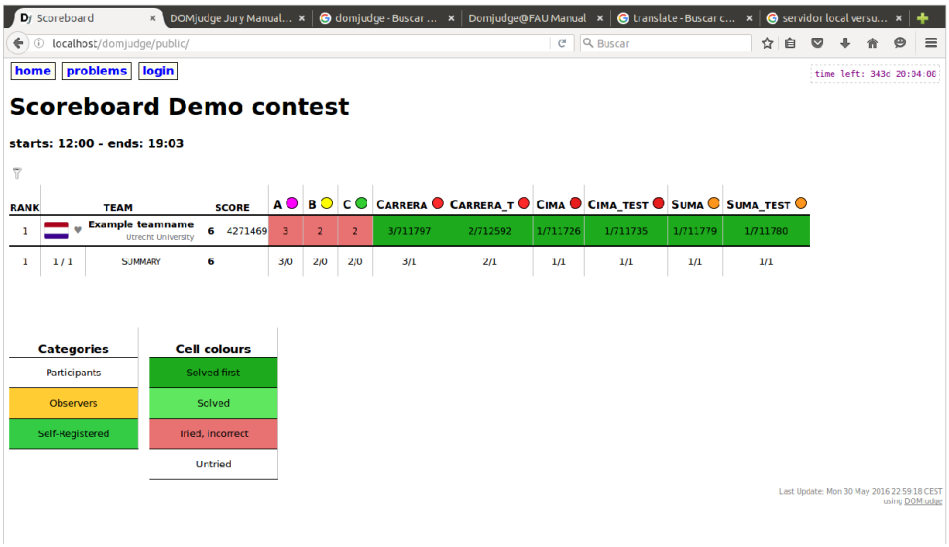

Figura 3.2: Pantalla principal

pdf de los problemas disponibles.

3.3.2. Panel del administrador

Con la instalación finalizada, se puede acceder al panel de control del administrador logueando con el nombre de usuario admin y contraseña admin. Al loguearnos observamos la siguiente figura 3.3.

Se puede observar las distintas configuraciones que se pueden hacer con

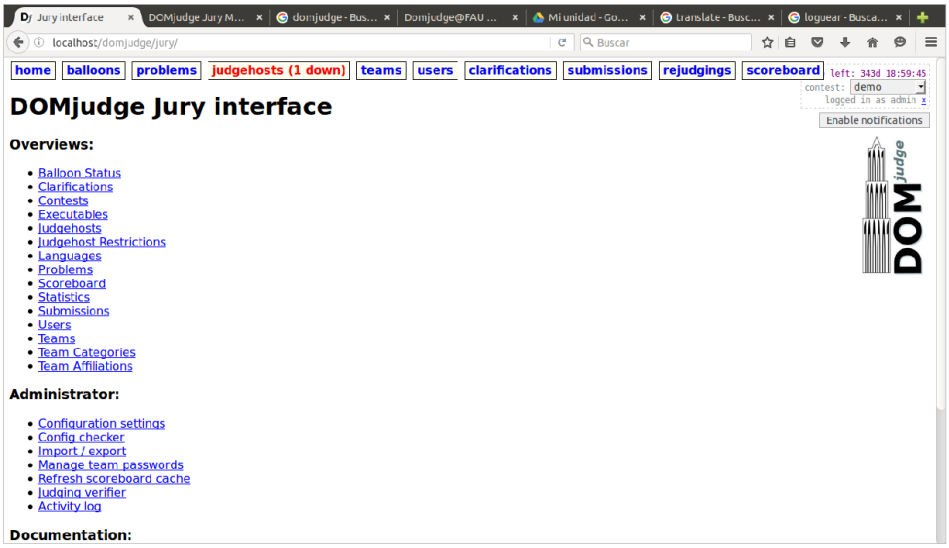

Figura 3.3: Panel del administrador

este rol. En este documento vamos a explicar con detalle sólo los ajustes situados en el menú superior. Para más información se puede consultar en la documentación de DomJudge.

1. Balloons: muestran los equipos que han resueltos los problemas con las horas exactas y algunas informaciones adicionales. A esta pantalla tendrán acceso todos los usuarios que se logueen independientemente del rol que tengan. Ver figura 3.4

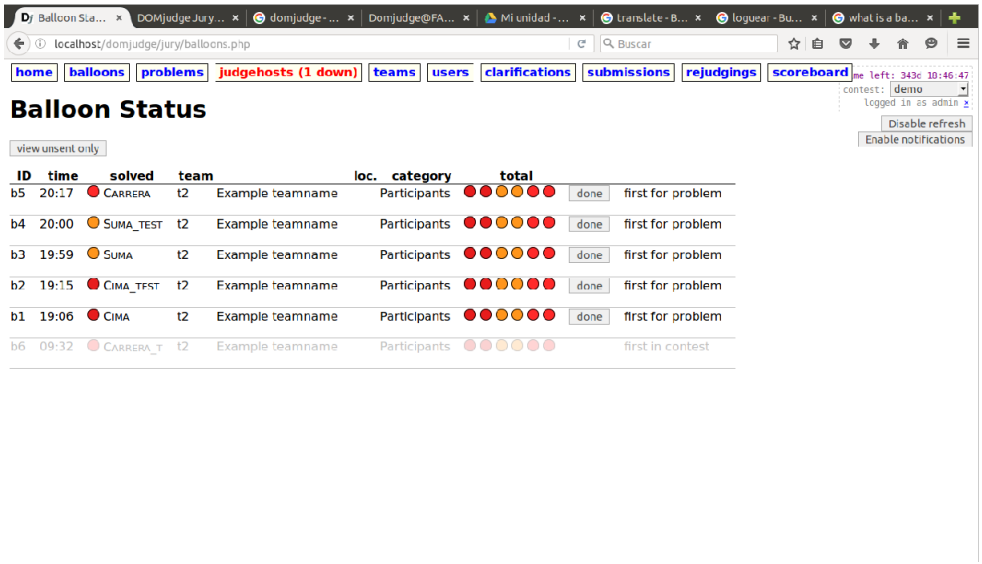

Figura 3.4: Ballons

2. Problems: en esta página se muestran los ejercicios que hay en la base de datos con el número del concurso al que pertenecen y las restricciones que tiene cada problema. El administrador además podrá añadir, modificar y eliminar los problemas. Ver figura 3.5

   a) Editar problema: se puede modificar datos del problema y añadir restricciones como por ejemplo limitando el tiempo de ejecución o condicionando la salida. También está disponible la opción de editar los casos de prueba. Este es el núcleo de nuestro proyecto y es donde se añadirá el generador automático de casos de prueba. Se verá con más detalle en el siguiente capítulo. Ver figura 3.6.

3. Judgehost: en esta página observamos los host instalados en DomJudge y podemos verificar los estados de cada uno de ellos. Si los hosts se

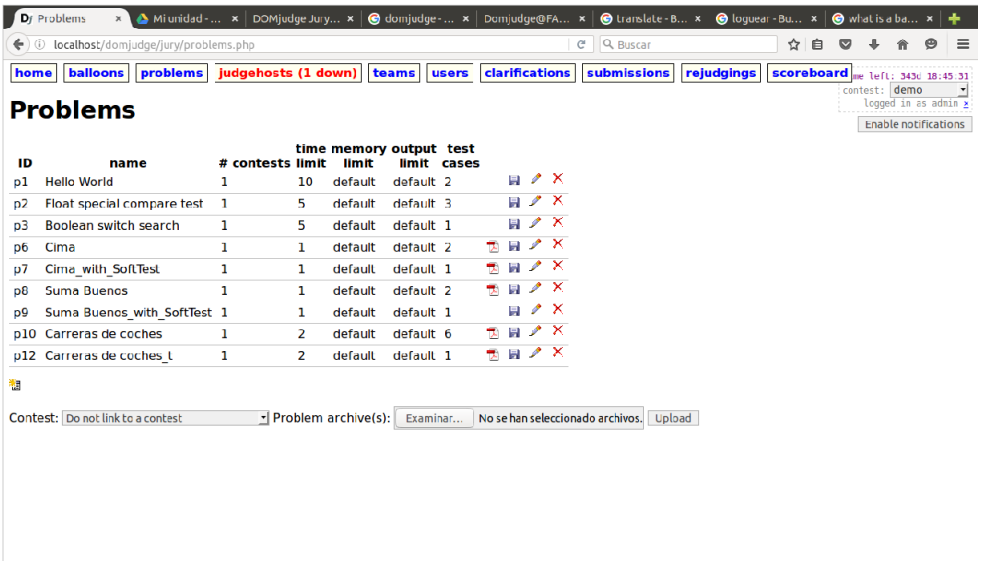

Figura 3.5: Problemas

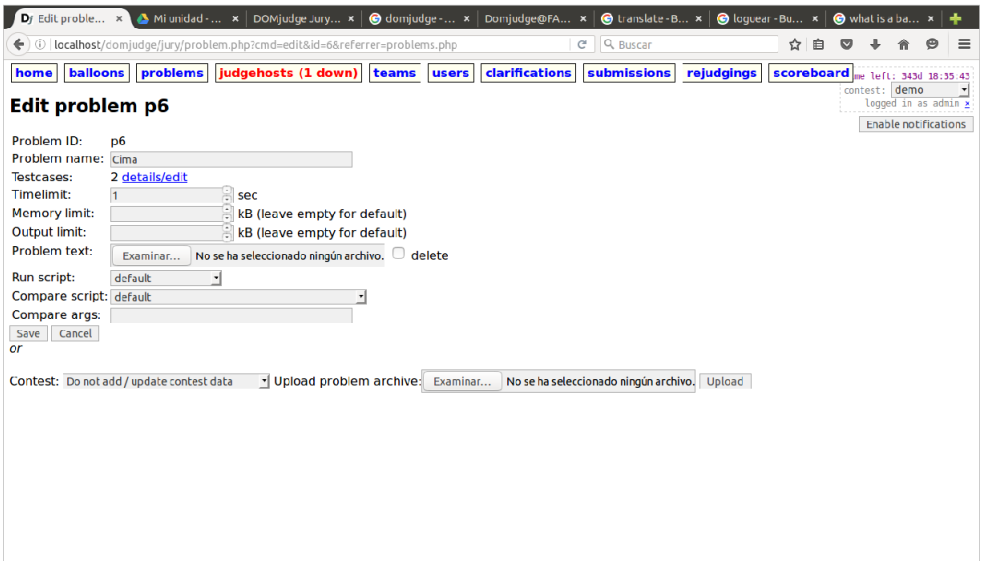

Figura 3.6: Editar problemas

encuentran activos estarán habilitados para recibir soluciones y corregirlas. Por el contrario, si no se encuentran activos no podrán procesar estas peticiones. Ver figura 3.7

4. Teams: muestra los equipos participantes que existen en la base de datos con sus categorías y los concursos en los cuales participan. Ver

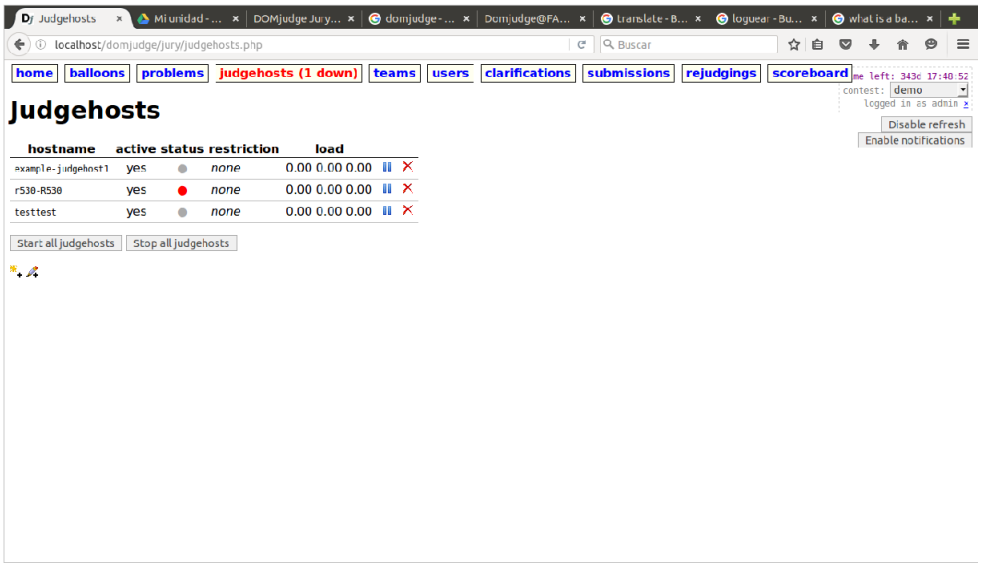

Figura 3.7: JudgeHost

figura 3.8

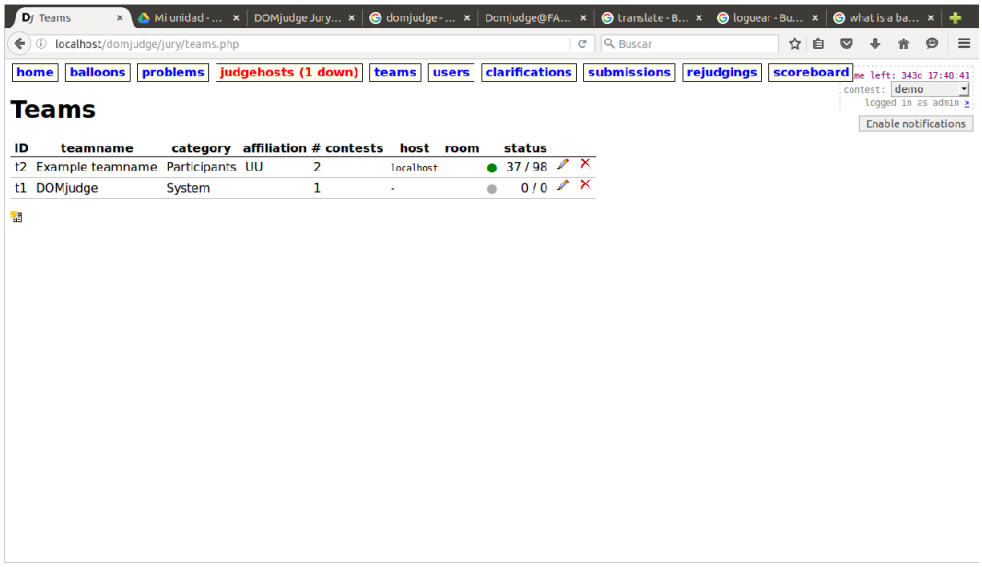

Figura 3.8: Equipos

5. Users: permite la modificación y eliminación de los usuarios que se encuentran en la base de datos. También se pueden crear nuevos usuarios con el rol de administrador y darles permisos de distintos tipos. Ver figura 3.9

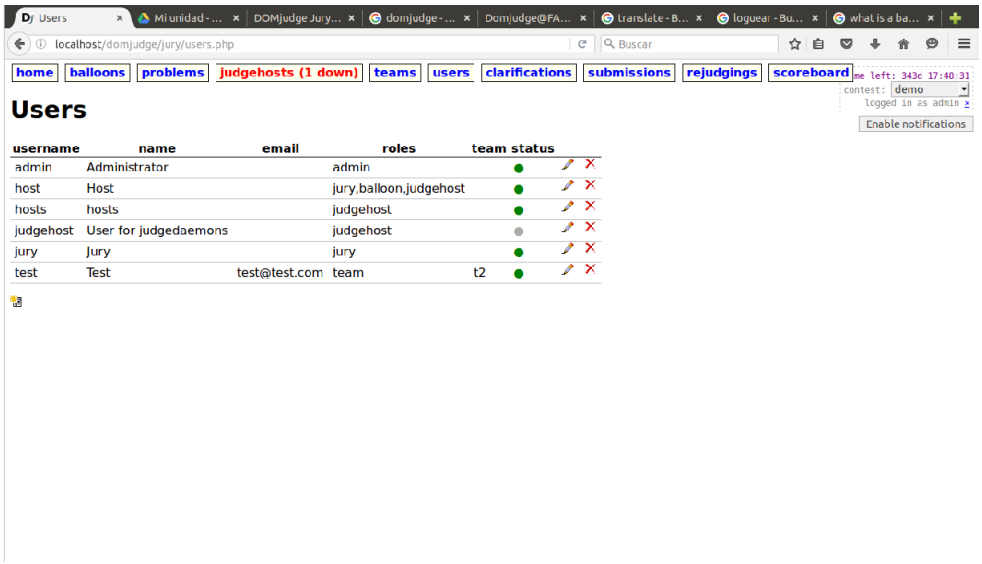

Figura 3.9: Usuarios

a) Editar o añadir usuario: en la captura se muestran las diferentes opciones a la hora de editar o añadir un usuario. En la casilla IP Address se le puede asignar la IP desde la cual puede tener acceso ese usuario. Esta es una de las muchas medidas de seguridad incluidas en Domjudge.Ver figura 3.10

6. Clarifications: permite al jurado envíar las clarificaciones sobre la corrección o el problema en general. También tiene la función de responder a las dudas que les pueden surgir a los participantes durante el concurso. Ver figura 3.11

7. Submissions: muestra la lista de envíos que han realizado los participantes con sus estados de resolución. Se puede ver los detalles de cada envío haciendo click sobre cada uno de ellos.Ver figura 3.12

a) Detalles de entrega: Se muestran todos los aspectos de la entrega tales como el equipo que la ha enviado, el concurso al que pertenece o el problema que se esta resolviendo. También permite el visionado del código fuente de la entrega. En la figura 3.13 abajo se muestran los mensajes de compilación y comparación de salidas con los casos de pruebas.

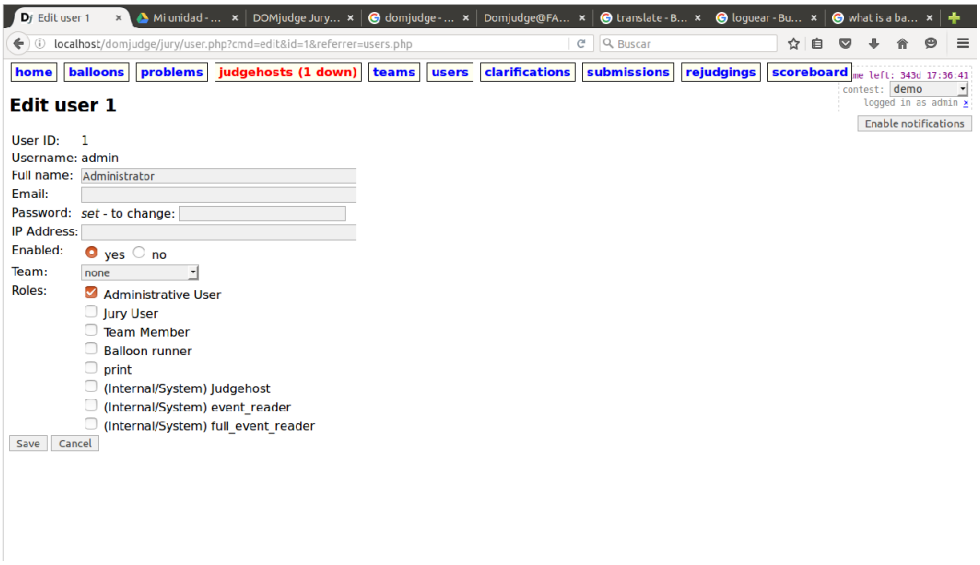

Figura 3.10: Editar o añadir usuarios

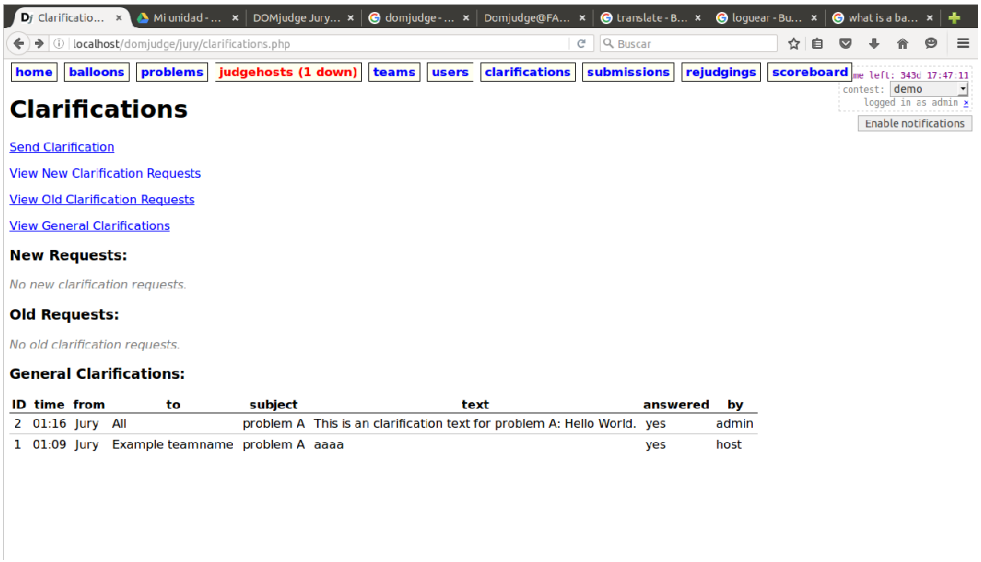

Figura 3.11: Clarificaciones

8. Rejudgings: todas las entregas tienen la opción de ser rejuzgadas ya sea por un error en los casos de prueba u otras causas. En nuestro caso no hemos vuelto a juzgar ninguna entrega por lo que se muestra la pantalla vacía. figura 3.14

9. Scoreboard: como se ha mencionado anteriormente se trata del mismo

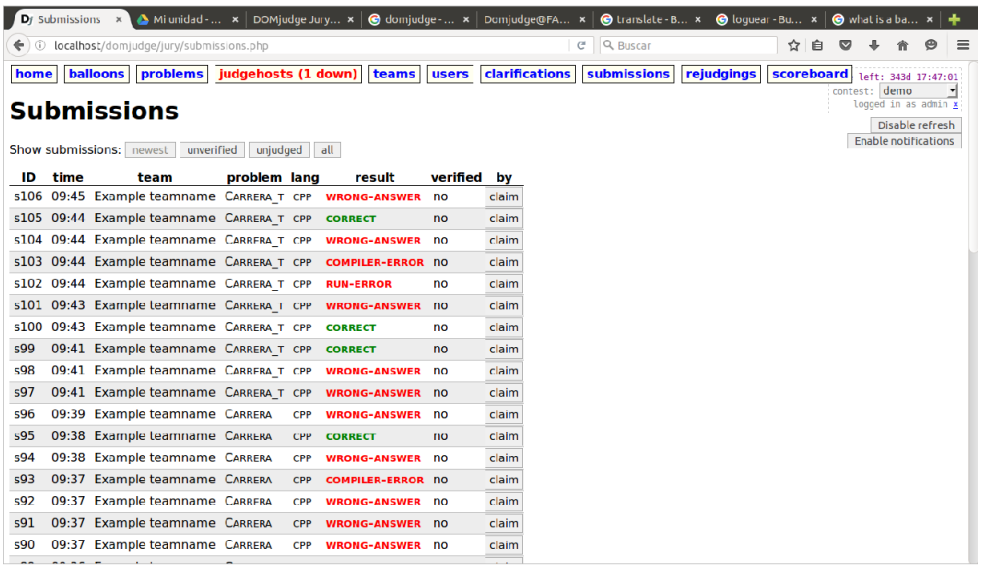

Figura 3.12: Entregas

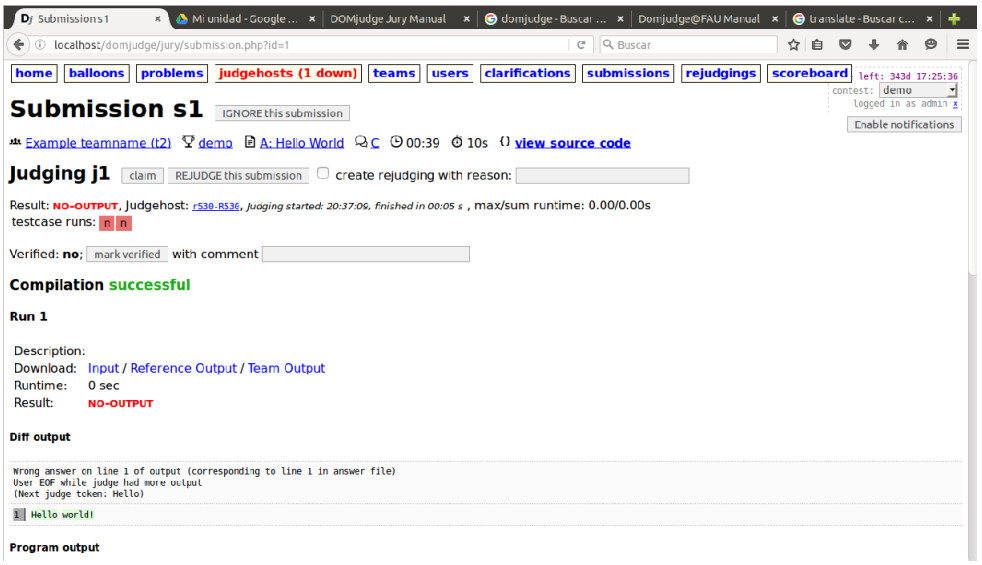

Figura 3.13: Detalles de entrega

marcador que puede ver el público.

3.3.3. Panel del jurado

El jurado es el que se encarga de supervisar el concurso evitando que ocurra algún tipo de error, es capaz de corregir y de responder a las dudas que puedan surgir durante el concurso. El panel de control del jurado es

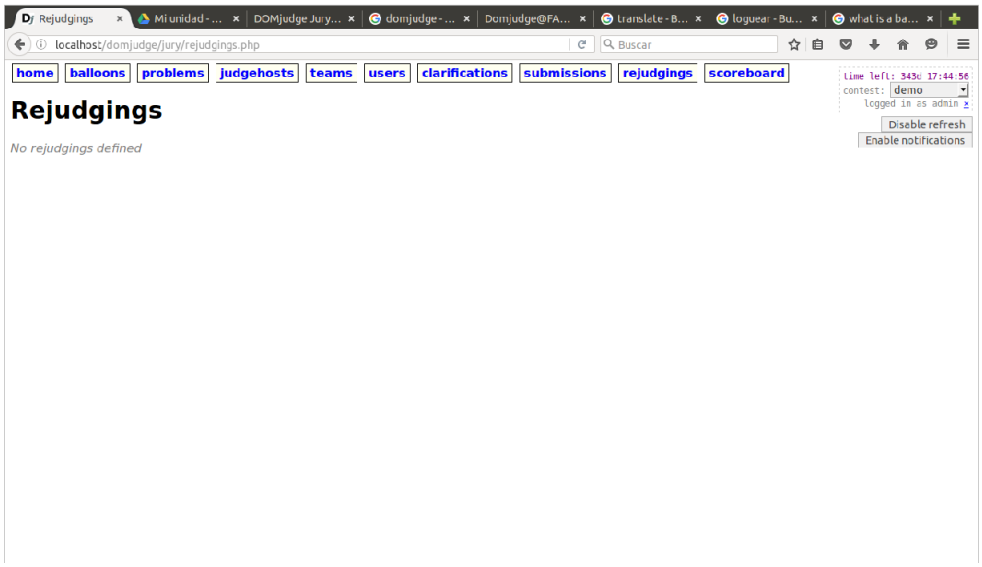

Figura 3.14: Re-juzgar

como se muestra en la figura 3.15.

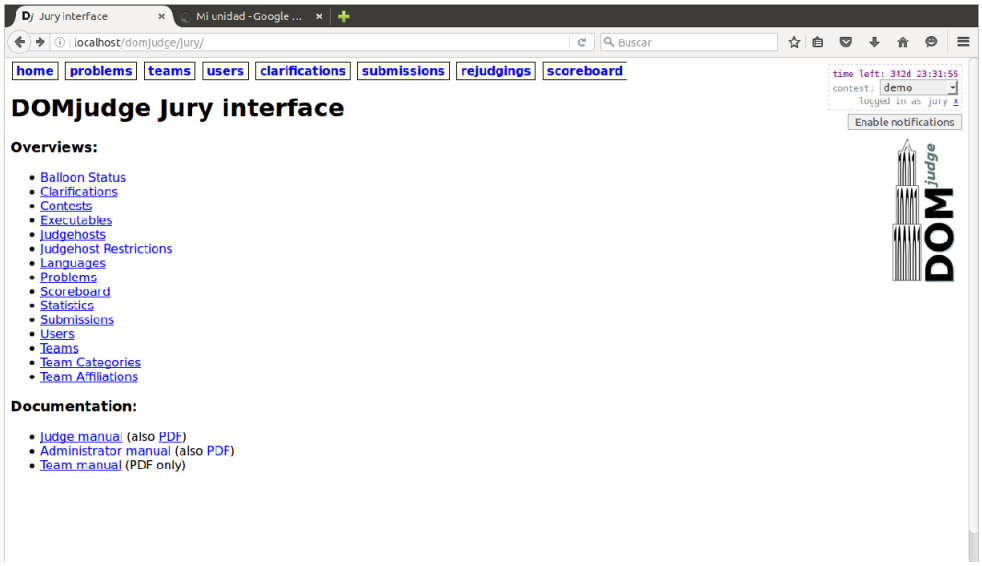

Figura 3.15: Panel del jurado

Como se puede observar, las opciones del menú superior se han reducido y no se tiene acceso a la configuración del administrador. El jurado tiene permisos para acceder a gran parte de la información del concurso pero carecen de los privilegios suficientes como para modificar dichos concursos.

3.3.4. Panel del equipo concursante

Para los concursantes el acceso es bastante restringido. Sólo se les pemite la participación en el concurso, limitando sus permisos en gran medida. El panel es como se muestra en la figura 3.16.

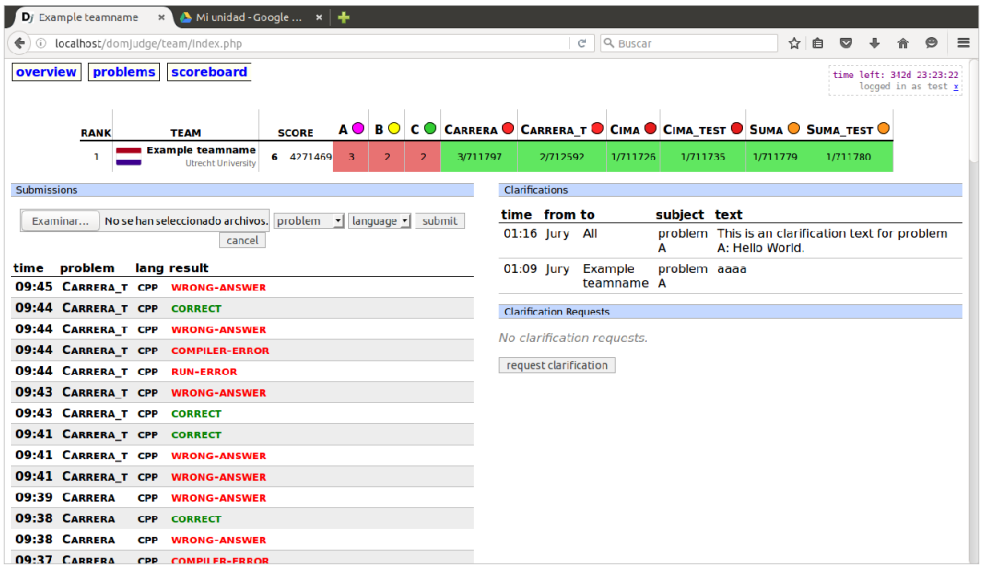

Figura 3.16: Panel del concursante

En el menú de la figura 3.16 sólo tiene acceso a la visión general, los problemas y el marcador. En la sección de visión general se muestran los equipos que participan y sus puntuaciones hasta el momento. Más abajo se encuentra el formulario de entrega, las aclaraciones y las dudas del concursante.
Es en esta pantalla donde los equipos son capaces de enviar las soluciones implementadas y quedarse a la espera de la respuesta.

3.4. Guía de utilización de DomJudge

A continuación veremos como organizar un concurso paso a paso de manera detallada:

1. El administrador introduce un problema nuevo y añade casos de prueba para el ejercicio. Ver figura 3.17.

2. El administrador crea un nuevo concurso con las configuraciones deseadas, como por ejemplo el que podemos apreciar en la siguiente captu-

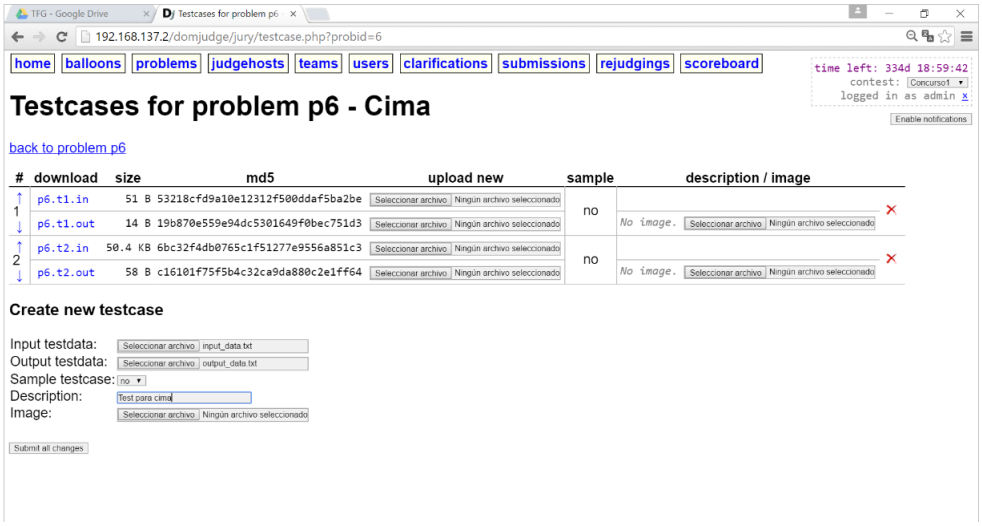

Figura 3.17: Introducir un problema nuevo

ra.Ver figura 3.18.

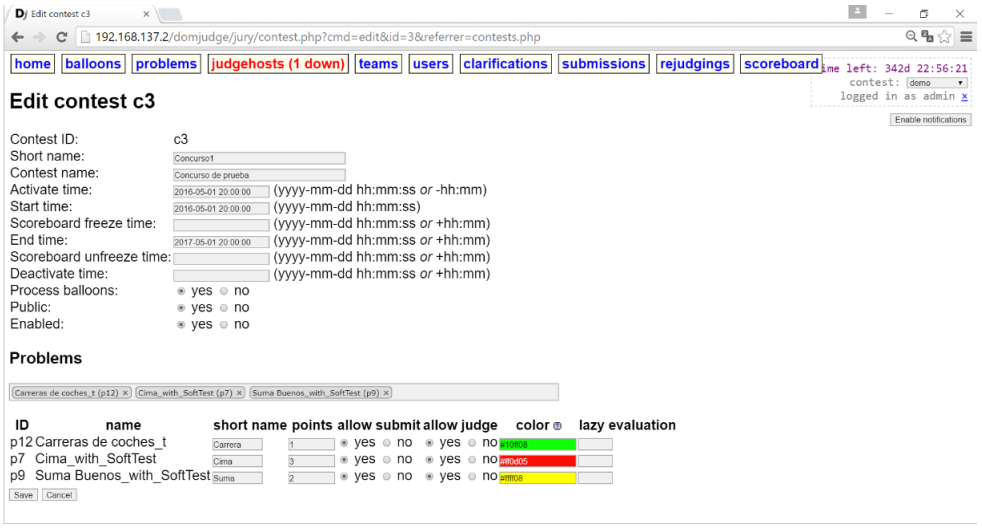

Figura 3.18: Crear un nuevo concurso

Tenemos un Concurso de prueba que ha sido iniciado el día 2016-05-01 a la hora 20:00:00 y acabará el 2017-05-01 a las 20:00:00. Esta configurado de manera que es visible al pblico con 3 problemas: Carreras de coches_t, Cima_with_SoftTest y Suma Buenos_with_SoftTest. A cada uno de los problemas se les puede asignar el nombre que se desee y la puntuaciń que tiene. Además de otras configuraciones es posible

asignar colores con el fin de diferenciarlos, esta utilidad suele usarse como elemento separador de diferentes metodologías de resolución de algoritmos, permitiendo agrupar aquellos que sigan una misma estrategia.

Una vez guardado el concurso el resto de usuarios podrá acceder a él cuando llegue la fecha de activación. Se podrá acceder al concurso en la parte superior derecha. Se puede observar en la figura 3.19.

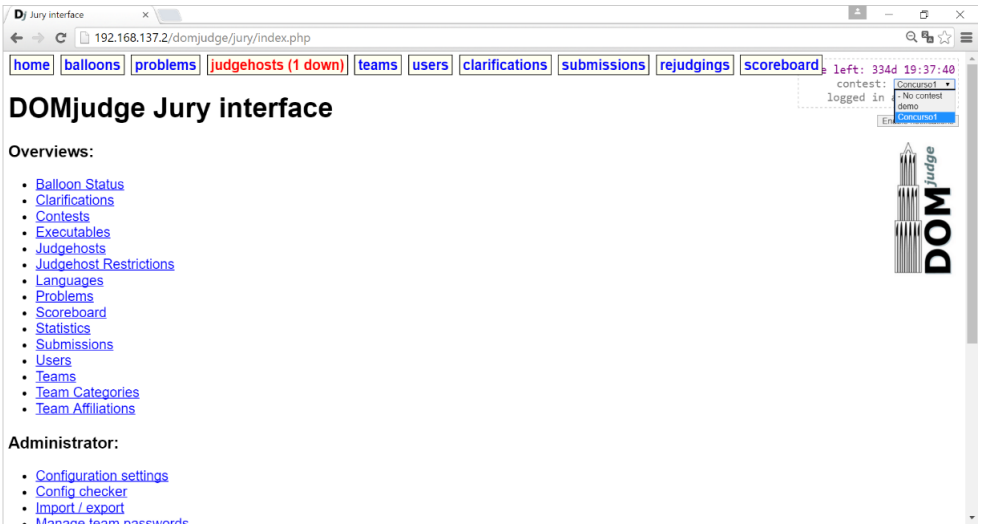

Figura 3.19: Acceder al concurso

3. Llegada la fecha de activación del concurso, el jurado podrá acceder a DomJudge y supervisar las entregas de los equipos. Ver figura 3.20.

   En el ejemplo de la figura 3.20, el jurado podrá observar que el equipo llamado Example teamname ha realizado una entrega con ID s107 a las 00:27 con lenguaje C++ al problema Carrera y el estado de la entrega está en espera a la corrección.

4. Finalmente el equipo recibe la corrección y se observa en la figura 3.21.

   La corrección devuelve un error, por lo tanto el concursante deberá intentarlo de nuevo.

5. Existe la posibilidad de pedir aclaraciones al jurado sobre la corrección. Como se puede ver en la figura 3.22.

Figura 3.20: Supervision del jurado

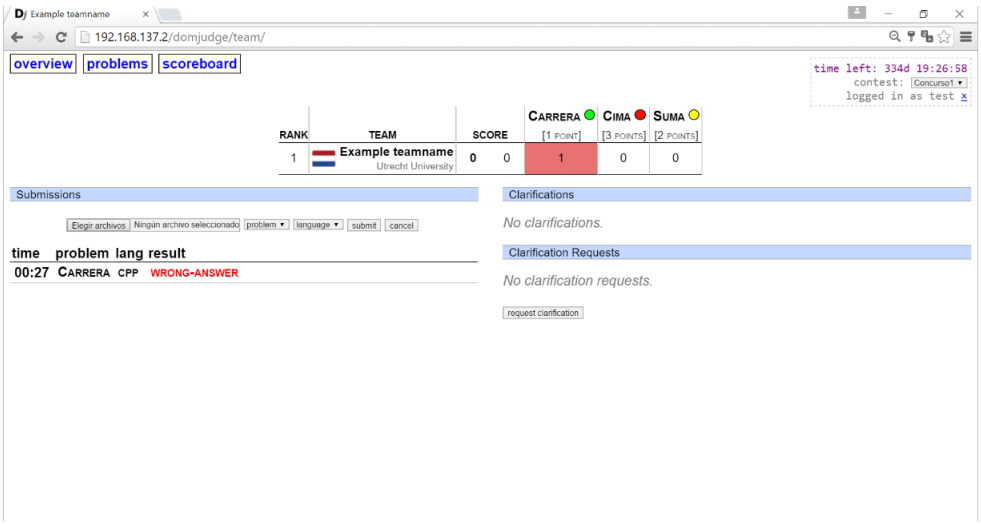

Figura 3.21: Corrección

6. El jurado en su panel podrá ver la duda y podrá responderle. Ver figura 3.23.

7. La clarificación podrá ser enviada únicamente al equipo que ha realizado la consulta o si el juez considera que esta duda interesa a todos los equipos podrá enviarla a todos los participantes del concurso. Ver figura 3.24.

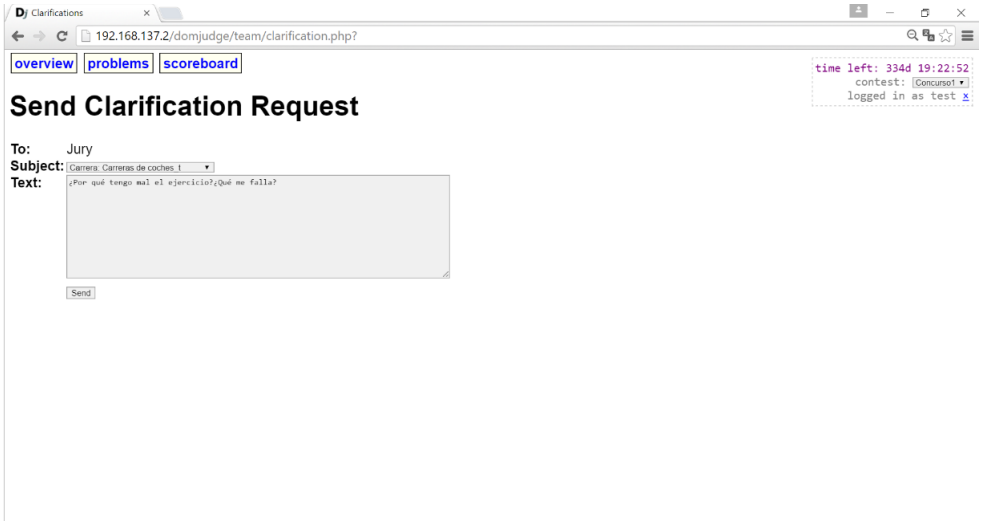

Figura 3.22: Realizar preguntas

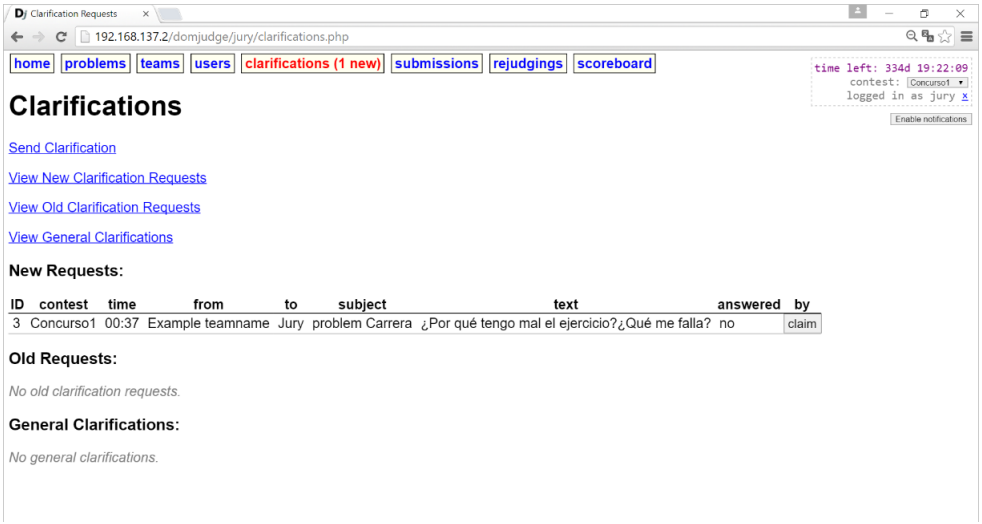

Figura 3.23: Panel de dudas del jurado

8. El concursante recibe la respuesta. Ver figura 3.25.

3.5. Resultados de las entregas

El resultado de cada entrega realizada por los usuarios, se determina mediante unos procesos que corren internamente dentro del sistema, estos procesos son capaces de decidir el estado final de la solución, no limitando su valor a correcto o erroneo, sino a una gran variedad de posibilidades, estas

Figura 3.24: Responder a las dudas

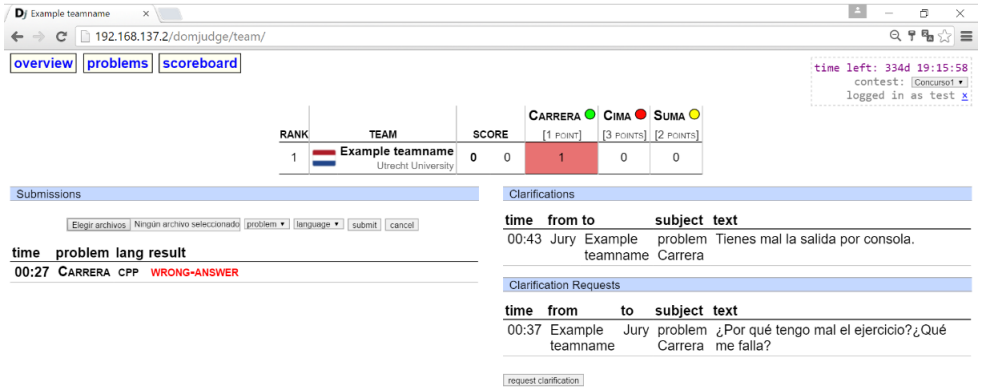

Figura 3.25: Visualizar las respuestas

son:

- QUEUED/PENDING: cuando la entrega está en la cola a la espera de corrección.
- JUDGING: cuando la entrega está en proceso de corrección.
- TOO-LATE: la entrega ha sido enviado después de que el concurso haya acabado.
- CORRECT: la entrega ha sido correcto.

- COMPILER-ERROR: ha producido un error de compilación.
- TIMELIMIT: el programa ha excedido el tiempo de respuesta.
- RUN-ERROR: ha producido un error al ejecutar el programa con los test.
- NO-OUTPUT: no hay salidas.
- WRONG-ANSWER: la salida no es correcta con la esperada.

3.6. Conclusión

Gracias a Domjudge el concursante podrá participar en los concursos que desee, saber si su solución es la correcta de manera casi inmediata y recibir aclaraciones por parte del jurado. Todo ello sin estar de forma presencial frente al jurado y pudiendo saber en todo momento como va comparado con el resto de participantes.

Este sistema cuenta con una gran versatilidad permitiendo su uso en las áreas de educación consiguiendo así un desacoplamiento del lenguaje utilizado, de estar forma resulta bastante sencilla utilizar el sistema de DomJudge como complemento a la educación. Debido a las características que presenta, también permite forzar a los alumnos a hacer que sus entregas cumplan criterios de eficiencia estableciendo un tiempo límite de ejecución, descartando todas aquellas que no sigan estos criterios. Otras opciones útiles del sistema es la capacidad para realizar anuncios globales para todos los alumnos, lo que permite dar ciertas clarificaciones que se crean recomendables para problemas concretos.

Debido a estas características su uso se ha ido extendiendo en la Facultad de Informática de la Universidad Complutense de Madrid, siendo utilizado en aquellas asignaturas que hacen un especial énfasis en la algoritmia, como puede ser Fundamentos de la Programación, Estructura de Datos y Algoritmos o Técnicas de la Ingeniería del Software. Su uso no ha quedado sólo en el ámbito académico, ya que también resulta una gran herramienta para la realización de concursos de programación, llegando a

Para nosotros representa una buena herramienta a la cual acoplar nuestro trabajo sobre jPET para agilizar aún mas todo el trabajo del jurado, en este caso el relacionado con la generación de tests para los problemas.

Notas bibliográficas

- Página oficial de DomJudge. www.domjudge.org

En el próximo capítulo...

En el próximo capítulo se explica la diferencia entre PET a jPET y una breve descripción de cada uno de ellos.

Capítulo 4

jPET

El testing de componentes puede ser muy efectivo para mostrar la presencia de errores, pero absolutamente inadecuado para demostrar su ausencia

Edsger Dijkstra

Resumen: En este capítulo se explicará brevemente la herramienta jPET, su origen y su uso.

4.1. Introducción

jPET es una extensión de PET que recibe ficheros codificados en el bytecode de java y lo muestra de manera comprensible a nivel de código fuente de Java. Para este propósito, jPET está integrado dentro del entorno de programación Eclipse ampliando de esta forma la funcionalidad de PET con el objetivo de ayudar a los desarrolladores a probar los programas de Java durante su desarrollo.

Además, la herramienta jPET dispone de las capacidades necesarias para llevar a cabo la comprensión del programa en Java o en bytecodes a nivel de método. La comprensión se puede hacer mediante la observación (usando el visor y observando las trazas explicados anteriormente) de los comportamientos de las entradas y salidas del método en cuestión con una serie de ejecuciones concretas.

4.2. PET

PET (Partial Evaluation-based Test Case Generator for Bytecode) es un prototipo de una investigación cuyo propósito es generar casos de prue-

ba de forma automática apoyándose en la técnica de evaluación parcial. La herramienta recibe como entrada un programa en bytecode y una serie de parámetros opcionales. PET ejecuta el bytecode simbólicamente y devuelve como salida un conjunto de casos de prueba. Cada caso de prueba está asociado a una rama del árbol de ejecución. Construido de acuerdo con el criterio de recubrimiento. Se expresa como un conjunto de restricciones sobre los valores de entrada y contiene una descripción de los contenidos de la memoria dinámica (o heap).

Las restricciones de la memoria dinámica imponen condiciones sobre la forma y contenidos de las estructuras de datos del programa alojadas en esta misma. PET hace uso de un resolutor de restricciones que genera valores concretos a partir de dichas restricciones, permitiendo la construcción de los tests. PET puede usarse a través de una interfaz de línea de comandos o bien usando una interfaz web. Además soporta una variedad de opciones interesantes, como la elección de criterios de recubrimiento o la generación de tests jUnit.

La generación de los casos de pruebas tiene 2 fases:

1. Descompilación del bytecode en programas con restricciones. La ventaja de la descompilación es la libertad de manejo sobre las restricciones.

2. Generación de casos de pruebas a partir del programa con restricciones generado anteriormente. Para ello, PET tiene integrado un evaluador de Programación con Restricciones con el que puede resolver los programas.

4.3. jPET

jPET es una herramienta de generación automática de casos de prueba integrada en el entorno de desarrollo Eclipse con el propósito de ayudar a los desarrolladores a probar sus programas en Java durante el proceso de desarrollo del software.

jPET dispone de varias funcionalidades principales:

- Permite ejecutar PET desde la interfaz de Eclipse y seleccionar los argumentos y los métodos a ejecutar a través de una ventana de diálogo.

- Incorpora un visor de casos de prueba (test-case viewer) para visualizar la información relativa a los casos de prueba. El visor muestra los contenidos de la memoria global representando las estructuras de datos alojadas en la memoria dinámica y sus relaciones (denominado heap). Puede mostrar el contenido de la memoria antes de la ejecución (heap de entrada) y después de la ejecución de cada caso de prueba (heap de salida).

- Puede mostrar la traza de ejecución de un caso de prueba dado (i.e., la secuencia de instrucciones que el caso de prueba ejecutaría) de dos formas distintas:

  1. Marcando todas las instrucciones implicadas
  2. Permitiendo al usuario reproducir la secuencia de instrucciones paso a paso usando la interfaz de depuración de Eclipse.

- Por último, puede analizar sintácticamente precondiciones de métodos escritas en JML (Java modeling language) y usarlas para evitar la generación de casos de prueba poco interesantes.

4.4. Conclusión

El uso de una herramienta de generación automática capaz de funcionar con tan solo un código fuente proporciona un gran abanico de oportunidades para su integración en diferentes contextos. Permitiendo aplicar técnicas de testing en diferentes ámbitos. En el caso de nuestro trabajo, jPET nos permitirá generar test-cases de manera automática. Siendo capaces de crear un conjunto de tests realmente significativo a partir de una solución implementada en Java. En este proyecto, se ha utilizado jPET para la obtención de las entradas y salidas de problemas planteados en el contexto de DomJudge. Para generar el xml con las salidas y entradas adecuadas jPET necesita un código fuente en Java. Una vez haya generado el xml dispondremos de las salidas y entradas correspondientes, ideales para los test-cases. Esto supone un problema debido a que en la actualidad Domjudge no está preparado para aceptar los ficheros de entrada y salida como XML. Por lo que deberemos realizar una traducción manual al formato requerido. En el capítulo 5 se mencionará como se ha seguido el proceso de integración y como se han solucionado todos los inconvenientes surgidos de esta adaptación. Se ha decidido hacer uso de jPET frente a otras herramientas, ya que contamos como tutor del proyecto con Miguel Gómez-Zamaolla, uno de los creadores de la herramienta, capaz de realizar tareas de mantenimiento y adaptación.y solucionar problemas que surjan relacionados con ella.

Notas bibliográficas

- Página oficial de PET. https://costa.ls.fi.upm.es/pet/pet.php

- Elvira Albert, Israel Cabañas, Antonio Flores-Montoya, Miguel Gómez-Zamalloa, Sergio Gutiérrez, *jPET: an Automatic Test-Case Generator for Java*, Complutense University of Madrid, Spain.

En el próximo capítulo...

En el próximo capítulo se centra en los detalles del desarrollo del módulo Solver y su integración en DomJudge.

Capítulo 5

Integración de jPET

No te preocupes si no funciona bien. Si todo estuviera correcto, serías despedido de tu trabajo

Ley de Mosher de la Ingeniería del Software

Resumen: A continuación se detallan la implementación del módulo SoftTest que se utiliza para traducir los ficheros xml resultantes del uso de jPET a ficheros in y out, así como las etapas del proceso de integración del módulo SoftTest a DomJudge.

5.1. Introducción

El núcleo de este proyecto es la integración de un generador automático de casos de prueba a una aplicación web de juez en línea. Para este proyecto se ha elegido DomJudge como el juez donde se integrará el módulo del generador automático de casos de prueba.

Este juez online permiten por un lado publicar problemas de programación, y que los alumnos entreguen sus soluciones, las cuales son evaluadas en el acto. El uso de estos jueces online está resultando muy práctico como medio para que los alumnos puedan autoevaluar ejercicios prácticos de programación, y también para los profesores como medio de evaluación a´ nadido a los exámenes finales.

Para que la autoevaluación realizada por estos jueces sea efectiva, los profesores deben proporcionar casos de prueba de calidad que son ejecutados automáticamente cuando los alumnos suben sus soluciones. Escribir estos casos de prueba resulta costoso y complejo, y es en este punto, donde el uso de jPET podría resultar muy útil.

La integración consiste en un trabajo investigación acerca de la viabilidad de incluir el generador automático jPET en distintos ámbitos de DomJudge, permitiendo al público que haga uso de el. Como se ha explicado con anterioridad, DomJudge se trata de un juez en línea con las características ideales para su ampliación ya que además de ser un sistema de código abierto, su uso se esta comenzando a extender en el ámbito de la Informática.

El objetivo a conseguir es realizar una integración de la herramienta de generación de casos automáticos al entorno con el fin de facilitar la tarea de la generación de los casos de prueba. En el hipotético caso de que el generador sea lo suficientemente sofisticado, se podrían llegar a conseguir test mejores o iguales que aquellos que pudieran aportar los profesores. Dado que se va a aplicar a un ámbito docente, su eficacia carece de tanta relevancia ya que por norma general, los casos de prueba generados automaticamente, irán acompañados de otros tests complementarios, generalmente diseñados mediante la generación aleatoria o aplicando técnicas de caja negra.

Esta idea innovadora está enfocada a integrar un generador automático sobre DomJudge, sin descartar nunca la opción de incluir otras herramientas auto generadoras que puedan servir como sustituto o complemento de la principal. En este proyecto se ha elegido jPET como generador principal y también incluye otras alternativas como EvoSuite que se explicará más adelante.

jPET nos permite analizar programas en java y obtener aquellas entradas que servirán para explorar todos los caminos posibles con sus correspondientes salidas. El fichero que proporciona es un archivo xml. Un archivo xml es un tipo de fichero que permite su utilización en cualquier contexto pero que genera ciertos inconvenientes a la hora de obtener sus datos ya que la lectura puede llegar a ser compleja, sin permitirnos realizar un análisis directo, asi como una traducción inmediata, teniendo el problema de que los jueces en línea en la actualidad no son capaces de admitir estos formatos de entrada y salida. Para aprovechar el potencial que ofrece jPET en el ámbito de los jueces virtuales y otro tipos de programas de testing decidimos llevar a cabo la traducción de los ficheros xml generados por jPET y transformarlos en ficheros in y out que son el estándar. Para ello creamos la herramienta SoftTest.

5.2. ¿Cómo funciona?

La herramienta SoftTest está pensada para manejar jPET y evitar así que el usuario inexperto se vea obligado a manejar los complicados parámetros de entrada de jPET. La herramienta puede ser utilizada por línea de comandos o a través de una interfaz gráfica.
Con la interfaz gráfica permitimos que otros usuarios que no vayan a usar el juez virtual puedan usar la herramienta de manera intuitiva. Mientras que

con la opción de línea comandos los usuarios más avanzados tienen acceso a más opciones como cambiar la ruta de destino del guardado y la del jPET que deseen utilizar. Además gracias a esta segunda opción el juez virtual (en este caso domjudge) puede lanzar la herramienta el mismo con una intervención mínima por parte del usuario y sin que el usuario se percate siquiera del uso de la herramienta para sacar los ficheros de prueba.

5.3. Las distintas fases de desarrollo

Durante la implementación de la herramienta esta paso por distintas fases y ha dado como resultado final el SoftTest ahora integrado en nuestra instalación de Domjudge. A continuación veremos estas distintas fases.

5.3.1. Primera fase: pruebas y definición

La primera versión se limitaba a facilitar el lanzamiento de la herramienta jPET y a guardar el fichero xml. Solo era capaz de funcionar para métodos con un parámetro de entrada int y que devolviera de nuevo un int. Se desarrolló también una primera interfaz gráfica para facilitar el manejo de la herramienta.

5.3.2. Segunda fase: Traducción xml y ficheros de salida

Tras estudiar cómo funciona jPET y cómo se estructuran los ficheros xml que genera se pasó al desarrollo de un primer sistema de traducción. Para recorrer el xml el sistema utiliza el paquete externo jdom. El traductor de nuevo esta orientado solo a int. La traducción se realiza recorriendo el xml entero y recogiendo los valores que se consideran necesarios (input y output) para generar los ficheros in y out.

5.3.3. Tercera fase: arrays de int

Se decide pasar de ints a arrays de ints. Se lleva a cabo una modificación general de la herramienta SoftTest tanto en la manera de lanzar el ejecutable jPET como en el proceso de traducción que ahora admite también arrays de int aunque conserva la capacidad de traducir xml de ints.

5.3.4. Cuarta fase: String

Una retirada a tiempo es una victoria.
Napoleón Bonaparte

Durante mucho tiempo trabajamos con el objetivo de que el método principal recibiera un String y devolviera un String. Ya que al recibir un String

podría traducirlo luego el propio método en el tipo de variable que necesitara y que devolviera un String nos permitía mostrar prácticamente cualquier salida que consideráramos oportuna.
La herramienta SoftTest recibió cambios para adaptarse a estas nuevas especificaciones. Sin embargo estos cambios tuvieron que ser desechados en su mayoría ya que jPET no fue capaz de manejar Strings de la manera prevista y nos obligó a dar un paso atrás con el fin de retomar la vía correcta en el desarrollo de la herramienta.

5.3.5. Quinta fase: Depuración de la traducción y vuelta a los arrays

Dado los problemas que origino el uso de Strings en jPET nos vimos obligados a limitar el uso de la herramienta a arrays de int que es donde daba los mejores y más prometedores resultados.
Tras haber decidido que limitaríamos el uso de la herramienta a este sector proseguimos a mejorar la traducción del xml y la generación de este. Eliminamos las entradas y salidas de los ficheros in y out que devolvieran una excepción lo que nos permite generar resultados más comprensibles. Corregimos errores puntuales que aparecían a la hora de hacer la traducción en unos casos específicos.
Dedicamos muchas horas al uso intensivo de jPET con el fin de averiguar que parámetros por defecto arrojaban mejores resultados de media en los casos de prueba. Finalmente hicimos la que sería la interfaz gráfica definitiva con el fin de hacer el manejo de la herramienta lo más sencillo e intuitivo posible.

5.3.6. Fase final: Preparando la herramienta para Domjudge

Con la herramienta plenamente funcional decidimos que era hora de integrar la herramienta con el juez online Domjudge que habíamos instalado previamente en uno de nuestros equipos.
Para ello le añadimos la opción de manejar la herramienta directamente desde la consola a través de parámetros de entrada. Al ser esta una opción más avanzada le permitimos más personalización a la hora de manejar el jPET y los ficheros in y out. Gracias a esta modificación Domjudge es capaz de utilizar directamente la herramienta SoftTest sin necesidad de un intermediario. Solo necesita unos parámetros que le debe proporcionar el usuario como por ejemplo el fichero que hay que analizar.

5.3.7. Parametros jPET

Como la herramienta SoftTest busca adaptar y hacer más sencillo el manejo de jPET gran parte de ese trabajo para simplificar el uso de jPET recae en hacer una entrada de argumentos estándar que funcione bien en la mayo-

ría de los casos. Esta es la entrada que hemos escogido:

-c bck 5 -td num -d 1 10 -l ff -v 2 -tr statements -cc yes

A continuación vamos a desglosar los parámetros de entrada utilizados al llamar a jPET y veremos que hace cada uno de los argumentos previamente citados.

- -c bck 5: Define el criterio de profundidad del código. Cuanto mayor es, mas resultados genera jPET. Sin embargo si el argumento asociado a la profundidad es demasiado grande jPET no conseguirá finalizar el análisis. Con una profundidad de 5 conseguimos unos resultados interesantes en un margen de tiempo razonable.

- -td num: El parámetro td nos permite elegir si queremos casos de prueba numéricos o path-constraints. En nuestro caso ya que queremos generar entradas y salidas nos conviene el generador de casos de prueba numéricos.

- -d 1 10: Define el dominio que jPET asigna a las variables. Es decir en este caso jPET asignara valores entre 1 y 10 a las variables de entrada del programa.

- -l ff: La estrategia de etiquetado afecta a la selección de variables. En este caso hemos elegido opción ff ya que es la que mejores resultados arroja.

- -v 2: El nivel de verbosidad muestra el grado de información que nos muestra jPET. En jPET varia de 0 a 2. Hemos elegido el nivel 2 ya que queremos que la información mostrada al usuario sea la máxima posible para que este sea capaz de realizar los análisis que considere oportunos.

- -tr statements: Cuando se utiliza, jPET muestra las salidas de los casos de pruebas. La opción statements proporcionaba mejores resultados. De nuevo utilizamos esta opción para que el usuario de la herramienta disponga de toda la información que le pueda ser útil.

- -cc yes: Muestra la información del code coverage. Así el usuario puede ver la información extraída por jPET y si recibe un code coverage muy bajo podría plantearse cambiar el código.

5.3.8. Adaptación ficheros in y out

Pese a que jPET genera resultados interesantes estos no son 100 % fiables. Los ficheros in son generados con un formato específico de nuestra aplicación.

Sin embargo, muchos de los ejercicios que utilizamos como casos de prueba tienen un formato concreto, por lo tanto estamos obligados a retocar los ficheros in generados por SoftTest para que tengan el formato correcto. En las últimas versiones se ha detectado cierta incongruencia entre la correspondencia de las entradas con las salidas del problema, a pesar de esta situación, las entradas siguen siendo las generadas por el Path Coverage del CFG, manteniendo su riqueza en el entorno del testing. La solución mas inmediata es la de utilizar estas entradas como entrada para el problema, y guardar las salidas en un fichero auxiliar. Un ejemplo de como hacerlo sería este:

```
public static void main(String[] args) throws Exception {
BufferedReader bf = new BufferedReader(new
    FileReader("carreras_in_adaptadas"));

//Haciendo uso de una clase ya existente llamada Printwriter,
    podemos escribir sobre fichero con relativa facilidad.
PrintWriter writer = new PrintWriter("carreras_out_adaptadas",
    "UTF-8");
String aux1 = bf.readLine();
int numero_de_casos = Integer.parseInt(aux1);

for(int i = 0; i < numero_de_casos; i++){
/*
.
.
Codigo encargado de leer la entrada de fichero y prepararla para el
    algoritmo
.
.
*/
Carreras a = new Carreras();
//Carreras es el nombre de la clase que trata el problema
writer.println(a.solve(entrada)[0]);
//De igual forma que se escribe sobre consola se hace sobre el
    fichero.
}
bf.close();
writer.close();
//Importante no olvidar cerrar el writer, ya que no se garantiza
    que se guarden los cambios realizados en el fichero.
}
```

En el código mostrado se hace uso de la clase PrintWriter que por usabilidad y simpleza se ha considerado la ideal para esta tarea, para ello, se declará un objeto PrintWriter usando el constructor que recibe como parámetro de entrada el fichero al que deseamos escribir, y como segundo parámetro la codificación de este. Después de ejecutar cada iteración del algoritmo por cada caso de entrada, se hará una llama a este objeto para guardar la salida en el formato deseado del problema, requiriendo de una gran dependencia del problema que se esté tratando. En el GitHub asociado al proyecto se incluyen varios ejemplos mas que leen diferentes tipos de entrada que contemplan muchas mas variaciones de formatos como ayuda para realizar la adaptación.

De esta forma, ya tendríamos generados los ficheros de entrada y salidas correspondientes y solo nos faltaría incluirlos a DomJudge.

5.4. Uso de la herramienta SoftTest

La herramienta SoftTest puede ser utilizada a través de la interfaz gráfica que hemos desarrollado o bien a través de la terminal. Obviamente ambas opciones comparten ciertas características.
Para empezar aunque la herramienta SoftTest por sí sola podría funcionar en Windows no es el caso de jPET. Y puesto que utilizamos jPET para la generación del xml solo podemos utilizar SoftTest en sistemas en base Linux. jPET solo admite archivos .class por lo tanto SoftTest solo funcionara correctamente con ese tipo de archivos. El usuario está obligado a compilar previamente el programa .java (salvo si lo hace a través de Domjudge). Además también debe de quitar el package del archivo java. El no hacerlo hará que la herramienta jPET no sea capaz de analizar correctamente el archivo debido a que ha recibido una ruta errónea.

5.4.1. Uso de la interfaz Gráfica

A continuación veremos como utilizar la interfaz gráfica de la herramienta SoftTest.

- Al arrancar la aplicación aparece la siguiente figura 5.1.
  La ventana esta compuesta por un área de texto y dos botones. En el

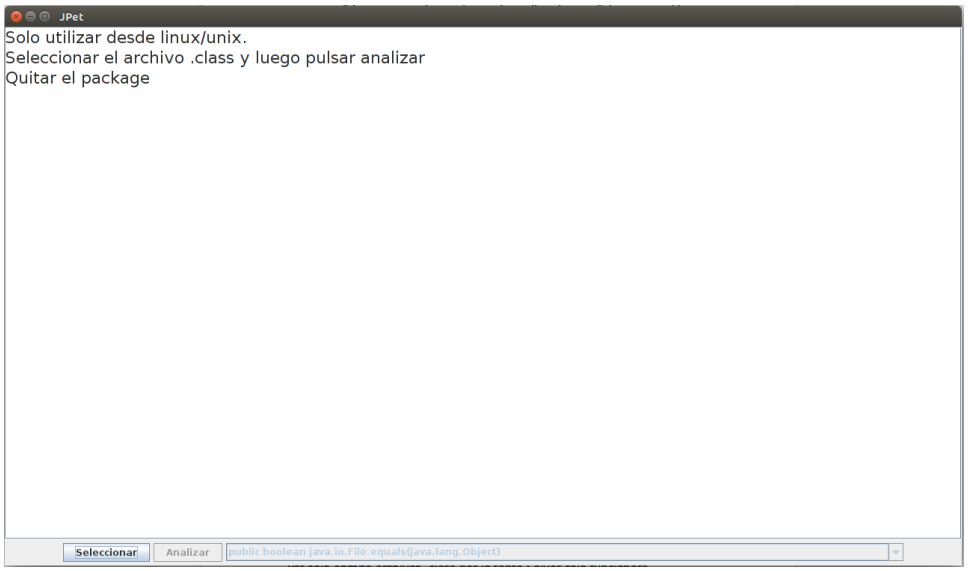

Figura 5.1: Interfáz gráfica de SoftTest

área de texto aparecen las siguientes indicaciones:

Solo utilizar desde linux/unix.
Seleccionar el archivo .class y luego pulsar analizar
Quitar el package

De los dos botones que hay solo uno esta habilitado, el de seleccionar. El de analizar permanecerá deshabilitado hasta que se seleccione un .class.

- Al pulsar el botón de seleccionar aparecerá una nueva ventana. Ver figura 5.2.
  Esta nueva ventana es un explorador de archivo configurada para mos-

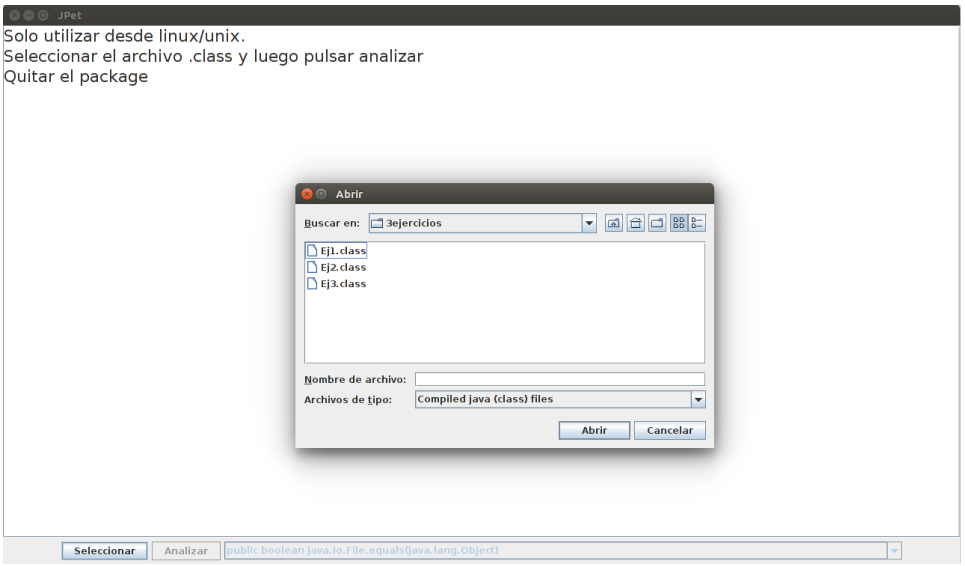

Figura 5.2: Al pulsar seleccionar

trar solo los archivos .class. Una vez que hemos encontrado el archivo que deseamos lo seleccionamos y pulsamos abrir.

- Al pulsar abrir observamos que hemos vuelto al panel inicial pero este ha sufrido ciertas modificaciones. Ver figura 5.3.
  Ahora el text area ya no muestras las instrucciones abreviadas de uso sino la ruta del archivo y su nombre. Además como ya tenemos un archivo seleccionado el botón de analizar ya esta habilitado. A continuación procedemos a hacer click en él.

- Observamos que la herramienta ahora nos muestra otros datos. Ver figura 5.4.
  La herramienta nos muestra directamente a través del text area entradas y las salidas generadas por jPET. Además nos muestra las rutas

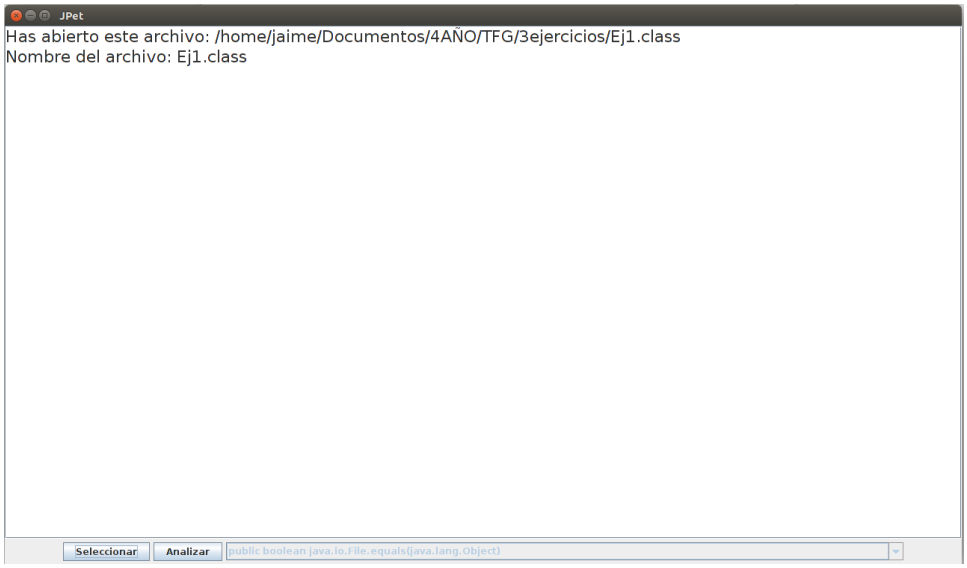

Figura 5.3: Archivo seleccionado

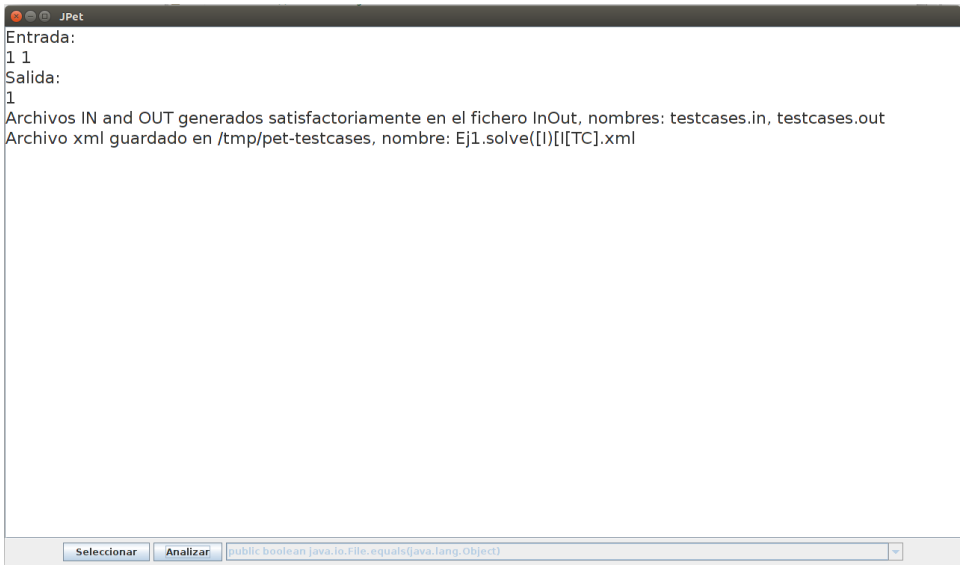

Figura 5.4: Mensajes de salida de SoftTest

donde se han guardado en xml que ha generado jPET durante su ejecución y los ficheros in and out generados por la herramienta SoftTest al traducir el xml.
Si nos desplazamos hasta la ruta proporcionada por SoftTest podemos observar dos carpetas. Ver figura 5.5.

En la carpeta InOut se almacenan los ficheros in y out y en la carpeta

Figura 5.5: Ficheros generados

pet-testcases se almacena el xml.
Si abrimos los archivos in y out comprobaremos que coincide con lo que nos ha mostrado previamente la herramienta SoftTest. Ahora el usuario puede utilizar estos archivos como deseé. Puede incluso subirlos directamente a Domjudge sin necesidad de utilizar el complemento que hemos añadido a Domjudge. Ver figura 5.6.

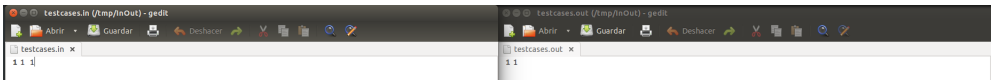

Figura 5.6: Archivos in y out

5.5. Uso de la herramienta a través de la terminal

La herramienta también puede ser usada a través de la terminal. Esta opción permite mas personalización como por ejemplo donde queremos que se guarden los in and out, el jPET que queremos utilizar, etc
Si llamamos al la herramienta sin ningún parámetro desde la terminal la interfaz gráfica se abrirá. Ver figura 5.7.
Si ponemos el parámetro -h se mostraran unas pequeñas instrucciones de uso. Ver figura 5.8.
El primer parámetro es el nombre del archivo .class sin la extensión. El segundo parámetro es la ruta del archivo en cuestión. Y el ultimo argumento es la ruta donde se guardaran los ficheros in out y donde se encuentra el jPET que se desea utilizar.
Una llamada correcta para hacer funcionar la herramienta desde la consola sería la siguiente. Ver figura 5.9.
Que mostraría el siguiente resultado. Ver figura 5.10.
El xml se guarda en este caso en la carpeta tmp/pet-testcases y los archivos in y out en la carpeta InOut localizada en home/jaime/workspace/prueba ya que es el tercer parametro.

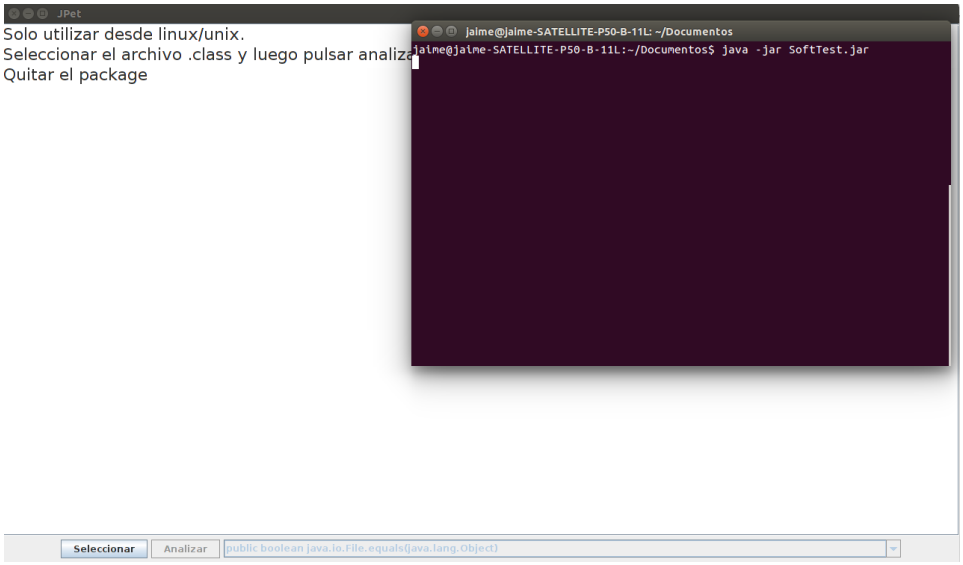

Figura 5.7: Interfáz gráfica de SoftTest

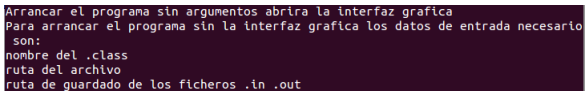

Figura 5.8: Instrucciones de uso

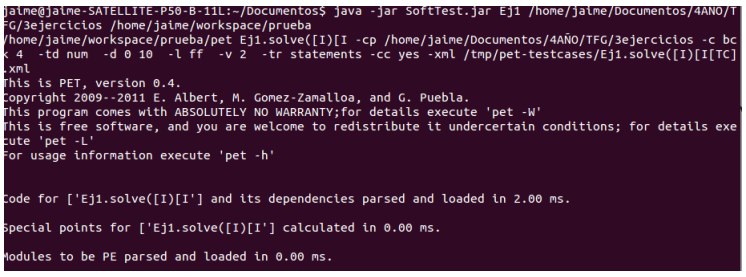

Figura 5.9: Ejemplo de llamada

5.6. Recapitulación

Los ficheros in y out generados por Domjudge pueden ser modificados posteriormente por el usuario si desea completarlos aún más puntualmente. La interfaz gráfica sigue siendo accesible si un usuario desea usar la herramienta SoftTest por su cuenta aunque también puede hacer uso de la línea de comandos si es un usuario más avanzado. Ya que como hemos visto antes

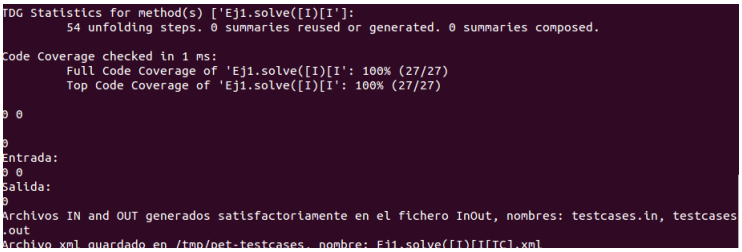

Figura 5.10: Ejemplo de resultados

le proporciona mas personalización.

5.7. Integración

La edición del código fuente de DomJudge se ha utilizado lenguajes de programación HTML y PHP en ficheros .php, que en este caso el fichero donde se van a hacer las ediciones es testcase.php.

El fichero testcase.php está localizado en el servidor donde se ha instalado DomJudge y se encarga de mostrar y editar los casos de pruebas. La estructura del fichero es en resumen:

- Declaración de los variables y extraer de la base de datos la información necesaria del problema al que desee editar proporcionando el ID del problema.
- Una vez confirmada la existencia del problema, se extrae de la base de datos todos los casos de prueba.
- Reordena los casos de pruebas según la importancia del test.
- Muestra por la interfaz web los casos de prueba con una estructura determinada.
- Muestra por la interfaz web el formulario para añadir un test nuevo.

La edición del fichero testcase.php se centra en la última parte de la estructura ya que se trata de incluir al formulario una nueva forma de añadir test. En concreto se han añadido el siguiente fragmento de código:

```
<h3>Create testcases automatically</h3>
<table>
<tr><td>Program:</td><td><?php echo
    addFileField('add_program')?></td></tr>
<tr><td>Generator:</td><td>
<select name="select_generator" id="select_generator">
```

```
<option value="jPET" selected="selected">jPET</option>
<option value="EvoSuite">EvoSuite (En desarrollo)</option>
</select>
</td></tr>
</table>
```

Éste fragmento de código simplemente integra al formulario una tabla nueva con la primera fila la entrada del fichero del código fuente del programa y la segunda fila las opciones para seleccionar el generador que se desee utilizar. Se puede dar cuenta de que existe en las opciones un generador llamado EvoSuite, esto se ha puesto como ejemplo de generadores adicionales que podrá haber en el futuro pero aún no está implementado.

El código anterior escrito en HTML sólo representa la parte de interfaz al usuario, las funciones que generan los casos de pruebas se detalla a continuación.

Primero comprobar que el usuario ha seleccionado el fichero del programa, en caso afirmativo pues se crean las variables necesarias y se comprueba si el fichero subido es vacía. Rank es el identificador de los casos de prueba y tienen que ser únicos.

```
if ( !empty($_FILES['add_program']['name']) ) {
$content = array();
$rank = $maxrank + 1;

if ( empty($_FILES['add_program']['name']) ) {
warning("No program file specified for new testcase,ignoring.");
} else {
```

La función checkFileUpload comprueba si el fichero ha sido subido correctamente sin ningún tipo de error.

```
checkFileUpload ( $_FILES['add_program']['error'] );
```

La función move_uploaded_file permite guardar el fichero en otro directorio que en este caso opinamos que sería mejor guardarlo en un directorio temporal para luego ser borrado.

```
move_uploaded_file( $_FILES['add_program']['tmp_name'], TMPDIR .
    '/' . $_FILES['add_program']['name']);
```

jPET recibe como programa de entrada sólo los archivos binarios, por lo que es necesario se compilados antes de pasar a jPET.

```
if( !system('javac ' . TMPDIR . '/' .
    $_FILES['add_program']['name'] ) ){
```

En el caso de que se haya compilado correctamente, pues se ejecuta jPET

pasándole los parámetros correspondientes que se han detallado en otro capítulo.

```
system(LIBDIR . '/SoftTest.jar ' .
    basename($_FILES['add_program']['name'],".java") . ' ' . TMPDIR
    . ' ' . LIBDIR );
```

Hemos decidido en borrar los archivos temporales que no se van a usar más para no ocupar la memoria del servidor.

```
unlink(TMPDIR . '/' .
    basename($_FILES['add_program']['name'],".java") . '.class');
}else{
warning("Compilation error");
}
unlink(TMPDIR . '/' . $_FILES['add_program']['name']);
}
```

Se comprueba que jPET ha generado los ficheros in y out correctamente, éstos ficheros es donde están los casos de pruebas.

```
if ( !file_exists(TMPDIR . '/testcases.in') || !file_exists(TMPDIR
    . '/testcases.out') ) {
warning("No input and output file generated for new testcase,
    ignoring.");
} else {
```

Se leen los contenidos de los ficheros in y out para luego guardarlos en la base de datos. Los ficheros in y out también se van a borrar por la misma causa de no acumular la memoria del servidor.

```
$content['input'] = file_get_contents(TMPDIR . '/testcases.in');
$content['output'] = file_get_contents(TMPDIR . '/testcases.out');
unlink(TMPDIR . '/testcases.in');
unlink(TMPDIR . '/testcases.out');
}
```

Se introduce en la base de datos toda la información del test. Los datos que no se han obtenido se dejan por defecto, que en caso de DomJudge los interpreta como null.

```
$DB->q("INSERT INTO testcase ( probid, rank, md5sum_input,
    md5sum_output, input, output, description, sample) VALUES
    (%i,%i,%s,%s,%s,%s,%s,%i)", $probid, $rank,
    md5(@$content['input']), md5(@$content['output']),
    @$content['input'], @$content['output'], @$_POST['add_desc'],
    @$_POST['add_sample']);
```

Se registra las acciones que se han hecho de introducir nuevo test y genera el mensaje de salida para el usuario según el tamaño de los test y si están vacías.

```
auditlog('testcase', $probid, 'added', "rank $rank");

$result .= "<li>Added new testcase $rank. ";
if (
    $_FILES['add_output']['size']>dbconfig_get('output_limit')*1024
    ) {
$result .= "<br /><b>Warning: output file size exceeds " .
    "<code>output_limit</code> of " . dbconfig_get('output_limit')
    . " kB. This will always result in wrong answers!</b>";
}
if ( empty($content['input']) || empty($content['output']) ) {
$result .= "<br /><b>Warning: empty testcase file(s)!</b>";
}
$result .= "</li>\n";
}
```

Todo el código anterior añadido al fichero testcase.php está desarrollado en base a las demás funciones que existe en el mismo fichero, de esta manera se puede asegurar su correcto funcionamiento con respecto a las comprobaciones que se deben de hacer para iniciar la siguiente acción.

5.8. Dificultades encontradas

A lo largo de la integración han surgido varios problemas que han sido resueltos uno a uno. A continuación se van a detallar cada uno de ellos con un orden cronológico de aparición de las dificultades:

1. Entender la ubicación de cada fichero. La instalación de DomJudge genera una gran cantidad de ficheros en la que cada uno tiene su funcionalidad. Al mismo tiempo, la instalación se divide en server, host y doc.

2. Encontrar el fichero donde se edita los casos de pruebas. DomJudge es una aplicación web bastante elaborada por parte de la comunidad DomJudge, por lo que su instalación añade más de 900 elementos al sistema.
   Afortunadamente, los URL de DomJudge son muy explicativos a la hora de encontrar la ubicación del fichero.

3. Entender el código del fichero testcase.php. Como se ha explicado en la sección anterior, el fichero testcase.php está estructurado de distintas funcionalidades bastante difícil de entender ya que utilizan funciones de

otros ficheros que se han cargado al inicio con funciones require(). Por ejemplo, al inicio del fichero testcase.php carga con require al init.php y en el fichero init.php también carga otros ficheros.

4. Analizar la base de datos. Los test en DomJudge son guardados en la base de datos, por lo que averiguar y entender la tabla o tablas que las guardan es primordial. Como se puede observar en la siguiente captura, en la base de datos de DomJudge hay una tabla con nombre testcase en el que se guarda todos los casos de prueba. Ver figura 5.11.

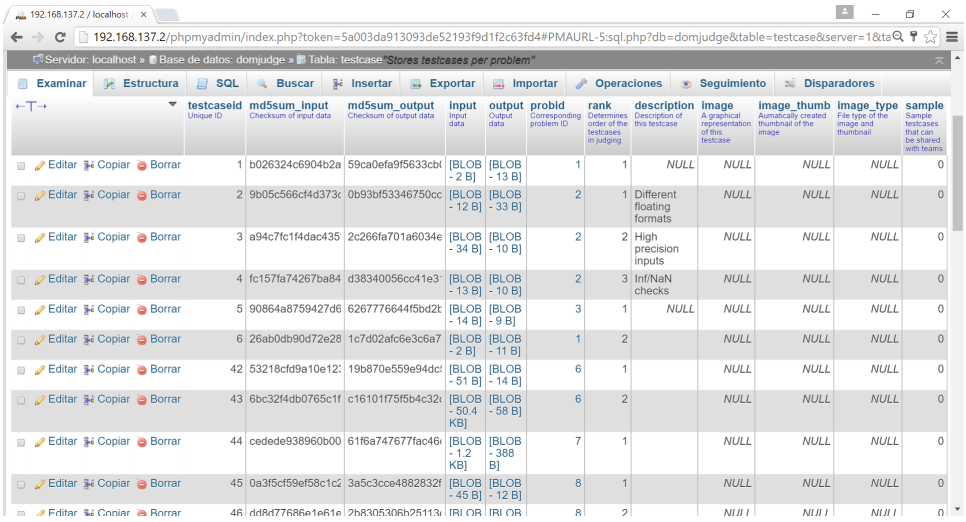

Figura 5.11: Bases de datos de DomJudge

Cada caso de prueba tiene un ID (testcaseid) que los diferencia, luego existe otro ID (probid) del problema al que pertenece y los test están ordenados por la importancia del test (rank). Las entradas y salidas de los test están guardados en forma de BLOB (Binary Large Objects).

5. Ubicación del ejecutable jPET. Como se ha explicado en el capítulo de jPET, el generador es un ejecutable que puede ser lanzado desde la consola o ejecutado gráficamente. La duda surge cuando hay que decidir en qué directorio hay que ubicar el ejecutable, porque según la instalación de DomJudge donde los archivos y ficheros están jerarquizado según los sistemas UNIX. La propuesta inicial fue ubicar en el directorio bin porque se entiende que jPET es un ejecutable o archivo binario. Luego se descartado la propuesta optando por ubicarlo en el directorio lib, a causa de que en un futuro jPET y otros generadores puedan funcionar como una librería.

5.9. Conclusión

Gracias a Domjudge el concursante podrá participar en los concursos que desee, saber si su solución es la correcta de manera casi inmediata y recibir aclaraciones por parte del jurado. Todo ello sin estar de forma presencial frente al jurado y pudiendo saber en todo momento como va comparado con el resto de participantes.

Los casos de prueba generados por jPET podrían servir como punto de partida a la hora de generar un conjunto de casos de prueba de calidad. Un aspecto muy interesante en este sentido, es que no es necesario que los programas de los alumnos utilicen el lenguaje Java. Lo único que sería necesario es que el profesor proporcione una solución escrita en Java. A partir de esta se podrían generar los casos de prueba iniciales. Si se diese el caso de que las soluciones de los alumnos viniesen escritas en Java, se podrían plantear enfoques más interesantes en los cuales los tests se forman a partir de ambas soluciones, la del alumno, para generar los datos de entrada, y la del profesor, para chequear que las salidas para esas entradas son las correctas.

En el próximo capítulo...

En el siguiente capítulo se nombra algunas posibilidad de desarrollo para el futuro y una vista general del alcance que podrá tener nuestra investigación.

Capítulo 6

Estadísticas

La estadística es el único tribunal de apelación para juzgar el nuevo conocimiento.

P. C. Mahalonibis

Resumen: Para corroborar la validez del trabajo realizado se ha aplicado la herramientra creada sobre problemas de asignaturas de programación y se han extraido estadísticas de ellos.

Se han utilizado problemas de algoritmia que están siendo objeto de evaluación en las asignaturas de Fundamentos de Programacón y Estructura de Datos y Algoritmos. Se ha escogido una muestra de ocho problemas significativos, y para cada uno de ellos se explicará brevemente lo que pide, la razón por la cual se considera interesante, los resultados obtenidos agrupados en una tabla, y por último breves comentarios acerca de los resultados. A la hora de mencionar el nombre de cada problema, las siglas FP corresponderán a Fundamentos de la Programación y las siglas EDA a Estructura de Datos y Algoritmos.

6.1. FP Carreras

6.1.1. Explicación

Se quiere jugar con el mayor número de coches posible, para ello, cada coche necesita de dos pilas, y además la suma de los voltajes de las pilas tiene que superar cierto voltaje.

Dada una sucesión de pilas con su voltaje y un valor V indicando el voltaje que se debe superar para que un coche funcione: Hallar el mayor número de coches que se pueden usar.

6.1.2. Razón de interés

Para hallar la solución del problema de una manera eficiente, es necesario el uso de un algoritmo voraz, previamente habiendo aplicado un método de ordenación a las pilas dadas por el enunciado ya que no se garantizaba en ningún momento su orden.

El método voraz consiste en dado un array con todas las pilas ordenadas, mantener un índice en cada extremo, de tal forma que si la suma de los indices llegan a ser superiores al voltaje requerido, se descartan ambas pilas y se suma un coche a la solución desplazando los índices una posición al interior. En caso de que la suma de los índices no sea suficiente, se desplazará el índice izquierdo una posición a la derecha hasta la siguiente pila que tendrá un voltaje mayor o igual. El algoritmo finaliza cuando los índices apunten a la misma posición o se hayan sobrepasado. Para maximizar la eficiencia se aplica el método de ordenación QuickSort dejando el coste de la solución en O(n * log n), siendo n el número de pilas.

6.1.3. Estadística

Carreras de Coches		Folder	
		Correct Answers	Wrong Answers
Detection with manual testcases	Correct	5	0
	Error	0	10
Detection with automatic testcases	Correct	5	1
	Error	0	9
Result of using SoftTest		100%	90%
Number of submissions		5	10

6.1.4. Comentarios

En este caso los tests han dado unos resultados muy positivos, aunque se ha alterado el dominio con el que trabaja jPET, ya que por defecto, tiende a usar el valor cero como valor entero predeterminado, forzando el dominio a valores mayores que cero se obtienen unos resultados mucho mas significativos.

6.2. EDA Cima

6.2.1. Explicación

Dada una sucesión de elementos de valores enteros, se quiere verificar que existe un punto tal que todos sus valores anteriores sean menores y a su vez decrecientes, y que todos los valores siguientes sean de la misma forma menores y decrecientes. En caso de que se cumpla esta propiedad, se pide

mostrar por pantalla además la posición que toma el elemento cima, si no se cumple, bastará tan solo con un "No".

6.2.2. Razón de interés

Para resolver este problema, se puede ver fácilmente que al menos es necesaria una iteración por todos los elementos del array, y para mantener un coste eficiente, se realizará todo en un tiempo lineal en función del número de elementos.

Cobra su importancia debido a que intuyendo el CFG del programa, podemos ver que puede haber multitud de casos en donde se encuentre el elemento cima, además de que el elemento que puede acabar con la propiedad puede encontrarse en cualquier parte del array. No se trata de un algoritmo especialmente complejo, pero si interesante de analizar.

6.2.3. Estadística

Cima		Folder	
		Correct Answers	Wrong Answers
Detection with manual testcases	Correct	4	0
	Error	0	9
Detection with automatic testcases	Correct	4	0
	Error	0	9
Result of using SoftTest		100%	100%
Number of submissions		4	9

6.2.4. Comentarios

En este caso aunque los resultados hayan sido correctos en su totalidad, es importante mencionar que la complejidad del algoritmo no era necesariamente elevada, y que por la naturaleza del problema, los tests resultantes iban a ser realmente significativos.

6.3. EDA SumaBuen

6.3.1. Explicación

Como entrada se recibe una sucesión de números enteros, no necesariamente ordenados, y se pide como resultado la suma de elementos de dentro de esa sucesión que cumplan la siguiente propiedad: Dado i que indica la posición del índice del array y siendo v dicho array, se tiene que un elemento v[i] es valido si cumple que

$$v[i] = 2^i$$

6.3.2. Razón de interés

El algoritmo se resuelve simplemente recorriendo todo el array llevando dos variables con la función de contador, una con el resultado de las sumas hasta el momento, y otra que vaya almacenando las potencias de 2. De esta forma se permite calcular el resultado de forma eficiente en un tiempo lineal.

El interés en este problema es que se obliga a jPET a que de ciertos valores a las variables para que permita que ocurran las situaciones esperadas, es un problema muy recomendable para el uso de la herramienta ya que necesita unos valores muy concretos para tener situaciones especiales influyentes en la salida que se vaya a generar.

6.3.3. Estadística

Suma Buenos		Folder	
		Correct Answers	Wrong Answers
Detection with manual testcases	Correct	5	0
	Error	0	8
Detection with automatic testcases	Correct	5	0
	Error	0	8
Result of using SoftTest		100%	100%
Number of submissions		5	8

6.3.4. Comentarios

Tal y como se esperaba los resultados obtenidos han sido realmente satisfactorios debido a la naturaleza del problema. Se recomienda su uso sobre esta clase de problemas en caso de que se identifiquen con facilidad, ya que los casos de prueba generados son realmente significativos y se puede asegurar un éxito cercano al cien por ciento en la mayoría de los casos

6.4. EDA Paréntesis balanceados

6.4.1. Explicación

La entrada consiste en una cadena de texto (que no incluye saltos de línea) en la cual se tiene que comprobar que para los paréntesis, corchetes y llaves, que por cada símbolo de apertura existe otro signo del mismo tipo pero de cierre y además se encuentran en el orden correcto situados, no dejando ningún paréntesis abierto sin cerrar, ni permitiendo paréntesis cerrados sin que hayan sido abiertos.

6.4.2. Razón de interés

La eficiencia de este problema reside en hacer uso de una pila para almacenar el último paréntesis que ha sido abierto, de esta forma, almacenando cada uno de ellos, al encontrarnos con uno de cierre, podemos comprobar si son del mismo tipo, y en caso de serlo, desapilar el último paréntesis de apertura añadido. Al llegar al final de la cadena, la pila debe quedar vacía para asegurarse que la los paréntesis están balanceados.

Este problema se ha a/ nadido porque es interesante el uso de estructuras de datos en la resolución de problemas y se quiere verificar que también puede asegurar un correcto funcionamiento para esta clase de problemas.

6.4.3. Estadística

Parentesis balanceado		Folder	
		Correct Answers	Wrong Answers
Detection with manual testcases	Correct	4	0
	Error	1	10
Detection with automatic testcases	Correct	4	3
	Error	1	7
Result of using SoftTest		100%	70%
Number of submissions		5	10

6.4.4. Comentarios

En este caso de las soluciones dadas, como correctas, se ha detectado una como incorrecta debido a que la configuración de los diferentes jueces Online (el usado por la facultad de Informática y el nuestro propio) difieren en ciertos aspectos, generando errores de compilación por lo que no se ha llegado a ejecutar. A pesar de esto, se ha mantenido la estadística tal y como se obtuvo, preservando la pureza de los resultados obtenidos.

En el caso del uso de estructuras auxiliares, hay ciertos parámetros de jPET que plantean limitaciones respecto al número de usos de las instrucciones, impidiendo generar casos realmente significativos en un periodo razonable de tiempo. Explotando la funcionalidad de jPET, se podrían modificar estos parámetros y asumir un coste de tiempo exponencial en la generación de casos con el fin de obtener resultados mucho mas relevantes. Debido a que la estadística esta enfocada a un uso ordinario, se ha descartado esta opción y se ha limitado su uso restringiendo el tiempo de generación de casos de prueba a un margen razonable.

6.5. EDA Vector divisible en pares impares

6.5.1. Explicación

Tal y como indica el nombre del problema, se pide hallar si un vector de elementos, esta formado por una sucesión de cero o mas elementos pares, y que a partir de ese elemento, todos los siguientes sean impares.

6.5.2. Razón de interés

Para la solución solo es necesario un recorrido del array una única vez, se comienza mirando elemento a elemento, si se trata de un número par, y en el caso de encontrar un número impar, se asegura que todos los siguientes hasta el final del vector, son impares.

Los valores del vector no son realmente significativos en el sentido de que cualquier valor par tiene el mismo significado que el resto, de la misma forma que cualquier valor impar se comporta igual que los otros impares. Debido a esta situación es un problema potencialmente recomendable para el uso de jPET. Generando resultados realmente significativos.

6.5.3. Estadística

ParImpar		Folder	
		Correct Answers	Wrong Answers
Detection with manual testcases	Correct	6	0
	Error	0	9
Detection with automatic testcases	Correct	6	0
	Error	0	9
Result of using SoftTest		100%	100%
Number of submissions		6	9

6.5.4. Comentarios

Como se esperaba, los resultados han sido satisfactorios.

6.6. EDA Sucesión de naturales

6.6.1. Explicación

Tomándo como base la sucesión de Fibonacci, se dan 3 valores enteros, tales que dos de ellos representan el valor que toma la función para cuando esta vale cero y uno, y el otro el número de valor del que se quiere calcular el resultado de la función. Es decir:

$$fob_0(x, y) = x$$

$$fob_1(x, y) = y$$

$$fob_n(x, y) = fob_{n-1}(x, y) + fob_{n-2}(x, y)$$

6.6.2. Razón de interés

Este problema es idealmente resuelto haciendo uso de técnicas de programación dinámica, tales que permitan almacenar el resultado que da la función para diferentes valores, y no tener que volver a calcular repetidas veces el mismo valor. Pero no se hizo con esa mentalidad, sino la de explotar la potencia de la recursividad, por lo que se implemento un algoritmo recursivo que calcula ineficientemente los valores, pero nos permite ver como trabaja jPET con algoritmos recursivos.

6.6.3. Estadística

Fibonacci		Folder	
		Correct Answers	Wrong Answers
Detection with manual testcases	Correct	4	0
	Error	0	9
Detection with automatic testcases	Correct	4	6
	Error	0	3
Result of using SoftTest		100%	33%
Number of submissions		4	9

6.6.4. Comentarios

En este caso los resultados tan malos obtenidos son debido a la asignación por defecto de los valores que da jPET a las variables, creando casos de pruebas con los valores para la función de la sucesión de valores de cero y uno iguales. Limitando así de sobremanera la eficacia de los casos de prueba generados.

6.7. EDA Complementario

6.7.1. Explicación

Dado un número entero, hallar su complementario. El complementario es un número tal que la suma de cada carácter uno a uno con el valor obtenido de entrada tome como resultado el nueve. Es decir, si como entrada recibimos el valor 100, se deberá devolver el valor 899.

6.7.2. Razón de interés

En el ámbito de la asignatura, fuerzan a que la solución implementada por el alumno interprete la entrada como un array de números. De tal forma que la solución tenga un coste lineal en el número de dígitos del valor de entrada. Para obtener la solución simplemente basta con recorrer elemento a elemento el array, hallando el valor necesario para que la suma llegue a valer 9, y de la misma forma que se ha descompuesto un valor en un array, obtener el resultado de un array a un elemento para mostrarlo por la salida.

6.7.3. Estadística

Complementario		Folder	
		Correct Answers	Wrong Answers
Detection with manual testcases	Correct	5	0
	Error	0	10
Detection with automatic testcases	Correct	5	0
	Error	0	10
Result of using SoftTest		100%	100%
Number of submissions		5	10

6.7.4. Comentarios

A pesar del problema de la elección de los valores para las variables, la complejidad del problema no resultaba ser excesiva y aquellas soluciones incorrectas eran suficientemente incorrectas como para no necesitar casos de prueba de gran valor.

6.8. EDA El hotel Pico

6.8.1. Explicación

Un hotel quiere calcular el cobro que se hace a los turistas en un intervalo par de días, para ello cuenta con la ocupación de habitaciones que ha tenido cada día el hotel, pero su sistema de pago es algo peculiar. Para el primer y el último día el coste de la estancia por turista es de un euro, para el segundo y el penúltimo día es de dos euros, de tal forma que para el tercer día y el antepenúltimo día el coste asciende a cuatro euros por cada turista. Así de manera sucesiva, multiplicando el precio por dos mientras nos acercamos a los días centrales. Se pide calcular el importe total.

6.8.2. Razón de interés

A pesar de ser otro algoritmo basado en el recorrido de un array, se ha realizado otra implementación recursiva que sustente la eficacia de jPET

frente a estos problemas debido al poco afortunado ejemplo del algoritmo recursivo en la sucesión de números naturales (Apartado 5.6).

6.8.3. Estadística

El hotel Pico		Folder	
		Correct Answers	Wrong Answers
Detection with manual testcases	Correct	5	0
	Error	0	9
Detection with automatic testcases	Correct	5	1
	Error	0	8
Result of using SoftTest		100%	89%
Number of submissions		5	9

6.8.4. Comentarios

En este caso a pesar de que vuelve a ser un problema la elección de las variables, los valores son realmente significativos como para permitir la creación de un conjunto fiables de testcases.

En el próximo capítulo...

En el próximo capítulo se hablará de las conclusiones obtenidas de todo el trabajo y se plantearan posibles ampliaciones.

Capítulo 7

Conclusiones y Líneas futuras

Si el tiempo es lo más valioso, la pérdida de tiempo es el mayor de los derroches.

Benjamin Franklin.

Resumen: Se hablarán de los resultados obtenidos del desarrollo de la integración elaborando una conclusión y ee listarán aquellas posibles ampliaciones del proyecto que se crean necesarias.

7.1. Conclusiones

El mayor punto positivo acerca de esta integración sobre DomJudge, es que efectivamente funciona, y se pueden llegar a conseguir realmente buenos resultados. Para ello nos hacemos eco de las estadísticas realizadas, que en ningún momento corrompen aquellas soluciones que son correctas. Por lo tanto la adición de los tests generados automáticamente nunca podría entorpecer la tarea de los otros tests. La media de detección de problemas incorrectos es de un 85,25 %. Siendo enormemente afectada por un problema que no era el idóneo para esta práctica. A pesar de esto, se sigue manteniendo un valor muy cercano a la totalidad. Es necesario ver que las estadísticas se han aplicado sobre problemas que algoritmicamente no presentan una gran complejidad. No es recomendable su uso sobre algoritmos complejos y sobre todo sobre aquellos que cuenten con muchas dependencias. Su uso recomendado está estrictamente restringido a problemas que pueden ser resueltos con un solo algoritmo.

A pesar de que su uso sea recomendable en estos aspectos, cuenta también con algunas desventajas.

1. No todos los códigos son iguales. Y por lo tanto, existen mas maneras para resolver un problema que la que plantearía el encargado del

desarrollo de los tests. Esto supone un problema debido a que todas las entradas están basadas en un único código fuente. El cual, si esta realizado de forma idónea no necesita tratar como casos especiales algunas entradas especificas que usuarios mas inexpertos en el mundo de la programación se ven obligados a hacer. Ignorando así una posible fuente de errores.

2. Las precondiciones en el entorno de DomJudge suelen tener una naturaleza realmente fuerte. Estas desinhiben al usuario del tratamiento de dominios no aceptados para el algoritmo, permitiendo centrar toda su atención en la resolución del algoritmo. Esta situación nos resulta favorable y desfavorable al mismo tiempo. Ya que el alumno no se deberá preocupar de situaciones extrañas, aunque cuando desarrolle código mas adelante si deberá hacerlo.
   Esta situación se ha solucionado incluyendo una detección de entradas que no cumplan las precondiciones. Generando internamente una excepción y filtrándose del fichero final de entradas así como en el de salidas. Una explicación mas detallada es incluida en la sección del desarrollo del complemento.

3. Requiere de un código correcto programado en Java, y que además cumpla una serie de requisitos.

4. Las entradas y las salidas requieren de una cierta modificación que les permita adaptarse al estilo necesario para el problema en cuestión o Domjudge incluso. Las entradas generadas, consisten en una única linea por cada caso. Desgraciadamente Domjudge muchas veces impone requisitos como que el primer valor determine el número de elementos de un vector, y a continuación en la siguiente línea se encuentren los valores de los elementos vector. Puede incluso llegar a exigir un carácter centinela que marque el fin del caso de prueba.
   Lamentablemente estas transformaciones deben ser realizadas a mano. Aunque una gran cantidad de ellas son incluidas en el GitHub asociado a este Trabajo de Fin de Grado a modo de ejemplo.

5. Tiene una limitación ligada a la complejidad del problema, ya que tiende a crecer de manera exponencial con la complejidad del código. Además encontrar tests cases significativos para esta clases de problemas puede resultar una tarea difícil haciendo uso de jPET en un tiempo razonable.

6. En ningún momento es capaz de tener en cuenta si una solución dada por el algoritmo es correcta en términos de eficiencia. Únicamente genera entradas y salidas generalmente de un tamaño significativamente pequeño que no permiten evaluar este criterio con exactitud.

A pesar de ser desventajas en ciertos aspectos costosas, muchas de ellas son el mal menor de otras posibles alternativas. Ya que nunca se esta forzando al usuario a tener que dar una especificación formal del problema, que resultaría en una tarea de una mayor complejidad.

Es importante observar que la primera desventaja puede ser fácilmente subsanada si se usa como apoyo la generación aleatoria de casos de prueba. También se pueden aplicar técnicas de caja negra para encontrar aquellos puntos conflictivos que puedan resultar en errores potenciales.

Como conclusión final, no se recomienda hacer unicamente uso de los casos de prueba generados de forma automática. Ya que a pesar de que cuenta con una alta probabilidad de exito, el dar una solución incorrecta como válida puede suponer realmente un gran problema. Se insta por lo tanto a hacer un acompañamiento de estos tests generados con otros, aplicando técnicas de caja negra que apunten a entradas conflictivas. Siempre compaginados con una cantidad significativa de casos de prueba generados aleatoriamente que corroboren la robustez de las soluciones dadas.

7.2. Conclusions

The most positive point about this integration on DomJudge, is that it actually works, and you actually get good results. To affirm this we rely on the statistics made, that never mark as incorrect a solution that is correct. Therefore the addition of the automatically generated tests could never hinder the work of other tests. The average detection of incorrect problems is 85.25 %. However it is greatly affected by a problem that was not the ideal for this project. Despite this, it still maintains a very close value to the whole. It is necessary to remind that the statistics have been applied to problems that doesn't have a great algorithmically complexity. It is not recommended for use on complex algorithms and especially on those who have many dependencies. Its recommended use is strictly restricted to problems that can be solved with a single algorithm.

Although its use is recommended in these aspects, it also has some disadvantages.

1. Not all codes are the same. And therefore, there are more ways to solve a problem that the one provided for the development of the tests. This is a problem because all entries are based on a single source code. Which, if it 's done in the most ideal way this ideal doesn't need to treat as special cases some specific inputs that most inexperienced users in the world of programming are forced to do. Ignoring that way a possible source of errors.

2. Preconditions in the environment of DomJudge usually have a really strong nature. These preconditions release the user of the treatment of

not accepted algorithm domains, focusing all their attention on solving the algorithm. This situation is favorable to us and unfavorable at the same time. Since the student should not worry about strange situations, but when developing code later he must do so.
This situation has been fixed including detection entries that do not meet the preconditions. Internally generating an exception and filtering the final input and output file. A more detailed explanation is included in the section of development of the complement.

3. It requires a correct code programmed in Java, and also meets a number of requirements.

4. The inputs and outputs require some modification to allow them to adapt to the style needed for the problem or even Domjudge.The generated inputs consist of a single line for each case. Unfortunately Domjudge often imposes requirements like the first value determines the number of elements of a vector, and then on the next line we find the values of the elements. It may even require a sentinel value that marks the end of the test case.
Sadly these changes must be made manually. Although a lot of them are included in the GitHub associated to this Final Degree Project as examples.

5. It has a limitation linked to the complexity of the problem, as it tends to grow exponentially with the complexity of the code. In addition finding significant tests cases for this kinds of problems can be a difficult task using Jpet within a reasonable time.

6. At no time it is able to take into account whether a given solution by the algorithm is correct in terms of efficiency. It only generates inputs and outputs of a generally small size that can not evaluate this criterion exactly.

Despite being costly disadvantages in certain aspects, many of them are the lesser evil of other possible alternatives. Since it is never forcing the user to give a formal specification of the problem, resulting in a more complex task.

It is important to note that the first disadvantage can be easily remedied if the random generation of test cases is used as a support. We can also apply black box techniques to find those spots that may result in errors.

As a final conclusion, it is not recommended to use only the automatically generated test cases. As though as it has a high probability of success, giving an incorrect solution as a valid one can be really a big problem. It therefore urged to make an accompaniment of these tests generated with others, using black box techniques aimed at conflicting entries. Always combined

with a significant amount of randomly generated test cases corroborating the robustness of the given solutions.

7.3. Líneas futuras

- Creación de estándares para las entradas de los problemas de DomJudge, que permitan ajustarse a ciertos patrones y puedan crearse herramientas que realicen una traducción inmediata al estándar sin obligar al usuario hacerlas a mano.
- Mantenimiento de la herramienta de PET, que permita el uso de nuevos tipos de variables así como una disminución de las restricciones que aporta.
- Ampliación de las capacidades de Pet y del complemento Solver para que admita más lenguajes que java.
- Enviar la modificación a Domjudge para que si la aceptan forme parte de la versión oficial.

Capítulo 8

Contribuciones personales

Unus pro omnibus, omnes pro uno

Resumen: Cada integrante del grupo explicará su trabajo realizado en el proyecto.

8.1. Trabajo realizado por Jaime Boixadós Martínez

Durante el desarrollo del TFG he llevado a cabo distintas tareas. Lo primero que tuve que hacer fue documentarme sobre jPET. Leí los manuales proporcionados por nuestro tutor y tras entender los conceptos de testing sobre los que se basaba empecé a hacer pruebas con jPET.
Las pruebas con jPET fueron mucho más largas de lo que me imagine en primera instancia. Muchas veces devolvía errores que no entendía y el hecho de disponer únicamente de la documentación del profesor (ya que la documentación online era escasa) dificultaba la tarea de identificación de estos errores. Finalmente logré entender a que se debían estos errores. En la mayoría de los casos se debían a tipos de parámetros no aceptados, tanto en los parámetros de entrada como en las variables de los métodos.
Una vez que considere que ya entendía al completo el manejo de jPET a través del plugin de eclipse me propuse aprender a utilizar el propio pet a través de línea de comandos. No fue tan complicado como esperaba y conseguí manejarlo rápidamente.
Para evitar a mis compañeros pasar por el calvario que yo pase para aprender a utilizar perfectamente pet decidí hacer una interfaz gráfica que facilitara la ejecución de la herramienta. Finalmente esta interfaz gráfica pasaría a ser

el complemento de traducción SoftTest para el sistema Domjudge y se convertiría en mi tarea central durante el resto del proyecto.
Como he dicho antes mi primera intención fue desarrollar una aplicación que facilitara el uso de pet a mi compañeros. Contaba con una interfaz gráfica muy simple y lanzaba un comando de pet por defecto para simplificarles la vida. Este almacenaba el fichero XML en el mismo directorio desde el que se lanzaba la aplicación y mostraba en un panel de la interfaz el XML obtenido.
Al poco tiempo determinamos que era necesario traducir el XML de manera automática si queríamos que fuera más legible. Así que el propósito de la aplicación cambió y empecé a desarrollar la parte de traducción del XML. Lo primero fue entender exactamente cómo se regía el XML. Si iba a traducirlo de manera automática debía entender porque saltaban en ciertas ocasiones distintas etiquetas y que estructura seguía. Tras estudiar el funcionamiento del XML generando en los distintos casos pude empezar a desarrollar el módulo de traducción.
En su primera implementación el módulo SoftTest solo era capaz de llevar a cabo traducciones de XMLs que fueran generados a partir de un método con un parámetro de entrada int y de salida out. Al principio solo mostraba los valores por pantalla aunque al poco tiempo ya generaba los fichero in y out. Las siguientes fases de desarrollo se centraron en mejorar el módulo de traducción. Al final de estas etapas el módulo era capaz de generar ficheros in y out de métodos con varios elementos de entrada e incluso de arrays de int.
Con el proyecto ya avanzado decidimos que tanto el parámetro de entrada como el de salida fueran por defecto un STRING. Sin embargo no fue posible debido a limitaciones propias de pet. Aun así la herramienta vivió cambios para adaptarse a esta nueva circunstancia. La mayoría tuvieron que ser desechados pero algunos como un recorrido más eficaz del xml fueron guardados.
Llegados a este punto decidimos que la mejor opción era establecer que los datos de entrada y de salida fueran dos arrays de int respectivamente. Durante las siguientes semanas me encargué de mejorar el código del complemento de traducción. Se corrigieron leves fallos y errores en la traducción.
El siguiente gran cambio que sufrió la herramienta fue el filtrado de resultados. Se decidió que todas aquellas entradas que generasen una excepción serían eliminadas de los ficheros in y out al igual que sus salidas. Esto nos permitió también que si deseamos filtrar un caso nos basta con lanzar una excepción desde el programa. SoftTest la detectara y eliminara la entrada de los ficheros resultantes.
Ahora que la herramienta funcionaba perfectamente y que disponíamos de una instalación de Domjudge en uno de nuestros equipos nos pusimos manos a la obra con la integración. Obviamente esto supuso grandes cambios en la aplicación ya que antes no estaba pensada para funcionar con Domjudge.
El primer gran cambio que recibió fue la posibilidad de ser lanzada desde

un terminal. Esta modificación no pisó todo lo desarrollado previamente y el módulo SoftTest podía seguir haciendo uso de la interfaz Grafica.
Los siguientes cambios fueron con el objetivo de adaptar su funcionamiento al sistema Domjudge. Le permiti al módulo cierto nivel de personaliación del comando por defecto utilizado para lanzar jPET. Ahora SoftTest puede elegir que jPET desea utilizar pasándole la ruta donde se encuentre el que desee. Además en esta ruta se guardaran los ficheros in y out. El fichero xml se guarda ahora en el directorio /tmp.
Mientras realizaba estas labores de adaptación trabaje en paralelo con un compañero para elegir los problemas con los que llevaríamos a cabo nuestro estudio estadístico. Tras realizar numerosas pruebas jugando con los distintos parámetros de jPET seleccionamos numerosos problemas prometedores. Los resultados del estudio final de esos problemas se encuentra en el capítulo de estadísticas.
Finalmente redacté junto a mis dos compañeros esta memoria.

8.2. Trabajo realizado por Benito Álvaro Cifuentes de la Torre

8.2.1. Testing

Realicé una tarea de investigación en la materia de Testing, La principal razón consistia en tener la capacidad de entender como podía funcionar la herramienta utilizada (jPET), y en caso de que diera complicaciones, ser siempre capaz de recurrir al concepto teórico que se aplica para la obtención de las entradas y salidas como último recurso. Para ello me matriculé en la asignatura optativa ofertada por la Universidad Complutense de Madrid Especificación Validación y Testing, donde se hizo especial énfasis a estos temas. Como complemento a esta tarea de investigación accedí a diferentes recursos bibliográficos que pudieran aportar otro enfoque o nueva información acerca de estos aspectos. Esta bibliografía queda mencionada en las notas bibliográficas del capítulo dedicado al Testing.

8.2.2. SoftTest

Participé activamente en el desarrollo de la herramienta una vez que fue creada una versión inicial muy primitiva que aun presentaba bugs y errores menores. Participé en la detección y corrección de uno de los principales problemas que presentó la aplicación ya que no realizaba correctamente una lectura del fichero XML generando ficheros de entrada y salida no válidos, inutilizando mediante un error lógico todo el mecanismo de generación y pudiendo llegar a corromper los resultados obtenidos.

Uno de los principales problemas que surgió durante el desarrollo de es-

ta aplicación fue la existencia de precondiciones fuertes en la mayoría de los enunciados de los problemas tipo de DomJudge, esta restricción chocaba directamente con el planteamiento inicial de uso jPET ya que ahora necesitariamos contar con algún mecanismo de filtro para estas entradas. Propuse tres posibles soluciones teniendo en cuenta que no se conoce ningún mecanismo capaz de especificar las precondiciones sin contar con una especificación formal del problema.

- Ampliación de jPET para permitirlas como parámetro de entrada en algun lenguaje.
- Obligar al programador a realizarlas el mismo de forma manual. Se pueden especificar mediante asserts o la elaboración de condiciones complejas sobre los datos de entrada.
- No tenerlas nunca en consideración.

Finalmente se optó por el uso de la segunda opción de forma que el programador deberá implementar una función precondición que verifique si la entrada cumple las especificaciones y lanzar una excepción en caso de no hacerlas.

Durante todo el proceso de desarrollo, proporcioné constantemente ejemplos de código que sustentaran los cambios realizados, haciendo un avance progresivo de la complejidad de estos, permitiendo seguir aumentando el alcance así como verificar si realmente podía ser usado para tratar problemas reales. Durante esta fase previa, programé siguiendo los requisitos impuestos (una nomenclatura concreta, evitar el uso de Strings, no incluir instrucciones potencialmente peligrosas para jPET,etc...) la gran mayoría de los ejercicios planteados en la asignatura de Fundamentos de la Programación, consiguiendo acceso al juez que utilizaban gracias al tutor del proyecto.

Una vez se tuvo una aplicación en un estado mas avanzado, se decidió crear unas estadísticas que corroboraran la eficacia de la aplicación que se había creado, pero para ello, se necesitaba una gran cantidad de problemas implementados, todos ellos siguiendo unas estrictas pautas de programación que aumentaban la complejidad del problema. Ya que para la realización de las estadísticas se necesitaban problemas de un mayor interés se descartaron todos los problemas que se resolvieron con anterioridad, y se comenzaron a hacer problemas de la asignatura de Estructura de Datos y Algoritmos. Realicé una totalidad de dieciocho problemas siguiendo las pautas, de los cuales diez de ellos se descartaron bien porque jPET no estaba preparado para trabajar con tipos que no fueran enteros, o bien porque la naturaleza del problema se alejaba de la algoritmia pura, apuntando mas a estilos de programación y funcionalidades.

Teniendo ya los problemas interesantes fijados y habiendo hecho uso de jPET para la generación de casos de prueba, seguía siendo necesaria una

adaptación de los formatos de los ficheros de entrada y salida. Como se comento con el tutor en las reuniones, estos ficheros podían presentar una mala correspondencia entre las entradas y las sálidas por lo que era necesario ejecutar las entradas y guardar las salidas generadas manualmente. Esta tarea fue mi responsabilidad, subí a GitHub diferentes programas que eran capaces de interpretar las entradas generadas por jPET y transformarlas en una gran variedad de formatos en función de la especificación que diera el problema subido a DomJudge. La obtención de las sálidas se obtuvo ejecutando el programa de solución y forzando a que su entrada estándar apuntara al fichero con el formato adecuado de DomJudge, guardando además las salidas generadas por el algoritmo en otro fichero aparte siguiendo de la misma forma un formato específico ligado al problema que se estaba resolviendo.

Tras esta realización de tareas, el compañero de Trabajo de Fin de Grado Yaofang, me facilitó las estadísticas generadas de estos ocho problemas. Realicé un análisis de estas y obtuve las conclusiones.

8.2.3. Memoria

Tomándo iniciativa propia me documenté acerca de las posibles opciones que existían para la creación de tesis o trabajos de fin de grado.

Tras una investigación conseguí hallar el proyecto Texis, realizado por el grupo de investigación GAIA asociados a la Universidad Complutense de Madrid. Texis consiste en una plantilla preparada para la elaboración de esta documentación en Latex, dando gran facilidad a su uso, así como incorporando una guía de utilización para usuarios inexpertos.

Facilité la transción de mis compañeros al uso de Latex mediante la implementación previa de un proyecto base, así como la resolución de dudas acerca de los nuevos aspectos que surgen al trabajar con Latex.

Sobre la memoria entragada realicé las siguientes tareas.

- Realización de los capítulos destinados al Testing, a la introducción,a las conclusiones y la elaboración de las estadísticas con los datos ya procesados por Yaofang.
- Revisión total del documento en reitaradas ocasiones
- Corrección del borrador en los apartados de Testing, Integración de jPET y DomJudge

8.3. Trabajo realizado por Yaofang Zhang

En un principio, la división de trabajo era bastante clara, aparte de los detalles que se han ido repartiendo a lo largo del año, los puntos principales a tratar se repartieron de la siguiente forma:

- Álvaro y Jaime se dedicarían a la parte de desarrollar SoftTest
- Yo me encargaría de instalar DomJudge.

8.3.1. DomJudge

La tarea investigación de DomJudge es la que ha necesitado una mayor inversión de tiempo. Empezar a entender el código de una gran comunidad como la de DomJudge costaba bastante tiempo. Por suerte, DomJudge cuenta con una documentación bastante buena y detallada sobre todos los aspectos del proyecto.

La documentación está en inglés, por lo que es necesario hacer un esfuerzo en entenderlo bien. La instalación comienza desde la instalación del sistema Linux, ya que es el sistema requerido para la instalación. Luego se fueron instalando e investigando a la vez los aspectos sobre el servidor y la base de datos.

Al finalizar la instalación, se dedico una gran parte de tiempo a la investigación de uso del sistema DomJudge. Desde crear un usuario administrador, crear un nuevo concurso, añadir nuevos problemas, hasta cambiar de rol para realizar entregas de prueba.

Por último, era necesario averiguar la parte de código fuente que era susceptible a modificaciones para permitir la integración de jPET. La tarea de encontrar como poder realizar esta ampliación resultó de gran complicación ya que no cuenta con la suficiente documentación acerca del código fuente. Tras un largo tiempo de búsqueda y entendimiento del código, pude añadir el fragmento de código que necesitaba para el correcto funcionamiento de jPET en DomJudge. Durante este último apartado no se obtuvo ningún tipo de ayuda de guias, ya que no se cuentan con la existencia de tutoriales y como se menciono anteriormente, a pesar de contar con una buena documentación de guia de uso y de instalación, descuidan las secciones de ampliación. Tuve que realizar un trabajo de entendimiento del código y de las bases de datos asociadas a DomJudge para poder entender el correcto funcionamiento de estas, para así poder ser capaz de hacer tareas como la de añadir nuevos casos de prueba que serán creados por la herramienta automática.

8.3.2. Estadísticas

Como la instalación de DomJudge se encuentra localizada en el servidor local de mi ordenador personal, resultaba imprescindible que me encargara de la elaboración de las estadísticas.

Las tareas exactas que hice son:

- Crear los problemas que se han fijado para realizar las estadísticas con los test diseñados por los profesores.

- Crear los problemas con test generados por SoftTest, con el propósito de comparar los distintos resultados que pueden tener.
- Realizar 15 entregas para cada ejercicio (8 ejercicios en total) en las 2 versiones.
- Anotar los resultados obtenidos en un Excel y de allí sacar la conclusión.

8.3.3. Memoria

La memoria está dividida en 7 capítulos de las cuales he participado en 3 de ellos:

- DomJudge por completo, aunque todo el contenido fue escrito por mí, se tuvo la necesidad de realizar un repaso de todo el capítulo por parte de mis compañeros para encontrar expresiones malas o faltas de ortografía.
- Integración. La 2^a mitad del capítulo fue escrito por mí debido a que me había encargado de la integración de SoftTest a DomJudge. En este capítulo, he explicado las distintas fases de la integración, añadiendo comentarios detallado del código que he insertado en DomJudge y los problemas que han surgido a lo largo de la instalación.
- jPET. La mayoría del contenido ha sido añadido por mi, basándome fuertemente en las bibliografías con las que contaba. De la misma forma que paso con los anteriores capítulos, estos requirieron también de una revisión en profundidad por parte de mis compañeros.