Generador automático de prácticas de Compiladores

Oriol Belmonte Castro

Resumen– En este proyecto se propone una herramienta de generación automática de prácticas, que evite uno de los problemas existentes en el grado de informática, la copia de prácticas. El objetivo es obtener una aplicación capaz de generar un conjunto de diferentes prácticas para cada grupo de alumnos de la asignatura de Compiladores de la titulación. Las diferentes variantes de prácticas tienen los mismos contenidos docentes que se tendrían en una sola. Dicha asignatura parte de una práctica que consiste en un compilador base igual para todos. La aplicación se encargará de encontrar para cada grupo, un conjunto de ampliaciones del compilador que se les pedirá que programen. Los resultados del proyecto son favorables, teniendo una aplicación funcional que será usada en la asignatura como herramienta para ayudar al profesorado.

Plabras clave– Generador automático; Prácticas; Ampliaciones; Backtracking; Java; Git; Compiladores; Librerias Swing; Apache POI.

Abstract– This project proposes an automatic generator of practices tool, to avoid one of the existing problems in the degree of computer science, the copy of practices. The final objective is to obtain an application capable of generating a set of different practices for every group of students of the Compilers subject of this degree.The different variants of practices have the same teaching contents that they would have in a single. This subject parts of from a practice that consists in a base compiler equal for all the groups. The aplication is responsible to finding for every group, a set of extensions of the compiler which there will be asked them to programme. The results of the application have been favorable, having a functional application that will be used in the subject as a tool to help teachers.

Keywords– Automatic generator; Practices, Extensions; Backtracking; Java; Git; Compilers; Swing libraries; Apache POI.

1 Introducción

Uno de los problemas existentes en el grado de informática es la copia de prácticas. Estas se pueden detectar mediante métodos como detectores de copias o exámenes que prueben que los alumnos han hecho ellos mismos las prácticas. Sin embargo, estos métodos no siempre son efectivos y los alumnos acaban aprobando las prácticas por copiarse de otros grupos, impidiendo que no lleguen a aprender los contenidos docentes importantes de la asignatura.

En este proyecto se pretende solucionar este problema mediante un generador de prácticas, el cual, partiendo de una práctica base igual para todos los alumnos, se generen diferentes extensiones de la práctica para cada grupo. Esta combinación de extensiones tendrá un nivel de dificultad igual para todos los grupos y además, harán aprender las mismas competencias que se esperan de la asignatura. De esta manera, se evitará la copia de las prácticas, pero al estar aprendiendo los mismos objetivos, favorecerá a que los grupos se ayuden entre ellos para llegar a solucionar sus propias prácticas.

Para hacer esto, se ha trabajado con las prácticas de la asignatura Compiladores de esta titulación [1], las cuales se adaptan a las características de este proyecto. En estas prácticas se parte de un compilador base que se les da a todos los alumnos y, a lo largo del semestre, se les pide que programen diferentes ampliaciones de dicho compilador, de tal forma que al final de la asignatura, el alumno aprenda a ver cómo funciona un compilador y cómo mejorarlo.

Con tal de explicar el trabajo realizado, este documento se estructura de la siguiente forma: para empezar, se dará la motivación de este proyecto, seguidamente se mostrarán los

• E-mail de contacto: Oriol.belmonte@e-campus.uab.cat
• Mención realizada: Computación
• Trabajo tutorizado por: Francisco Javier Sánchez Pujades (Ciencias de la computación)
• Curso 2016/17

objetivos a cumplir y acto seguido, se comentará la planificación del proyecto. Se seguirá el documento con la explicación del estado del arte y seguidamente se pasará a explicar la metodología seguida para hacer este proyecto. Se argumentarán los resultados obtenidos y finalmente se darán unas conclusiones y líneas futuras del proyecto.

2 MOTIVACIÓN

La motivación principal del proyecto ha sido la de poder aplicar las técnicas y herramientas vistas durante el grado a la hora de resolver problemas reales, como es en este caso, crear una aplicación de generación automática de prácticas. Asimismo, otra motivación ha sido poder ayudar en una de las asignaturas del grado de informática, como en este caso, Compiladores, con tal de que los futuros alumnos que la cursen mejoren y profundicen en el aprendizaje del funcionamiento de un compilador.

3 OBJETIVOS

El principal objetivo de este proyecto es crear una aplicación capaz de generar de forma automática dicho conjunto de prácticas.

Para ello, se dividió el proyecto en las siguientes tareas, con el fin de poder alcanzar el objetivo principal de forma más eficiente y de forma que al profesorado de Compiladores le sea fácil de usar dicha aplicación.

- Generar una base de datos de todas las ampliaciones con la siguiente información:
  - **Nombre de la ampliación**: nombre que servirá para identificar de que trata la ampliación.
  - **Etiqueta o Tipo**: palabras clave que clasificará el tipo de ampliación para saber cual está siendo usada. Cada etiqueta forma una agrupación de ampliaciones en las que, todas hacen aprender el mismo objetivo al alumno, pero cada ampliación tendrá diferente forma de enseñarle.
  - **Ampliaciones incompatibles**: cada ampliación tendrá ciertas incompatibilidades con otras debido a que puede generar errores en el código.
  - **Dificultad**: este dato se usa a modo informativo para el usuario final, de cara a un análisis estadístico para saber si dichas ampliaciones le suponen al alumno el nivel de dificultad apuntado.
- Crear una estructura en forma de árbol que contenga en sus ramas todas las ampliaciones que se hayan guardado en la base de datos del punto anterior [2]. Cada rama deberá tener:
  - Enunciado de la ampliación.
  - Test de corrección.
  - Código solucionado.

  Para crear esta estructura, se usarán las herramientas de ramificación y fusión que nos porporciona los repositorios Git. Por tanto, la base de datos ayudará a encontrar rápidamente en el repositorio Git, donde se encuentra cada ampliación para posteriormente fusionar el código, enunciado y los test con otras ramas.
- Creación de un programa encargado de crear todas las prácticas. Dicho programa combinará de forma aleatoria las diferentes ampliaciones del repositorio Git para crear cada práctica, de forma que sean compatibles entre ellas y que generen en conjunto:
  - Un enunciado que explique todas las ampliaciones que deberá hacer el grupo.
  - El código solucionado para el profesorado.
  - Los tests que se usarán para corregir las prácticas.

  Para ello, se usará el algoritmo de Bactracking [3], el cual se adapta a las necesidades del problema. Cuando acabe de ejecutarse el algoritmo, se generará una estructura de directorios con las prácticas generadas dentro y un fichero Excel o csv que indique a que alumnos les pertenece las prácticas generadas.
- Comprobación de que todas las prácticas creadas se generen y funcionen con todas las posibles combinaciones que se puedan formar, arreglando los errores en el programa o conflictos entre ampliaciones que puedan surgir en el repositorio o en la base de datos.
- Programación de una interfaz gráfica para el usuario usando Java como lenguaje de programación principal, con la que se pueda indicar cuantas prácticas diferentes quiere y con qué tipo de ampliaciones. Esta interfaz integrará los anteriores puntos con tal de que sea más fácil e intuitivo de usar para el usuario final.
- Integración del generador a la aplicación web de corrección de prácticas de Compiladores.

4 PLANIFICACIÓN

A lo largo del proyecto, se ha completado todos los objetivos mencionados siguiendo de forma aproximada la distribución de horas especificada en la Tabla 1.

TABLA 1: PLANIFICACIÓN DEL PROYECTO EN HORAS

Bloques	Horas
Redacción de informes	40
Búsqueda de información	15
Generación de base de datos	5
Creación repositorio Git	10
Programación Backtracking	70
Pruebas generales	25
Interfaz de usuario	60
Integración	35
Redacción del artículo final	20
Preparación de la presentación	20
Total	**300**

La mayoría del tiempo se ha dedicado al programa de Backtracking que junto a la generación de la base de datos y el repositorio Git, se forma el núcleo de la aplicación. El resto del tiempo se ha dedicado a facilitar el uso de la aplicación al profesorado, creando la interfaz que integrará el

núcleo y a la integración en la aplicación web para corrección de prácticas. Además, durante el proyecto, se ha dedicado tiempo a la redacción de los informes de progreso, del articulo y a la preparación de la defensa del proyecto.

Para cumplir con cada objetivo de la distribución de las horas establecida, se ha seguido la metodología SCRUM ya que tiene como objetivo la mejora continua e incremental del proyecto en cada sprint. El proyecto se ha planificado a partir de una serie de sprints de una duración de entre 2 y 3 semanas, coincidiendo así con las entregas de los informes. En el apéndice A.3 se puede ver un diagrama de Gantt en dónde se aprecian las tareas propuestas en la distribución de horas para la realización de este proyecto.

5 Estado del arte

Des de la creación de los primeros lenguajes de programación tales como CLU en 1975 [4], se han ido inventando una gran variedad de aplicaciones para diversos usos científicos, educativos o de ocio. De entre estas aplicaciones están los generadores automáticos, los cuales existe una gran variedad de aplicaciones disponibles que pueden pasar des de algo más técnico como generadores de contraseñas para un usuario o generadores de tráfico en la web [5], hasta aplicaciones para el ocio como generadores de partidas de juegos de cartas, crucigramas.

Como referencias a generadores en el ámbito académico, hay plataformas web donde un profesor puede generar diversos exámenes, ya sean tipo test o escritos, y poderlos guardar en una base de datos en la nube, permitiendo también que los alumnos hagan estos exámenes y que se corrija online [6]. Para estudiantes de matemáticas o ciencias, existe la aplicación web de Wolfram Alpha que soluciona diversas ecuaciones o problemas matemáticos de forma automática [7]. Actualmente, dicha aplicación también dispone de un generador de problemas matemáticos para que el alumno pueda aprender por su cuenta.

Como plataformas de corrección, por un lado se tienen los que la corrección de prácticas o trabajos se hace manualmente, como las plataformas del Campus Virtual [8] o Cerbero [9] de la Escuela de Ingeniería para la entrega de prácticas de los alumnos. Por otro lado, se tienen los de corrección automática, como las webs de correctores ortográficos o correctores de test de teoría de autoescuela [10].

6 Metodología

Este apartado explicará cómo se han cumplido los objetivos planteados, argumentando cada decisión y su implicación en los siguientes puntos, llegando a la aplicación final.

6.1. Generación de base de datos

Para la creación de la base de datos, se decidió guardar todos los datos relacionados con las ampliaciones de la práctica en un fichero Excel o csv con el formato de la Figura 1.

Como el usuario final será el profesorado de Compiladores, la base de datos se puede gestionar de forma local sin tener ninguna clase de servidor. Dicho fichero será el encargado de guardar la información relacionada a cada ampliación que se vaya a usar en el algoritmo de Backtracking.

	A	B	C	D
1	GIT REPOSITORI	TFG repositori		
2				
3	Nom	Tipus	Incompatibilitat	Dificultat
4	A1	array	A1, A2	1
5	A2	array	A2, A1, A4	2
6	A3	operador	A3	3
7	A4	instrucció	A4, A3	4

Fig. 1: Ejemplo de base de datos de ampliaciones

En la primera fila del fichero, tendremos el nombre del repositorio Git que se usará para esta base de datos. Esto simplifica la interfaz ya que, una vez se busque esta base de datos, la interfaz se encargará de buscar el repositorio Git según el nombre de la carpeta que lo contenga. Dicho repositorio deberá estar en el mismo directorio que el de la base de datos con tal de que la búsqueda sea directa.

A continuación, el fichero contendrá en 4 columnas, la siguiente información relacionada a cada ampliación (anteriormente ya comentada):

- **Nombre de la ampliación**: nombre identificativo la ampliación.
- **Etiqueta o Tipo**: palabras clave que clasificará la ampliación para saber qué tipo está siendo usada. Junto a la interfaz, el profesor podrá usar estas etiquetas para decidir qué tipo de ampliaciones deseará que se creen.
- **Ampliaciones incompatibles**: cada ampliación tiene incompatibilidades con otras que puede generar errores en el código. También especificar que cada ampliación se anota incompatibilidad con ella misma, con tal de facilitar la generación de prácticas sin repetición, ahorrando código innecesario.
- **Dificultad**: usado a modo informativo para el usuario final, de cara a un análisis estadístico. El usuario final puede indicar un rango de dificultad y, la aplicación, se encarga de generar prácticas las cuales la suma de las dificultades de cada ampliación esté en el rango.

El algoritmo de Backtracking recoge toda la información de este fichero con tal de saber si la ampliación seleccionada tiene la etiqueta que ha pedido el usuario final y también, si es compatible con las siguientes que seleccione. Luego, mediante comandos que nos proporciona el repositorio Git, fusionará dichas ampliaciones y generará las prácticas.

Toda estas gestiones se consiguen hacer gracias a las librerías Apache POI que se proporcionan para Java para la manipulación de ficheros de Microsoft Office [11] y la librería JGit para usar comandos Git en programas Java [12].

6.2. Creación del repositorio Git

Como Subversion o CVS, los repositorios Git son sistemas controladores de versiones que permiten guardar el trabajo de los proyectos teniendo un control de los cambios producidos y de las mejoras implementadas [13]. Para este proyecto se ha decidido usar Git por ser OpenSource, por facilidad de poder clonar el repositorio Git a otros dispositivos y por sus herramientas de creación de ramas y fusión.

El repositorio que se ha creado tiene la forma de árbol donde, la raíz contiene la práctica base con el enunciado que se les da a los alumnos. Cada hoja de una rama del repositorio tendrá una ampliación de la práctica con lo siguiente:

- **Enunciado de la ampliación**: texto que se unirá a la explicación base de la práctica, que comentará que se tiene que programar en la práctica base.
- **Solucion de la ampliación**: código que se quedarán los profesores de la asignatura con tal de tener una guía a la hora de ayudar al grupo de dicha práctica.
- **Test de corrección**: código que usará el profesorado y el aplicativo web de Compiladores para corregir de manera automática las prácticas de los alumnos.

Por tanto, la estructura principal que tendrá el repositorio será como se muestra en la Figura 2.

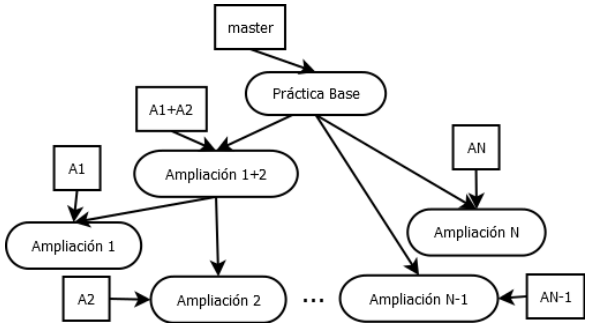

Fig. 2: Estructura del repositorio Git

Todos los óvalos contienen la información comentada: enunciado, código y test. Los cuadrados son índices que usa Git para identificar las ramas creadas. El índice master es uno predefinido en Git para especificar la rama principal, como se tiene una práctica base inicial, se ha decidido que el master sea el de la Figura 2. Los demás índices tendrán el mismo nombre que en la base de datos de las ampliaciones, de tal manera de que, cuando se busque una ampliación en concreto en la base de datos y usando la comanda Git para saltar entre ramas, *git checkout "etiqueta"*, salte directamente a la ampliación que se quiera usar.

Un problema considerado en el repositorio que usará el profesorado, es la existencia de ampliaciones que son compatibles entre ellas pero que una dependa de otra para funcionar correctamente, pudiendo dar el caso de que al fusionarse entre ellas, aparezcan errores en el código que impida ejecutarse. Para solucionar este problema, se han considerado en el repositorio como si fueran una sola. Es decir, hay ramas del repositorio que consisten en dos o más ampliaciones unidas ya que unas pueden depender de las otras.

Como el código solucionado se lo quedará el profesorado, no importará que este conjunto se fusione con las demás ampliaciones. Sin embargo, lo que si que importará serán los test, ya que para la corrección del alumno, será necesario tener los de la ampliación correspondiente.

Para la base de datos no afectará esta unión ya que cada ampliación se tratará como si estuvieran separadas. En el repositorio sucederá lo mismo, ya que se buscará por nombre, y dicha rama contendrá el conjunto de código unido pero con los test de una sola ampliación. De esta forma, si Backtracking selecciona dos o más ampliaciones que formen esta unión de ampliaciones, no generará conflicto ya que el código será el mismo y los test se fusionarán pudiendo corregir la práctica para las ampliaciones seleccionadas.

En el ejemplo de la Figura 2 podemos observar dicho caso en las ampliación A1 y A2, las cuales se observa que están en la misma rama debido a que una depende de otra. La ampliación A1+A2 únicamente servirá de cara al profesorado para tener el repositorio estructurado y no se tendrá en consideración en la base de datos ni en el algoritmo de Bactracking. Esta ampliación contendrá el código, enunciado y los test unidos. Lo que interesará de cara a la base de datos serán las ampliaciones A1 y A2 por separado, los cuales tendrán el código de su unión, sin embargo, los test y enunciado serán los de la ampliación correspondiente. Como se ha comentado, la base de datos solo guardará estos nombres por separado de forma que Backtracking pueda seleccionarlos sin preocuparnos de que al fusionar estas ramas no genere ningún conflicto.

6.2.1. Tipos de ficheros

En esta clase de repositorios hay que tener en cuenta el tipo de ficheros que se debe colgar. Los ficheros binarios como los de Microsoft Word provocan que a la hora de fusionar ramas en repositorios, no funcione correctamente impidiendo generar el enunciado de la práctica correctamente.

En el caso del código de Compiladores no existe problema ya que al no ser binarios se pueden leer en texto plano y fusionar ramas con código sin generar problemas. Para el enunciado, se ha decidido usar ficheros LaTeX [14]. Esta clase de ficheros no son binarios y, al igual que con los ficheros Microsoft Word, se puede añadir imágenes o texto formateado al gusto del usuario.

6.2.2. Conflictos de fusión de ramas

Un punto importante que sucede en este proyecto, son los conflictos de fusión de ramas en los repositorios Git. Estos conflictos aparecen con ficheros que tienen el mismo nombre en cada rama pero que parte de su contenido es diferente y ocupa el mismo sitio, implicando que al fusionar se deba seleccionar una de las dos versiones o ambas de ese fichero tal y como se muestra en la Figura 3.

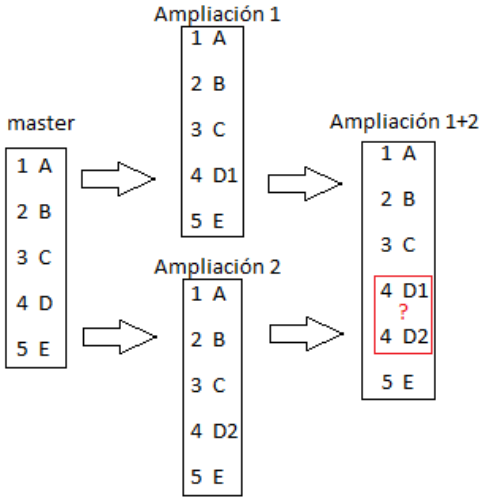

Fig. 3: Ejemplo de fusión de ramas y su conflicto.

Cada rectángulo de la figura representa el contenido de un fichero, ya sea trozos de código o parágrafos de texto, que es modificado en cada rama de diferente forma en el repositorio Git. En la rama master se tiene el fichero original,

pero al crear nuevas ampliaciones y ramificando el repositorio, dicho fichero se le ha modificado el contenido del punto 4 de diferente forma en cada ampliación. El problema surge cuando queremos fusionarlas, ya que aparece el conflicto descrito e impide tener un fichero sin errores.

De forma predeterminada, los repositorios Git no fusionan sin decidir manualmente que contenido es el que se quiere en estos conflictos. Esto se puede configurar modificando el fichero .gitattributes, que viene en cada repositorio Git que se genere. En este fichero se puede decidir si se quiere la versión con el contenido D1, con D2 o ambas. La principal causa de que los ficheros binarios den problemas, es debido a que un fichero binario no se puede crear uniendo otros ficheros binarios ya que no asegura que se genere con el formato correcto. Por ello, dichos ficheros ignoran .gitattributes, impidiendo que se pueda fusionar correctamente.

En este proyecto, al fusionar ramas se necesitará configurar el fichero .gitattributes para que se quede con el contenido de ambas partes, ya que como se ha comentado, se necesitarán los diferentes enunciados y bloques de código de cada rama al fusionarlas. Para hacerlo, se escribe en .gitattributes *"*.csl merge=union"*. Esta línea significa que, al fusionar todos los ficheros que sean csl (ficheros de código de las prácticas de Compiladores), se quedará con las dos variantes del mismo fichero de cada rama. Dicho procedimiento se puede hacer igual con los ficheros LaTeX y con los demás ficheros que no sean binarios.

6.3. Programación Backtracking

El algoritmo de Backtracking se adapta a las necesidades del proyecto ya que el algoritmo explora en profundidad un árbol tomando ciertas decisiones a cada paso. En este caso, el árbol generado dependerá de la cantidad de ampliaciones que tenga la base de datos y por tanto, el primer nivel del árbol serán todas ellas. Para explicar como sigue a partir del primer nivel para encontrar una solución, se ha programado un Backtracking siguiendo el pseudocódigo de la Figura 4.

```
Function Backtracking(database,neededTypes, difficulty ,neededIndex, sol, solutions)
    If (neededTypes = sol.size()) then
        validSolution = isUniqueSolution(database, sol, solutions);
        If(not validSolution) then
            sol.removeTail();
            Return(sol);
        FIf
        Return(sol);
    FIf
    Var= neededTypes[neededIndex];
    for ( database value ) do
        auxSol = null;
        If (satisfyRestrictions(database, Var, sol, auxSol)) then
            sol.append(auxSol);
            neededIndex++;
            Res=Backtracking(database,
            neededTypes ,difficulty,neededIndex, sol, solutions);
            If (Res is a complete solution) then
                Return(Res);
            FIf
        FIf
    FFor
    Return(Error)
FFunction
```

Fig. 4: Pseudocódigo del algoritmo de Backtracking usado.

Esta versión se encarga de encontrar 1 sola solución que satisface dos puntos: que la cantidad y tipos de ampliación sean las que ha exigido el usuario y además, que no existan repeticiones en el conjunto de ampliaciones seleccionadas. Por tanto, para generar las demás prácticas que necesite el usuario se ha decidido crear un bucle en el cual se ejecuta el Backtracking en cada iteración. El objetivo de esto es el de mezclar en cada iteración del bucle y, de manera aleatoria, el orden de la lista donde se guardan todas las ampliaciones.

Con esto se evita sacar todas las soluciones del mismo árbol, provocando que salieran combinaciones con más peso que otras. Mezclando las ampliaciones, la probabilidad de que una ampliación sea seleccionada es la misma que las demás y, que el árbol en el que se buscará una nueva solución sea siempre diferente.

Cada solución encontrada se guardará en una lista, comprobando que la combinación encontrada no se repita con las demás. Cuando acaba el bucle, la lista es usada posteriormente para buscar las ampliaciones en el repositorio Git y luego generar las prácticas.

El algoritmo usa la recursividad para recorrer el árbol y encontrar la solución. En cada recursión, se seleccionan las ampliaciones que cumplan con la petición del usuario. Para ello, se irá mirando en cada ampliación de la base de datos si es compatible con las ampliaciones exigidas por el usuario, mirando si el tipo coincide con la petición y si es compatible con las ampliaciones ya seleccionadas en otras recursiones. Una vez escogido un conjunto de ampliaciones que cumplan con la petición se entrará en el caso base. En este caso, se comprueba si la solución es única mirando si, de todas las soluciones ya encontradas en anteriores ejecuciones de Backtracking, la encontrada sea diferente de las demás. Además, se comprueba si la suma de dificultades de las ampliaciones escogidas está en el rango de dificultad que el usuario ha pedido. Si las condiciones se cumplen, la ejecución de Backtracking finalizará y se guardará esta solución al conjunto de soluciones ya encontradas.

En esta versión del algoritmo existen dos casos por los cuales la ejecución puede acabar sin dar una solución. El primero sucede cuando no se encuentra en la base de datos otra ampliación compatible con la solución propuesta. Esto se debe a que no hay combinación posible en la base de datos que cumpla con la petición del usuario significando que la ejecución del algoritmo debe acabar. El segundo caso aparece cuando la petición del usuario es compleja, implicando que el algoritmo no pueda dar nuevas combinaciones y que queden grupos sin prácticas. En este caso se ha decidido que para el resto de prácticas que necesite el usuario, sean copias de otras ya generadas, implicando por tanto, que existan combinaciones repetidas.

6.4. Interfaz gráfica

Inicialmente, al ejecutar la interfaz se muestra una ventana como la de la Figura 5. En ella se observan tres botones: dos botones de *"Abrir..."*, uno para seleccionar una base de datos válida, la cual al escogerla, permite ver la interfaz de usuario al completo, y el otro para seleccionar un fichero donde se guarden la información de los grupos de prácticas. El tercer botón de *"Generar prácticas"* permite crear las prácticas una vez seleccionado una base de datos válida, los grupos de prácticas y las ampliaciones deseadas.

Si el usuario decidiera directamente generar prácticas sin haber seleccionado una base de datos y grupos de prácticas, la interfaz mostraría un mensaje de error por no tenerlas cargas. Cuando el usuario selecciona un fichero de base de

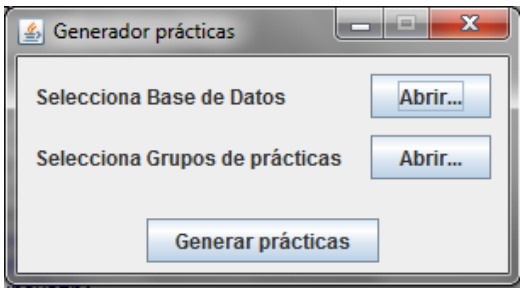

Fig. 5: Interfaz de usuario inicial.

datos, se comprueba que cumple el formato de la Figura 1. Para ello se debe cumplir que el repositorio Git sea válido y que la información de las ampliaciones sean del tipo correspondiente, además de que esté en la columna correcta. En caso de no cumplir con el formato, la base de datos no se considera válida y la interfaz muestra un mensaje de error especificando en donde está.

Cuando se selecciona una base de datos, se comprueba que la carpeta donde se guarda el repositorio Git es válida. En el repositorio se comprueban 2 cosas: que exista la carpeta *.git* (carpeta generada por defecto al crear un repositorio Git), si no existe significa que no es un repositorio y que las ramas del repositorio se nombren igual que las ampliaciones del fichero de la base de datos. Si el nombre de una ampliación no coincide con ninguno de las ramas, significará que dicha ampliación no existe realmente, provocando que no se cargue el repositorio y la interfaz se encargue de enviar un mensaje especificando que ampliación no existe.

Respecto al fichero de grupos de prácticas se hacen ciertas comprobaciones, siendo válida una estructura de fichero como la de la Figura 6. Este fichero debe cumplir que la información de los alumnos sea del tipo correspondiente y que cada grupo contenga al menos un alumno.

	A	B	C	D
1	Grup1			
2		Nom1	Cognom1	11111111
3		Nom2	Cognom2	11111112
4	Grup2			
5		Nom3	Cognom3	11111113
6		Nom4	Cognom4	11111114

Fig. 6: Estructura del fichero de grupos de prácticas.

Si todas las comprobaciones son pasadas, la interfaz cargará los componentes restantes de la interfaz, dando información y opciones para generar las prácticas tal y como se muestra en la Figura 7. En caso contrario, la interfaz no cargará nada y se deberá arreglar el problema correspondiente.

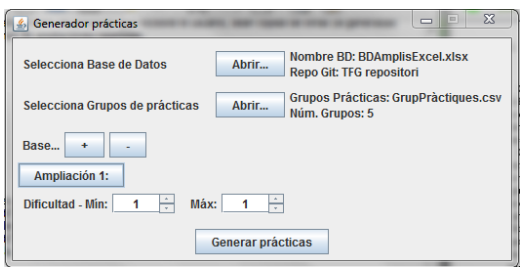

Fig. 7: Interfaz de usuario completa.

Los nuevos componentes, servirán para generar la petición del usuario. Por un lado se tiene los botones de -"para añadir ampliaciones a la petición y de "para eliminar ampliaciones de la petición. Por cada ampliación añadida se crea un botón el cual pulsándolo aparecer una nueva ventana que da la opción de especificar los tipos de ampliación que el usuario quiera tal y como se muestra en la Figura 8.

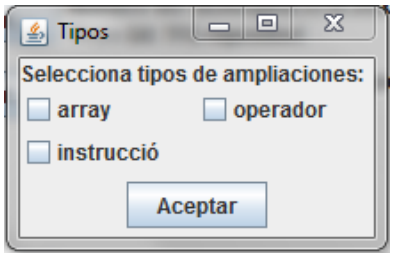

Fig. 8: Ventana de selección de tipos de ampliación.

Finalmente los spinners marcarán el rango de dificultad que quiera el usuario, pudiendo así crear prácticas de menor o mayor dificultad. Por otro lado, el componente que aparece en caso de error es un mensaje que sale debajo del botón de generación de prácticas. Este mensaje será diferente según el error que haya podido pasar durante la ejecución de la aplicación, para dar la información necesaria para poder arregralo y poder generar la prácticas.

6.5. Generación de las prácticas

Cuando el usuario decida qué clase de prácticas necesita, quedará una interfaz como se muestra en la Figura 9.

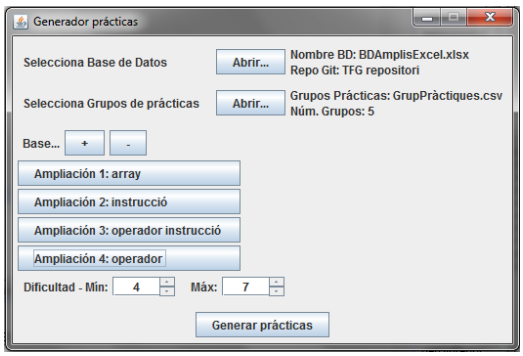

Fig. 9: Interfaz completa con ampliaciones seleccionadas.

Cuando el usuario decida generar las prácticas, la interfaz dejará seleccionar la carpeta en que se desea generar las prácticas y su nombre, dando paso a que se ejecute el algoritmo de Backtracking. Cuando el algoritmo haya dado con la solución, la interfaz crea los directorios donde cada carpeta vendrá con el nombre *NombreGrupo_NIU1_NIU2*. Dicha información se recogerá del fichero de grupos de prácticas, dejando finalmente una estructura de directorios tal y como se muestra en la Figura 10.

En el momento en que se genere la estructura, se generarán las prácticas usando el conjunto de nombres de ampliaciones que ha dado Backtracking como solución. Para ello, se buscarán en el repositorio Git estas ampliaciones mediante el comando *git checkout "nombre índice"*, y fusionarla con las otras ramas de Git mediante el comando *git merge "nombre de rama a fusionar"* formando la práctica

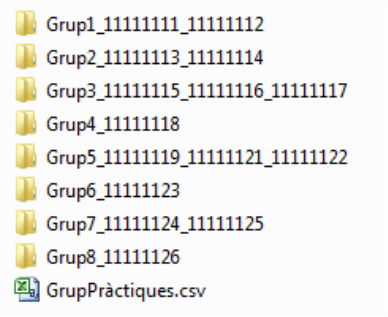

Fig. 10: Estructura de directorios.

de un grupo. Cuando se tiene la práctica formada, todo el contenido se copia a la carpeta del grupo de prácticas correspondiente. Dichas gestiones se han conseguido combinando las librerías de JGit y Apache POI.

Esta generación de prácticas genera un problema. Al fusionar en Git unas ramas, se genera un nuevo nodo en el árbol del repositorio que contiene dicha unión. Si no se borran estos nodos, se genera un árbol con demasiadas ramas innecesarias. La solución tomada es simple: cada vez que se genere una práctica, se aplicará el comando *git reset –hard "Nombre de la rama original"* el cual reiniciará el estado del repositorio al que tenía antes de hacer las fusiones.

Finalmente, cuando todas las carpetas tengan el código y enunciado completo de la práctica asignada al grupo, la aplicación se encargará también de modificar el fichero de grupos de prácticas. Primero copiara dicho fichero a la estructura de directorios generada por la interfaz tal y como se muestra en la Figura 10. Después, en cada fila del fichero donde este el nombre del grupo apuntado, se escribirán las ampliaciones asignadas a la práctica del grupo correspondiente. Este formato se puede ver en la Figura 11 donde se observa que en las siguientes columnas al nombre del grupo, están los nombres de las ampliaciones asignadas.

	A	B	C	D
1	Grup1	Multiples_Variables	potencia	Ins_Switch
2		Nom1	Cognom1	11111111
3		Nom2	Cognom2	11111112
4	Grup2	Multiples_Campos	Factorial	Ins_Switch
5		Nom3	Cognom3	11111113
6		Nom4	Cognom4	11111114

Fig. 11: Grupos de prácticas con ampliaciones asignadas.

6.6. Pruebas generales

Cabe la posibilidad de que la selección del usuario de las ampliaciones que desee produzca un árbol donde en algún punto haya una incompatibilidad no detectada anteriormente o que el formato de los ficheros de las bases de datos no cumpla con el establecido. El objetivo de estas pruebas es la de encontrar estos errores que produzcan prácticas que no funcionen, tanto en el algoritmo de Backtracking, como en las bases de datos o como en la interfaz de usuario y comprobar que se da el mensaje correspondiente.

La base de datos que se ha usado está compuesta por 5 ampliaciones de 3 tipos diferentes. A partir de estas ampliaciones, el repositorio Git se ha creado con la estructura que se muestra en la Figura 12.

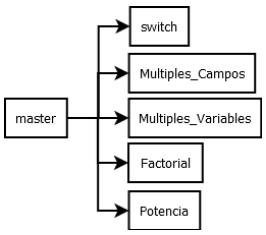

Fig. 12: Estructura del repositorio actual.

Cada rama de este repositorio contiene los ficheros necesarios para poder generar una práctica completa, los cuales se muestran en la Figura 13.

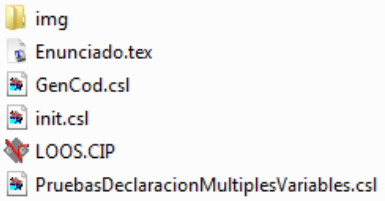

Fig. 13: Contenido de una carpeta de grupo de prácticas.

El fichero *Enunciat.tex* y la carpeta *img*, son los ficheros LaTeX necesarios para generar el enunciado de la práctica. El fichero *GenCod.csl* guarda el código del compilador. En este fichero se incluye el código de cada ampliación que haya sido usada al fusionar las ramas del repositorio. Por otro lado, el fichero *init.csl* contiene las líneas de código para cargar el fichero *GenCod.csl*, los test y las llamadas a las funciones que generar los informes que muestran las notas de los alumnos. El fichero *LOOS.CIP*, se encarga de ejecutar las líneas de código del fichero *init.csl* a modo de ejecutable. Finalmente el fichero *PruebaInstruccionSwitch.csl* contiene los test que deberá pasar la práctica. Cada ampliación tendrá un fichero como este pero con diferente nombre, de modo que al fusionar las ramas para generar una práctica, se tendrá varios ficheros de este tipo, cada uno con test propios de la ampliación correspondiente.

6.6.1. Pruebas sobre la interfaz

Cuando se ejecuta por primera vez la interfaz, se ha probado los casos en que debería dar un mensaje de error impidiendo generar prácticas. Primero se ha probado que se dé el mensaje de error al pulsar directamente a generar práctica sin seleccionar los ficheros de base de datos y de grupos.

A continuación, se ha probado al seleccionar una base de datos, que la interfaz da el mensaje correspondiente cuando no cumpla el formato especificado. Se ha probado los casos en que: el fichero de la base de datos tenga más de 4 columnas, los valores no corresponden con lo que se pide, el nombre del repositorio Git no conduzca a un repositorio y finalmente, si existen espacios en blanco. En todos los casos da el mensaje correspondiente y se especifica la clase de error. Para el repositorio Git se ha comprobado que el número de ramas es el mismo que el de ampliaciones descritas en el fichero de la base de datos y que los nombres se corresponden con cada una de las ampliaciones.

Cuando la base de datos cumple con el formato, se muestran correctamente los demás componentes de la interfaz.

Se ha podido probar que se puede añadir y quitar ampliaciones en la petición y cada botón da acceso a la ventana donde se seleccionan los tipos que se quieren. Estos tipos se ha comprobado que son los mismos que los de la base de datos usada: *instrucció*, *operació* y *declaració*. Además, se ha probado que en los spinners de dificultad no se puedan introducir valores erróneos. Para ello se ha probado que no se pueda poner caracteres ni números reales. Finalmente se ha comprobado que poniendo en el spinner de dificultad mínima un valor superior al spinner de dificultad máxima impida generar prácticas.

Para acabar se ha comprobado que seleccionando un fichero de grupos de prácticas, cumpla su formato. Para ello se ha comprobado que en caso de error, se especifique el mensaje de error correspondiente. Dichos mensajes saltarán en caso de que el fichero de datos tenga más de 4 columnas, los valores no correspondan a los exigidos, existan grupos sin alumnos asignados y la fila donde contiene el nombre del grupo, dicho nombre estará en la primera columna.

6.6.2. Pruebas en la generación de prácticas

Una vez el usuario decida generar prácticas, se ha comprobado los casos en los que debería dar un mensaje de error impidiendo generarlas. Para ello, se ha probado un conjunto de peticiones las cuales no son posibles de hacer con la base de datos actual tales como: pedir una cantidad de tipos de ampliación las cuales no existe combinación posible en la base de datos, pedir un rango de dificultad con valores excesivos y el caso de no pedir ninguna ampliación. En los dos primeros casos se ha dado el mensaje correspondiente informando de que no hay combinación posible y para el tercer caso el mensaje de petición de prácticas incorrecta.

A continuación, se han hecho pruebas más específicas para observar si las soluciones dadas son correctas. Las primeras han sido centradas en comprobar si los rangos de dificultad, tipos seleccionado y ampliaciones seleccionadas de cada solución eran correctas. Probando en casos sencillos de peticiones simples y fáciles de corregir con la vista, se ha podido comprobar que las soluciones daban con el tipo de petición que el usuario ha pedido, dentro del rango de dificultad, todas las combinaciones de ampliaciones diferentes y con los tipos de ampliación exigidas. Además ejecutando los test que deben pasar las prácticas para que el alumno obtenga la nota de su práctica, se puede comprobar si los pasa todos, significando que no hay errores en el código. Para ello al ejecutar los test debe dar una respuesta como en la Figura 14.

```
Resultados sintactico ----------------------------------
Test sintactico NO OK.............: []
Test sintactico obligatorios NO OK: []
Test sintactico Ok................: 11 (100%)
Test sintactico obligatorio.......: CORRECTO
Resultados semantico -----------------------------------
Test semantico NO OK.............: []
Test semantico obligatorios NO OK: []
Test semantico Ok................: 12 (100%)
Test semantico obligatorio.......: CORRECTO
Resultados generacion de codigo ------------------------
Test generacion de codigo NO OK.............: []
Test generacion de codigo obligatorios NO OK: []
Test generacion de codigo Ok................: 8 (100%)
Test generacion de codigo obligatorio.......: CORRECTO
NOTAS
SINTACTICO=10
SEMANTICO=10
GENCOD=10
```

Fig. 14: Resultado de ejecutar los test de una práctica.

Estos test se encargan de probar las tres fases de la práctica de la asignatura de Compiladores, prueba test sintácticos, semánticos y de generación de código del compilador. En esta figura se observa que si todos los test han pasado, muestra al final las notas de la práctica y que nota máxima de 10 querrá decir que el código es totalmente funcional y sin errores. En el caso de este proyecto, se ha ido probando a ejecutar los test de todas las combinaciones, y ninguno de los casos ha dado problemas, dando nota máxima.

Para acabar, se ha probado de pedir un nombre elevado de prácticas para ver si se crean prácticas repetidas. Para ello se ha creado un fichero de grupos de prácticas con 5 grupos ya que en la base de datos que se ha usado, el máximo número de combinaciones dadas es de 4. Como se comentó, en el caso de que se necesite repetir prácticas, el resto que no se han generado por el Backtracking, serán copias de otras soluciones ya propuestas y en este caso, se ha comprobado que el quinto grupo obtiene siempre el mismo tipo de práctica que el del primer grupo.

Paralelamente se ha probando varias combinaciones de código en LaTeX para que al fusionar ramas se compruebe que el texto resultado sigue compilando y teniendo sentido.

6.7. Integración

Con las prácticas generadas, se ha adaptado el aplicativo web de la asignatura con tal de que sea capaz de corregir todas las prácticas de manera automática, permitiendo que la web se encargue automáticamente de usar los test de corrección correspondientes para puntuar la práctica del alumno pudiendo ver su nota y los test que no haya superado.

Anteriormente para corregir las prácticas, el profesor creaba una entrega de prácticas en el aplicativo rellenando los campos tal y como se muestra en la Figura 15. El archivo a colgar debía ser una carpeta comprimida, el cual contenía el fichero de los test que debe superar el alumno y un fichero *init.csl*, encargado de buscar el fichero de código que colgaba el alumno y pasar los test. Con los resultados generados, el aplicativo los guarda en una carpeta separada.

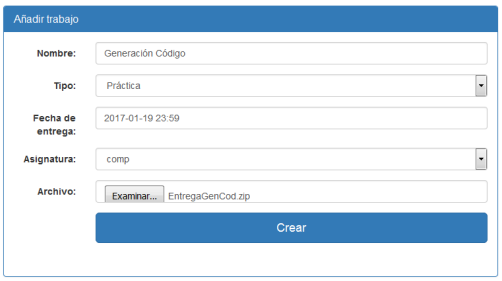

Fig. 15: Formulario de generación de entrega.

Una vez generada la entrega, los alumnos podían colgar sus prácticas para que fueran evaluadas, pudiendo ver su última entrega para comprobar el script que colgó y sus resultados. Como antes solo había un fichero de test que debían superar todos los alumnos, el fichero *init.csl* se simplificaba en cargar el fichero de código del alumno y los tests del profesor para ejecutarse y dar los resultados. Actualmente se tiene un conjunto de test diferentes, divididos en distintos ficheros y agrupados en carpetas según los grupo de prácticas que se tengan tal y como se mostró en la

Figura 10 de la estructura de directorios, con lo que el trabajo de *init.csl* se centrará esta vez en saber dónde buscar los test correspondientes.

En primer lugar se ha sido modificar el código del aplicativo web, para que además de indicarle al programa *init.csl* donde se encuentra el fichero que ha colgado el alumno, también indique el NIU del alumno. Con esto, no hará falta modificar más el aplicativo, por tanto, la importancia está en programar el *init.csl* para que busque los test.

Para ello, la estructura de la carpeta comprimida que debe colgar el profesor, puede ser el mismo que el de la Figura 10 borrando el fichero csv, ya que el aplicativo no permite enviar ficheros que no sean de código. Los ficheros de *GenCod.csl* de cada carpeta deben borrarse también ya que si nó, cuando el alumno cuelgue su práctica se corregirá su *GenCod* y además el de la carpeta, dando así que el alumno siempre saque máxima nota. Finalmente, junto a las carpetas añadiremos el fichero *init.csl* encargado de buscar los test correspondientes. Dicho fichero se programará siguiendo el pseudocódigo de la Figura 16

```
var rutas;
foreach (nombre_carpeta) do
        if (nombre_carpeta contiene NIU) then
                rutas.append (nombre_carpeta.ruta);
                rutas.append (carpeta_base.ruta);
                rutas.append (carpeta_códigoAlumno.ruta);
                ejecutaTest (rutas);
        Fif
FiForeach
```

Fig. 16: Pseudocódigo init.csl.

El objetivo es que el programa busque la carpeta que en su nombre contenga el NIU del alumno. Como en la generación de las prácticas se crean las carpetas con el formato *NombreGrupo_NIU1_NIU2*, se asegurá que el programa encontrará la carpeta del alumno que ha colgado la práctica. Con la carpeta encontrada, se guarda en una lista la ruta de la carpeta, la ruta donde se encuentra los test de la práctica base y finalmente la ruta donde el código del alumno ha sido guardado. Con estas rutas solo hace falta ejecutar los test, los cuales llamando a los *init.csl* de la carpeta del grupo y de la práctica base, se cargan y se ejecutan solos, generando el informe con los resultados. Finalmente y como ya se hacía, el aplicativo web, recoge el informe y lo guarda en la carpeta donde está el código del alumno. Con esto, el alumno podrá descargarse su informe y se le mostrará el mismo contenido que el de la Figura 14. Cuando la fecha de entrega expire, el alumno podrá descargarse el mismo informe, pero esta vez, incluyendo la información que ya contenía, se le mostrará los tests fallados y su causa.

7 Resultados

En primer lugar, se ha conseguido especificar una estructura que debe tener la base de datos y que información debe contener tal y como se ha mostrado en la Figura 1. Dicho fichero será un Excel o csv y contendrá la información de las ampliaciones que se usarán para crear las ramas: nombre de ampliación, tipo, sus incompatibilidades y su dificultad.

Seguidamente se ha hablado del repositorio Git y su estructura en forma de árbol como en la Figura 2. Se ha especificado que tipo de ficheros tendrá cada rama, el motivo de que no sean ficheros binarios y como debe estar configurado el repositorio con tal de evitar los conflictos de ramas modificando el fichero *.gitattributes*. Por ello se ha conseguido crear un repositorio adaptado a las necesidades del proyecto, configurándolo para que no genere problemas al fusionar ramas.

Una vez creada la base de datos y su repositorio, se ha adaptado el algoritmo de Backtracking para poder usar dicha base de datos y generar un conjunto de ampliaciones para generar la prácticas. Con esto se ha conseguido un núcleo capaz de cumplir con el objetivo principal del proyecto.

Con todos estos elementos, el siguiente paso ha sido integrarlo en una interfaz gráfica. Dicha interfaz es la que se ha mostrado en la Figura 7 y será el diseño actual de la aplicación. Se ha comentado los componentes que tiene y su función. Los botones para buscar una base de datos y un fichero de grupos de prácticas que sean Excel o csv. También se han creado los botones para añadir y quitar ampliaciones, además de los botones para cada ampliación que muestran una nueva ventana de selección del tipo de ampliación que se desea. Luego los spinners para seleccionar el rango de dificultad que tendrá el conjunto de ampliaciones al generar la práctica completa y finalmente, el botón de generar prácticas, el cual ejecuta el algoritmo de Backtracking usando los datos de las ampliaciones para la creación de prácticas.

Una vez obtenida un conjunto de prácticas a crear, se ha conseguido que se genere una estructura de directorios como la que se mostró en la Figura 10, donde cada carpeta se guardará una práctica que incluye todas las ampliaciones que haya decidido el algoritmo de Backtracking. Para ello se usará JGit para la fusión de ramas del repositorio Git, generando así una práctica completa que se copiará del repositorio a la carpeta correspondiente del grupo. Dichas carpetas únicamente contendrán lo que se mostró en la Figura 13.

Llegados a este punto, se ha hecho el periodo de pruebas para esta base de datos. Como se ha comentado, se ha conseguido probar tanto la interfaz gráfica observando que los mensajes de error los diera en el momento oportuno, como los propios resultados de las fusiones de ramas del repositorio Git, usando los test como fuente fiable para confirmar que el código no tiene errores y que los test los pasa todos.

Finalmente se ha integrado toda la aplicación al aplicativo web de corrección de prácticas de Compiladores. Para ello se ha modificado los ficheros que ejecutan los test de corrección para que cuando el alumno cuelgue su práctica para que se corrija, el aplicativo identifique que práctica tiene asignada mirando el NIU y pase los test adecuados.

8 Conclusiones

El desarrollo del proyecto ha cumplido la distribución de horas que se ha impuesto. Aun así el proyecto ha ido sufriendo cambios en la preferencia de los objetivos debido a que no se tenía en su momento el material necesario para su cumplimiento. Todos estos cambios han sido necesarios para conseguir llegar a la fecha límite habiendo cumplido todos los objetivos.

En este proyecto se ha cumplido todos los objetivos principales y secundarios que se habían exigido al principio de manera satisfactoria, puesto que se ha conseguido crear

una aplicación con una interfaz que genere un conjunto de prácticas de la asignatura de Compiladores, de manera aleatoria pero sin que se repita ninguna de las prácticas generadas adaptando el algoritmo de Backtracking a las necesidades de este proyecto. Por otro lado se ha podido conocer nuevas librerías y aplicaciones que han ayudado en la ejecución de este proyecto tales como el sistema control de versiones Git o las librerías Apache POI para gestionar ficheros de Microsoft.

Finalmente se ha podido probar que las ampliaciones proporcionadas por el profesorado son todas compatibles entre ellas, dando así una aplicación que no tendrá ningún fallo a la hora de generar las prácticas y siendo optimistas para las futuras ampliaciones que se añadirán a este conjunto inicial de ampliaciones.

8.1. Líneas futuras

La base de datos aún le quedan muchas ampliaciones por añadir, así que la colaboración con el profesorado de la asignatura de Compiladores implicará seguir creando nuevas ampliaciones, consiguiendo el ir aumentando la base de datos durante este tiempo.

Además, esta aplicación se puede adaptar para otras asignaturas de la titulación, creando otra base de datos que se adapte a las necesidades de la asignatura. Por otra parte, también se puede usar como herramienta para los alumnos. Se les proporciona la aplicación y una base de datos que genere problemas de la asignatura, consiguiendo que los alumnos puedan generarse y corregir sus propios ejercicios para estudiar.

Para la interfaz gráfica se puede proporcionar nuevas funcionalidades. Un punto pendiente en este proyecto sería generar un informe en el momento de generar prácticas que proporcione información en los casos en que surgan incompatibilidades no detectadas, avisando que ampliación ha producido el error.

Finalmente, en la colaboración con el profesorado se está haciendo una página web de challenge para congresos de investigación, el cual su contenido es generado cargando un fichero de texto donde viene escrito la información relacionada con el challenge. Dicho contenido se podría generar adaptando la aplicación del proyecto para crear diversos challenge utilizando las condiciones que el usuario desee.

AGRADECIMIENTOS

En primer lugar, agradecer a Javier Sánchez, tutor de este proyecto, por su gran paciencia y apoyo que han hecho este trabajo posible. Agradecer también al equipo docente de la Escuela de Ingeniería de la Universidad Autónoma de Barcelona, pos su dedicación en formar y educar a esta generación de ingenieros y las que están por venir.

Finalmente, y no menos importante, agradecer a mi familia el cariño y su apoyo que han tenido conmigo, ya que sin ellos hubiera sido improbable haber cursado esta titulación sin ningún problema.

REFERENCIAS

[1] Escuela de Ingeniería, "Compiladors," 2016, accessed: 2016-9-28. [Online]. Available: http://www.uab.cat/guiesdocents/2016-17/g102782a2016-17iCAT.pdf

[2] Software Freedom Conservancy, "Git –fast-version-control," 2009, accessed: 2016-9-18. [Online]. Available: https://git-scm.com/about

[3] D. Matuszek, "Backtracking," 2002, accessed: 2016-9-22. [Online]. Available: https://www.cis.upenn.edu/ matuszek/cit594-2012/Pages/backtracking.html

[4] Barbara Liskov, "A History of CLU." Apr. 1992, accessed: 2017-1-4. [Online]. Available: http://publications.csail.mit.edu/lcs/pubs/pdf/MIT-LCS-TR-561.pdf

[5] James Parsons, "The top 10 traffic generator applications on the web." Jan. 2015, accessed: 2017-1-4. [Online]. Available: http://growtraffic.com/blog/2015/01/top-10-traffic-generator-applications-web

[6] Fidenia srl., "Questbase." accessed: 2017-2-2. [Online]. Available: http://www.questbase.com/product/features.aspx

[7] Wolfram Research, "WolframAlpha," accessed: 2017-2-2. [Online]. Available: https://www.wolframalpha.com/

[8] Universidad Autónoma de Barcelona, "Campus Virtual de la UAB," accessed: 2017-2-6. [Online]. Available: https://cv.uab.cat/portada/ca/index.html

[9] ——, "Cerbero - Entorn de Gestió Documental," accessed: 2017-2-6. [Online]. Available: http://cerbero.uab.cat/moodle2/

[10] Vialtest, "Vialtest.com," accessed: 2017-1-4. [Online]. Available: https://vialtest.com/dgt-examenes

[11] The Apache Software Foundation, "Apache POI - the Java API for Microsoft Documents," 2002, accessed: 2016-9-21. [Online]. Available: http://poi.apache.org/index.html

[12] Software Freedom Conservancy, "Embedding git in your applications - jgit," accessed: 2016-9-21. [Online]. Available: https://git-scm.com/book/es/v2/Embedding-Git-in-your-Applications-JGit

[13] M. McCullough, "What is version control?" accessed: 2016-9-19. [Online]. Available: https://git-scm.com/video/what-is-version-control

[14] LaTeX3 Project, "The LaTeX project." accessed: 2016-12-12. [Online]. Available: https://www.latex-project.org/

[15] Tutorialspoint, "Swing tutorial," accessed: 2016-9-22. [Online]. Available: http://www.tutorialspoint.com/swing/

[16] The Eclipse Foundation, "Eclipse," accessed: 2016-9-30. [Online]. Available: https://eclipse.org/home/index.php

APÉNDICES

A.1. Diagrama de clases de la interfaz

Para conseguir gestionar toda la interfaz incluyendo las bases de datos y el repositorio Git, se ha programado siguiendo el diagrama de clases mostrado en la Figura 17.

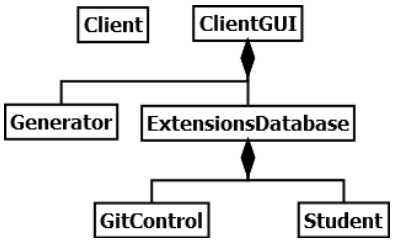

Fig. 17: Diagrama de clases de la aplicación.

La mayoría del trabajo la tiene la clase *ClientGUI* por ser el centro que controla todos los botones y spinners, los cuales estos gestionan las bases de datos y llaman al algoritmo de Backtracking para finalmente generar las prácticas. A continuación, se explica la función que tiene cada clase.

- **Client:** Esta clase se encarga de llamar a la clase ClientGUI para empezar con la ejecución.
- **ClientGUI:** Clase principal de la aplicación. Esta clase se gestiona cada componente de la interfaz, la base de datos de ampliaciones y la generación de las prácticas. La interfaz gráfica se ha programado usando las librerías Swing que proporciona Java por simplicidad en las necesidades de este proyecto [15]. Una vez el usuario haya escogido una base de datos válida, se crea un objeto *ExtensionsDatabase*. En dicho objeto se guarda toda la información relacionada con la base de datos, asi que una vez se haya creado este objeto, la clase *ClientGUI* se encarga de mostrar al usuario el resto de los componentes de la interfaz. Una vez se haya seleccionado las ampliaciones deseadas y un rango de dificultad, al pulsar el botón de generación de prácticas, se crea un objeto *Generator* el cual se encarga de ejecutar el algoritmo de Backtracking usando los valores que haya puesto el usuario en la interfaz como datos para generar las prácticas.
- **ExtensionsDatabase:** Clase encargada de leer la base de datos de las ampliaciones, los datos del fichero de los grupos de prácticas, los cuales se guardarán en un objeto *Student*, y el repositorio Git, el cual generará un objeto *GitControl* donde se guardan los datos de este repositorio. Esta clase se creará en el momento en que el usuario haya seleccionado una base de datos válida.
- **GitControl:** Encargado de gestionar el repositorio Git y por tanto, encargada de proporcionar métodos capaces de aplicar comandos Git con tal de poder gestionar el repositorio. Una vez se ha leído la base de datos, se crea un objeto de esta clase que guardará la ruta de donde está el repositorio y las ramas que contiene. Esta clase junto a la clase *Generator*, se encargan de crear todas las prácticas y de guardarlas en la estructura de directorios que se generará al darle al botón de generación de prácticas. Para conseguirlo se han usado las librerías JGit que proporciona Git para gestionar repositorios en aplicaciones Java.
- **Student:** Clase encargada de guardar la información relacionada a los grupos de prácticas. Esta clase se crea un objeto en el momento de leer la base de datos de las ampliaciones. Utilizada en la generación de los directorios ya que se usa el nombre de grupo y los NIUs de los integrantes del grupo a modo de nombre de las carpetas.
- **Generator:** Clase que aplica el algoritmo de Backtracking. En esta clase se crea un objeto en el momento en que el usuario pulse el botón de generar las prácticas. Dicho objeto se encarga principalmente de crear una solución a la petición del usuario usando el algoritmo de Backtraking y, acto seguido, generar la estructura de directorios que contendrá las prácticas. Una vez se crea la estructura, se llaman a los métodos de la clase *GitControl* para fusionar las ramas del repositorio y generar las prácticas que se guardarán en cada carpeta de dicha estructura.

A.2. Herramientas usadas

- **Microsoft Excel**: ficheros Excel que ayudan para crear la base de datos que guarda toda la información relacionada con las ampliaciones y para relacionar los grupos de prácticas con las prácticas generadas.
- **Git**: sistema control de versiones con herramientas de ramificación y fusión que ayudan a crear un repositorio en forma de árbol, el cual se ira fusionando dichas ramas para unir ampliaciones y así formar las diferentes prácticas para los alumnos.
- **Eclipse MARS.1**: framework para la programación de aplicaciones en Java que ha servido para crear el núcleo de toda la aplicación [16].
- **JGit**: librerias de Java usadas para poder llamar comandos Git en proyectos de Java.
- **Apache POI v3.15**: librerias de Java necessarias para la manipulación de ficheros de Microsoft tales como Word o Excel [11].
- **Librerias AWT y Swing**: librerías de Java utilizadas para la creación de interfaces gráficas.

A.3. Diagrama de Gantt

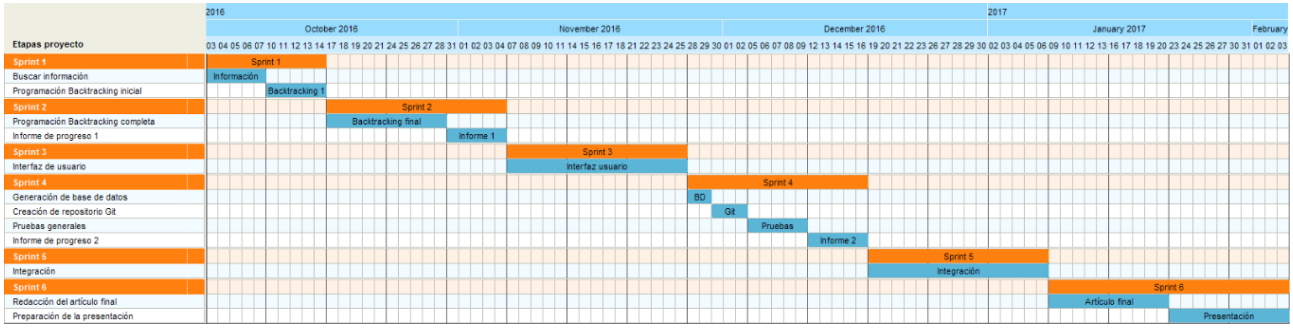

Fig. 18: Diagrama de Gantt del proyecto