UNIVERSIDAD
COMPLUTENSE
MADRID

Proyecto de Innovación y Mejora de la Calidad Docente

Convocatoria 2015

Proyecto nº 32

Implementación de un entorno de aprendizaje colaborativo de lenguajes de programación mediante traducción

Responsable del proyecto: Adrián Riesco

Facultad de Informática

Departamento de Sistemas Informáticos y Computación

1. Objetivos propuestos en la presentación del proyecto

Los objetivos de este proyecto son:

- O1: Implementar una herramienta que sea capaz de comparar 2 programas y decidir si son equivalentes. Dicha comparación no será de caja negra, sino que estará basada en la comparación de (i) los diagramas de control de flujo; (ii) las comparación simbólica de las aristas; y (iii) la comparación simbólica de los valores al final de cada bloque, dadas las restricciones generadas por las aristas.

- O2: Implementar una interfaz que dé soporte a la herramienta diseñada en el apartado anterior. Dicha interfaz será en principio un servicio web, como se indicó en el PIMCD 97/2014.

- O3: Diseñar e implementar una base de datos para guardar la información sobre usuarios, temas, candidatos y soluciones.

- O4: Implementar el login de tal manera que asegure la seguridad de los usuarios. Nuestra propuesta consiste en usar las cuentas institucionales de los estudiantes (Google), lo que nos proporciona las siguientes ventajas: (i) no es necesario registro; (ii) los docentes pueden identificar a los estudiantes, facilitando el seguimiento; (iii) se puede usar la red social propia de la cuenta (Google+) para publicación de logros.

- O5: Implementar un conjunto de problemas en todos los lenguajes soportados por la herramienta. En principio la herramienta solo soportará dos lenguajes de programación: Python y Java, y por ello todos los problemas deberán estar implementados en ambos lenguajes. Este conjunto de problemas, que estará separado por temas y lecciones según su contenido y complejidad, incluirá tanto problemas clásicos de programación para aprendizaje de las estructuras de control básicas como problemas sobre estructuras de datos y métodos algorítmicos.

- O6: Escribir las explicaciones correspondientes a cada uno de los temas, de tal manera que la herramienta permita un aprendizaje autónomo a los alumnos.

2. Objetivos alcanzados

Todos los objetivos han sido alcanzados exitosamente siguiendo las pautas recopiladas en el PIMCD 97/2014, *Estudio de viabilidad de un entorno de aprendizaje colaborativo de lenguajes de programación*. En particular:

- R1: Como se indica en las secciones 6.1. y 6.2., se ha desarrollado un comparador en Java que recibe el código fuente de un programa escrito en Java o en Python y determina si dicho programa es equivalente a otro almacenado previamente como solución. Dado que esta es una herramienta para aprender lenguajes de programación, el concepto de "iguales" no se refiere simplemente a que los programas se comporten igual, sino a que tengan una estructura parecida. Sin embargo, forzar demasiado el parecido en las estructuras puede resultar frustrante para los estudiantes, por lo que decidimos exigir el mismo Diagrama de Control de Flujo (en el cual se deben usar los mismos tipos de bucles) pero flexibilizar el orden de asignación y el nombre de las variables; la igualdad de los valores de las variables se comprueba posteriormente usando el resolutor de restricciones Z3. Este resultado cubre el objetivo O1.
- R2: Como se detalla en la sección 6.3, se ha diseñado una base de datos para almacenar problemas y usuarios e implementado una interfaz que conecta a los usuarios, la base de datos y el comparador descrito en la página anterior. Además, el login de usuario en esta aplicación se puede realizar usando cuentas tanto de Gmail (dado que actualmente todos los estudiantes de la UCM cuentan con estas cuentas) como de Facebook (ya que consideramos que esta es la red social con más usuarios en la actualidad). Esta conexión permite a los usuarios conectar entre ellos e incremente la gamificación, un concepto que consideramos necesario para "enganchar" a los usuarios en el uso de la herramienta. Este resultado cubre los objetivos O2, O3 y O4.
- R3: Hemos compilado un manual con explicaciones sobre las estructuras comunes a Java y Python, tal y como se detalla en la sección 6.4. Estas explicaciones, divididas en temas tal y como se desarrolló en el PIMCD 97/2014, cuentan cada una con una serie de problemas para practicar. Además, todos estos problemas cuentan con (i) soluciones en ambos lenguajes, lo que nos permite comparar la respuesta del usuario siguiente el método descrito en el resultado R1 y (ii) casos de test, lo que además de darnos confianza en su corrección nos permitirá en el futuro realizar otros tipos de comprobaciones de caja negra. Este resultado cubre los objetivos O5 y O6.

3. Metodología empleada en el proyecto

Durante el desarrollo de este proyecto hemos diferenciado 6 tareas las cuales cubren los diferentes objetivos. Hemos podido desarrollar gran parte de estas tareas en paralelo gracias a su independencia, lo cual nos ha permitido obtener todos los objetivos descritos en el capítulo anterior en el periodo mayo-diciembre de 2015. A continuación describimos brevemente cada una de estas tareas, junto con sus períodos de ejecución, participantes y resultados.

- **Tarea 1. Desarrollo de la componente que comprueba la equivalencia de 2 programas (O1):** El desarrollo y consecución de esta tarea viene principalmente representado por el objetivo cumplido R1. Para esta tarea se desarrolló un proyecto final de grado con título *Comparador de Java y Python para Duocode*, de los alumnos David García Domínguez y Ángel Villalba Fernández-Paniagua, dirigido por los participantes de este PIMCD Adrián Riesco Rodríguez y Manuel Montenegro Montes (responsable de la tarea) presentado en junio de 2015. En todo momento se contó con la continua colaboración de Salvador Tamarit tanto para consulta como ejecución.

- **Tarea 2, Desarrollo del servicio web que proporcionará la funcionalidad del sistema (O2), Tarea 3, Implementación una interfaz gráfica que utilice haga uso del servicio web (O2 y O4) y Tarea 4, Diseño e implementación la base de datos para almacenar la información del sistema (O3):** Estas 3 tareas muy relacionadas han sido principalmente desarrolladas mediantes un proyecto final de grado con título *Desarrollo de un front-end para DuoCode*, de los alumnos Julián F. Calleja da Silva, Johana Gabriela Ferreira Yagua y José Carlos Valera Villalba, dirigido por los profesores Enrique Martín Martín (responsable de T2) y Adrián Riesco Rodríguez (responsable de T3), ambos participantes del PIMCD, y presentado en junio de 2015. Los resultado de estas tareas se ven reflejados principalmente en el objetivo cumplido R2. A parte de los directores del proyecto, estas tareas han contado con la colaboración tanto en planteamiento como en desarrollo de los participantes Rafael Caballero y Manuel Montenegro (T2, T3 y T4), y de Salvador Tamarit (responsable de T4 y participante en T2 y T3).

- **Tarea 5, Generación de contenido: problemas y explicaciones (O5 y O6):** Para la consecución de esta tarea se desarrolladas mediante el proyecto final de grado titulado *Implementación de ejemplos para DuoCode*, de la alumna Laura Muñoz, dirigida por el profesor Adrián Riesco, y presentado en septiembre de 2015. Los objetivos conseguidos con esta tarea quedan descritos por el objetivo cumplido R3. En esta tarea ha sido de vital importancia la colaboración y experiencia de los profesores Rafael Caballero (responsable de la tarea) y Enrique Martín Martín.

- **Tarea 6, Integración:** Los últimos meses del proyecto se dedicaron a juntar resultados obtenidos en las anteriores tareas mediante la colaboración de todos los participantes, siendo Adrián Riesco el responsable de coordinación.

4. Recursos humanos

A continuación detallamos la participación de cada miembro del proyecto en las tareas detalladas en el punto anterior.

- **Rafael Caballero:** Ha participado en la obtención del objetivo cumplido R2 que corresponde con los objetivos iniciales O2, O3 y O4, y que se ha llevado a cabo durante la elaboración del trabajo fin de grado mencionado anteriormente y presentado en junio de 2015. Como responsable de la tarea T5 ha participado en la generación del contenido de la aplicación, tanto problemas como sus explicaciones, junto con Adrián Riesco y Enrique Martín.

- **Enrique Martín Martín:** Responsable de la tarea T2, que se ha llevado a cabo como parte del trabajo fin de grado *Desarrollo de un front-end para DuoCode*, codirigido junto con Adrián Riesco. El mismo trabajo fin de grado ha servido para desarrollar la tarea T3 y la tarea T4, cubriendo de esta forma los objetivos O2, O3 y O4. También ha colaborado en el desarrollo de la tarea T5.

- **Manuel Montenegro Montes:** Responsable de la tarea 1, que ha llevado a cabo co-dirigiendo junto con Adrián Riesco el trabajo fin de grado C*omparador de Java y Python para Duocode*, presentado en Junio de 2015. Además, su labor ha sido fundamental en el desarrollo de las tareas T2, T3 y T4, ocupándose del desarrollo de la interfaz web y colaborando en el desarrollo de la funcionalidad del sistema.

- **Adrián Riesco:** Ha actuado de coordinador del proyecto (tarea 6), interaccionando con el resto de miembros para asegurar el éxito de cada una de las tarea y por tanto lograr la consecución de los objetivos propuestos. Por otra parte, ha sido codirector del Trabajo Fin de Grado *Comparador de Java y Python para Duocode*, de los alumnos David García Domínguez y Ángel Villalba Fernández-Paniagua, que ha permitido llevar a cabo la tarea 1, y por consiguiente el objetivo 1, y del trabajo *Desarrollo de un front-end para DuoCode*, de los alumnos Julián F. Calleja da Silva, Johana Gabriela Ferreira Yagua y José Carlos Valera Villalba, fundamental para el desarrollo de las tareas 2 y 3 (esta última en la que figura como responsable). Finalmente también ha sido fundamental en el desarrollo de la tarea 5, dirigiendo el trabajo fin de grado *Implementación de ejemplos para DuoCode*, cuya temática cubre los objetivos de esta tarea.

- **Salvador Tamarit:** Responsable de la tarea T4, ha participado de forma activa en el diseño e implementación de la base de datos que almacena tanto los datos de los usuarios como los de los problemas. Su labor también ha sido fundamental para el desarrollo de las tareas T2 y T3, desarrollando la

funcionalidad del sistema y colaborando en la implementación de los servicios web.

5. Desarrollo de las actividades

Como se ha explicado en las secciones anteriores, los resultados obtenidos en este PIMCD han sido en gran medida desarrollados por estudiantes de la Facultad de Informática y de la Facultad de Ciencias Matemáticas como parte de sus Trabajos Fin de Grado. Por ello, gran parte del desarrollo ha tenido lugar hasta el mes de octubre, cuando se terminaron de presentar los trabajos, mientras que los meses posteriores han servido para poner la herramienta a punto para poder ser evaluado por los estudiantes.

Para la Tarea 1 hubo reuniones quincenales entre Manuel Montenegro y Adrián Riesco con los estudiantes del trabajo *Comparador de Java y Python para Duocode*. Para preparar los contenidos de estas reuniones se hicieron reuniones con Salvador Tamarit, para fijar objetivos y aclarar conceptos antes de asignárselos a los estudiantes.

Asímismo, para las Tareas 2, 3 y 4 se efectuaron reuniones quincenales entre Enrique Martín y Adrián Riesco con los estudiantes del trabajo *Desarrollo de un front-end para DuoCode*. Como preparación y a continuación de las reuniones anteriores se realizaron reuniones entre todos los componentes del grupo. Estas reuniones sirvieron tanto para planificar estas tareas como para decidir la futura integración de todas las partes del proyecto.

La Tarea 5 se llevó a cabo en parte en paralelo con las anteriores tareas pero continuó una vez estas acabaron, ya que este TFG se presentó en octubre mientras los anteriores se presentaron en junio. La estudiante implementó los problemas previamente recopilados por los profesores Rafael Caballero, Enrique Martín y Adrián Riesco. Además, cada vez que un tema finalizaba dichos profesores repasaban los problemas y la teoría generada y proporcionaban los materiales necesarios para el siguiente.

Por último, en la Tarea 6, que se extendió desde la finalización de los primeros TFGs en junio hasta final de año, todos los miembros del grupo se encargaron de unificar los resultados obtenidos para dejar la herramienta preparada para su uso en futuros cursos.

6. Anexos

En esta sección presentamos en detalle los resultados alcanzados durante el desarrollo de este Proyecto de Innovación y Mejora de la Calidad Docente. En primer lugar describimos cómo se implementó el comparador de programas, continuaremos con la implementación de la interfaz y por último presentaremos brevemente cómo se pobló la base de datos.

6.1. Construcción del Diagrama de Control de Flujo

El principal componente de nuestra herramienta es el comparador de programas. Ya vimos en el PIMCD 97/2014 que existen muchos métodos para comparar programas, aunque el tipo específico de comparador que nosotros necesitamos varía ligeramente de los estudiados hasta ahora: era necesario un comparador que se fije no solo en los resultados (por lo que debemos descartar los comparadores de caja negra, aunque bien podrían reforzar la comparación en el futuro) sino que se fije en la estructura. Necesitamos que la comparación sea estricta para las estructuras (como estamos diseñando una herramienta educativa no nos vale que el usuario solucione un problema que requiere bucles **while** usando bucles **for**) pero por ejemplo laxa con los nombres de las variables y el orden de asignaciones, siempre y cuando dicho orden no afecte al resultado final.

Por estas razones, decidimos crear los Diagramas de Control de Flujo (CFG) correspondientes a los dos programas que queremos comparar, indicando en las aristas el tipo de condición que la genera (**while**, **for**, **if**, **switch**, etc.) y dejando como nodos los bloques de código usados en cada una de estas construcciones. Una vez construidos los CFGs la comparación tiene 2 fases: en primer lugar comprobamos que la estructura sea la misma (es decir, que tengamos un isomorfismo de grafos), dejando sin comprobar los valores que toman las variables. En una segunda fase, que describimos en la sección 6.2., comprobaremos que realmente los valores que toman las variables son equivalentes.

La implementación de la comparación se llevó a cabo en el trabajo de fin de grado *Comparador de Java y Python para Duocode*[1]. Para ello se decidió usar el lenguaje de programación Java, ya que disponía de las bibliotecas adecuadas tanto para la generación del árbol de sintaxis abstracta como para la resolución de restricciones. El primer paso para la construcción del CFG es el análisis sintáctico del programa introducido y la generación del árbol de sintaxis abstracta (AST). Para ello utilizamos la herramienta ANTLR, que permite realizar el análisis sintáctico de cualquier programa cuya gramática haya sido previamente cargada. Una de las razones que nos llevaron a utilizar esta herramienta es su amplia biblioteca de gramáticas para lenguajes de programación, que ya incluía las de Java y Python 3, así como muchas otras que nos permitirán ampliar el trabajo en el futuro. Una vez generado el AST y definidas las clases para almacenar el CFG (todas ellas definidas por nosotros) se recorre el AST, fusionando nodos se encuentran secuencias de asignaciones y generando nuevos nodos y aristas cuando nos encontramos con instrucciones de control de flujo. El resultado final del trabajo nos permite trabajar con el cuerpo de cualquier método que no contenga excepciones.

Una vez construido el CFG de los programas que queremos comparar, se recorren para comprobar si existe un isomorfismo entre ellos. Este isomorfismo no sigue el algoritmo general para comprobar isomorfismos de grafos porque nos encontramos ante un caso especial: en el grafo existe un orden (el orden secuencial en el que se ha

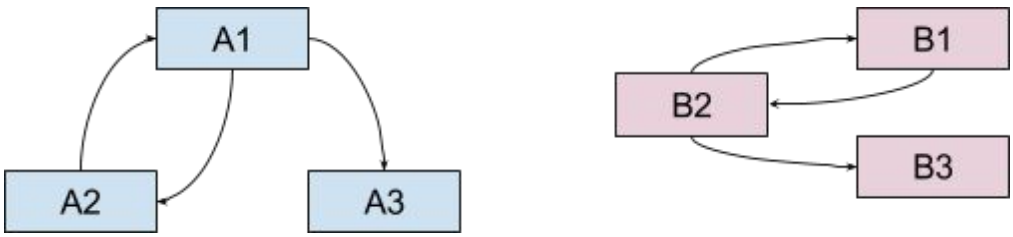
[1] http://cisne.sim.ucm.es/record=b3438656~S6*spi

programado) y por tanto solo unos pocos isomorfismos son legales. Teniendo esta restricción en cuenta, es posible realizar la comparación de manera eficiente.

6.2. Comparación mediante resolución de restricciones

Tras el proceso de construcción del CFG hemos obtenido una representación de cada programa a comparar como un conjunto de bloques. Un bloque es una secuencia de instrucciones básicas que siempre se ejecutan secuencialmente. Los bloques están unidos mediante aristas, cada una de ellas etiquetada con una expresión lógica que indica la condición bajo las cual la ejecución salta del bloque origen de la arista al bloque destino.

Supongamos que queremos comparar dos programas A y B, de los cuales hemos obtenido sus respectivos CFGs. Recordemos que, como se ha dicho en la sección anterior, la comparación consta de dos fases. En primer lugar ha de determinarse una correspondencia (*isomorfismo*) entre los bloques de ambos CFGs. Se trata de asociar cada bloque del programa A con un bloque del programa B, de modo que si dos bloques cualesquiera están conectados en el programa A, sus homólogos en el programa B también deben estar conectados. Por ejemplo, dada la siguiente representación de los bloques de los CFGs de dos programas:



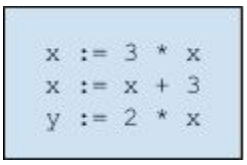
El bloque A1 del programa A se corresponde con el bloque B2 del programa B. El bloque A2 se corresponde con B1, mientras que A3 se corresponde con B3.

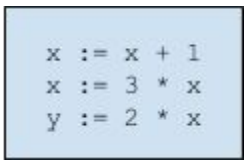
A continuación hemos de comprobar, en base a esta correspondencia, la equivalencia entre los programas. Las siguientes condiciones aseguran dicha equivalencia:

- Cada bloque del programa A es *equivalente* a su bloque asociado en el programa B.
- Si dos bloques en el programa A están conectados mediante una arista, también existirá una arista *equivalente* que conecte los bloques correspondientes del programa B.

Para comprobar estas condiciones debemos aclarar qué significa el hecho de que dos bloques sean equivalentes y que dos aristas sean equivalentes. Intuitivamente, dos bloques son equivalentes si producen el mismo efecto en el estado de las variables del programa. Por ejemplo, los siguientes bloques son equivalentes, ya que para

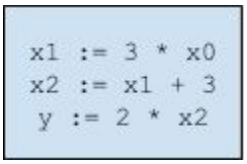
cualesquiera valores iniciales de *x* e *y*, los valores finales de ambas variables tras la ejecución de cada bloque son los mismos.

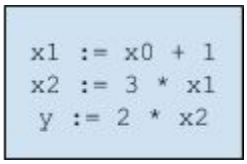




En particular, si el valor de inicial de *x* es 4, y el valor inicial de *y* es 3, tras la ejecución del primer bloque se tiene *x = 15, y = 30*, e igualmente tras la ejecución del segundo bloque. Esta igualdad de resultados ocurre para cualesquiera valores iniciales de *x* e *y*.

La comparación entre dos bloques resulta problemática cuando éstos contienen varias asignaciones de una misma variable. Así ocurre en los bloques del ejemplo anterior, en los que la variable *x* es asignada dos veces. Para evitar este tipo de asignaciones múltiples se realiza una transformación previa a llamada forma SSA (*Static Single Assignment*)[2]. En esta forma se hacen explícitas las distintas versiones de una misma variable a lo largo del bloque. En nuestro ejemplo, la transformación SSA daría lugar al siguiente resultado:





Para comprobar la equivalencia entre los dos bloques se genera una fórmula lógica a partir de las asignaciones de los bloques en forma SSA. La validez de esta fórmula lógica se comprueba posteriormente mediante la herramienta Z3[3].

La comprobación de equivalencia entre dos aristas se realiza de manera similar. Cada arista del CFG va asociada a una condición y, de nuevo, basta comprobar la equivalencia lógica de ambas condiciones. Por ejemplo, es lo mismo afirmar que una variable *x* tiene un valor entero comprendido entre 0 y 3 (ambos inclusive) que afirmar que la variable *x* tiene un valor entero que no es negativo ni mayor o igual que 4. Ambas condiciones son equivalentes porque para cualquier valor de *x*, o ambas condiciones son ciertas, o ambas son falsas, pero no existe ningún valor de *x* que satisfaga una de las condiciones y no satisfaga la otra. Al igual que en el caso de la equivalencia de bloques, podemos utilizar la herramienta Z3 para comprobar que dos condiciones son lógicamente equivalentes.

La implementación de la generación de CFG y la comparación de programas se ha realizado en el contexto de un Trabajo de Fin de Grado (*Comparador de Java y Python para DuoCode*) cuyos autores son David García Domínguez y Ángel Villalba Fernández-Paniagua. Actualmente esta implementación soporta los lenguajes Java y Python.

[2] https://en.wikipedia.org/wiki/Static_single_assignment_form
[3] https://github.com/Z3Prover/z3

6.3. Interfaz

Un aspecto muy importante del entorno de aprendizaje interactivo es su interfaz, ya que es la capa con la que interactúan los usuarios directamente. Esta interfaz debe contemplar una organización cliente-servidor: el servidor contiene toda la colección de programas y la lógica software destinada a comprobar que las respuestas son correctas, mientras que el cliente debe permitir interactuar con esta lógica. Además, el servidor debe almacenar toda la información necesaria sobre los usuarios y sus logros obtenidos.

De entre todas las posibilidades consideradas se ha decido implementar el servidor como un servicio RESTful. Esta tecnología nos permite crear un servidor muy versátil, ya que únicamente recibe y genera mensajes HTTP que transportan documentos JSON. De esta manera el mismo servidor puede dar soporte a múltiples front-ends tales como:

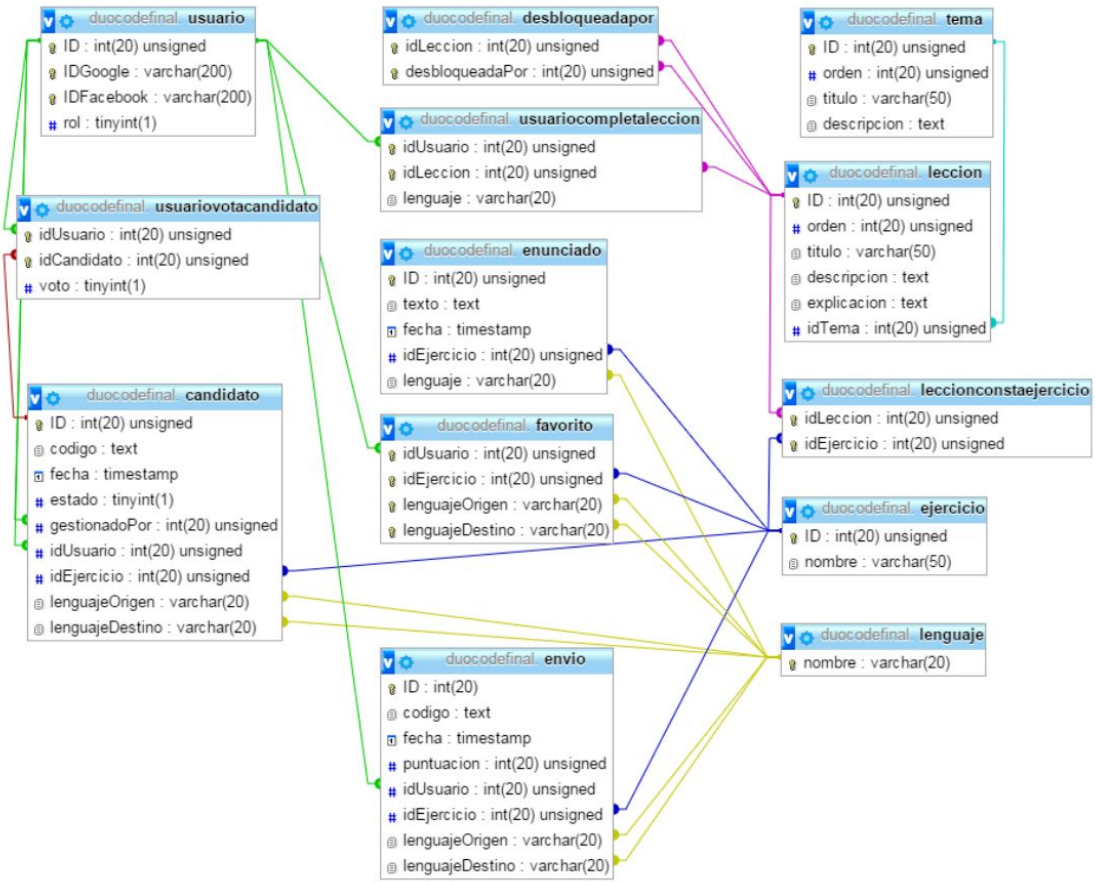
- Aplicaciones web que realizan peticiones asíncronas AJAX al servidor
- Aplicaciones móviles, ya sea mediante la inclusión de la aplicación web dentro de una *web views* o mediante aplicaciones nativas que realizan peticiones HTTP al servidor.
- Aplicaciones de escritorio usuales, por ejemplo realizadas en Java.

Para el desarrollo de la interfaz del entorno de aprendizaje interactivo contamos con la ayuda de los alumnos Julián F. Calleja da Silva, Johana Gabriela Ferreira Yagua y José Carlos Valera Villalba, que en su Trabajo de Fin de Grado titulado “Desarrollo de un front-end para DuoCode”[4] desarrollaron sus principales componentes. El código fuente de los distintos componentes mencionados en esta sección se puede encontrar en https://github.com/jucallej/DuoCode.

Servidor RESTful

El servidor RESTful encargado recibir peticiones HTTP, conectar con la lógica de la aplicación generar el resultado adecuado ha sido implementado en Java, concretamente utilizando el *framework* Jersey. Este *framework* permite un manejo sencillo de los distintos recursos URL permitidos y permite la inclusión de nuevas funcionalidades de manera sencilla. Para su correcto funcionamiento, este servidor RESTful necesita de un sistema de persistencia de datos. En este caso nos decantamos por una base de datos relacional alojada en un servidor MySQL debido a su estandarización y a la adecuación de este Sistema de Gestión de Bases de Datos a las características de eficiencia esperadas. El diseño concreto de la base de datos se realizó mediante refinamientos sucesivos en los que se iban eliminando duplicidades hasta llegar al diseño mostrado en la siguiente figura. Como se puede observar las entidades más importantes son las relacionadas con los usuarios y con los problemas (temas, lecciones, enunciados y candidatos).

[4] http://cisne.sim.ucm.es/record=b3438656~S6*spi



Interfaz web

La interfaz web es un conjunto de página HTML y *scripts* que permiten al usuario interactuar con el servidor RESTful directamente desde el navegador. Idealmente este conjunto de páginas web debe mostrar una alta cohesión y actuar como una sola unidad, de manera similar a una aplicación estándar de escritorio. Para conseguir este objetivo hemos utilizado el *framework* AngularJS, ya que nos permite realizar peticiones *asíncronas* al servidor de manera sencilla, por tanto permitiendo una utilización fluida de la interfaz. Además AngularJS nos permite la modificación dinámica de la página web para mostrar u ocultar fragmentos, moverse de una pantalla a otra, etc… Se ha utilizado también la librería *highlight.js* para el coloreado del código fuente de cara a mejorar la experiencia visual del alumno.

Finalmente la interfaz web ha considerado la integración con redes sociales como Google+ (totalmente compatible con las cuentas de la UCM) o Facebook utilizando para ello la librería HelloJS. De esta manera los usuarios de la herramienta de aprendizaje colaborativo no tendrían que crear una nueva cuenta en este sistema sino que podrían utilizar su cuenta en alguna red social. Esta característica nos proporciona una mejora adicional: publicar los logros en la mencionada red social. Aunque pueda parecer una característica menor, esta integración abre un abanico de posibilidades motivacionales y de ludificación que a nuestro juicio aumentará en gran medida el impacto del aprendizaje.

6.4. Desarrollo de la base de datos de teoría y problemas

Siguiendo la planificación desarrollada en el PIMCD 97/2014, dividimos los temas a desarrollar en el trabajo *Implementación de problemas para DuoCode*, elaborado por Laura Muñoz y dirigido por el profesor Adrián Riesco, en (1) Expresiones simples; (2) Listas y arrays; (3) Funciones y procedimientos; (4) Estructuras de control de flujo, con (4.1) Condicionales, (4.2) Bucles y (4.3) Bucles compuestos; (5) Recursión; (6) Clases; y (7) Métodos algorítmicos, con (7.1) Divide y vencerás y (7.2) Algoritmos voraces. Todos estos apartados se han desarrollado tanto para Java como para Python. En cada tema se expone en primer lugar la teoría que nos permitirá implementar posteriormente los ejercicios propuestos, ejemplificando cada punto con problemas sencillos que permitan al usuario ver las nociones teóricas en uso. Al final de cada tema se propone una serie de ejercicios (al menos 10 para cada uno de los puntos de los apartados (1.) - (6.) y 5 para métodos algorítmicos). Para estos ejercicios proporcionamos la solución correcta en Java y Python, lo que permite el uso del comparador descrito en las secciones 6.1. y 6.2. Además de la solución correcta hemos implementado tests de unidad para, en primer lugar, verificar en la medida de lo posible que las soluciones propuestas son correctas y, en segundo lugar, para facilitar futuras mejoras en el comparador mediante caja negra, ya que los casos de test implementados aquí servirían también de entrada para comparar la implementación del usuario contra la implementación correcta.

Esta base de conocimiento se introduce en la herramienta usando las precondiciones indicadas en el PIMCD 97/2014, suponiendo el punto final para la puesta en marcha de la herramienta.