UNIVERSIDAD
COMPLUTENSE
MADRID

Prog-Advisor

Trabajo de Fin de Grado – Facultad de Informátca – Universidad Complutense

8 de junio de 2018

—

Alumnos:
Aykaz Esayan
Sebastián Oleaga Quintas
Directores:
Jesús Correas Fernández (Departamento de Sistemas Informáticos y Computación - UCM)
Guillermo Román Díez (Departamento de Lenguajes y Sistemas Informáticos e Ingeniería de Software - UPM)

Índice

Resumen

En este proyecto se han tocado los dos extremos de la arquitectura cliente-servidor para llevar a cabo un sistema de subida y bajada de prácticas, incluyendo el análisis y corrección del código fuente subido.

Por una parte, se ha comprobado la ***capacidad de modificación*** que permite el entorno de desarrollo Eclipse para extender sus funcionalidades con un plugin.

Se ha entendido mejor el proceso de ***comunicación cliente-servidor*** que comunica una máquina local con una remota, que ya se había visto en distintos sistemas y asignaturas pero de forma más superficial.

Se ha utilizado el ***flujo básico*** en cualquier asignatura de programación, que es la subida y bajada de prácticas por parte de alumnos y profesores, y se ha comprobado que es posible su simplificación para ahorrar tiempo a ambas partes usando el mismo entorno de desarrollo.

Se ha comprobado que se puede ***reconfigurar*** el servidor para añadir, modificar o eliminar cambios en la salida del plugin sin necesidad de reinstalar una nueva versión o complementos adicionales, lo que facilita el añadir nuevas asignaturas, grupos o prácticas al sistema de forma rápida.

Se ha aprendido a usar comandos en ***lenguaje shell*** para la gestión de archivos, la descompresión de comprimidos, y la utilización de distintos parámetros para cambiar la utilización de comandos básicos.

En resumen, en este trabajo se han tocado numerosos lenguajes y protocolos, envolviendolo en una experiencia que ha permitido llevar a cabo el flujo deseado.

Palabras clave

Entorno de desarrollo integrado (IDE)
Eclipse
Enseñanza de la programación
Arquitectura cliente-servidor
Analizador de código fuente
Corrector de prácticas de programación

1. Introduction

This End-of-Degree Project will be developed to allow the delivery of programming practices directly from the development environment and its correction by the teacher.

1.1. Background

In this project we will study the extensibility of the Eclipse development environment through plugins

A development environment, or IDE (Integrated Development Environment)[1], is a computer application which purpose is to facilitate the creation of code to the developer by providing a series of comprehensive services and reducing the necessary configurations when developing, offering a service that encompasses them all.

An IDE is made up of a source code editor, a compiler, a debugger and automatic generation and auto-completion tools for source code, among others.

In addition to having what is mentioned above, it also includes a series of additional tools to extend its functionality. In Eclipse, this is done through plugins.

Plugins are code packages that interact with the IDE extending its functionalities [2].

Being independent programs from the development environment, the user will have to install them in the IDE to be able to use them.

In this End-of-Degree Project, the implementation of the EasyInterface system has been used in order to develop the new system.

The goal of EasyInterface is to create a communication channel through a plugin in order to use external tools, such as PMD (a Java source code analyzer), CheckStyle and FindBugs, which are installed on external servers on the Internet.

Using the EasyInterface architecture, which in turn is based on the European project Envisage [3], the possibility of guiding the development environment to remote services through a server was considered. These services are provided by resident applications on the server that process the source code sent from the development environment.

This approach provides advantages over the IDE's traditional approach:

- The applications for the treatment of source code found in the server do not need a specific interface, but they can be controlled by parameters and the results that they provide are directed to the standard exit of the application.
- The development environment only requires one component, which would be the Prog-Advisor plugin that manages all the results from the applications installed on the server.
- No additional installation is needed on the client's computer, except for the plugin. Server updates are transparent to installations on the computer where Prog-Advisor is installed.

1.2 Objectives

With this inherited system, we want to study the client-server architecture. We want to implement the Prog-Advisor plugin starting with the communication with the server based on requests with JSON schema and XML responses. The answers include actions and commands that the plugin must interpret to provide interactivity with the server.

This architecture has been used to carry out the objective of this End-of-Degree Project: a practice correction system integrated in the Eclipse development environment. This system has been based

on the development of two new servers, with two new functionalities in each one, which allow the desired upload-download flow.

We also want to modify the inherited plugin to adapt it to this new functionality, changing its source code and preparing it for the main advantages that the Prog-Advisor system will offer:

- **Simple and safe upload:** the students will save time by omitting the need to have to identify themselves in a virtual campus or online repository to upload a practice. In addition, it will provide security since the plugin will have an authentication system installed in the server, in which there will be the possibilities of executing the compilation to detect compilation errors, automatic correction programs, test execution and program analysis to suggest good programming practices.
- **Simple correction:** in the case of the teachers, it will save them the downloading time of numerous student practices one by one and its subsequent saving and opening in Eclipse. It will also help to organize different downloads in different folders to keep a record and a progress of each student. The upload of the corrected practices will also be simplified since the plugin will offer the option of selecting a folder and uploading all its content to the server for the future download of the students. The teacher can also add comments, associated JUnit tests, or a simple text file with comments, which will be very useful for students to help in the progress of the practice.
- **Simple download**: the student will only have to select an option in the plugin to be able to download his correction and have it created as a new project in his Eclipse.

2. Introducción

Este TFG se va a realizar para permitir la entrega de prácticas de programación directamente desde el entorno de desarrollo y su corrección por parte del profesor.

2.1. Antecedentes

En este trabajo se va a estudiar la extensibilidad del entorno de desarrollo Eclipse mediante plugins.

Un entorno de desarrollo, o IDE (Integrated Development Environment)[1], es una aplicación informática cuya finalidad es facilitar al desarrollador la creación de código proporcionando una serie de servicios integrales y reduciendo las configuraciones necesarias a la hora de desarrollar, ofreciendo un servicio que engloba todas ellas.

Un IDE está formado por un editor de código fuente, un compilador, un depurador y herramientas automáticas de generación y auto-completado de código fuente, entre otras.

Además de tener todo lo mencionado anteriormente, también incluye una serie de herramientas adicionales para extender la funcionalidad de la misma. En Eclipse, esto se lleva a cabo mediante plugins.

Los **plugins** [2] son paquetes de código que interactúan con el IDE extendiendo las funcionalidades del mismo (modificando el coloreado y el aspecto del código fuente del editor, por ejemplo).

Al ser programas independientes al entorno de desarrollo, será el usuario el que tendrá que instalarlos en el IDE para poder hacer uso de ellos.

En este Trabajo de Fin de Grado se ha partido de la implementación del sistema de **EasyInterface** para llevar a cabo el desarrollo del nuevo sistema.

El objetivo de **EasyInterface** es crear un canal de comunicación mediante un plugin para el uso de herramientas externas, como PMD (un analizador de código fuente de Java), CheckStyle y FindBugs, instalados en servidores externos en Internet.

Utilizando la arquitectura de **EasyInterface**, que a su vez se basa en el proyecto europeo **Envisage** [3], se ha visto la posibilidad de orientar el entorno de desarrollo a servicios remotos mediante un servidor. Estos servicios los proporcionan aplicaciones residentes en el servidor que procesan el código fuente enviado desde el entorno de desarrollo.

Este enfoque proporciona ventajas frente al enfoque tradicional de los IDEs:

- Las aplicaciones de tratamiento de código fuente halladas en el servidor no necesitan una interfaz específica, si no que se pueden controlar mediante parámetros y los resultados que proporcionan son dirigidos a la salida estándar de la aplicación.
- El entorno de desarrollo sólo requiere un componente, que sería el plugin **Prog-Advisor** que gestiona todos los resultados de las aplicaciones instaladas en el servidor.
- No se necesita ninguna instalación adicional en el ordenador del cliente, exceptuando el plugin. Las actualizaciones del servidor son transparentes a las instalaciones en el ordenador donde esté instalado **Prog-Advisor**.

2.2. Objetivos

Con este sistema heredado, se quiere estudiar la arquitectura cliente-servidor. Se quiere implementar el plugin **Prog-Advisor** partiendo de la comunicación con el servidor basada en solicitudes con esquema JSON y respuestas XML. Las respuestas incluyen acciones y comandos que debe interpretar el plugin para proporcionar interactividad con el servidor.

Se ha aprovechado esta arquitectura para llevar a cabo el objetivo de este TFG: *un sistema de corrección de prácticas integrado en el entorno de desarrollo Eclipse*. Este sistema se ha basado en el desarrollo de dos servidores nuevos, con dos funcionalidades también nuevas en cada uno, que permiten el flujo de subida-bajada deseado.

También se quiere modificar el plugin heredado para que se adapte a esta nueva funcionalidad, cambiando su código fuente y preparándolo para las principales ventajas que ofrecerá el sistema **Prog-Advisor**:

- **Subida simple y segura**: ahorrará tiempo al omitir la necesidad de que un alumno tenga que identificarse en un campus virtual o repositorio online para realizar la subida de una práctica. Además, proporcionará seguridad ya que el plugin tendrá un sistema de autenticación instalado en el servidor, en el cual existirá la posibilidad
de ejecutar la compilación para detectar errores de compilación, programas de corrección automática, ejecución de tests y análisis de programas para sugerir buenas prácticas de programación.
- **Corrección simple:** en el caso del profesor, le ahorrará el tiempo de bajada de numerosas prácticas de alumnos una por una y su posterior guardado y apertura en Eclipse. También ayudará a organizar distintas bajadas en distintas carpetas para mantener un registro y progreso de cada alumno. La subida de las prácticas corregidas también se simplificará ya que el plugin ofrecerá la opción de seleccionar una carpeta y subir todo su contenido al servidor para la futura bajada de los alumnos. El profesor podrá además añadir comentarios, pruebas de JUnit asociadas, o un simple fichero de texto con comentarios, lo cual resultará muy útil a los alumnos para ayudar en el avance de la práctica.
- **Bajada simple:** el alumno solo tendrá que seleccionar la opción de descargar su corrección en el plugin para que se le cree un nuevo proyecto en su Eclipse con la práctica que subió, corregida.

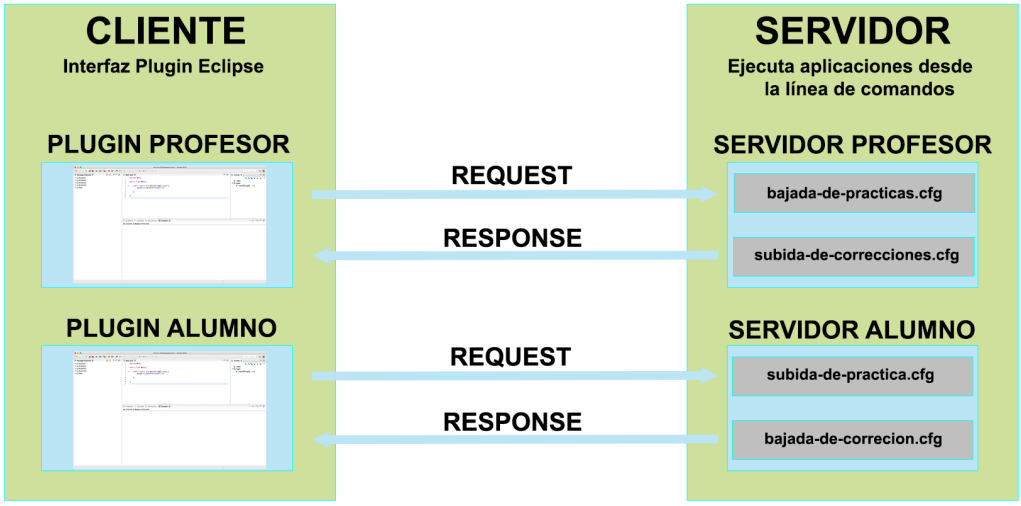
3. Arquitectura

La **arquitectura cliente-servidor** consiste en repartir las tareas entre los proveedores de recursos (servidores) y los demandantes (clientes). Los clientes van a realizar una petición de un recurso al servidor y se van a quedar esperando la respuesta de este. El servidor aceptará la petición y devolverá el resultado al cliente.

Las principales ventajas de esta arquitectura son:

- El servidor va a controlar los accesos a los datos, evitando así el acceso de un cliente defectuoso o no autorizado, el cual puede llegar a dañar el sistema.
- Se pueden añadir nuevos clientes y servidores sin complicación alguna y llevar a cabo la mejora de los ya existentes por separado y en cualquier momento.
- El fácil mantenimiento es otra de las ventajas principales de esta arquitectura, pues al tener distribuidos las funcionalidades en varias máquinas independientes el uno del otro, será posible, por ejemplo, reemplazar, actualizar o reparar el servidor sin que los clientes se vean afectados.

En la siguiente figura se muestra la arquitectura cliente-servidor del sistema Prog-Advisor.



3.1. Servidor

El servidor, basado en apache, estará instalado en una máquina (Mac o Linux) junto con varias aplicaciones.

El servidor contendrá aplicaciones que se ejecutarán cuando el usuario lo solicite desde el plugin de Eclipse. Una aplicación será un programa que se ejecutará en el servidor utilizando un conjunto de parámetros configurables en el plugin de Eclipse, así como el código fuente del proyecto de Eclipse enviado por el plugin. En este proyecto, la estructura de los parámetros de configuración estará definida en un archivo de configuración (.cfg) y tendrá asociado un Script de Shell de Linux que lo ejecutará.

En el archivo de configuración se especificarán los parámetros que necesitarán cada una de las aplicaciones para llevar a cabo su ejecución y el cómo deben ejecutarse. Los ficheros de configuración de las aplicaciones del servidor van a almacenar la información en formato XML.

Las aplicaciones del servidor se ejecutarán desde la línea de comandos correspondiente a cada una de ellas y especificada en los ficheros de configuración bajo la etiqueta XML **<cmdlineapp> </cmdlineapp>**. El contenido de esta etiqueta XML corresponderá al Script de Shell de Linux a ejecutar.

En la imágen que viene a continuación, se puede observar un fichero de configuración básico dentro del servidor:

```
<app visible="true">
    <appinfo uploadtype="active">
        <acronym>Prog-Advisor Alumno Subida</acronym>
        <title>Prog-Advisor Alumno Subida</title>

        <desc>
            <short>Comprobador de codigo fuente</short>
            <long>Comprobador de codigo fuente</long>
        </desc>
    </appinfo>

    <apphelp>
        <content format='html'>
            Esto es la ayuda del comprobador de codigo fuente.
        </content>
    </apphelp>

    <execinfo method="cmdline">
        <cmdlineapp>
            ./run_padvisor_alumno_subida.sh -f _ei_files _ei_parameters
        </cmdlineapp>
    </execinfo>

    <parameters prefix = "-" check="false">
        <textfield name="dni">
            <desc>
                <short>DNI/NIE</short>
                <long>Indique su DNI/NIE</long>
            </desc>
        </textfield>
    </parameters>
</app>
```

El significado de cada una de las etiquetas XML que compone un fichero de configuración se explicará en detalle en la sección 4.3 de esta memoria.

Se va a disponer de dos servidores: el servidor del alumno y el servidor del profesor. Ambos van a tener un fichero de configuración propio de cada uno (**apps.cfg**) en el cual aparecerá, dentro de la etiqueta XML <**apps**> </**apps**>, la información sobre sus aplicaciones.

- SERVIDOR ALUMNO

  En el servidor del alumno estarán alojadas las aplicaciones propias del alumno (subida-de-practica.cfg y bajada-de-correccion.cfg) y sus correspondientes líneas de comando (run_subida_de_practica.sh y run_bajada_de_correccion.sh) para poder ejecutarlas.

- SERVIDOR PROFESOR

  Al igual que en el servidor del alumno, en el del profesor estarán alojadas las sus propias aplicaciones (bajada-de-practicas.cfg y subida-de-correcciones.cfg) y sus correspondientes líneas de comando (run_bajada_de_practicas.sh y run_subida_de_correcciones.sh) para poder ejecutarlas.

Ambos van a estar instalados en una misma máquina en la cual compartirán directorios. Van a disponer de URLs distintas, con el fin de poder proporcionar a cada uno sus propias aplicaciones. Cada plugin tendrá configurado por defecto la URL del servidor.

A su vez, en el servidor existirán dos subdirectorios en los cuales se alojarán prácticas de alumnos: Prog-Advisor-Data y Prog-Advisor-Data-Correcciones. Estos van a ser accesibles tanto por el servidor del profesor como el del alumno.

- **Prog-Advisor-Data**:

  El alumno subirá sus prácticas a este subdirectorio del servidor. El profesor podrá acceder a él para descargarse las prácticas y corregirlas.

- **Prog-Advisor-Data-Correcciones**:

  Una vez que el profesor haya concluido con la corrección de las prácticas, los subirá a este subdirectorio para que los alumnos, accediendo a él, puedan descargarlos.

En las aplicaciones de subida de prácticas de alumno y profesor se ha incluido, como ejemplo de uso de herramientas externas, una herramienta de análisis de programas llamado **PMD** [4]. Este es un analizador de código fuente de java que incluye una serie de reglas en base de las cuales informa sobre malos hábitos de programación. PMD no informa errores de compilación, ya que solo puede procesar código fuente bien formado.

3.2. Cliente

Los tipos de clientes de nuestra aplicación van a ser dos: alumno y profesor. El alumno tendrá su servidor, con sus propias aplicaciones alojadas en él (inaccesibles para el profesor) y viceversa. El plugin es idéntico para ambos y la única diferencia es el servidor al que se conectan. El cliente tendrá el plugin instalado en su Eclipse.

El cliente va a disponer de distintas aplicaciones mediante los cuales va a poder realizar las siguientes operaciones.

- Solicitar una lista de aplicaciones disponibles para el tipo de cliente en concreto.
- Una vez recibida la lista de aplicaciones, el cliente va a ser capaz de seleccionar una aplicación de su lista de aplicaciones y configurar sus parámetros de configuración.
- Generar la petición de ejecución y enviarla al servidor.
- Mostrar la respuesta del servidor en un entorno de usuario (consola de Eclipse).

El profesor será capaz de acceder a la máquina donde está instalado el servidor y modificar los parámetros de configuración de las aplicaciones. De este modo, en las aplicaciones del plugin del alumno se van a mostrar las opciones que configure el profesor.

4. Estructura cliente-servidor

La formación de la solicitud, el lenguaje de salida y los formatos de respuesta a las solicitudes proceden de la implementación existente en **EasyInterface** [5], extendidos para adaptarse a los requisitos de este Trabajo de Fin de Grado, como se detalla en los siguientes apartados.

4.1. Formación de la solicitud

La comunicación con el servidor se va a llevar a cabo mediante solicitudes HTTP de tipo POST. Los datos que se envían estarán incluidos en la petición. La solicitud va a incluir un atributo llamado “**eirequest”**, cuyo valor será un JSON que variará según el tipo de solicitud.

Existen dos formatos de solicitud dependiendo de si se quiere obtener información sobre una aplicación o si lo que se quiere es ejecutar una aplicación determinada.

4.1.1. Solicitud de información sobre las aplicaciones disponibles

Se utilizará para recuperar información sobre una aplicación determinada o sobre todas las disponibles disponibles. Para ello se hará uso de la siguiente solicitud:

```
{ "command" : "app_details",  "app_id":"_ei_all" }
```

Los atributos principales de esta solicitud son:

- **command:** Para poder consultar la información sobre una o todas las aplicaciones, el comando a ejecutar será **“app-details”**.
- **app-id:** El valor de este atributo variará en función de si queremos obtener información sobre todas las aplicaciones o de una aplicación determinada. Para el primer caso su valor será **“_ei_all”**, y para el segundo, el identificador de la aplicación determinada.

Por último, el valor de **“eirequest”** será el JSON de la solicitud anterior.

```
"eirequest="+JSON.stringify( {"command": "app_details", "app_id": "_ei_all"} )
```

4.1.2. Solicitud de ejecución de una aplicación disponible en el servidor

Dicha solicitud se utilizará para ejecutar una aplicación determinada disponible en el servidor. La estructura de la solicitud será similar a la vista anteriormente, solo que en este caso tendremos que configurar los parámetros de la aplicación. Supongamos que queremos ejecutar la aplicación cuyo identificador es **“Subida de práctica”**. La petición tendrá el siguiente aspecto:

```
{
        "command": "execute",
        "app_id": "Subida de práctica",
        "parameters": {
                "asignatura": ["PII"],
                "password": ["1234"],
                "_ei_clientid": "web_client",
                "grupo": ["A"],
                "practica": ["Prac1"],
                "dni": ["Alumno1"],
                "_ei_files":
                [
                        {
                                "id": "10",
                                "name": "Prueba.txt",
                                "type": "text",
                                "content": "Base64 de la práctica del alumno"
                        },
                ]
        }
}
```

Los atributos principales de esta solicitud son:

- **command:** Para poder ejecutar una aplicación determinada, se hará uso de comando **“execute”**.
- **app-id:** Identificador de la aplicación a ejecutar.
- **parameters:** Será un JSON de los parámetros de ejecución de la aplicación.
    - Como podemos observar, tenemos la **asignatura**, la **contraseña**, el **grupo**, la **práctica** y el **dni**, que son parámetros que el cliente rellena al hacer uso de la aplicación.
    - **_ei_clientid**: identificador del cliente que hace la petición al servidor.
    - **_ei_outline**: va a contener los nombres de los métodos que el servidor va a tratar.
    - **_ei_files**: va a contener información sobre el envío realizado por el alumno. El contenido se codificará en base64 y se almacenará en practica.txt.

Esta solicitud se almacenará en formato JSON en el atributo **“eirequest”**.

4.2. Lenguaje de salida de Prog-Advisor

Es el lenguaje que van a utilizar las aplicaciones del servidor para mostrar los resultados en un interfaz gráfico de usuario (Eclipse).

La respuesta del servidor consistirá en un documento XML que tiene el siguiente aspecto:

```
<ei_response>
    <ei_server_output></ei_server_output>
    <ei_output></ei_output>
    <ei_error></ei_error>
</ei_response>
```

Donde:

- **ei_server_output**: contendrá mensajes de depuración escritos por el servidor.
- **ei_output**: contendrá la respuesta a la petición llevada a cabo al servidor.
- **ei_error:** contendrá mensajes de error relacionados a la solicitud.

En los siguientes apartados se mostrarán los formatos de respuesta de **EasyInterface** que serán utilizados por **Prog-Advisor**. Estos formatos estarán contenidos dentro de la etiqueta XML **<ei_output></ei_output>**.

4.3. Formato de respuesta de la solicitud de información de las aplicaciones disponibles

El formato de respuesta correspondiente a la solicitud de información de las aplicaciones disponibles en el servidor contendrá la información de estas aplicaciones dentro de la etiqueta XML **<apps></apps>**. Dentro de ella, se utilizarán fundamentalmente las siguientes etiquetas:

- APPS

  Esta etiqueta XML estará definida dentro del fichero de configuración **apps.cfg**, en el cual se van a declarar una lista de aplicaciones disponibles en el servidor. Al tener dos servidores, cada uno de ellos tendrá su propio fichero de configuración **apps.cfg** con la correspondiente declaración de las aplicaciones. En el interior de esta etiqueta XML nos encontraremos con una etiqueta *app* con el identificador de la aplicación y *src* que indicará la ruta de la misma.

Etiqueta apps para el alumno:

```
<apps>
 <app id="Subida de práctica"              src="./subida-de-practica.cfg" />
 <app id="Bajada de corrección"            src="./bajada-de-correccion.cfg" />
</apps>
```

Etiqueta apps para el profesor:

```
<apps>
 <app id="Subida de correcciones"          src="./subida-de-correcciones.cfg" />
 <app id="Bajada de prácticas"             src="./bajada-de-practicas.cfg" />
</apps>
```

Las etiquetas que vienen a continuación se encuentran definidas en los archivos de configuración de cada aplicación (.cfg).

- APP

  Está etiqueta XML se utilizará para definir una aplicación determinada. En su interior se definirán el resto de etiquetas XML relacionadas con la aplicación.

Consta de un atributo booleano que indicará si la aplicación es visible o no.

```
<app visible="true">
    <appinfo>…</appinfo>

    <apphelp>…</apphelp>

    <execinfo>…</execinfo>

    <parameters>…</parameters>
</app>
```

- APPINFO

  Esta etiqueta contendrá la información sobre la aplicación determinada. Sus atributos a destacar son: <**acronym**>, <**title**> y <**desc**>.

```
<appinfo uploadtype="…">
    <acronym>…</acronym>

    <title>…</title>

    <desc>
        <short>…</short>

        <long>…</long>
    </desc>
</appinfo>
```

- APPHELP

  Esta etiqueta XML va a contener información de ayuda sobre una aplicación. Cada una tendrá su ayuda asociada, el cual se podrá consultar accediendo a la aplicación desde la interfaz del usuario. Por ejemplo, para la subida de la práctica del alumno, esta etiqueta contendrá el siguiente texto: "*Esta aplicación te permite subir tu práctica al servidor con los parámetros insertados, permitiendo que el profesor se la descargue y pueda subir una futura corrección*".

```
<apphelp>
    <content format='html'>
        …
    </content>
</apphelp>
```

- EXECINFO

  El contenido de esta etiqueta será la correspondiente línea de comando para ejecutar la aplicación asociada.

```
<execinfo method="cmdline">
    <cmdlineapp>…</cmdlineapp>
</execinfo>
```

- PARAMETERS

  Esta etiqueta XML definirá los parámetros que debe recibir la aplicación cuando se ejecute. Estos se le piden al usuario cuando va a solicitar la ejecución de una aplicación. De entre todos los parámetros disponibles, caben destacar los siguientes:

  - **TEXTFIELD:** Esta etiqueta definirá un parámetro cuyo valor será un texto introducido por el usuario.

```
<textfield name="">
    <desc>
        <short>…</short>
        <long>… </long>
    </desc>
</textfield>
```

  - **SELECTONE:** Esta etiqueta definirá listas desplegables. Su valor será aque que se haya elegido de un conjunto de valores disponibles.

```
<selectone name="" widget="">
    <default>…</default>
    <desc>
        <short>…</short>
        <long>…</long>
    </desc>

    <option value="">
        <desc>
            <short>…</short>
            <long>…</long>
        </desc>
    </option>

    <option value="">
        <desc>
            <short>…</short>
            <long>…</long>
        </desc>
    </option>
</selectone>
```

  - **PASSWORDFIELD:** Esta etiqueta definirá un nuevo parámetro que extenderá el formato de respuesta de **EasyInterface** añadiendo una nueva funcionalidad en el sistema Prog-Advisor. Su valor será una contraseña introducida por el alumno o el profesor. Este parámetro servirá para la autenticación del usuario que vaya a utilizar la aplicación. Tanto

las aplicaciones del profesor como las del alumno tendrán este campo y será obligatorio para poder hacer uso de la aplicación y realizar envíos o descargas de prácticas en el servidor. La contraseña introducida por el usuario aparecerá cifrada con asteriscos.

```
<passwordfield name="">
    <desc>
        <short>…</short>
        <long>…</long>
    </desc>
</passwordfield>
```

4.4. Formato de respuesta a la solicitud de ejecución de una aplicación disponible en el servidor

La respuesta del servidor se encuentra dentro de la etiqueta **eiout**, que está contenida en la etiqueta **ei_output** y este dentro de la etiqueta **ei_response**. La estructura de este conjunto de etiquetas será la siguiente:

```
<ei_response>
    <ei_output>
        <eiout>
            <eicommands>
                <printonconsole>
                    <content format="text">
                            ================================================================
                            Prog-Advisor: Repositorio de prácticas
                            DNI: Alumno10 ASIGNATURA: TP GRUPO: A Practica: Pr1
                            Proyecto: Test
                            ================================================================
                    </content>
                </printonconsole>
            </eicommands>
        </eiout>
    </ei_output>
</ei_response>
```

Esta etiqueta contendrá una lista de comandos **eicommands** ejecutados en el procesamiento de la petición y una serie de acciones **eiactions**. Primero se ejecutarán los comandos y a continuación las acciones.

En esta sección solo se van a explicar los **eicommands** utilizados para el plugin Prog-Advisor. No se va a ha hacer uso de los **eiactions.**

EICOMMANDS

De toda la lista de comandos existente, sólo se hará uso del comando **PrintOnConsoleCommand**. A esta lista de comandos se le añadirá un comando nuevo llamado **GetProjectCommand.**

- PrintOnConsoleCommand

  Este comando se utilizará para escribir texto en la consola de Eclipse del cliente. El texto que se escribirá en la consola estará dentro de la etiqueta **content**.

```
<eiout>
    <eicommands>
        <printonconsole>
            <content>...</content>
        </printonconsole>
    </eicommands>
</eiout>
```

- GetProjectCommand

  Este comando ha sido creado para el plugin Prog-Advisor con la finalidad de poder obtener una ruta local y su contenido comprimido y codificado en base64 en el Eclipse de desarrollo.

  La etiqueta XML asociada este comando será <**getproject**></**getproject**>, cuyos atributos serán: **path** y **content**. En la primera se guardará la ruta introducida por el profesor o el alumno, y en la segunda un contenido comprimido y codificado en Base64. Este contenido corresponderá a un conjunto de prácticas que se quiere descargar del servidor.

```
<eiout>
    <eicommands>
        <getproject path="">

        </getproject>
    </eicommands>
</eiout>
```

5. Implementación del plugin de Prog-Advisor

El objetivo de este Trabajo de Fin de Grado es el desarrollo e implementación del plugin que actúa como repositorio de subidas y correcciones **Prog-Advisor**, el cual tendrá dos versiones implementadas y con diferentes parámetros de entrada dependiendo del cliente, una para alumnos y otra para profesores. En primer lugar se presentan los distintos casos de uso en el sistema **Prog-Advisor** heredados del plugin **EasyInterface**, mencionado algunas de las modificaciones que se han realizado.

En la siguiente sección se describirán los casos de uso de las nuevas aplicaciones creadas e implementadas en el código fuente del plugin.

5.1. Casos de uso heredados de **EasyInterface**

En el sistema **EasyInterface** se identificaban los siguientes casos de uso [5], los explicaremos con ejemplos del plugin desarrollado en este TFG:

- **Establecer parámetros:**

  Este caso de uso se usaba en el plugin heredado **EasyInterface** para poder realizar cambios de valores de los parámetros desde la página de preferencias del plugin para las distintas

aplicaciones disponibles en el servidor remoto. La lista de aplicaciones viene dada por dicho servidor, por lo que el plugin que se esté usando (alumno o profesor), deberá obtener del servidor la lista de aplicaciones y sus parámetros de configuración para poder tratarlos de forma dinámica.

- **Solicitar la ejecución de la aplicación desde la barra de menú:**

  El usuario puede pulsar el botón correspondiente al plugin que esté utilizando. Esta opción muestra una ventana de diálogo con las aplicaciones disponibles en el servidor para subir o bajar código fuente al servidor. Desde esta ventana se pueden modificar los parámetros de las aplicaciones ofrecidas por el servidor y solicitar su ejecución.

- **Solicitar la ejecución de la aplicación desde la barra de herramientas:**

  Parecido al caso de uso anterior, pero en este caso la acción se solicita desde la barra de herramientas de Eclipse.

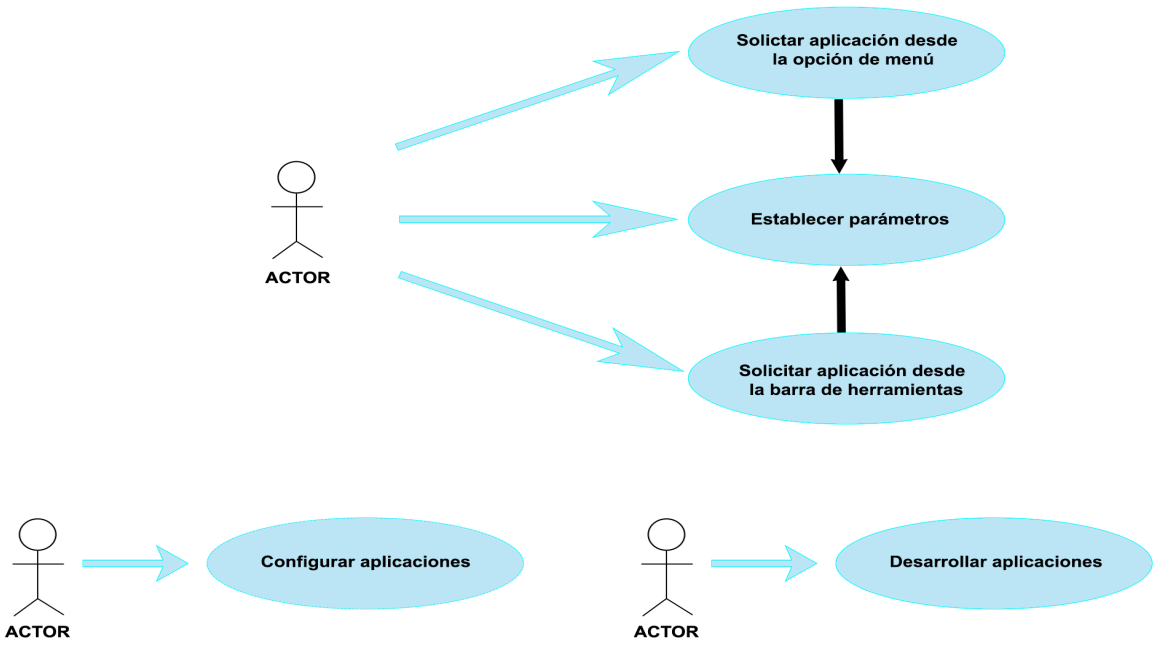
- **Desarrollar Aplicaciones:**

  Como en el caso anterior, este caso también se desarrollará íntegro en el servidor, y consistirá en el desarrollo de aplicaciones que el usuario deseé que estén disponibles en el servidor de **Prog-Advisor**.

- **Configurar Aplicaciones:**

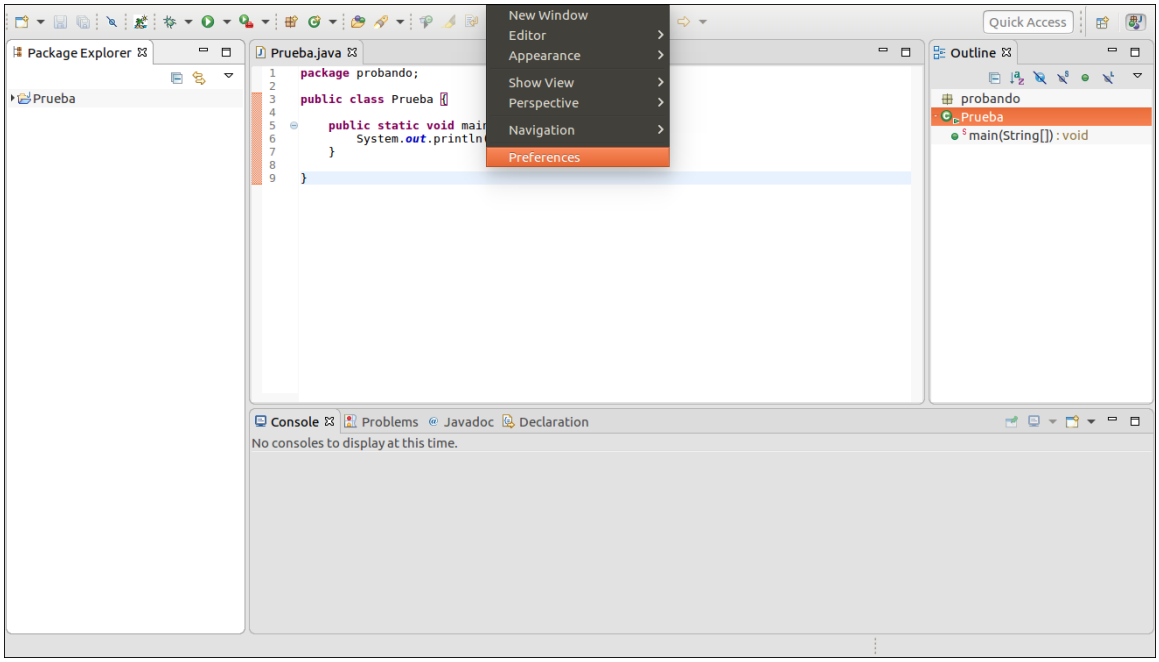
  El usuario de este caso de uso podrá configurar el servidor de **Prog-Advisor** para ejecutar las aplicaciones que estén instaladas en el servidor. Este caso se realiza íntegramente en el servidor.

A continuación se muestra el diagrama UML de los casos de uso del sistema **Prog-Advisor**:

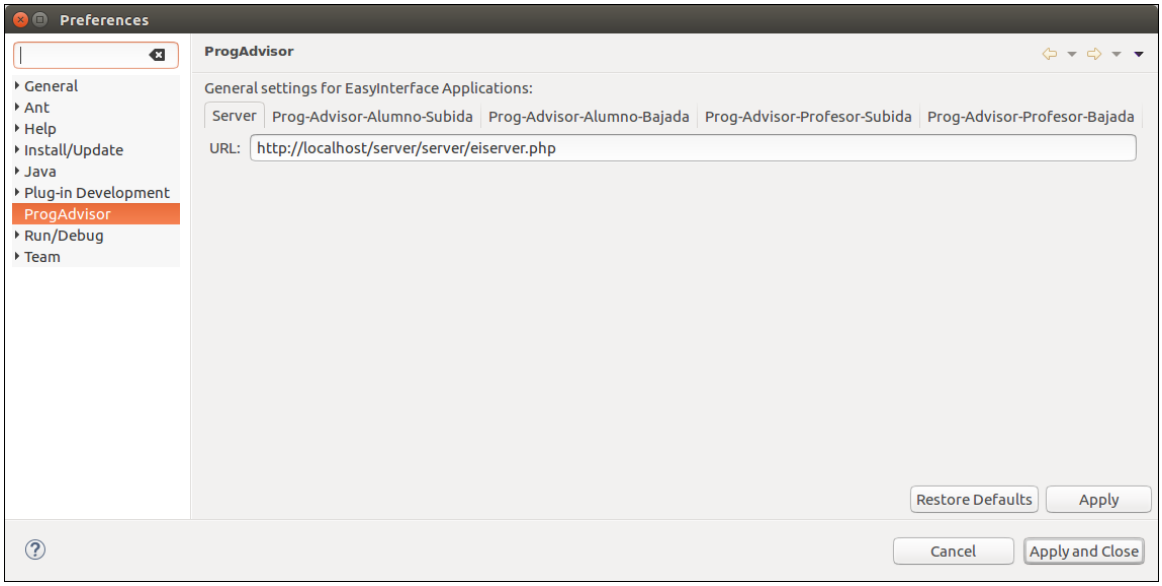


5.1.1. Caso de uso: Establecer Parámetros

Las preferencias de todos los componentes de Eclipse se encontrarán en la opción de menú **Window > Preferences**, como se muestra en la siguiente captura:



Una vez pulsada esta opción , Eclipse mostrará una ventana con las preferencias de todos los componentes y plugins que estén instalados en Eclipse. Entre ellos se encontrará la página de preferencias del plugin **Prog-Advisor**, como se muestra a continuación:



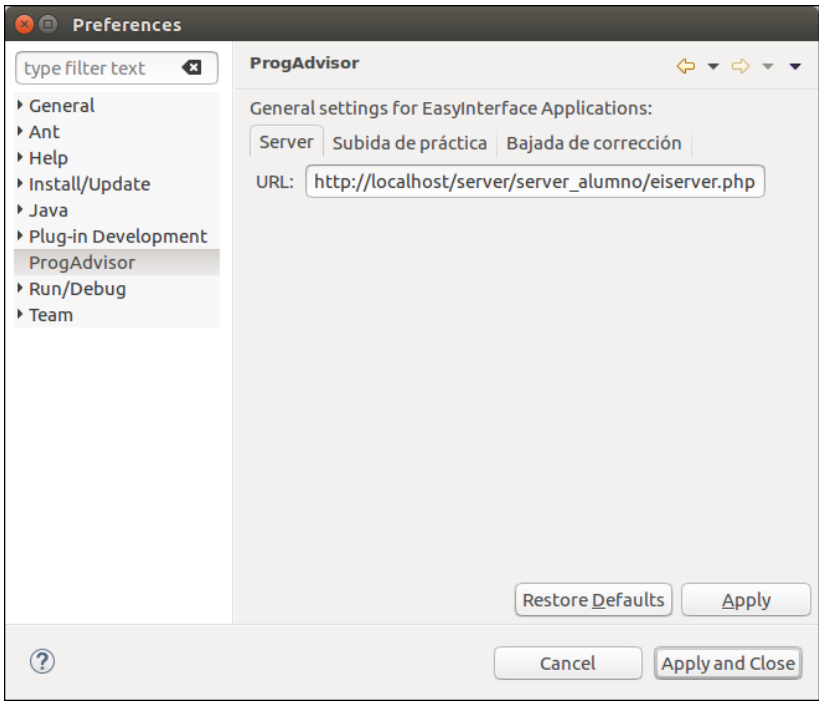
Lo primero que se verá en la página de preferencias de **Prog-Advisor** será el campo de entrada de la URL del servidor. En el plugin heredado **EasyInterface**, este campo se puede cambiar dependiendo del servidor al que el usuario quisiera conectarse y del que recibir diferentes aplicaciones. En el caso de nuestro plugin, este campo no ha de tocarse ya que estará apuntando al servidor de subida y bajada de prácticas correspondiente al usuario alumno o profesor que hemos desarrollado en este Trabajo de Fin de Grado.

Con la URL introducida, la respuesta proporcionada por el servidor generará dinámicamente las páginas de preferencias de cada una de las aplicaciones disponibles en ese servidor. Las aplicaciones se mostrarán en distintas pestañas en la misma página de preferencias del plugin **Prog-Advisor**.

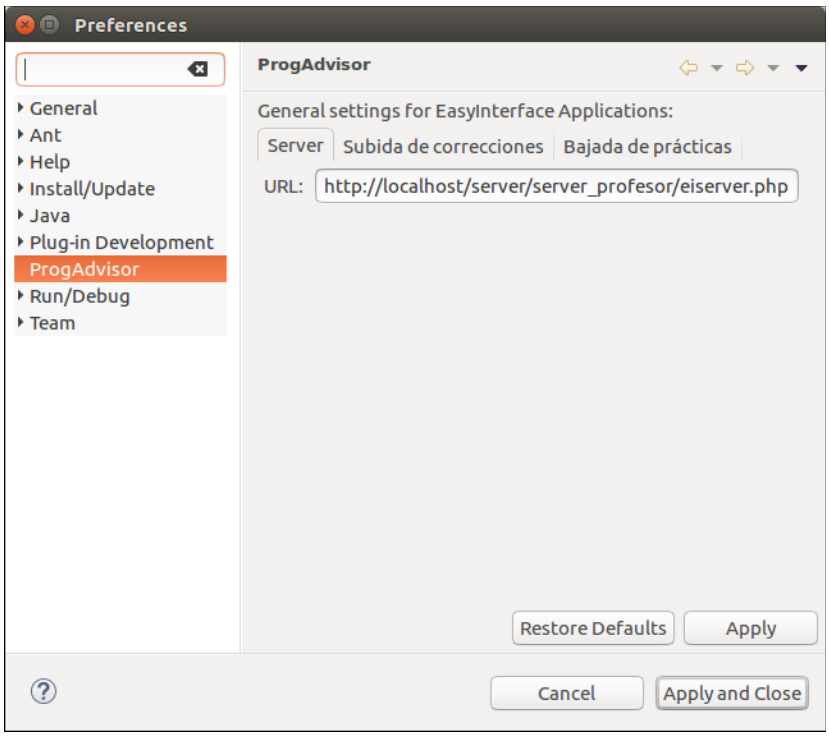
De este modo, las páginas de preferencias de cada una de las aplicaciones serán completamente parametrizables en el servidor sin necesidad de modificar la instalación del plugin. En en caso de nuestro trabajo, estas aplicaciones estarán fijadas, aun que podrán ser modificadas en versiones futuras del plugin desde el servidor por otro actor que no corresponde a ninguno de los dos mencionados anteriormente. Lo único que estará programado directamente en el plugin de **Prog-Advisor** será el campo de entrada para establecer la URL del servidor en la página de preferencias inicial mostrada anteriormente.

Una vez cargadas todas las aplicaciones del servidor seleccionado, el usuario dispondrá de las páginas de preferencias que corresponden a cada una de ellas, en las que podrá modificar los parámetros con los valores por defecto asignados a cada uno de ellos.

En la siguiente imagen se muestra el aspecto que tendrá la página de preferencias de **Prog-Advisor** con la información que ha devuelto el servidor. En este caso tenemos las aplicaciones de la versión del plugin que utilizará el alumno. En la imagen se aprecian las pestañas que corresponden a las dos aplicaciones que tendrá el servidor de esta versión del plugin: **Subida de práctica, Bajada de corrección**.

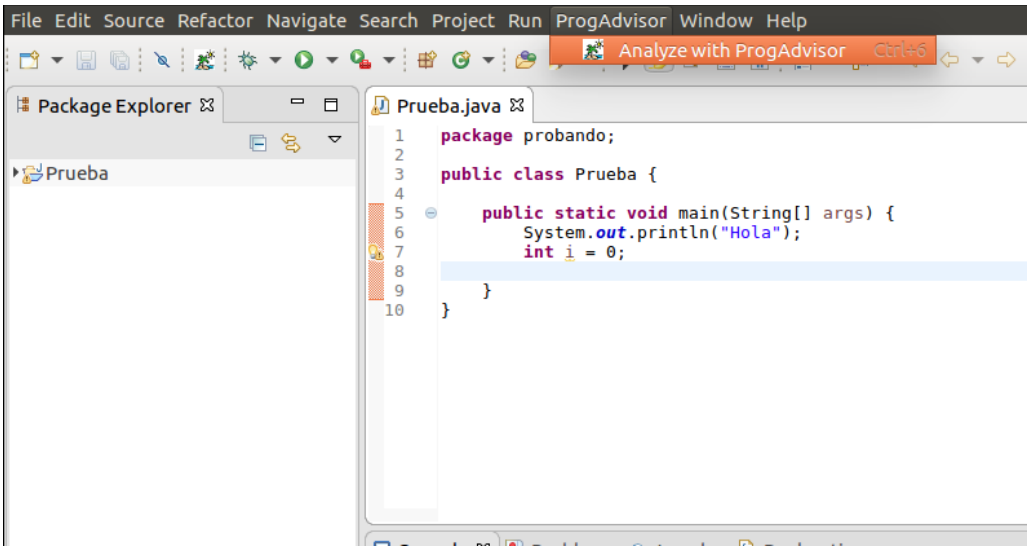


En el caso del plugin utilizado por el profesor, se mostrarán las páginas de preferencias de las aplicaciones correspondientes a esta versión del plugin, que tendrán un nombre parecido pero diferentes funcionalidades. En la imagen se aprecia el cambio de aplicaciones dependiendo del servidor al que se apunte:



5.1.2. Caso de uso: Opción de menú

El plugin **Prog-Advisor** incluye en la barra de menú una entrada con el mismo nombre del plugin que contiene la opción de "Analyze with Prog-Advisor", como se muestra a continuación.

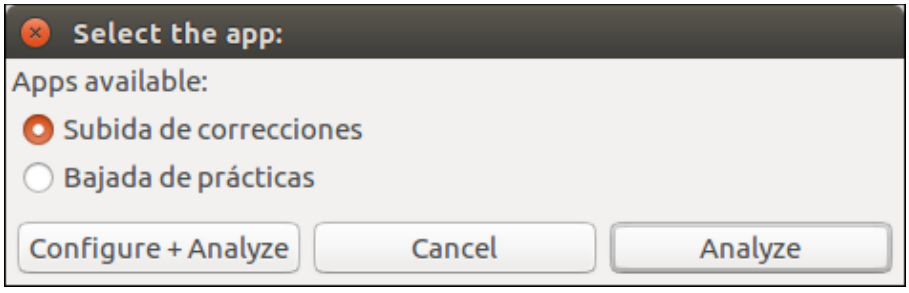


Los componentes básicos de Eclipse se pueden extender con plugins utilizando los *puntos de extensión*, que son entradas en el fichero de descripción del plugin (plugin.xml). De esta forma se puede modificar el aspecto de diferentes componentes de Eclipse como la barra de menú, la barra de herramientas, etc.

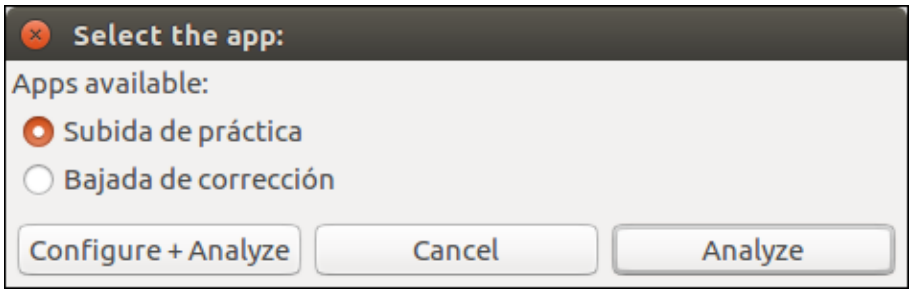
En este caso de uso, la entrada de **Prog-Advisor** en la barra de menú se crea con la especificación de las extensiones de *org.eclipse.ui.menus* en el fichero plugin.xml:

```
52  <extension
53        point="org.eclipse.ui.menus">
54     <menuContribution
55           locationURI="menu:org.eclipse.ui.main.menu?after=additions">
56        <menu
57              label="ProgAdvisor"
58              id="org.costa.progadvisor.menus.mainMenu">
59           <command
60                 commandId="org.costa.progadvisor.commands.analyzeMethod"
61                 id="org.costa.progadvisor.menus.noStdCommand"
62                 label="Analyze with ProgAdvisor"
63                 icon="icons/costa_icon.png">
64           </command>
65        </menu>
66     </menuContribution>
67     <menuContribution
68           locationURI="toolbar:org.eclipse.ui.main.toolbar?after=additions">
69        <toolbar
70              id="org.costa.progadvisor.toolbars.mainToolbar">
71           <command
72                 commandId="org.costa.progadvisor.commands.analyzeMethod"
73                 icon="icons/costa_icon.png"
74                 id="org.costa.progadvisor.toolbars.noStdCommand"
75                 tooltip="Analyze with ProgAdvisor">
```

La opción del menú de **Prog-Advisor**, “Analyze with Prog-Advisor”, abrirá una ventana de diálogo que dependerá de las aplicaciones instaladas en el servidor al que esté apuntando. En la siguiente imagen se aprecian las aplicaciones devueltas para el servidor de **Prog-Advisor** del usuario *alumno*.



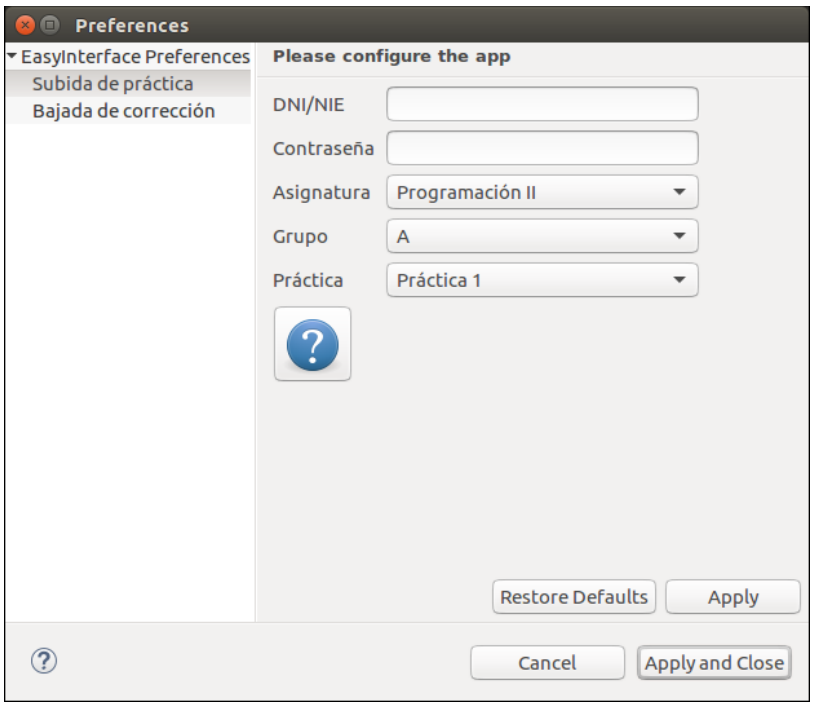
En el caso de profesor, se verán las siguientes:



Como en el caso anterior, esta ventana se creará de forma dinámica con la información de las aplicaciones devuelta por el servidor, que se solicita con una petición en formato JSON.

El botón *Configure + Analyze* permitirá configurar los parámetros de las aplicaciones del servidor antes de ser ejecutadas. Por cada aplicación, se creará dinámicamente una ventana con las preferencias de la aplicación correspondiente que podrán ser modificadas según las necesidades del usuario. En las siguientes figuras se mostrarán las ventanas de configuración de las cuatro aplicaciones desarrolladas para este Trabajo de Fin de Grado, que corresponden a las dos que usará el alumno y las dos del profesor.

Con esta implementación, cuando se seleccione la aplicación, por ejemplo, de *Subida de práctica* del plugin del alumno y se pulse en el botón de *Configure + Analyze*, se mostrará una ventana con los valores de los parámetros de esta aplicación:

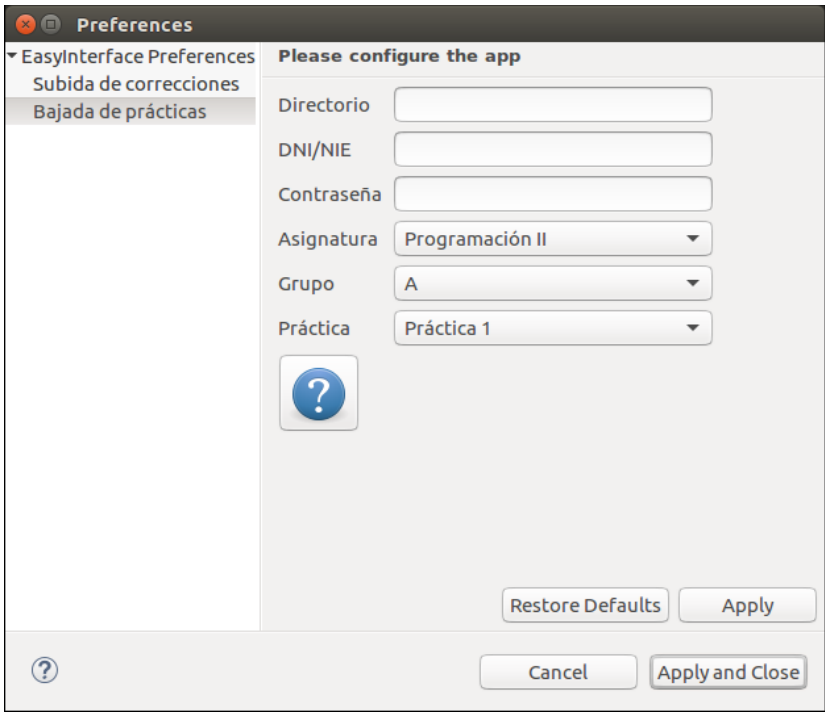


Esta ventana se generará de forma automática con los datos devueltos por el servidor con la etiqueta XML <*parameters*>. Cada campo de entrada se configurará dinámicamente con las opciones que estén disponibles. En la siguiente imagen se muestra el fragmento del mensaje de respuesta que correspondería al último parámetro de la imagen anterior, *Práctica*:

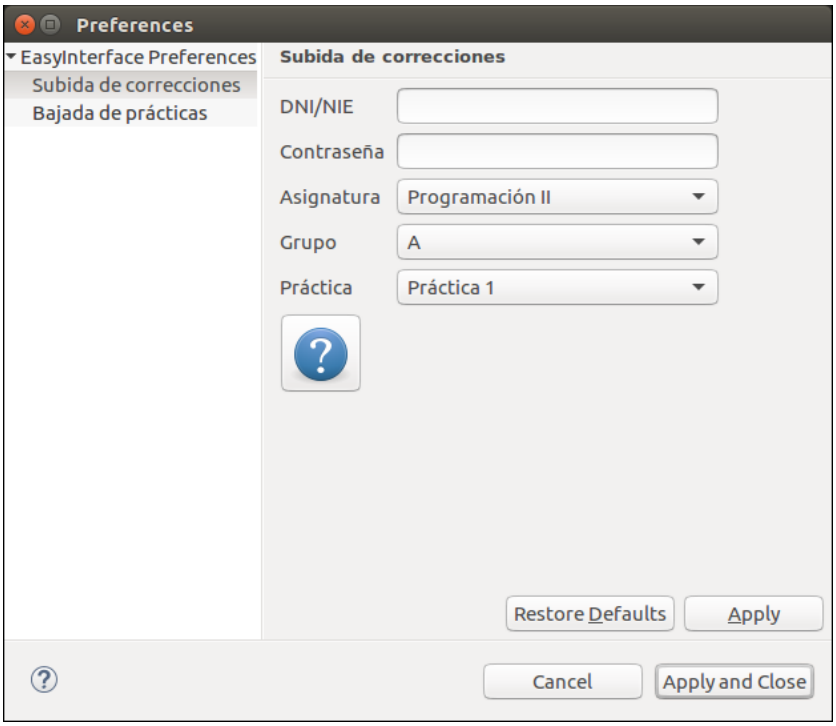
Se observa que para este parámetro se podrán elegir entre tres prácticas a subir: *Práctica 1*, *Práctica 2 y Práctica 3*, y que por defecto estará seleccionada la *Práctica 1*.

```
<selectone name="practica" widget="combo">
  <default>Pr1</default>
  <desc>
    <short>Práctica</short>
    <long>Práctica</long>
  </desc>
  <option value="Prac1">
    <desc>
      <short>Práctica 1</short>
      <long>Práctica 1</long>
    </desc>
  </option>
  <option value="Prac2">
    <desc>
      <short>Práctica 2</short>
      <long>Práctica 2</long>
    </desc>
  </option>
  <option value="Prac3">
    <desc>
      <short>Práctica 3</short>
      <long>Práctica 3</long>
    </desc>
  </option>
</selectone>
```

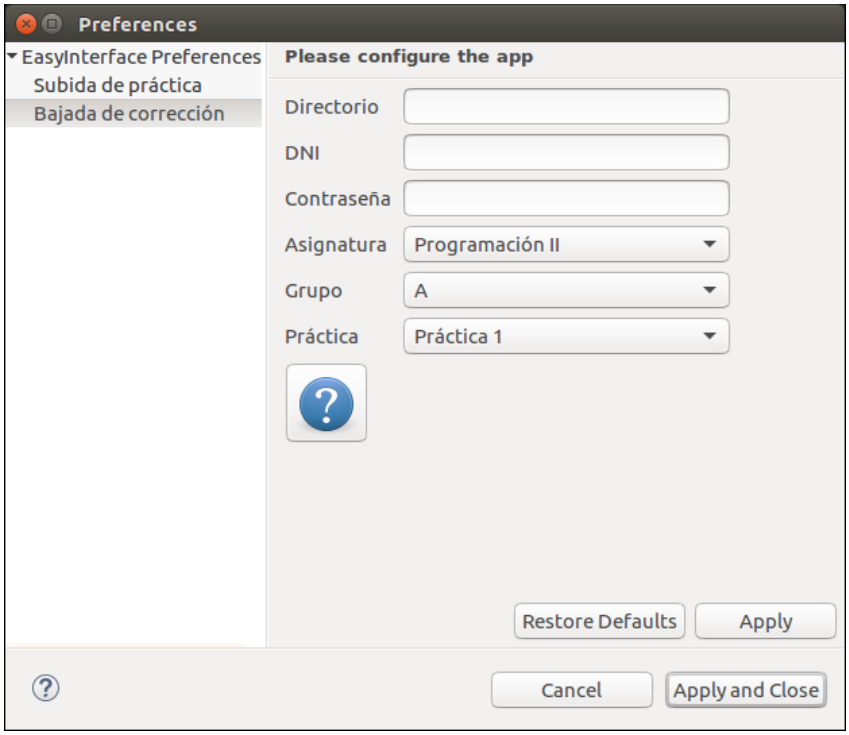
En otras aplicaciones se usarán diferentes parámetros y se compartirán otros, para mantener el flujo del plugin de subida de prácticas. En el caso de la aplicación de *Bajada de prácticas* del profesor, por ejemplo, se verán los siguientes parámetros:



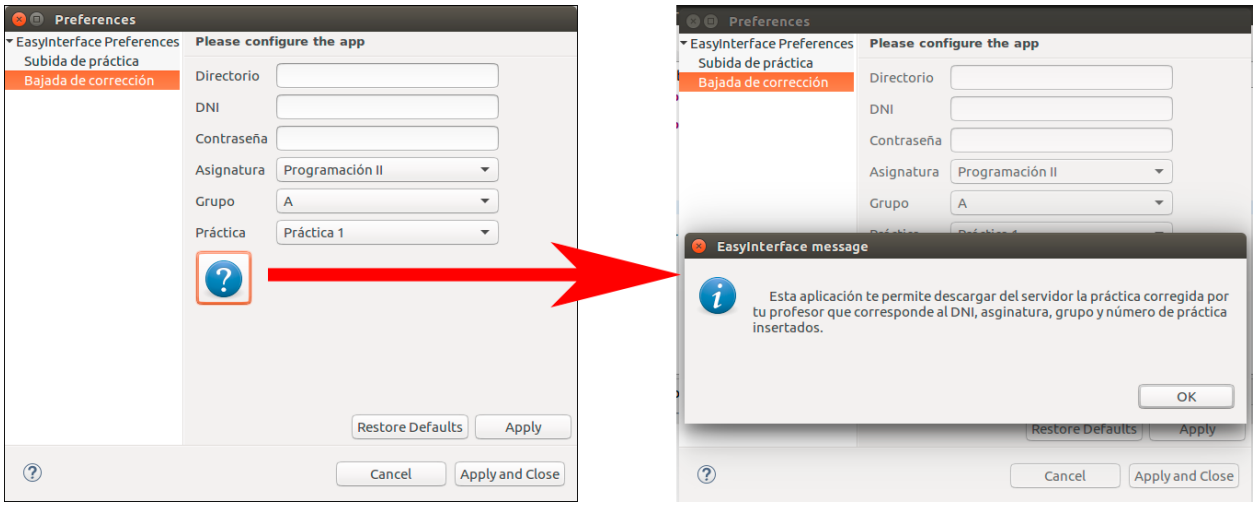
En la de *Subida de correcciones* de profesor serán estos:



Y por último, en la aplicación de *Bajada de corrección* por parte del alumno, se verán estos:

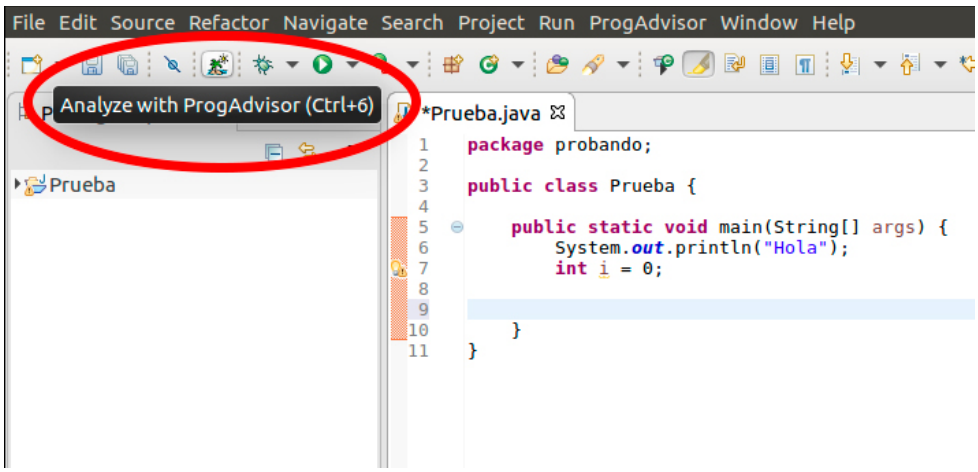


En todas se dispondrá de un botón de ayuda con un resumen de lo que va hacer cada aplicación. El texto que se mostrará en dicho mensaje forma parte también de la respuesta recibida del servidor al solicitar las aplicaciones disponibles:



5.1.3. Caso de uso: Opción de Barra de herramientas

El plugin **Prog-Advisor** también incluirá un acceso directo ubicado en la barra de herramientas, como se ve en la siguiente imagen:



Como en el anterior caso, se usarán los puntos de extensión *org.eclipse.ui.command*, *org.eclipse.ui.handlers* y *org.eclipse.ui.menus* para esta extensión de la barra de herramientas de Eclipse.

Al pulsar el nuevo botón **Prog-Advisor**, se mostrarán las mismas ventanas que en el caso de uso anterior.

5.1.4. Caso de uso: Desarrollar aplicaciones

Como ya se menciona en el resumen anterior, este caso se realizará enteramente en la parte del servidor, por lo que el usuario (que en nuestro TFG será el profesor) que vaya a hacer uso de este caso, tendrá que tener acceso y permisos de edición en el mismo.

Este caso de uso consistirá en el desarrollo de las aplicaciones que se quiera que tenga el servidor donde se están creando para un futuro uso en el plugin **Prog-Advisor**. Se trata de crear archivos de configuración del tipo *nombre_archivo.cfg* en el que se describan los parámetros de los que dispondrá la nueva aplicación (o la edición de archivos de configuración anteriores).

Como ejemplo, se muestra el archivo de configuración correspondiente a la aplicación del servidor del alumno, *Subida de Práctica*:

```
<app visible="true">
  <appinfo uploadtype="active">
    <acronym>Subida de práctica</acronym>
    <title>Subida de práctica</title>
    <desc>
      <short>Comprobador de código fuente</short>
      <long>Comprobador de código fuente</long>
    </desc>
  </appinfo>
  <apphelp>
    <content format='html'>
      Esta aplicación te permite subir tu práctica al servidor con los parámetros insertados, permitiendo que tu profesor se la descargue y
pueda subir una futura corrección.
    </content>
  </apphelp>
  <execinfo method="cmdline">
    <cmdlineapp>./run_subida_de_practica.sh -f _ei_files _ei_parameters</cmdlineapp>
  </execinfo>
  <parameters prefix = "-" check="false">
    <textfield name="dni">
      <desc>
        <short>DNI/NIE</short>
        <long>Indique su DNI/NIE</long>
      </desc>
    </textfield>
    <passwordfield name="password">
      <desc>
        <short>Contraseña</short>
        <long>Inserte la contraseña asignada</long>
      </desc>
    </passwordfield>
    <selectone name="asignatura" widget="combo">
      <default>PII</default>
      <desc>
        <short>Asignatura</short>
        <long>Asignatura</long>
      </desc>
      <option value="PII">
        <desc>
```

Se aprecian en la imagen los distintos parámetros y cómo se han declarado para que después el plugin **Prog-Advisor** instalado en Eclipse pueda mostrarlos como en los casos de uso anteriores.

Para las aplicaciones que se han desarrollado para este TFG, se ha utilizado un *script de shell* asociado a cada aplicación desarrollada, que consiste en un archivo de tipo *".sh"* que contiene líneas en lenguaje del shell de comandos de UNIX. Este shell recibirá los parámetros con los valores que introduzca el usuario que ejecute la aplicación desde el plugin, y los procesará para llevar a cabo la función de la aplicación.

```
mkdir -p $SRCDIR
mkdir -p $BCKDIR

cp -f $TEMP $SRCDIR

echo "[$DNI-$DATE]: Envio realizado $SRCDIR" >> $GLOG

echo "============================================================" | tee -a $LOG
echo "PROG-ADVISOR: Revisor de estilo de programación." | tee -a $LOG
echo "DNI: $DNI  ASIGNATURA: $ASIG  GRUPO: $GRUPO  PRACTICA: $PRAC" | tee -a $LOG

ext="${BASENAME##*.}"

ENCODED_PROJECT_PATH=$SRCDIR$BASENAME
PROJECT=${BASENAME%.$ext}
echo "   Proyecto: $PROJECT" | tee -a $LOG

base64 --decode $ENCODED_PROJECT_PATH > $SRCDIR$PROJECT.zip
unzip -qq $SRCDIR$PROJECT.zip -d $SRCDIR

cp $SRCDIR$PROJECT.zip $BCKDIR$DNI-$DATE.zip

cd $SRCDIR$PROJECT

# COMPILAR PROYECTO ENTERO
echo "==================Resultados de la compilación==================" | tee -a $LOG

mm="javac -cp ".:lib/*" -d bin $(find ./src/* | grep .java) 2>&1"
eval $mm
ret_code=$?
if [ $ret_code != 0 ]
then
        echo "Error: El proyecto no se ha subido al servidor porque no compila."
        echo "============================================================" | tee -a $LOG
        cd ..
        rm -rf $PROJECT
        rm -f $PROJECT.zip
        rm -f $PROJECT.txt
        echo "</content>"
```

En la imagen se ve un ejemplo del shell asociado a la aplicación de *Subida de Práctica* del servidor del alumno, el cual se explicará más adelante.

5.1.5. Caso de uso: Configurar aplicaciones

Este caso de uso, como se ha resumido anteriormente, consistirá en configurar el servidor de **Prog-Advisor** para ejecutar las aplicaciones que estén instaladas. Para realizar este caso, el usuario tendrá que tener acceso a los archivos del servidor, y poder editar el archivo de *apps.cfg* correspondiente al servidor que quiera configurar.

```
<apps>
 <app id="Subida de correcciones"              src="./subida-de-correcciones.cfg" />
 <app id="Bajada de prácticas"              src="./bajada-de-practicas.cfg" />
</apps>
```

Como se ve en la imagen, en el servidor del alumno estarán instaladas dos aplicaciones: *Subida de Práctica* y *Bajada de Corrección*. En el caso del servidor del profesor, estarán instaladas las de *Bajada de Prácticas* y *Subida de Correcciones*.

Para configurar el servidor, será necesario crear un archivo de configuración o usar uno creado y añadirlo al archivo de *apps.cfg* para que el plugin pueda cargarlo en la interfaz que lanza desde Eclipse, vista en los casos anteriores.

5.2. Casos de uso de Prog-Advisor

Los casos de uso de **Prog-Advisor** son casos particulares de **EasyInterface**, para unas configuraciones de aplicaciones específicas. **EasyInterface** proporciona un framework genérico. Lo que hemos hecho en este TFG es particularizarlo para los casos concretos de **Prog-Advisor**, extendiendo algunos de los comandos y respuestas para proporcionar la funcionalidad necesaria.

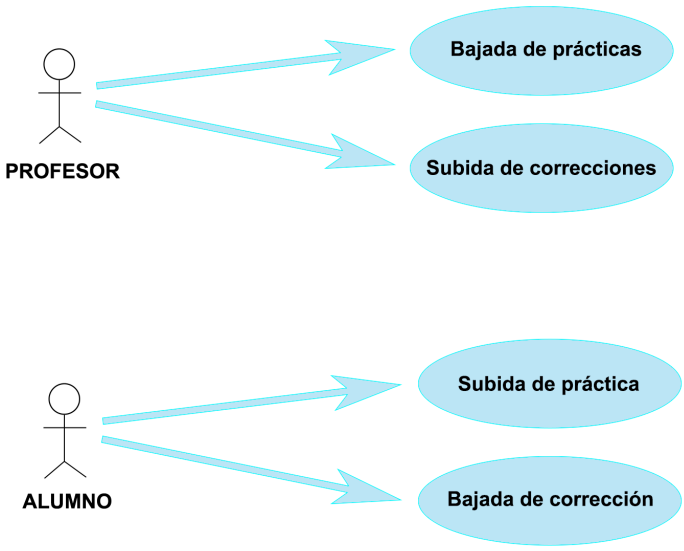
En el nuevo flujo de subida-bajada de prácticas creado para este proyecto, identificaremos los siguientes casos de uso para el *alumno*:

- **Subida de práctica:** Este caso consistirá, como su nombre indica, en la subida de la práctica activa en el Eclipse del *alumno* a un servidor remoto para la futura revisión y/o corrección del profesor.
- **Bajada de corrección:** Este caso cerrará el flujo de subida-bajada mencionado anteriormente. Consistirá en la bajada de la práctica subida con el caso anterior, corregida por el profesor, y la creación de un nuevo proyecto en Eclipse con dicha práctica.

Para el *profesor*, los casos son muy parecidos pero con distinta implementación:

- **Bajada de prácticas:** El profesor se bajará todas las prácticas correspondientes a la asignatura, grupo y número de práctica elegidos, y el plugin le creará y abrirá los proyectos en su Eclipse.
- **Subida de correcciones:** En este caso, el profesor subirá el contenido de una carpeta donde estén guardadas las prácticas que quiere subir, y el plugin se encargará de dejarlas en el servidor para la futura descarga de los alumnos.

Los actores serán los mismos que los explicados en los casos de uso heredados: *alumno* y *profesor*:



Por seguir el flujo de la aplicación, se describirán a continuación los casos de uso en el siguiente orden:

- *Subida de práctica*, *Bajada de prácticas*, *Subida de correcciones*, *Bajada de corrección*.

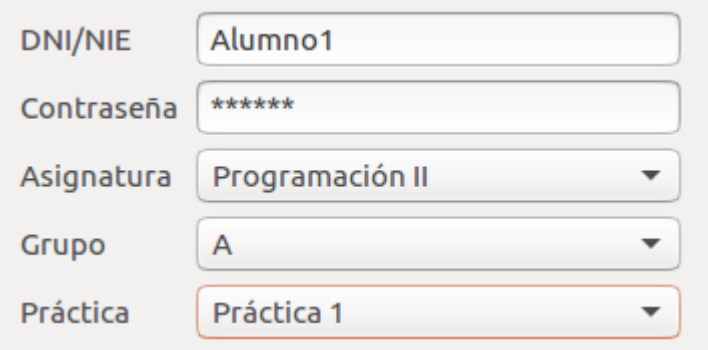
5.2.1. Caso de uso: Subida de práctica

Este caso supondrá el principio del flujo subida-bajada creado en este TFG.

El alumno subirá al servidor una práctica para que el profesor pueda corregirla.

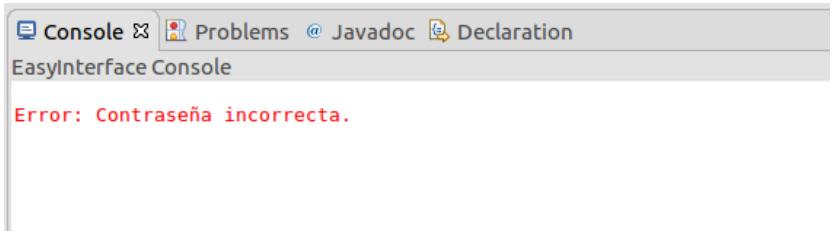
Primero, el *alumno* tendrá que acceder al plugin **Prog-Advisor** instalado en su Eclipse por la barra de menú o la barra de herramientas (como se ha explicado anteriormente en la sección 4.1.1 y 4.1.2), y rellenar los parámetros de la nueva aplicación desarrollada para este proyecto (como en el caso de uso *establecer parámetros*) con los datos correspondientes.

Los datos que tendrá que insertar el alumno son: *DNI*, *Contraseña*, *Asignatura*, *Grupo* y *Número de práctica*, los cuales vienen dados de la aplicación con el mismo nombre que este caso de uso instalada en el servidor del alumno.

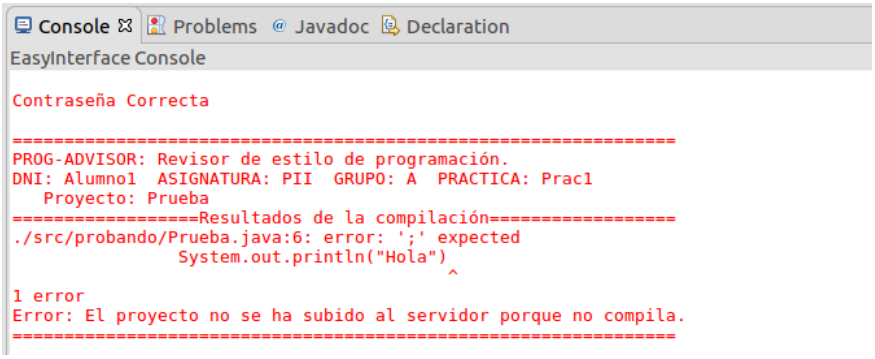


Una vez rellenados los parámetros de la aplicación con los valores correspondientes, la práctica se subirá al servidor. En él, el script de shell asociado a la aplicación que se ejecuta en el servidor realizará todo el proceso de verificación del usuario.

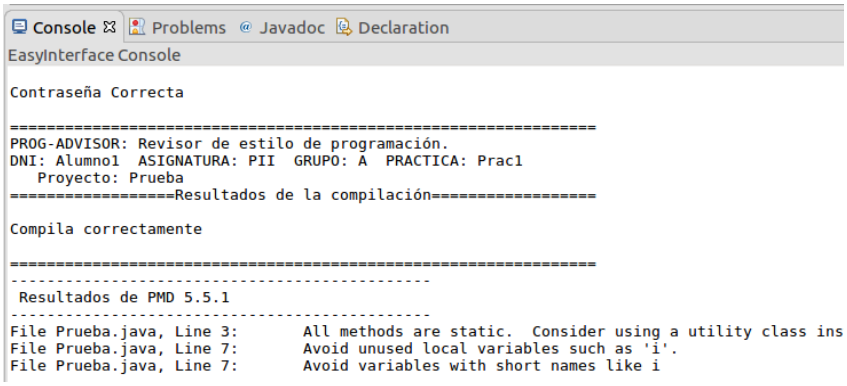
En el caso de que el alumno introduzca un DNI que no esté registrado o una contraseña diferente a la que se le ha asignado, el script del servidor devolverá una respuesta para que el plugin muestre por la consola de Eclipse un error notificando al alumno. Para el proceso de validación de usuario-contraseña del servidor se ha utilizará el comando htpasswd, tomado de la página web de Apache[9].



Si los datos de autenticación son correctos, la práctica se compilará en el servidor. Si la compilación tuviera errores, se notificará al alumno y la práctica no se subirá al servidor.



En caso contrario, la práctica subida pasará por el analizador de código instalado en el servidor, **PMD**, el cual devolverá unos resultados que se pasarán también al plugin y se mostrarán por la consola. Una vez realizadas estas dos acciones, se mostrará un texto informativo con los valores de los parámetros asociados a la práctica que se han subido al servidor.

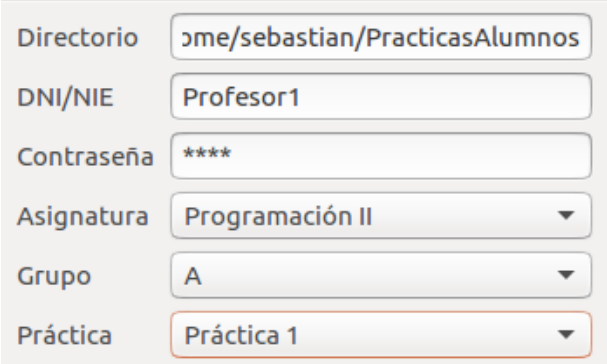


5.2.2. Caso de uso: Bajada de prácticas

Este caso corresponde al siguiente paso en el flujo de subida-bajada que propone **Prog-Advisor**.

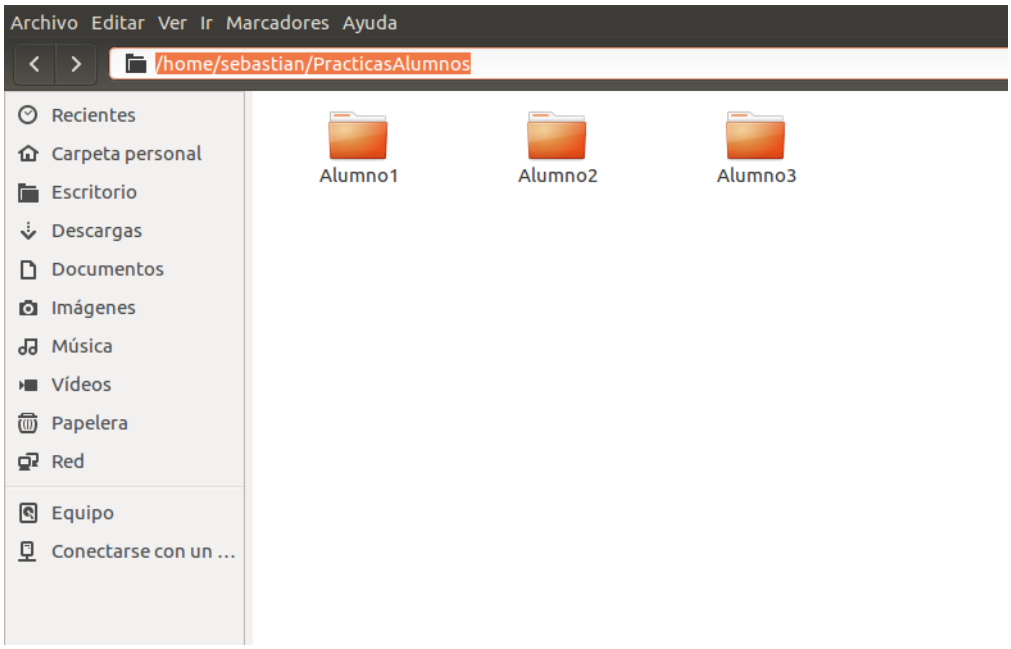
El profesor se descargará las prácticas subidas al servidor que correspondan a su asignatura y grupo.

Como en el anterior, el profesor tendrá que rellenar los parámetros de la aplicación *Bajada de prácticas*, pero en esta aplicación se pedirá un campo más, que corresponde a la dirección del directorio donde se quiere realizar la bajada masiva de prácticas.

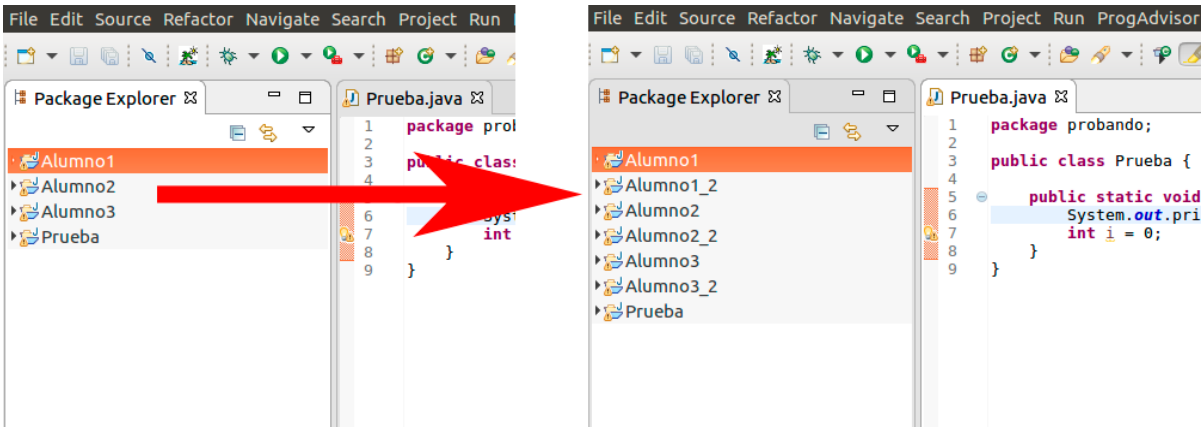


Una vez rellenados los parámetros, el plugin lanza la petición al servidor, en el cual el script de shell remoto es el que se encarga de gestionar los parámetros. La parte de autenticación se llevará a cabo de la misma forma que en el alumno. El servidor devolverá un texto codificado en base64 del

archivo comprimido en *zip* con todas las prácticas que estuvieran subidas en el servidor que cumplieran los valores de los parámetros anteriormente introducidos. El plugin tratará este texto y descomprimirá las prácticas en el directorio que el profesor haya elegido.



Con las prácticas descomprimidas, el plugin se encargará de montarlas en el Eclipse abierto. En el caso de que el profesor ya tuviera una práctica montada con el DNI de un alumno que se ha vuelto a bajar, el plugin creará otro proyecto con el nombre: *DNI_v(número de versión correspondiente)*.

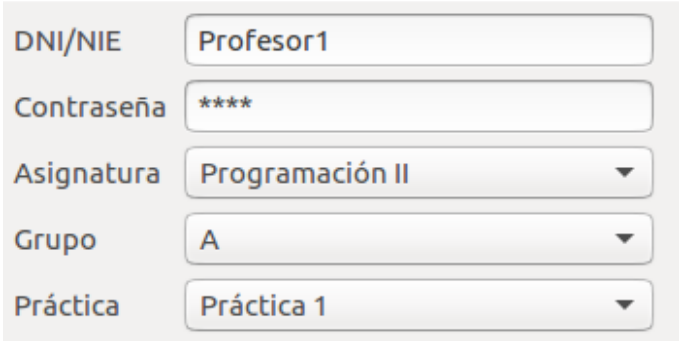


5.2.3. Caso de uso: Subida de correcciones

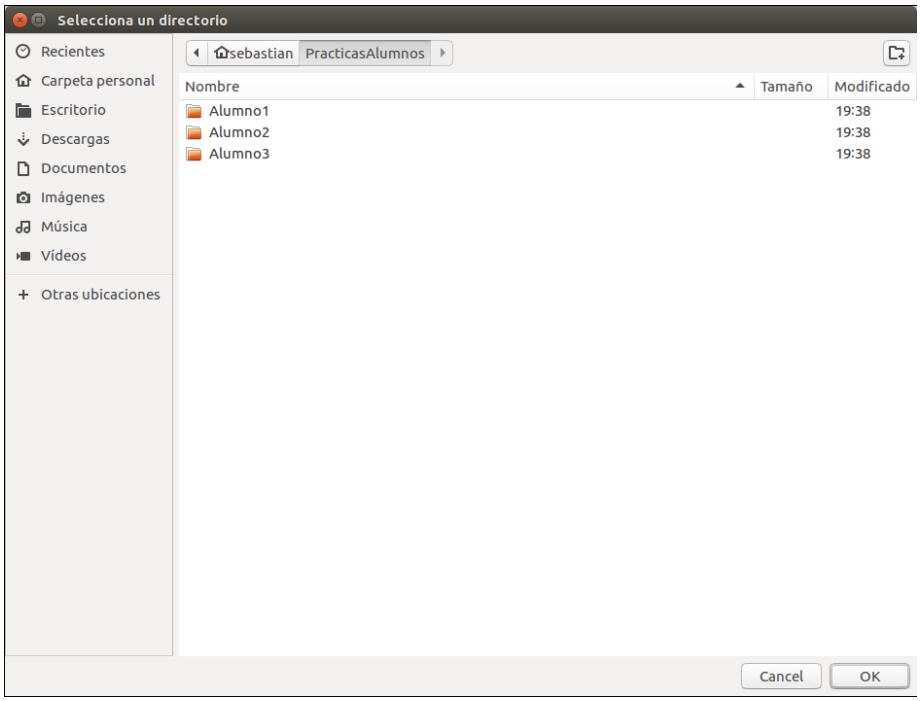
Como ya se ha mencionado en el anterior, este caso corresponderá al siguiente paso del flujo.

El profesor podrá realizar cualquier cambio sobre el proyecto del alumno. Por ejemplo, podrá añadir comentarios al código, añadir casos de prueba de JUnit que hace que falle su programa, corregir algunos errores del código fuente, o añadir archivos con comentarios sobre la práctica.

Parecido al caso de *Subida de práctica*, este caso subirá las correcciones de una carpeta a seleccionar por el profesor.



Una vez rellenados los valores de los parámetros de la aplicación *Subida de correcciones*, el plugin abrirá un diálogo de selección de carpeta (Directory Dialog SWT) [6], en el que el profesor deberá seleccionar la carpeta donde estén guardadas las prácticas corregidas.



Como en el caso de subida por parte del alumno, el servidor recibirá los valores y verificará el DNI y la contraseña introducidas, sacando el correspondiente mensaje de error en caso de que alguno de los valores sea incorrecto. También pasará por el analizador de código **PMD**, instalado también en el servidor del profesor, y enviando la respuesta que el plugin después sacará por la consola de Eclipse.

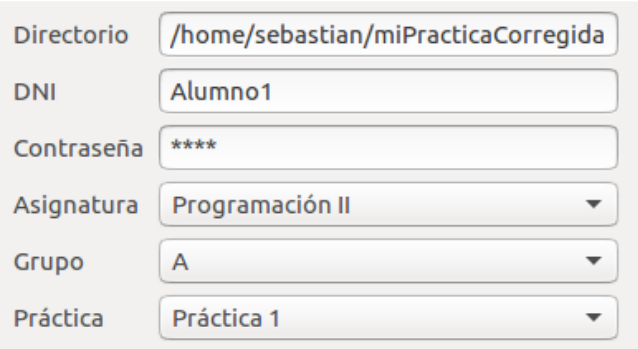
Estas prácticas quedarán guardadas en el servidor para su futura descarga por parte de los alumnos.

5.2.4. Caso de uso: Bajada de corrección

Este caso será el último correspondiente al flujo de subida-baja de **Prog-Advisor**.

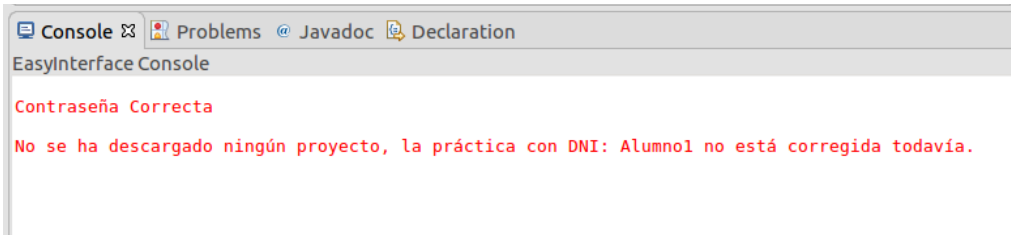
El alumno se bajará la práctica corregida por el profesor, del servidor.

Se parecerá al caso de *Bajada de prácticas* del profesor, ya que la aplicación *Bajada de corrección* también incluirá el parámetro “Directorio” en el que habrá que indicar el directorio donde se quiera descargar la corrección.



Como en el caso del profesor, los valores de los parámetros se procesarán en el shell remoto en el servidor, el cual devolverá los mensajes de error de haber fallado la autenticación, o mensaje de éxito en el caso de que la práctica se hubiera descargado y creado correctamente.

En el caso de que en el servidor todavía no existiera una corrección para su práctica, el servidor devolverá un error notificándoselo al alumno:

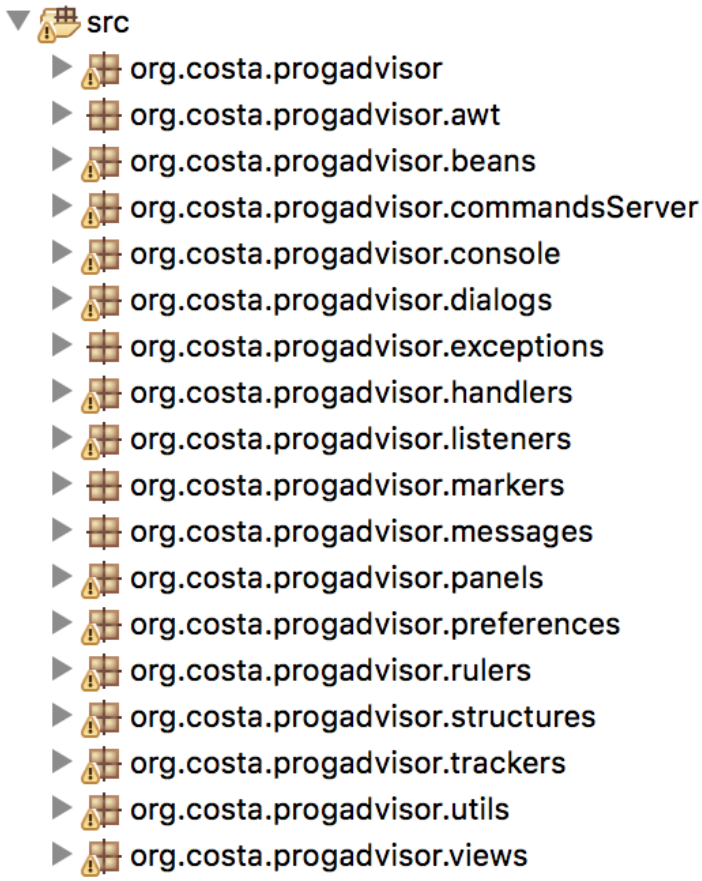


El proceso será el mismo que en el caso de uso *bajada de prácticas*. El plugin recibirá un *zip* en base64 que decodificará y descomprimirá en la carpeta que el alumno haya indicado. Después, creará el proyecto con nombre DNI en el Eclipse abierto y el alumno podrá ver los cambios que haya añadido el profesor.

6. Paquetes y estructuras de clases del plugin Prog-Advisor

La estructura de paquetes del plugin Prog-Advisor proceden de la implementación de **EasyInterface**.

En la siguiente imagen se muestra la estructura de los paquetes existentes en **Prog-Advisor**:



Para el desarrollo del plugin Prog-Advisor han sido necesarios los siguientes cambios en los paquetes ya existentes.

6.1. Paquete org.costa.progadvisor.beans

En este paquete se encuentran las clases necesarias para llevar a cabo la serialización de XML en objetos Java y viceversa mediante la utilización de anotaciones con la tecnología JAXB [7].

Se han llevado a cabo las siguientes modificaciones en este paquete:

AppInfo.java

Se ha modificado la clase **AppInfo.java**, en el cual se ha añadido un nuevo atributo llamado **uploadtype**. Este atributo se identificará dentro de un XML con la anotación “uploadtype”.

```
@XmlAttribute(name = "uploadtype")
private String uploadtype;
```

Param.java

Clase encargada de serializar en objetos java los parámetros XML que contienen las aplicaciones del plugin Prog-Advisor (selectone, selectmany, textfield, etc).

Esta clase ha sido modificada para añadir un nuevo atributo llamado **passwordfield**. Este atributo se identificará dentro de un XML con la anotación **passwordfield**.

La finalidad de añadir este nuevo parámetro es permitir a los clientes del plugin (profesores y alumnos) poder identificarse para poder hacer uso de las aplicaciones disponibles en el servidor de cada uno.

```
@XmlElement(name = "passwordfield", type = Passwordfield.class)
private Passwordfield passwordfield;
```

GetProjectCommand.java

En este paquete se ha añadido una nueva clase llamada GetProjectCommand.java, el cual se va a encargar de llevar a cabo la serialización de la etiqueta XML **<getproject> </getProject>** en un objeto java.

Sus atributos son:

- **path:** Atributo en el que se almacenará una ruta local.
- **texto:** Atributo en el que se almacenará contenido obtenido del servidor, comprimido y codificado en Base64 [8].

6.2. Paquete org.costa.progadvisor.console

En este paquete se han hecho modificaciones en la clase **CostabsShellCommand.java**.

Como se ha mencionado anteriormente, en la clase AppInfo.java se ha añadido el atributo **uploadtype**. Este se rescatará en el método buildCommand() de la clase CostabsShellCommand.java.

Dependiendo del valor de este atributo, el plugin Prog-Advisor hará una cosa u otra:

- **active**: la etiqueta uploadtype tomará este valor en el caso de que se quiera subir una sola práctica al servidor. Debido a esto, en la aplicación de subida del alumno se utilizará este valor mediante el cual se subirá al subdirectorio de Prog-Advisor-Data del servidor la práctica actualmente abierta del alumno.
- **many**: la etiqueta uploadtype tomará este valor en el caso de que se quiera subir un conjunto de prácticas al subdirectorio Prog-Advisor-Data-Correcciones del servidor. Solo se utilizará en la aplicación de subida del profesor, puesto que el profesor va a subir todas las correcciones de las prácticas de los alumnos asociados a una asignatura y a un grupo.
- **none**: la etiqueta uploadtype tomará este valor en el caso de que se quiera descargar prácticas del servidor. Lo usarán las aplicaciones de profesor bajada y alumno bajada. Si la etiqueta uploadtype tiene este valor, es por qué se va a hacer uso del nuevo tracker creado para el plugin Prog-Advisor llamado GetProjectTracker.java, el cual se explicará más adelante.

6.3. Paquete org.costa.progadvisor.preferences

En este paquete se encuentran las clases encargadas de implementar las páginas de preferencias del plugin Prog-Advisor.

La clase **CostabsPreferences0.java** se va a encargar de implementar la página de preferencias que encontraremos en la barra de menú de Eclipse.

La clase **CostabsPreferences2.java** se va a encargar de implementar la página de preferencias de la barra de herramientas de Eclipse.

Los campos que contienen las páginas de preferencias de cada una de las aplicaciones coinciden con los parámetros que tienen estas en el servidor.

Tal y como se ha mencionado en la sección “Estructura cliente-servidor Prog-Advisor”, para recuperar estos parámetros y construir las páginas de preferencias del plugin Prog-Advisor, se realizará una petición de solicitud de información sobre todas las aplicaciones disponibles en el servidor del alumno o del profesor. La respuesta del servidor será en formato XML y contendrá los parámetros de configuración de las aplicaciones disponibles en el servidor del alumno o del profesor.

Las clases mencionadas anteriormente se encargarán de construir las páginas de preferencias mediante el XML obtenido del servidor. Lo utilizarán para crear una lista de objetos java que los representa. Mediante esta lista de objetos se construirán los controles gráficos de las páginas de preferencias con la información recogida de las aplicaciones.

Puesto que tanto el alumno como el profesor tienen que autenticarse mediante una contraseña para poder utilizar las aplicaciones propias del servidor de cada uno, ha sido necesario añadir en las clases mencionadas anteriormente un nuevo atributo para la cotraseña llamado **OPT_PASSWORDFIELD.** Este será un nuevo campo para las páginas de preferencias del plugin Prog-Advisor, el cual será obtenido desde la petición XML del servidor.

```
public static final String OPT_PASSWORDFIELD = "passwordfield";
```

6.4. Paquete org.costa.progadvisor.trackers

En este paquete se encuentran todas las clases que se van a encargar de gestionar las respuestas del servidor como comandos y acciones.

En el plugin Prog-Advisor se ha añadido una nueva clase en este paquete llamada **GetProjectTracker.java**.

Los atributos de esta clase son **path** y **zipfile**, los cuales se obtendrán mediante la clase **GetProjectCommand.java**.

El objetivo de este nuevo tracker es permitir obtener en el Eclipse de desarrollo una ruta local (indicada por el profesor o el alumno) y un contenido indicado. El contenido estará comprimido y codificado en Base64 y corresponderá a una serie de prácticas obtenidas del servidor. La ruta local se almacenará en el atributo **path** y el contenido en el atributo **zipfile**.

El método **track()** se encargará de decodificar y descomprimir el contenido en la ruta indicada. Trás tener todas las prácticas descomprimidas, se importaran uno a uno en el Eclipse del profesor o del alumno como proyectos java.

6.5. Paquete org.costa.progadvisor.utils

Este paquete va a contener clases útiles para el plugin. Además de las disponibles, se han añadido dos clases nuevas: **ZipUtils.java** y **Unzip.java** [10].

- ZipUtils.java

  Esta clase se encargará de comprimir, en formato .zip, un directorio determinado, localizado por su ruta.

- Unizp.java

  Esta clase se encargará de descomprimir un archivo, con compresión .zip, en un directorio indicado mediante su ruta.

7. Sistemas utilizados

En el desarrollo y la implementación del plugin **Prog-Advisor** se ha hecho uso de los siguientes sistemas y tecnologías:

- **Ubuntu 16.04 y MacOs High Sierra:** el desarrollo y las pruebas de funcionamiento del plugin se han llevado a cabo en estos dos sistemas operativos.
- **Eclipse Oxygen.1a versión (4.7.1.M20170906-1700):** IDE de Eclipse utilizado para la implementación del código fuente del plugin.
- **Apache Subversion (SVN):** repositorio utilizado para llevar a cabo un control de versiones del código fuente del plugin.
- **Servidor Apache:** tecnología en la cual está basado el servidor del plugin.

8. Trabajo realizado por cada alumno

El trabajo se ha repartido de una forma equitativa, tocando ambos alumnos tanto la parte del servidor como la del cliente. A continuación se especifica el trabajo realizado por cada uno:

8.1. Sebastián Oleaga Quintas

Explicaré mi contribución a este proyecto muchas veces hablando en plural ya que prácticamente todo el desarrollo del código lo realizamos conjuntamente los dos participantes quedándonos en la biblioteca de la facultad, rara vez trabajando de forma individual en casa.

Empecé el trabajo junto con mi compañero sin tener mucha idea de plugins ni otras extensiones de Eclipse, ya que era algo que no había utilizado en toda la carrera. En un principio tuvimos que acudir a varias reuniones con nuestros directores sólo para poder instalar correctamente el entorno en nuestros ordenadores. La dificultad en la instalación venía dada por las múltiples partes que teníamos que tener instaladas antes de ponernos a desarrollar las nuevas aplicaciones de nuestro proyecto. Una vez tuvimos instaladas las partes de servidor y descargado del repositorio de SVN el código fuente del plugin base EasyInterface, ya pudimos ponernos a ejecutar y probar algunos de los casos de uso que íbamos a utilizar y cambiar para el proyecto.

En un principio mi compañero y yo realizábamos el mismo trabajo al mismo tiempo, ya que solo entender el flujo del plugin, y lo que hacían las distintas clases que no habíamos desarrollado nosotros nos llevó mucho esfuerzo y tiempo. A partir del primer mes, las reuniones semanales con nuestros directores nos ayudaron a ponernos al día y poder empezar a hacer cambios. El plugin de EasyInterface tenía desarrollada una aplicación para la subida de un único fichero Java, así que partimos de ahí para empezar a hacer cambios para que el plugin pudiera subir al servidor más de un archivo.

La solución con la que dimos, y aprovechando la estructura ya creada de subida de un archivo, fue la de comprimir y pasar a base64 todos los archivos del proyecto que estuviera activo en el momento de la subida, guardando este base64 en un archivo txt de texto plano que sería el que subiríamos al servidor. Durante este proceso, tuvimos que tocar también la parte del servidor para

que se adaptara a la nueva subida, creando una nueva aplicación con más parámetros (asignatura, grupo, número de práctica y DNI), y cambiando casi por completo el archivo shell asociado a la aplicación.

Este último archivo fue de los que más tiempo nos llevó implementar, ya que en él quisimos añadir la compilación Java a través de la línea de comandos y tratar el txt para transformarlo en un zip que después pudiéramos descomprimir para que el profesor pudiera bajarse la práctica. Mi compañero se encargó de la parte de la compilación mientras que yo desarrollé la parte de tratado del txt.

Una vez esta parte fue avanzando y pudimos realizar las primeras subidas de proyectos correctamente, pasamos a la parte de la bajada de estas prácticas por parte del profesor.

Este caso requería otra visión totalmente diferente, ya que ahora sería del servidor donde tendríamos que conseguir los datos, y el plugin donde tratarlos. Con la dificultad de no poder navegar por carpetas desde el plugin como hacíamos en el servidor (ya que la estructura de carpetas sería diferente para cada usuario), topamos con la idea de que el profesor subiera al servidor una ruta de la carpeta donde quisiera guardar las prácticas descargadas, y luego este se la devolviera sin tocarla. No era la solución más correcta pero fue la única que nos funcionó bien y nos permitió seguir desarrollando.

Para conseguir que el profesor recuperara las prácticas subidas por los alumnos, creamos una nueva aplicación que recibía como parámetros la ruta mencionada anteriormente, y los datos de la asignatura, grupo y práctica y que quisiera descargar. En el shell asociado, hicimos que hiciera el proceso inverso a la subida: crear un zip con todas las prácticas y pasarlo a base64 para que después el plugin lo tratara.

Para que el plugin convirtiera este base64 a una lista de prácticas abiertas en el Eclipse, creamos un nuevo comando y un tracker en el plugin que recibiera el texto en base64, lo convirtiera en un zip, lo descomprimiera y después creara los proyectos en el Eclipse desde el que se hubiera ejecutado la aplicación. Esta parte del trabajo la realizamos ambos participantes simultáneamente, quedándonos varias tardes en la biblioteca y reuniéndonos con los profesores para que nos ayudaran en los temas de implementación que nos costaran.

La siguiente parte, que era la de subida de las correcciones por parte del profesor, la realizamos más rápido comparado con las anteriores, ya que se parecía bastante la subida por parte de un alumno y el código se podía reutilizar. Lo único que variaba era que en lugar de subir una única práctica, el profesor subiría varias, por lo que decidimos utilizar un JFrame para que el profesor escogiera la carpeta donde se encontraban las prácticas que quería subir. Para mantener el diseño ya utilizado por el plugin heredado, al final optamos por un DialogBox del sistema para la elección de esta carpeta.

El proceso de subida desde el shell era muy parecido, tratando un txt que subía el plugin al servidor y transformándolo a un zip que después se descomprimiría en una carpeta a la que tendría que acceder después del plugin del alumno.

La última parte, que era la bajada de la corrección por parte del alumno, no tuvo mayor dificultad que buscar el DNI que insertara el alumno para descargar una práctica con ese nombre.

Esta parte fue importante, ya que hasta el momento habíamos guardado las prácticas subidas en el servidor con la fecha y el DNI como nombre de la carpeta. Al tener que descargar el alumno una única corrección (la última subida) y para que el profesor no descargara varias versiones de una misma práctica de un alumno (al principio se bajaba y montaba todas las versiones que estuvieran subidas), decidimos eliminar la fecha del nombre y guardar una única subida, eliminando del servidor versiones anteriores con un mismo DNI si las hubiera. De esta forma, el servidor estaba menos cargado y las funciones de subida y bajada se simplificaban enormemente.

Para mantener un control de versiones, decidimos hacer una nueva carpeta en el servidor que guardara en formato zip todas las subidas realizadas, aunque después éstas no fueran a ser utilizadas por ninguna aplicación.

Con esto, ya solo quedaba dar unos últimos retoques, como el análisis del código con PMD y los ficheros de usuarios y contraseñas para mantener un mínimo de seguridad en la subida y bajada de prácticas.

Esta última parte la desarrollé yo con los profesores, ya que mi compañero tuvo unos problemas con su ordenador y aprovechó este tiempo para empezar a escribir esta memoria.

La memoria sí la dividimos en partes para poder realizarla cuanto antes. Yo me encargué de la parte de “Implementación del plugin” y “Conclusiones y trabajo futuro”, como de la traducción al inglés de las partes realizadas por mi compañero.

8.2. Aykaz Esayan

Antes de empezar a desarrollar el Trabajo de Fin de Grado, empezamos aprendiendo cosas sobre plugins, con la documentación proporcionada por los profesores. Para practicar y entender cómo es un plugin por dentro, hice un plugin que simplemente pintaba por consola de Eclipse “Hola Mundo”.

Una vez que comprendimos el concepto de plugin y habiendo practicado con ellos, comencé a profundizar en el plugin EasyInterface. La ayuda de los profesores fue fundamental para entenderlo y poder llevar a cabo el desarrollo del plugin Prog-Advisor.

El primer paso que dimos fue configurar, con la ayuda de los profesores, nuestros ordenadores para tener el plugin funcionando en ellos. A su vez, se configuró SVN, de donde se obtuvo el código inicial del plugin.

Ya teniendo todo listo para empezar a programar, acordamos con los profesores tener reuniones semanales en los que nos informaban de que cosas teníamos que hacer y para enseñarles nuestros avances. Estas reuniones fueron muy útiles para poder avanzar en el desarrollo del plugin.

Nuestro primer objetivo fue aprender a programar en el lenguaje Shell Script de Linux, para poder llevar a cabo el desarrollo de las aplicaciones del servidor. No me llevó mucho tiempo aprender dicho lenguaje gracias a la documentación de Internet y ciertos conocimientos de manejo de comandos de linux adquiridos en diferentes asignaturas de la carrera.

La primera tarea que nos encomendaron los profesores fue la de poder subir la práctica actual en la que estaba trabajando el alumno al servidor. En un principio, en el plugin sólo se podía subir un único fichero java al servidor. Fue la más costosa, puesto que recién estábamos familiarizándonos con el funcionamiento del plugin. Ambos participamos por igual en el desarrollo de esta primera tarea. Una vez acabada, los profesores nos dijeron que además de subir la práctica del alumno, teníamos que compilar dicha práctica. De esto me encargue yo. Fue un trabajo realmente costoso, puesto que los comandos que había que utilizar para llevarlo a cabo los desconocía por completo.

Con esta primera tarea, es cuando realmente empecé a entender el funcionamiento y el flujo de ejecución del plugin. Aprendí bastantes comandos del lenguaje Shell Script de Bash, además de comprender muchas cosas del código fuente del plugin que teníamos entre manos.

La segunda tarea que tuvimos que llevar a cabo fue que el profesor fuera capaz de bajarse un conjunto de prácticas de alumnos. Estas prácticas se tenían que importar automáticamente en el Eclipse del profesor, con el fin de evitar importar cada práctica de manera individual. Tuvimos ciertas complicaciones a la hora de llevarlo a cabo, puesto que no sabíamos cómo obtener el directorio donde el profesor se quería descargar las prácticas en el Eclipse de desarrollo. Tras hablar

con los profesores, llegamos a la conclusión de que era necesario crear un nuevo tracker y un nuevo comando para propagar esa información.

Nos pusimos juntos a desarrollar dicha tarea en la biblioteca de la facultad. Basándonos en en los trackers y los comandos ya implementados en EasyInterface, conseguimos llevar a cabo la implementación.

Teniendo el tracker implementado, solo nos faltaba implementar la importación de las prácticas descargadas por el profesor en su Eclipse. Esta parte fue un poco complicada, puesto que tuvimos que fijarnos bastante en las documentaciones de Eclipse y encontrar la manera adecuada de importar el proyecto entero con todas sus librerías en un determinado workspace de Eclipse. El problema que teníamos en un principio era que el proyecto se importaba pero no como proyecto java. Investigando un poco más, encontré la solución. Al conseguir importar un proyecto en formato java, se solucionó el problema de las librerías.

Las dos últimas tareas que nos faltaban para cerrar el flujo de ejecución del plugin, fue hacer posible que el profesor subiera las correcciones de las prácticas de los alumnos y que el alumno pudiera bajarse su corrección y que esta se importará automáticamente en su Eclipse.

Llegados a este punto, decidimos crear dos subdirectorios en el servidor. La idea fue que el alumnos pudieran subir sus prácticas a un subdirectorio dentro del servidor y que los profesores pudieran acceder y descargarse las prácticas de los alumnos para corregirlos. Una vez corregidas las prácticas, el profesor los subiría al subdirectorio de correcciones, de donde los alumnos se descargarían su práctica corregida.

Lo primero que hicimos fue la aplicación encargada de subir las correcciones del profesor al servidor. Esto no fue muy costoso, debido a que ya teníamos cierta base procedente de la subida de una única práctica por parte del alumno. Teníamos que proporcionar una interfaz al profesor para que pudiera escoger la carpeta donde tenía guardadas las correcciones de los alumnos. En un principio escogimos JDialogBox, el cual no funcionó del todo bien en mi ordenador por motivos de sincronización de hilos. Además, los profesores nos dijeron de coger una interfaz que se pareciera a las ventanas del sistema operativo. Por ese motivo decidimos escoger DirectoryDialog SWT de java. Tanto mi compañero como yo trabajamos de forma equitativa en el desarrollo de esta aplicación.

Tras terminar con la subida de del profesor, nos pusimos a desarrollar la aplicación que cerraría el ciclo de ejecución del plugin Prog-Advisor: la bajada de la corrección por parte del alumno desde el subdirectorio de correcciones. Esto no fue tan complicado gracias al nuevo comando y tracker que habíamos desarrollado anteriormente para la bajada del profesor. Durante el desarrollo de esta tarea, yo tuvie ciertas complicaciones con el funcionamiento del plugin en mi ordenador. Debido a esto me centre en el avance de la memoria mientras que mi compañero se encargaba de desarrollar el código del plugin.

En esta última fase del trabajo de fin de grado, tuvimos reuniones más frecuentes con los profesores con el fin de dejar cosas claras sobre la memoria y el código. Fue en estas últimas reuniones donde decidimos mantener la herramienta externa del revisor de estilos de programación PMD.

En cuanto a la memoria, nos repartimos el trabajo. Yo me encargue de elaborar la introducción, los antecedentes y los objetivos, de explicar el funcionamiento de la arquitectura cliente-servidor de Prog-Advisor y de explicar la estructura de paquetes de nuestro plugin. A su vez, también me centré en el apartado de los sistemas utilizados y en la bibliografía.

9. Conclusions

The completion of this End-of-Degree Project has not been an easy task. It began from the already existing EasyInterface system implementation, a very complex system, which has had to be studied in detail before carrying out the implementation of the Prog-Advisor system.

Our objectives were:

- **The simple and safe upload:** we wanted to save the student time avoiding the need to himself in a virtual campus or repository to upload a practice. In addition, we wanted to provide security at the time of uploading the practice, integrating an authentication system on the server to avoid fraudulent use of the plugin.
    - These two objectives have been achieved almost completely. The upload of the practice is as simple as expected, and can be done by pressing a simple button. But the authentication system is not as secure as it was originally intended, although each teacher and student do have a username and password that they must enter to access the system.
- **Simple correction:** in the case of the teacher, we wanted to optimize the download of numerous student practices and avoid him the tedious task of having to mount one by one in their environment. We also wanted to help organize the downloads in different folders to keep track and progress of each student.
    - These objectives have also been carried out almost entirely. At first, we wanted the teacher to have the option of being able to choose between downloading all the practices of a group, or a single practice of a student that he was interested in. The first option has been implemented, but the second option has not.
- **Simple download:** it was also intended to simplify the recovery of the corrected practice by the student, making it similar to the upload, where the student would only have to press a button to recover it.
    - This objective has been achieved in its entirety, achieving the download of a correction by simply pressing a button.

10. Conclusiones y trabajo futuro

10.1. Conclusiones

La realización de este Trabajo de Fin de Grado no ha sido tarea sencilla. Se ha partido de la implementación ya existente del sistema EasyInterface, un sistema muy complejo, el cual se ha tenido que estudiar en detalle antes de llevar a cabo la implementación del sistema Prog-Advisor.

Nuestros objetivos eran:

- **La subida simple y segura**: se quería ahorrar tiempo al alumno evitando que se tuviera que identificar en un campus virtual o repositorio para subir una práctica. Además, se quería proporcionar seguridad a la hora de la subida, integrando un sistema de autenticación en el servidor para evitar un uso fraudulento del plugin.
    - Estos dos objetivos se han conseguido casi por completo. La subida de la práctica resulta tan simple como se esperaba, pudiendo realizarla pulsando un simple botón. Pero el sistema de autenticación no es tan seguro como se pretendía en un principio, aunque se ha conseguido que cada profesor y alumno dispongan de un usuario y contraseña que tienen que usar para acceder al sistema.
- **Corrección simple:** en el caso del profesor, se quería optimizar el tiempo de bajada de numerosas prácticas de alumnos y evitarle la tediosa labor de tener que montar una por una en su entorno. También se quería ayudar a organizar las bajadas en distintas carpetas para así mantener un registro y progreso de cada alumno.
    - Estos objetivos también se han podido llevar a cabo casi en su totalidad. En un principio se quería dar la opción al profesor de poder elegir entre descargarse todas las prácticas de un grupo, o una única práctica de un alumno que le interesara. Se ha conseguido implementar la primera, pero la segunda opción no ha llegado a realizarse.
- **Bajada simple:** se quería también simplificar la recuperación de la práctica corregida por parte del alumno, haciendo que como en la subida, sólo tuviera que pulsar un botón para recuperarla.
    - Este objetivo se ha conseguido al completo, consiguiendo realizar la bajada de una corrección simplemente pulsando un botón.

10.2. Trabajo futuro

Aunque se ha conseguido el flujo deseado, aún queda trabajo por hacer para perfeccionar el flujo y el control de errores que se pueden producir en el proceso. Destacan los siguientes puntos a realizar en un futuro:

- Controlar que el directorio que selecciona un profesor o un alumno para la bajada de prácticas es correcto, cosa que la versión actual no comprueba.
- Añadir más parámetros a las aplicaciones que puedan ser de utilidad para el profesor a la hora de corregir la práctica o de devolver el resultado a los alumnos, como la posibilidad de resaltar en el código devuelto por el profesor los cambios realizados en el código fuente.
- Siguiendo la línea de lo anteriormente mencionado, poder realizar estadísticas de entregas, notas, correcciones, etc. y después mostrar los resultados o las gráficas a través del plugin.
- Desarrollar una nueva aplicación para la entrega definitiva de una práctica, con la que el alumno pueda realizar una entrega única y el profesor devolverle una calificación que considere oportuna.

- Permitir que el profesor se baje una única práctica de un alumno introducido, sin necesidad de descargarse todas las prácticas subidas al servidor.
- Expandir la seguridad del sistema incluyendo una autenticación específica para cada asignatura y grupo.
- En el caso de que un profesor se descargue dos prácticas con un mismo DNI en una misma carpeta, el plugin no montará bien el proyecto en su Eclipse. Se puede mejorar esta bajada creando carpetas automáticamente con la fecha de bajada desde el plugin donde se guarden las distintas bajadas.

11. Bibliografía y referencias

[1] - Wikipedia: Entorno de desarrollo integrado (IDE):
- https://es.wikipedia.org/wiki/Entorno_de_desarrollo_integrado

[2] - Utilización y aprendizaje de plugins para Eclipse:
- http://www.vogella.com/tutorials/EclipsePlugin/article.html

[3] - Proyecto de investigación europea Envisage:
- http://www.envisage-project.eu/

[4] - Analizador de código fuente de java
- https://pmd.github.io/

[5] - Beatriz Bescós Calleja (2016). Trabajo Fin de Carrera: *EasyInterface plugin: orientando Eclipse a servicios*. Universidad Politécnica de Madrid.

[6] - Utilización de DirectoryDialog SWT:
- https://help.eclipse.org/luna/index.jsp?topic=%2Forg.eclipse.platform.doc.isv%2Freference%2Fapi%2Forg%2Feclipse%2Fswt%2Fwidgets%2FDirectoryDialog.html

[7] - JAXB: Utilización de XML con java:
http://www.vogella.com/tutorials/JAXB/article.html

[8] - Codificación en Base64
- Mediante la información del comando desde consola
  https://docs.oracle.com/javase/8/docs/api/java/util/Base64.html

[9] - Aprendizaje del lenguaje bash:
- Captura de argumentos (shift command):
  https://www.computerhope.com/unix/bash/shift.htm
- htpasswd:
  https://httpd.apache.org/docs/2.4/programs/htpasswd.html

[10] - Compresión y descompresión de archivos en formato .zip
- https://es.stackoverflow.com/questions/3082/compresi%C3%B3n-zip-directorios-extras
- https://es.stackoverflow.com/questions/23213/descomprimir-una-carpeta-desde-java

[11] - Utilización y aprendizaje de JSON:
- https://www.w3schools.com/js/js_json_intro.asp

12. Repositorio TFG Prog-Advisor

Se ha creado un repositorio en GitHub que contendrá:

- El código fuente asociado al plugin Prog-Advisor.
- Una imagen del sistema operativo Ubuntu (Linux). Dentro de este habrá un archivo de texto con las instrucciones de uso del plugin.

Enlace: https://github.com/TFG-Prog-Advisor/progadvisor