UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Desarrollo de una interfaz web para la depuración de programas Prolog

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Jorge Martín Morant

Tutor: Germán Francisco Vidal Oriola

2020/21

Resumen

Este proyecto consiste en el desarrollo de una interfaz web para el programa Rever, el cual permite la depuración reversible de código Prolog. Dicha interfaz, permitirá al usuario el uso del programa Rever sin necesidad de ningún tipo de instalación ni conocimiento previo de su funcionamiento.
Para su diseño, crearemos una estructura cliente-servidor que nos permita la comunicación entre la interfaz y un servidor con estructura REST, el cual se encargará de comunicarse con los procesos Rever.
Para la implementación del proyecto se utilizarán las tecnologías más utilizadas en el ámbito de las aplicaciones web. Entre ellas, NodeJs, PHP, HTML5, JavaScript y Bootstrap5.

Palabras clave: interfaz, JavaScript, NodeJs, HTML, interfaz web, Prolog, depuración

Abstract

This project consists of the development of a web interface for the program Rever, which allows reversible debugging of Prolog code. This interface will allow the user to use Rever without the need of any type of installation or prior knowledge of its operation.
For its design, we will create a client-server structure that allows communication between the interface and a server with a REST structure, which will be in charge of communicating with the Rever processes.
Fort he implementation of the project, the most used technologies in the field of web applications will be used. Among them, NodeJs, PHP, HTML5, JavaScript and Bootstrap5

Keywords : interface, JavaScript, NodeJs, HTML, web interface, Prolog, debuging

Tabla de contenidos

Índice de figuras

Índice de tablas

1. Introducción

Desde su creación en los 70, y su posterior desarrollo hasta la actualidad, Internet ha permitido el avance de innumerables tecnologías y servicios basados en la transmisión de información, hasta el punto de que algunas de éstas han arraigado tanto en nuestra sociedad que sería impensable para cualquier persona imaginarse una realidad en la que éstas no existieran.

Entre ellas, podríamos nombrar las más conocidas, como la mensajería instantánea, el correo electrónico, las redes sociales, las plataformas de retransmisión de audio o vídeo, ya sea en directo o en diferido, etc.

Todas las nombradas anteriormente tienen además algo en común: todas ellas suelen utilizarse junto al sistema World Wide Web, que establece los estándares de creación y transferencia de páginas web.

Las páginas web permiten a su vez la interacción de los usuarios con los servicios nombrados anteriormente, ofreciéndoles una interfaz accesible desde cualquier parte del mundo y en cualquier momento siempre que se disponga de conexión a Internet.

En este proyecto haremos uso de las tecnologías web para crear una interfaz sencilla y accesible para el depurador del lenguaje Prolog llamado Rever, del cual hablaremos en el capítulo 3 de esta misma memoria.

De esta manera, el principal objetivo del proyecto es diseñar y desarrollar dicha interfaz cumpliendo los requisitos marcados por el tutor, el cual toma el papel de cliente, y toda la infraestructura software necesaria para el funcionamiento de ésta.

En este capítulo hablaremos de la motivación que me ha llevado realizar este proyecto y de los objetivos que pretende cumplir. A continuación, haremos un breve análisis de las ventajas que ofrece dicha interfaz y definiremos la metodología a seguir. Finalmente, describiremos un breve resumen de la estructura de este documento.

1.1 Motivación

Mi motivación para la realización de este proyecto puede dividirse en 2 aspectos fundamentales, el aspecto personal, y el aspecto académico.

En cuanto al ámbito personal, este proyecto me permitía no solo hacer uso de todos aquellos conocimientos obtenidos durante la carrera, sino además centrarme en el ámbito de las Tecnologías de la Información y el desarrollo web, los cuales caen dentro de mis campos de especialización y son además mi principal interés en el ámbito de la informática.

Aparte de lo anterior, este proyecto me permitía investigar en las actuales tecnologías relacionadas con la web y aprender otras nuevas (como el lenguaje de programación Prolog empleado en el área de inteligencia artificial).

Por otra parte, en lo referente al ámbito académico, este proyecto me ha permitido profundizar en las tecnologías relacionadas con la red, permitiéndome ampliar mis conocimientos en estas, al mismo tiempo que ha hecho posible el obtener como resultado un producto funcional que ayudará a aquellos estudiantes e investigadores que quieran hacer uso de Rever como herramienta para su investigación o como apoyo para la demostración de ejemplos en otros trabajos académicos.

1.2 Objetivos

Desde la creación de los entornos de usuario, las interfaces han permitido la visualización de datos y el control de programas de una manera sencilla. Además, las interfaces web añaden la ventaja de que éstas sean mucho más accesibles.

Por ello, el principal objetivo del proyecto es el diseño de una interfaz web siguiendo los requisitos marcados por el tutor, que como ya hemos comentado anteriormente podemos considerarlo el cliente final de este desarrollo.

Dicha interfaz habrá de cumplir los siguientes requisitos:

- Deberá adaptarse a cualquier monitor o dispositivo sin importar su tamaño, sin que se esto reduzca su usabilidad; es decir, debe ser *responsive*.
- Deberá permitir realizar todas las acciones implementadas en Rever; es decir, no debe reducir la funcionalidad original del depurador.
- Deberá ser intuitiva y fácil de usar.
- Deberá poder usarse en varios idiomas (español e inglés).
- Deberá disponer de ejemplos para probar el depurador.
- Deberá permitir al usuario descargar el programa depurado y cargar programas propios desde un archivo.

Por otra parte, para que la interfaz funcione necesitará un servidor al cual enviar las peticiones y recibir la información procesada.

Por ello, se desarrollará una parte que actúe como servidor y que cumpla los siguientes requisitos:

- Deberá comunicarse con la interfaz y enviar los datos que ésta solicite.
- Deberá comunicarse con el programa Rever para poder transmitirle los datos y recibir las respuestas procesadas.
- Deberá crear, eliminar y modificar archivos temporales con los datos de la depuración de manera segura.
- Deberá poder manejar varios usuarios al mismo tiempo.
- Deberá ofrecer la interfaz web sobre el puerto 80, al igual que otros servidores web.

Además de todo lo anterior, se desarrollará una estructura para los mensajes que utilizan el cliente y el servidor para comunicarse, y se estudiarán que implementaciones serán necesarias para asegurar la seguridad de la interfaz y del servidor a nivel informático.

1.3 Impacto Esperado

Se espera que este proyecto aumente el uso de Rever como depurador Prolog, dado que permitirá un mejor acceso y control de este.

Por ello, ayudará a usuarios del lenguaje de programación Prolog a depurar sus programas sin necesidad de instalaciones ni complicaciones a la hora de aprender el funcionamiento de Rever.

Además, permitirá a aquellos usuarios que usen Rever con objetivos académicos a la hora de comprobar el funcionamiento de sus ejemplos. A estos usuarios, al igual que a los usuarios nombrados al inicio del apartado, les ayudará en el uso de Rever sin necesidad de instalaciones ni aprendizajes previos de cómo usar el programa.

1.4 Metodología

Dado que el proyecto tiene como objetivo principal el desarrollo de una aplicación software que actúe como interfaz gráfica web para Rever, seguiremos una de las metodologías estudiadas en la rama de ingeniería del software.

La metodología elegida en este caso es la metodología ágil conocida como Scrum [12.].

Esta decisión se debe a que dicha metodología se adapta perfectamente al proyecto que se ha llevado a cabo dado que permite tratar cada uno de los objetivos en iteraciones separadas, en las cuales se comparte con el cliente el avance del desarrollo.

En cada una de las iteraciones se seleccionará uno de los objetivos o un conjunto de estos, se decidirá la mejor manera de completar dichos requisitos, se realizará un *Sprint* para desarrollarlos y finalmente se realizarán las pruebas necesarias y se le presentará dicha implementación al cliente.

Sin embargo, no seguimos estrictamente todas las características de dicha metodología, y la causa de esto es que no contamos con un equipo de desarrollo, sino con una única persona que realizará dicho proyecto. Por ello, no serán necesarias las reuniones de planificación con el equipo, pero sí sesiones de planificación para decidir cómo completar el desarrollo de dicha iteración.

Cada ciclo se realizará en un periodo de dos a tres semanas en función de la dificultad y cantidad de requisitos a completar.

Más adelante en el capítulo cuatro (Análisis del problema) se mostrarán cada una de las iteraciones llevadas a cabo y los resultados de éstas.

1.5 Estructura

A continuación, definiré la estructura de este documento con una pequeña explicación de cada uno de sus capítulos:

- Introducción: Se trata de este mismo capítulo. Se presenta brevemente el proyecto, las motivaciones tras éste, los objetivos generales, el impacto que se espera tras su desarrollo, la metodología a seguir y se define la estructura general de este documento.
- Estado del arte: Se definirá la situación actual de cada una de las tecnologías aplicadas al proyecto, su desarrollo a lo largo de los años y se estudiará si existe alguna solución existente al problema.
- Rever: Se mostrará un pequeño resumen de que es Rever y cómo funciona, con el objetivo de tener un contexto claro de la funcionalidad de dicho programa.
- Análisis del problema: En primer lugar, se realizará una reflexión del problema a resolver y se especificarán los requisitos necesarios para el cumplimiento de los objetivos. Tras esto, se analizarán las posibles soluciones y se expondrá la solución elegida. Finalmente, se definirá un plan de trabajo acorde a la metodología escogida.
- Diseño: Se detallará la solución propuesta junto a todos los detalles y características de ésta. Se definirá el diseñó de la interfaz de usuario, la arquitectura del sistema y aquellas herramientas y tecnologías con las que se implementará el proyecto.
- Desarrollo: En este apartado se explicará el funcionamiento y estructura de la solución final del proyecto.
- Implantación: Se expondrá una guía de cómo instalar todo lo necesario para el funcionamiento de la interfaz. A continuación, se mostrará además un pequeño tutorial de utilización del resultado final.

- Pruebas: Se presentarán las pruebas realizadas al resultado del desarrollo para comprobar su correcta funcionalidad.
- Conclusiones: Finalmente, en este apartado, se reflexionará sobre el resultado obtenido, si este cumple los objetivos acordados, así como las posibles mejoras o cambios para futuras implementaciones.

2. Estado del arte

En este apartado estudiaremos la situación actual de las tecnologías relacionadas con nuestro proyecto. Tras esto, analizaremos si hay alguna solución existente que resuelva el problema que nos plantea nuestro proyecto.

2.1 Tecnologías actuales

En primer lugar, separaremos en dos ámbitos las tecnologías relacionadas con nuestro proyecto. Por un lado, analizaremos las tecnologías relacionadas con Prolog. Y, por otro lado, investigaremos las relacionadas con las aplicaciones web.

2.1.1 Tecnologías relacionadas con Prolog

Prolog

Prolog es un lenguaje de programación del paradigma de programación lógica, el cual es un paradigma de programación que se encuentra dentro del conjunto de paradigmas declarativos. Prolog está basado en la lógica de predicados [1.][4.].

La sintaxis de Prolog en términos de las llamadas "cláusulas", las cuales pueden ser "hechos" o "reglas".

En cuanto a los hechos, estos definen la información que tiene el programa; por ejemplo, "*perro(tobby)*" o "*mascota(tobby,jose)*". La primera define que *tobby* es un perro, mientras que la segunda define que *Tobby* es mascota de Jose.

Las reglas definen relaciones entre datos. Por ejemplo:

"*pasea(X,Y) :- perro(Y), mascota(Y,X)*"

Esta regla indicaría que X pasea a Y siempre que Y sea un perro y, además, Y sea mascota de X. Como podemos ver, la parte izquierda de la regla se cumple siempre que todas las llamadas de la parte derecha tengan éxito.

Además, cualquier conjunto de caracteres empezado por mayúscula se considerará como una variable, mientras que aquellos comenzados por minúscula se considerarán funciones o constantes.

Una vez definimos los hechos y las reglas, podremos realizar preguntas al programa. Por ejemplo, "*?- perro(Y).*" nos devolverá todos los valores posibles de Y.

Aunque se inventó en la década de los 70, su uso actual se ha incrementado gracias al auge del desarrollo de tecnologías como la inteligencia artificial, el *machine learning*, el procesamiento de lenguaje natural, etc. Esto se debe al uso extendido de Prolog en estos campos.

SWI-Prolog

SWI-Prolog es una implementación de código abierto del lenguaje de programación Prolog, con multitud de bibliotecas, herramientas y un entorno de trabajo que permite tanto la edición de código como la ejecución de este.

Además, permite que los programas Prolog desarrollados sean compatibles con múltiples plataformas, dado que SWI-Prolog se puede ejecutar sobre Linux, Windows, MacOS, Android, etc.

En su página web podemos encontrar todo lo relacionado con SWI-Prolog y la programación en Prolog, como tutoriales, documentación, foros, etc. [11.]

2.1.2 Tecnologías relacionadas con las aplicaciones web

Dado que nuestro objetivo principal es crear una interfaz web para Rever, analizaremos las tecnologías actuales relacionadas con las aplicaciones web.

HTML5

Se trata de la quinta versión del lenguaje de marcas HTML (*Hypertext Markup Language*), el cual es el lenguaje utilizado para la creación de páginas web dentro de la WWW (*World wide web*).

A partir de octubre de 2014 el W3C (*World Wide Web Consortium*) publicó la versión definitiva de esta quinta revisión y HTML5 paso a convertirse en el estándar. Podemos encontrar sus especificaciones en la página web de recopilación de estándares *WHATWG* [8.].

HTML permite la definición de la estructura de una página web y los elementos que la componen mediante una serie de etiquetas delimitadas por los caracteres “<” y “/>”. Entre estas etiquetas podemos encontrar elementos como tablas, párrafos, encabezados, imágenes, vídeos, listas, enlaces, etc.

HTML5 presenta una serie de mejoras y nuevos elementos frente su antecesor, las cuales se adaptan a las webs actuales, como por ejemplo etiquetas para el manejo de grandes conjuntos de datos, una mejora en los formularios, funcionalidades como el arrastre de imágenes y la incorporación de etiquetas para la reproducción de contenidos multimedia.

A parte de la definición de la estructura, HTML permite la inclusión de hojas de estilo CSS y del lenguaje JavaScript para la implementación de la lógica en las páginas web.

Las hojas de estilo CSS (*Cascading Style Sheets*) permiten la definición del diseño visual de los documentos HTML mediante la definición de atributos de los elementos encontrados en dichos ficheros. La separación entre los ficheros CSS y HTML se debe a que un único fichero CSS puede definir el estilo de múltiples ficheros HTML.

La sintaxis de CSS se basa en bloques formados por un selector, el cual indicará a que elementos se aplica el bloque, y un conjunto de declaraciones que asignarán valores a los atributos de los elementos HTML mediante tuplas “atributo:valor”.

El estándar de CSS también es definido por W3C y se puede encontrar en su propia página web [3.]

JavaScript

El lenguaje de programación interpretado JavaScript es un lenguaje orientado a objetos, débilmente tipado y dinámico [7.].

Este lenguaje es ampliamente utilizado debido a que los navegadores web soportan desde 2012 el lenguaje de programación ECMAScript 5.1, del cual JavaScript es un dialecto. Desde 2016 pasó a utilizarse la versión 6 de este lenguaje.

Esta tecnología permite a los navegadores web modificar dinámicamente la estructura de la página, crear nuevos elementos, eliminarlos, modificar sus atributos o enviar y recibir información de algún servidor mediante peticiones HTTP.

NodeJs

Se trata de un entorno de ejecución de JavaScript construido mediante el motor de JavaScript V8 de Chrome.

Según podemos encontrar en su página oficial, está orientado a eventos asíncronos y está diseñado para crear aplicaciones de red escalables. Además, contrasta con los demás sistemas concurrentes comunes hoy en día por el hecho de que no emplea hilos, ya que las redes basadas en hilos son relativamente ineficientes y difíciles de usar.

El desarrollo de NodeJs fue muy influenciado por entornos con un propósito similar como EventMachine de Ruby o Twisted de Python.

Además NodeJs incluye varios módulos básicos como el modulo de red, el del sistema de ficheros, el de temporizadores, y otros módulos que al igual que los anteriormente nombrados le permiten interactuar con el sistema operativo.

En la página oficial de NodeJs podemos encontrar una gran cantidad de documentación sobre su uso y sobre sus librerías [6.].

PHP

El lenguaje de programación de código abierto PHP (*Hypertext Preprocessor*) es muy popular en el desarrollo de aplicaciones web. Esto se debe principalmente al hecho de que, tal y como indica su nombre, las páginas web son preprocesadas antes de enviarse al cliente.

Esto permite crear paginas personalizadas en función del usuario que las requiera o la petición de datos a un servidor web y su implementación sobre la página de manera transparente para el usuario.

Dicho código php se encuentra incrustado dentro de los ficheros HTML entre las etiquetas especiales "<*?php*" y "*?*>", lo cual permitirá que sea detectado por nuestro servidor web, el cual lo procesará, y lo servirá al cliente.

Una gran ventaja de PHP es que tiene la capacidad de ser ejecutado en la mayoría de los servidores web mediante módulos adaptados para este fin.

Además PHP puede ser utilizado como un lenguaje de programación común y ser ejecutado fuera de un servidor [9.].

Bootstrap 5

Se trata de un *framework* de código abierto para el desarrollo de páginas web que facilita el diseño gráfico de la web mediante plantillas de diseño con tipografía, formularios, botones, cuadros, menús, y otros elementos.

Fue inicialmente desarrollado bajo el nombre *Blueprint* de *Twitter*, pero en 2011 esta compañía lo liberó como código abierto.

Además de las plantillas comentadas anteriormente, Bootstrap permite la creación de diseños *responsive* mediante un sistema de cuadriculas que divide la pantalla en 12 columnas cuyo ancho es relativo al ancho de la pantalla.

En esta última versión se ha eliminado la dependencia de la librería de JavaScript "*jQuery*" por lo que ya no contiene dependencias a ninguna librería y funciona completamente sobre JavaScript y CSS.

Podemos encontrar tutoriales, documentación y ejemplos en la página web oficial de Bootstrap [5.].

2.2 Soluciones existentes

Tras una extensiva búsqueda no hemos logrado encontrar ninguna solución existente al problema que nos concierne. No parece haber ningún proyecto previo que aborde la creación de una interfaz web para una aplicación Prolog.

Esto puede deberse a que el propio SWI-Prolog implementa librerías de conexión TCP, por lo que los desarrolladores que se propongan realizar una aplicación web pueden optar por diseñar el programa directamente para ese propósito.

En nuestro caso, la modificación de Rever está descartada, dado que nuestra intención es modificar en lo mínimo posible el propio programa original.

3. Rever

Para poder desarrollar una interfaz gráfica para Rever primero debemos entender qué es y cómo funciona.

Rever es un programa de depuración reversible escrito en Prolog, desarrollado por el investigador Germán Francisco Vidal Oriola, el cual es además el tutor de este proyecto.

Dicho programa se presenta en el trabajo "*Reversible Debugging in Logic Programming*" [14.], el cual a fecha de desarrollo de este proyecto se encuentra en espera de ser publicado. Dicho artículo extiende los resultados obtenidos en el artículo "*Reversible computations in logic programming*" [13.] del mismo autor.

El objetivo de Rever es mostrar la utilidad de la depuración reversible en el campo de los lenguajes de programación lógicos, y define un posible modelo de depuración reversible.

Primero, realizaremos un pequeño resumen del artículo en cuestión y, tras ello, nos centraremos en el funcionamiento de Rever desde el punto de vista del usuario.

En cuanto al resumen, no entraremos en detalle, dado que el cumplimiento de los objetivos de este proyecto no necesita de una comprensión profunda del funcionamiento de Rever ni de la semántica utilizada para ello. Si se quiere profundizar más en el tema, recomendamos la lectura del propio artículo una vez publicado.

3.1 Resumen del artículo

En primer lugar, se nos muestra como la depuración reversible es útil en cualquier tipo de lenguaje de programación, permitiéndonos encontrar el punto exacto de fallo en nuestro programa retrocediendo mediante una depuración inversa desde el problema observado. Además, nos ahorra tiempo ya que nos permite realizar dicha depuración sin necesidad de ejecutar el programa entero de nuevo paso a paso desde el inicio de su ejecución.

Sin embargo, en la programación lógica nos encontramos con dos problemas exclusivos de este tipo de lenguajes: el indeterminismo y la unificación.

Para resolver dichos problemas, el artículo contribuye con:

- La definición de una semántica operacional para programas lógicos, la cual permite tanto avanzar como retroceder pasos en su ejecución.
- La declaración y prueba de propiedades formales para dicha semántica, demostrando además que es determinista y, por ello, cualquier paso computacional puede ser revertido.
- La presentación de un diseño de depurador reversible para Prolog, Rever, basado en la semántica presentada.

Dicha semántica define estados de la ejecución, los cuales contienen la información necesaria para la continuación de la ejecución además de para poder revertir el último paso dado. Además, cada estado se marca con una etiqueta en función del tipo de estado en el que se encuentre (*backtrack, next, choice, choice_fail, unfold*).

Tras la definición de dicha semántica se exponen y prueban los resultados que demuestran que dicha semántica es reversible.

En cuanto a Rever y su funcionamiento, se explica que el diseño de Rever está basado en el estándar *4-port tracer* introducido por Byrd[2.][4.], con los *ports* "*call*", "*exit*", "*redo*" y "*fail*"; y funciona mediante la implementación de la semántica presentada en el mismo artículo.

3.2 Funcionamiento de Rever

En primer lugar, deberemos iniciar SWI-Prolog, e incluir tanto el programa Rever como el código a depurar mediante las cláusulas "*[rever].*" y "*[code]*", siendo *code* el nombre del fichero a depurar. Estos pasos los podemos observar en la figura 3.2.1

Figura 3.2.1 – Inicio del programa Rever

Según muestra el artículo, Rever puede ser ejecutado en dos modos:

- *"Debug mode"*: Este modo ejecutará el programa sin mostrar ninguna información hasta que este no alcance el predicado *"rtrace/0"* o una excepción, y en este momento se mostrará dicha información. Se puede iniciar mediante la llamada *"rdebug(query)."*.
- *"Trace mode"*: En este modo se mostrará cada uno de los pasos de la ejecución y se permitirá al usuario avanzar o retroceder en ésta. Se puede iniciar mediante la llamada *"rtrace(query)."*.

En ambos modos, los controles sobre la ejecución son los siguientes:

- Tecla flecha abajo (↓): Avanza un paso en la ejecución.
- Tecla flecha arriba (↑): Retrocede un paso en la ejecución.
- Tecla "q": Finaliza la ejecución.
- Tecla "s": Omite la depuración, mostrando todos los pasos posteriores hasta el final de esta.

Finalmente, cabe destacar que Rever está diseñado para depurar programas relativamente pequeños, por lo que no hay garantías de que escale a aplicaciones de tamaño medio o grande.

4. Análisis del problema

A lo largo de este capítulo analizaremos el problema que pretendemos resolver mediante este proyecto, tras lo cual, listaremos los requisitos necesarios para el cumplimiento de los objetivos marcados. A continuación, estudiaremos posibles soluciones y expondremos la solución elegida. Finalmente, se definirá el plan de trabajo a seguir.

La definición del problema que encontramos en este proyecto es la siguiente: Debemos diseñar e implementar una interfaz web que permita el control del programa de depuración Rever para el lenguaje Prolog. Al tratarse de una interfaz web necesitaremos además un servidor intermedio que permita a la interfaz realizar las peticiones correspondientes, y que envíe a esta los datos del programa. Además, también necesitaremos un servidor web que permita a los usuarios recibir la interfaz y todos los recursos necesarios para su funcionamiento.

En primer lugar, especificaremos los requisitos del sistema para más adelante poder centrarnos en posibles soluciones que cumplan dichos requisitos.

4.1 Especificación de requisitos

Dividiremos los requisitos de nuestro sistema en funcionales y no funcionales. Los requisitos funcionales son aquellos que definen los servicios que ofrecerá nuestro sistema, mientras que los no funcionales son aquellos que se refieren a las propiedades del sistema.

4.1.1 Requisitos funcionales

RF-01	Inicializar usuario
Descripción	Guardar los datos introducidos en la interfaz (código del programa y objetivo inicial) para su posterior ejecución.

Tabla 4.1.1 - Inicializar usuario

RF-02	Iniciar depuración
Descripción	Iniciar Rever con los datos del usuario y comunicarlo a la interfaz.

Tabla 4.1.2 – Iniciar depuración

RF-03	Siguiente paso
Descripción	Ejecutar el siguiente paso de la depuración y transmitir la información a la interfaz.

Tabla 4.1.3 – Siguiente paso

RF-04	Paso anterior
Descripción	Volver al paso anterior de la depuración y transmitir la información a la interfaz.

Tabla 4.1.4 – Paso anterior

RF-05	Omitir
Descripción	Omitir todos los pasos de la depuración y transmitir la información a la interfaz.

Tabla 4.1.5 - Omitir

RF-06	Finalizar depuración
Descripción	Finalizar la depuración actual y borrar los datos del usuario referentes a ésta.

Tabla 4.1.6 – Finalizar depuración

RF-07	Cargar programa
Descripción	Cargar un programa Prolog desde un fichero.

Tabla 4.1.7 – Cargar programa

RF-08	Descargar programa
Descripción	Descargar el programa Prolog a analizar en un fichero.

Tabla 4.1.8 – Descargar programa

RF-09	Cargar ejemplo
Descripción	Cargar programas ejemplo para analizar.

Tabla 4.1.9 – Cargar ejemplo

RF-10	Descargar traza
Descripción	Descargar en un fichero la traza de la depuración.

Tabla 4.1.10 – Descargar traza

RF-11	Enviar interfaz web
Descripción	Recibir peticiones web y enviar la interfaz.

Tabla 4.1.11 – Enviar interfaz web

RF-12	Cambiar idioma
Descripción	Mostrar la interfaz en el idioma seleccionado

Tabla 4.1.12 – Cambiar idioma

4.1.2 Requisitos no funcionales

Como requisitos no funcionales encontramos los siguientes:

- La interfaz debe ser *responsive*; es decir, debe adaptarse a toda la gama de tamaños de pantalla existentes sin que reduzca su usabilidad.
- El usuario no podrá crear ningún archivo en el servidor excepto el que contenga el programa Prolog. Además, el usuario no podrá decidir el nombre del archivo ni su localización, asegurando así la seguridad del servidor en este aspecto.
- La interfaz debe ser sencilla e intuitiva, el usuario debe ser capaz de utilizarla sin necesidad de entrenamiento.

4.2 Posibles soluciones

La resolución del problema que nos presenta este proyecto, el crear una interfaz web para Rever, es equivalente en nuestro caso a convertir Rever en una aplicación web. Por ello, ambas posibles soluciones utilizarán el esquema común de la mayoría de estas aplicaciones, una parte ejecutada sobre el navegador del cliente, y otra sobre el servidor.

Una opción descartada desde un inicio proponía ejecutar Rever directamente sobre el navegador del cliente, utilizando para ello una librería de ejecución de código Prolog sobre javascript. Esta idea se desechó rapidamente debido a que conllevaba un rediseño del programa Rever.

En cuanto a las posibles soluciones, nos encontramos ante dos opciones diferenciadas por el tipo de conexión que el servidor de la aplicación y la interfaz utilizarán para comunicarse. Por un lado podemos mantener la conexión durante toda la depuración del programa y, por otro, podemos utilizar una conexión distinta para cada uno de los pasos de la depuración.

En este apartado, explicaremos ambas soluciones y analizaremos sus ventajas e inconvenientes.

4.2.1 Solución 1 – Conexión persistente

En esta primera solución consideramos una única conexión entre el cliente y el servidor, que se mantendrá activa por cada cliente desde que dicho cliente inicia la depuración y hasta que la finaliza.

Las ventajas de esta opción se resumen en que evita que el cliente y el servidor tengan que crear una conexión por cada petición, lo que ahorraría al servidor y al cliente el tiempo que se tarda en crear e iniciar la conexión con protocolo TCP.

Por otra parte, nos encontraríamos con la desventaja de que sería necesario implementar controles que verificaran que la conexión sigue activa, tanto por el lado del cliente como del servidor. Esto se debe a que si en algún momento el cliente sufriera algún problema como un apagón o un cierre inesperado del navegador en ningún momento finalizaría la conexión con el servidor, y el servidor se quedaría con una conexión inútil ocupando recursos. Si por el contrario el que sufriera el problema fuera el servidor, los clientes no se enterarían de que esto ha ocurrido, generando un error al intentar enviar cualquier mensaje por dicha conexión.

4.2.2 Solución 2 – Conexión no persistente

Esta segunda opción creará una conexión TCP por cada una de las peticiones que los clientes realicen al servidor. Estas peticiones pueden ser para iniciar la depuración, finalizarla, hacerla avanzar, etc.

Como ventaja principal, esta solución permite ahorrar recursos del servidor, dado que solo ocupará los recursos necesarios para contestar a cada petición y, tras completar dicha tarea, liberará los recursos utilizados.

Como inconveniente, nuestro servidor y el cliente tendrán que crear una nueva conexión por cada petición, por lo que si el servidor recibe muchas peticiones en poco tiempo, este gastaría más recursos que con la solución 1.

4.3 Solución elegida

En nuestro caso se espera que la cantidad de usuarios activos al mismo tiempo sea reducido. Por ello, la unica diferencia notable entre las dos soluciones es el hecho de que la primera opción requeriría implementar código que compruebe si las conexiones siguen activas, mientras que en la segunda opción no será necesario.

En definitiva, la solución elegida es la segunda; crearemos una conexión distinta por cada petición realizada al servidor. Para ello, necesitaremos en primer lugar la interfaz de usuario, en segundo lugar un servidor web para transmitir la interfaz de usuario a los clientes, y finalmente el servidor de la aplicación y una manera que éste se comunique con el programa Rever, lo cual conseguiremos mediante una interfaz que conecte el servidor con Rever.

Así pues, necesitaremos implementar 4 bloques principales:

- La interfaz de usuario web, que permitirá al usuario controlar el programa.
- El servidor web, el cual distribuirá la interfaz web.
- El servidor, al cual llegarán las peticiones, las resolverá y transmitira la respuesta a la interfaz de usuario.
- La interfaz servidor-Rever, que servira de intermediario entre el servidor REST y los procesos Rever.

Finalmente, cabe destacar que esta solución (con pequeñas modificaciones) nos permite crear una interfaz web para cualquier aplicación de consola, dado que dicha solución simula la interacción del usuario mediante la interfaz servidor-proceso.

De este modo, si modificamos la interfaz de usuario, la interfaz servidor-proceso, y la interpretación de peticiones por parte del servidor, para que pueda realizar todas las acciones de las que dispone un usuario en consola, podremos ajustar esta solución para cualquier programa de consola existente.

4.4 Planificación de trabajo

Siguiendo la metodología Scrum, realizaremos múltiples ciclos que durarán una o dos semanas en función de los objetivos a cumplir, intentando en cada uno de estos implementar uno o varios requisitos. En el caso de que al presentar la implementación al cliente este decida que es necesario algún cambio, este se realizará en un ciclo más corto de una única semana, y se volverá a presentar al cliente. Una vez el cliente esté de acuerdo con la implementación, se continuará con el siguiente conjunto de requisitos.

Los ciclos que cumplir en cada uno de los ciclos son los siguientes:

Ciclo 1	Creación de la interfaz servidor-Rever
Objetivos	Crear una interfaz de comunicación que permita: • Crear procesos Rever. • Transmitir información de entrada al proceso. • Recibir datos procesados por el proceso. • Cerrar procesos Rever. En esta primera iteración tanto el código a depurar como el objetivo inicial estarán prefijados.
Duración	Dos semanas.

Tabla 4.4.1 – Creación de la interfaz servidor-Rever

Ciclo 2	Ampliación de la interfaz servidor-Rever
Objetivos	Modificar la interfaz para que sea capaz de iniciar un proceso Rever dados un objetivo inicial y la localización de un archivo con el código a depurar.
Duración	Una semana

Tabla 4.4.2 – Ampliación de la interfaz servidor-Rever

Ciclo 3	Creación del servidor
Objetivos	Crear un servidor HTTP que reciba peticiones y las procese utilizando la interfaz servidor-Rever. El servidor debe ser capaz de: - Iniciar una depuración. - Ejecutar el siguiente paso de la depuración. - Ejecutar el paso anterior. - Omitir una depuración. - Finalizar una depuración. Además de estas funciones, el servidor debe enviar el resultado de cada una al cliente que realizó la petición. En esta iteración tanto el código a depurar como el objetivo inicial estarán prefijados.
Duración	Dos semanas

Tabla 4.4.3 – Creación del servidor REST

Ciclo 4	Ampliación del servidor
Objetivos	Ampliar el servidor para que sea capaz de: - Manejar múltiples depuraciones identificadas por una id de usuario única para cada cliente. - Crear un archivo con el código Prolog enviado por el cliente. Este archivo tendrá un nombre fijado por la id de usuario. - Iniciar una depuración dado un objetivo inicial y la id de usuario.
Duración	Dos semanas

Tabla 4.4.4 – Ampliación del servidor REST

Ciclo 5	Servidor web
Objetivos	Buscar un servidor web existente y configurarlo o implementar uno de ser necesario, para poder servir la interfaz de usuario al cliente.
Duración	Una semana

Tabla 4.4.5 – Servidor web

Ciclo 6	Diseño de la interfaz web de usuario
Objetivos	Diseñar un mockup de la estructura de la interfaz web y reunirse con el cliente para estudiar posibles cambios.
Duración	Una semana

Tabla 4.4.6 – Diseño de la interfaz web de usuario

Es muy posible que el ciclo 6 sea necesario repetirlo múltiples veces, dado que es necesario asegurarse de que el cliente esta conforme con la distribución de la interfaz antes de empezar su desarrollo.

Ciclo 7	Desarrollo del aspecto gráfico de la interfaz
Objetivos	Instalar Bootstrap 5 y desarrollar el aspecto gráfico de la interfaz siguiendo los prototipos creados en el ciclo anterior.
Duración	Dos semanas

Tabla 4.4.7 – Desarrollo del aspecto gráfico de la interfaz

Ciclo 8	Desarrollo del aspecto lógico de la interfaz
Objetivos	Desarrollar toda la lógica necesaria para que la interfaz se comunique con el servidor. La interfaz debe ser capaz de: - Enviar el código a depurar y el objetivo inicial para iniciar la depuración. - Realizar la petición del siguiente paso de la depuración y mostrar el resultado. - Realizar la petición del anterior paso de la depuración y mostrar el resultado. - Realizar la petición para omitir la depuración y mostrar el resultado. - Realizar la petición para finalizar la depuración. - Cambiar el idioma de la interfaz. - Descargar el código a depurar a un archivo. - Cargar el código a depurar desde un archivo. - Descargar la traza de la depuración a un archivo.
Duración	Dos semanas

Tabla 4.4.8 – Desarrollo del aspecto lógico de la interfaz

La duración total del proyecto según la planificación será de 13 semanas, pero debemos tener en cuenta que es posible que sea necesario repetir varios ciclos. Por ello, la planificación podría durar entre 2 y 4 semanas extra, lo cual consideraremos dentro de plazo.

5. Diseño

En este capítulo hablaremos del diseño de la solución elegida en el capítulo anterior. En primer lugar, mostraremos la arquitectura del sistema y tras esto detallaremos cada una de las partes que lo componen, comentando además aquellas tecnologías que utilizaremos para su desarrollo.

5.1 Arquitectura del sistema

En cuanto a la plataforma o sistema operativo, diseñaremos nuestro proyecto para funcionar sobre Ubuntu 20.04, aunque funcionará en cualquier Linux que contenga el mismo conjunto de llamadas al sistema que Ubuntu.

La arquitectura de nuestro proyecto constará de cuatro componentes:

- Servidor web: Este servidor se encargará de distribuir la interfaz y todos los archivos necesarios para su funcionamiento a cualquier cliente que acceda al puerto 80 del computador que estemos usando como servidor.
- Servidor REST: Su trabajo será recibir las peticiones de la interfaz de usuario, comunicarse con el proceso Rever correspondiente mediante la interfaz servidor-Rever, y enviar los resultados de vuelta a la interfaz de usuario.
- Interfaz servidor-Rever: Permitirá al servidor REST crear, eliminar y comunicarse con procesos del programa Rever, además de crear y eliminar los ficheros con el código a depurar.
- Interfaz de usuario: Permitirá al usuario manejar el programa Rever con transparencia, de una manera sencilla y visual. Esta se comunicará con el servidor REST para enviar peticiones y recibir la información de la ejecución.

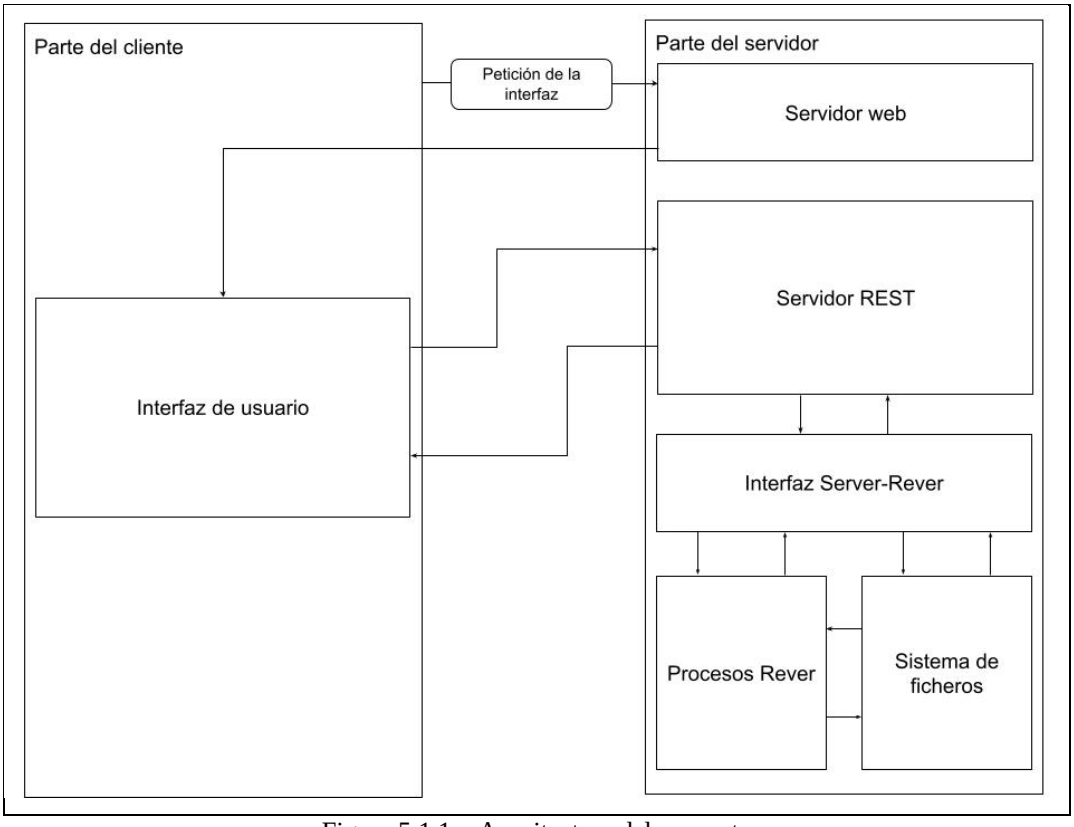

Figura 5.1.1 – Arquitectura del proyecto

Como podemos ver en la figura 5.1.1, el cliente realizará una petición HTTP a nuestro servidor web, el cual le enviará la interfaz de usuario. Tras esto dicha interfaz se comunicará con el servidor REST, el cual utilizará la interfaz Server-Rever para interactuar con los procesos Rever y con el sistema de archivos del sistema operativo.

5.2 Servidor web

Antes de diseñar e implementar un servidor web, decidimos estudiar las opciones existentes de código abierto, y dado que existen múltiples opciones que realizan las funciones que necesitamos, decidimos utilizar uno de estos como servidor web.

Netcraft realizó un estudio en febrero de 2021 en el que se investigó que servidores web eran los más utilizados en la actualidad. Tras analizar la respuesta de 1.204.252.411 páginas distribuidas en 263.042.054 dominios se descubrió que aproximadamente el 85% de páginas analizadas utilizaban uno de los siguientes 6 servidores web:

Nombre	%	¿Código abierto?	Sistema operativo
Nginx	35	Si	Linux, Windows
Apache	26	Si	Linux, Windows
Microsoft	7	No	Windows
OpenResty	6	Si	Linux, Windows
Cloudflare Server	5	No	Servicio en la nube
Google	4	No	Servicio en la nube

Tabla 5.2.1 – Cuota de mercado de servidores web

Como podemos observar en la tabla anterior, las únicas opciones de código abierto para sistemas operativos Linux (entre los cuales se encuentra Ubuntu) son Nginx, Apache y OpenResty. Sin embargo, Netcraft realiza también en el mismo estudio un análisis de mercado marcando las páginas a las que denomina “activas”.

Netcraft considera activas todas aquellas páginas que no compartan una estructura similar bajo una misma IP. Esta distinción se debe a que multitud de empresas que ofertan dominios crean paginas por defecto muy similares que solo se diferencian por el texto que muestra el nombre del dominio, el nombre del propietario, la fecha de registro, etc.

Por ello, para eliminar dichas páginas de la lista se realiza este segundo análisis, que muestra un punto de vista más certero sobre el uso de los distintos servidores web.

En dicho análisis, Apache ocupa un 25% de la cuota de mercado, mientras que Nginx cae a un 20% y OpenResty a un 3%. Lo que nos da a entender que el 35% de cuota de mercado visto anteriormente por parte de Nginx se debe al uso de este por parte de compañías de venta de dominios web para promocionar su producto mediante el uso de webs autogeneradas.

Finalmente, decidimos que utilizaríamos el servidor web Apache, dado que es el más usado según el estudio de Netcraft; y por su facilidad de instalación y configuración, además del conocimiento previo y práctica de uso del equipo de desarrollo con dicho servidor web.

5.3 Servidor REST

Se trata del servidor de la aplicación comentado en el capítulo anterior, el cual se encargará de recibir las peticiones de la interfaz y enviarle el resultado de la ejecución del programa Rever.

La denominación REST viene dada por el hecho de que seguirá el estilo de arquitectura software denominado REST; es decir, el servidor quedará a la escucha sobre un puerto definido, abrirá una conexión TCP por cada petición recibida, contestará a dicha petición y cerrará la conexión. Tanto para las peticiones recibidas como para las respuestas del servidor se utilizará el protocolo HTTP tal y como queda especificado por la arquitectura REST.

Por otra parte, en cuanto al lenguaje de programación para el desarrollo e implementación del servidor, estudiaremos cuales son los lenguajes más conocidos en el entorno de servidores, y de estos decidiremos el que mejor se adapte a nuestro proyecto.

Entre los lenguajes más comunes a la hora de desarrollar servidores encontramos Javascript, Python, Java y PHP, de los cuales hablamos en el capítulo 2.

Tras investigar las características de cada uno de ellos, decidimos que la implementación del servidor se realizará en Javascript mediante el entorno de ejecución Node. Esa elección se debe a que la naturaleza asíncrona de JavaScript permite implementar servidores de una manera sencilla y eficiente.

En cuanto al diseño del servidor, primero especificaremos la estructura de los mensajes de petición y de los de respuesta que recibirá y transmitirá. Estos mensajes serán objetos de JavaScript enviados como cadena de caracteres en formato JSON, y su estructura será la siguiente:

Para las respuestas:

Nombre del atributo	**Descripción**
output	Salida estándar del programa.
error	Salida de error del programa.

Tabla 5.3.1 – Estructura de las respuestas

Para las peticiones:

Nombre del atributo	**Descripción**
id	ID del cliente que realiza la petición.
cmd	Instrucción de la petición.
code	Código a depurar.
ini_obj	Objetivo inicial.

Tabla 5.3.2 – Estructura de las peticiones

Una vez definida la estructura de los mensajes, definiremos el diseño a implementar del servidor REST. Dicho servidor se programará en un único ejecutable JavaScript. En primer lugar iniciaremos el servidor, y tras esto filtraremos las peticiones que le lleguen en función del método HTTP de estas. Entre los métodos HTTP definidos por el estándar encontramos los métodos GET, POST, HEAD, OPTIONS, PUT, DELETE, TRACE y CONNECT [].

Dado que nuestro objetivo es realizar peticiones enviando nuestra propia estructura de mensajes en el campo de datos, utilizaremos únicamente el método POST, y haremos que el servidor responda a las peticiones de los demás métodos con un mensaje de error.

Tras filtrar las peticiones de tipo POST, analizaremos los atributos del mensaje para identificar la id del cliente y qué instrucción debe procesar, y mandará la información

correspondiente al proceso Rever de dicho cliente mediante la interfaz servidor-Rever, la cual definiremos en el apartado 5.4 de este capítulo.

Finalmente, enviaremos la respuesta al cliente con las cabeceras necesarias y finalizaremos la conexión TCP.

5.4 Interfaz Servidor-Rever

La interfaz Servidor-Rever es la encargada de permitir al servidor crear procesos rever, comunicarse con estos, y cerrarlos en caso de ser necesario.

Al igual que el servidor, esta parte del proyecto se implementará en JavaScript, dado que deben comunicarse y NodeJs nos permite ejecutar las llamadas al sistema necesarias para ejecutar procesos.

Para controlar los procesos de Rever, y dado que este se ejecuta sobre SWI-Prolog, ejecutaremos un proceso de SWI-Prolog y le introduciremos los datos necesarios para iniciar Rever.

En cuanto al diseño, la interfaz constará de una clase cuyos atributos serán los siguientes:

Nombre del atributo	**Descripción**
id	Salida estándar del programa.
process	Salida de error del programa.
io	Objetivo inicial del programa a depurar.

Tabla 5.4.1 – Atributos de la clase interfaz Server-Rever

Por otra parte, la clase tendrá las siguientes funciones:

Nombre de la función	**Descripción**
constructor	Constructor del objeto de la interfaz. Este inicializa todos los atributos de la clase y inicia el proceso SwiProlog
send_in	Enviar datos de entrada al proceso
read_out	Leer datos de la salida estándar del proceso
read_err	Leer datos de la salida de error del proceso
delete_proc	Borra el proceso
start_proc	Inicia Rever enviando los datos necesarios al proceso SwiProlog.
exc_proc	Mezcla de write, read_out y read_err. Envia datos de entrada y devuelve la salida del programa.

Tabla 5.4.2 – Funciones de la clase interfaz Server-Rever

Finalmente, para finalizar la descripción del diseño de la interfaz server-Rever, necesitaremos varias funciones que nos permitan crear ficheros dada una id de usuario, y volcar en estos ficheros el código del programa a depurar.

Para ello, crearemos 2 funciones, *create_file* que creará ficheros en una carpeta especifica que tendrán como nombre la id del usuario, y que tras crearlo escribirá sobre estos el código del programa a depurar; y *delete_file*, que dada una id de usuario, borrará el archivo correspondiente.

5.5 Interfaz de usuario

En cuanto a la interfaz de usuario utilizaremos una combinación de varias tecnologías ampliamente conocidas en el ámbito web, tal y como hemos podido ver en el capítulo 2.

En primer lugar, utilizaremos PHP para generar las ID de usuario y la implementación del cambio de idioma. Tras esto, crearemos el esqueleto de la página con HTML5 y toda la parte lógica mediante JavaScript. Finalmente, para cumplir con el objetivo de que la pagina sea *responsive* y para el apartado gráfico, instalaremos Bootstrap 5.

En cuanto al diseño de la interfaz, empezaremos especificando qué acciones debe permitir hacer al usuario; después definiremos el diseño gráfico de ésta, y finalmente hablaremos del diseño de la lógica.

5.5.1 Acciones del usuario

Para cumplir los requisitos del sistema, el apartado gráfico y el apartado lógico en conjunto deben permitir al usuario las siguientes acciones:

- Introducir el código del programa a depurar.
- Introducir el objetivo inicial del programa a depurar.
- Cargar un programa desde un archivo.
- Descargar el programa a un archivo.
- Cargar ejemplos predefinidos.
- Iniciar una depuración.
- Ejecutar el siguiente paso de la depuración.
- Ejecutar el paso anterior de la depuración.
- Omitir la depuración.
- Descargar la traza de la depuración.
- Finalizar depuración.
- Cambiar el idioma de la página.

Además, separaremos dichas acciones en grupos en función de las dependencias entre estas. En nuestro caso el mapa de dependencias sería el siguiente:

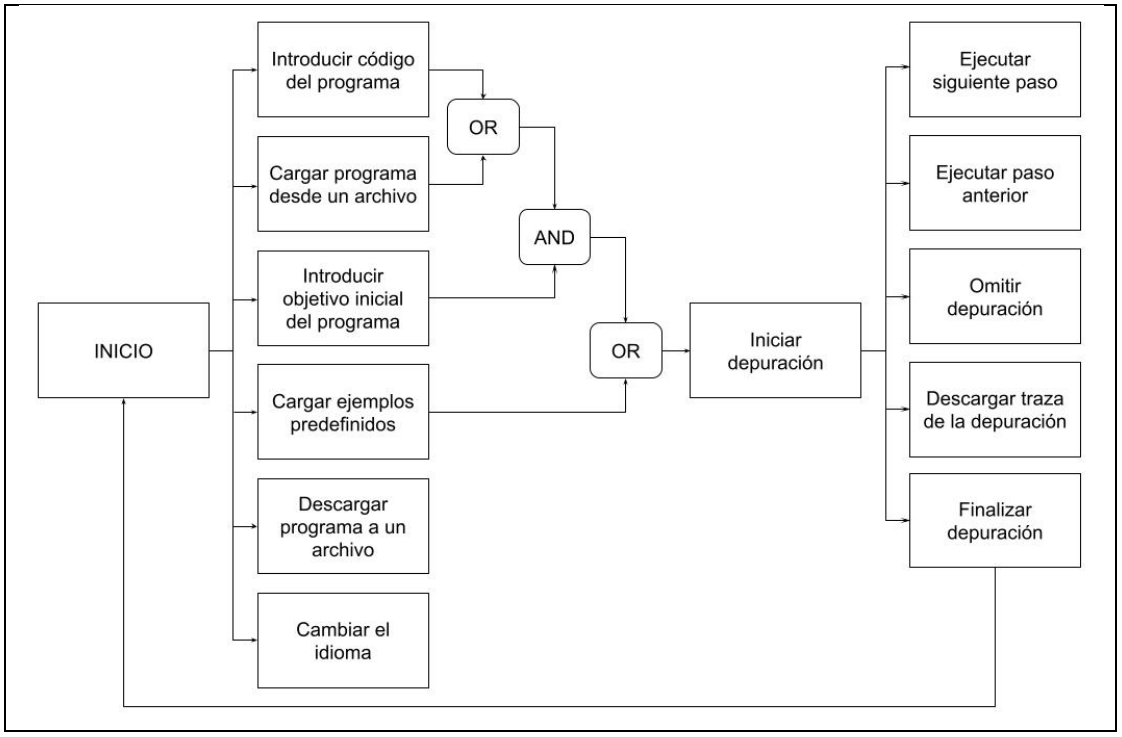

Figura 5.5.1.1 – Diagrama de dependencias de la interfaz

Como podemos observar en la Figura 5.5.1.1, tras el inicio del programa podremos introducir código del programa, cargar el programa desde un archivo, introducir el objetivo inicial, etc; Pero no podremos iniciar la depuración hasta que en la interfaz esté definido tanto el código del programa a depurar como el objetivo inicial.

Esto último lo podemos conseguir de varias maneras, tal y como se indica en la figura. Para introducir el código podemos tanto escribirlo manualmente como cargarlo desde un archivo. En cuanto al objetivo inicial, deberemos introducirlo a manualmente.

Otra opción en cuanto a inicializar dichos parámetros será utilizar la opción de cargar ejemplos, la cual nos introducirá automáticamente el código y el objetivo inicial en la interfaz.

Tras iniciar la depuración tendremos acceso a los controles de esta, es decir, ejecutar un paso adelante o atrás, omitirla, finalizarla o descargar la traza hasta el momento. Pero no podremos volver a realizar las acciones anteriores a la iniciación a la depuración hasta que no la finalicemos.

La información analizada nos permite ver que la mejor manera de organizar la interfaz gráfica es en dos o más ventanas; una para las acciones tras iniciar la depuración, y una o varias para las acciones antes de iniciar la depuración.

5.5.2 Diseño gráfico

Para el diseño de la interfaz utilizaremos prototipos de tipo *mockup* dado que nos permite realizar cambios rápidos de manera sencilla antes de desarrollar la interfaz final, y nos permite hacernos una idea de cómo será dicha interfaz.

En un primer lugar se presentó al tutor una primera opción en la cual la interfaz se separaba en 3 ventanas. Una ventana inicial que permitía introducir el objetivo inicial, cambiar el idioma, acceder a otra ventana para la edición del código del programa, e iniciar la depuración. Por otra parte, nos encontrábamos con una ventana para editar el código a depurar, cargarlo desde un fichero y descargarlo a un fichero. Finalmente, como tercera ventana nos encontrábamos la parte de depuración, en la cual podíamos realizar todas las acciones permitidas durante la depuración.

Además, como extra tras comunicárselo al tutor, decidimos añadir en los laterales columnas con información sobre Rever y sobre Prolog.

Como podemos observar en la figura 5.5.2.1, la cual se encuentra a continuación, en primer lugar encontramos una cabecera con el nombre del programa en la parte superior, y tras esto la pantalla se separaría en 3 columnas, los dos laterales con la información sobre Rever y Prolog, y la parte central en la cual se encuentra un desplegable para elegir programas predeterminados, el código del programa a analizar, un botón para abrir la ventana de edición, una caja de texto para introducir el objetivo inicial, y un botón para iniciar la depuración.

Además, podemos ver como el *mockup* contiene comentarios donde se especifican las acciones o características de la ventana.

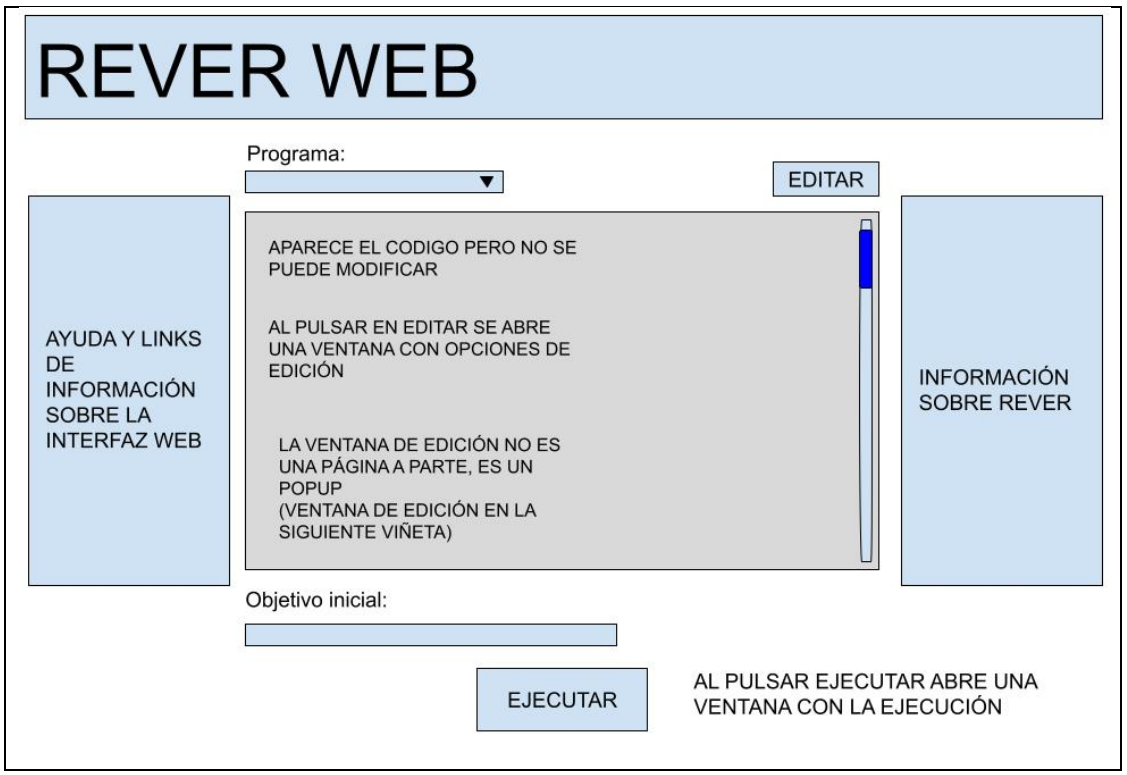

Figura 5.5.2.1 – Mockup v1 página principal

Figura 5.5.2.2 – Mockup v1 edición del código a depurar

En la figura 5.5.2.2 podemos ver cómo la ventana de edición se abriría como un *popup* por encima de la ventana principal. En esta podemos encontrar una caja de texto para editar el código del programa, botones para cargar y descargar el programa, y botones para aceptar o cancelar la edición.

Figura 5.5.2.3 – Mockup v1 ventana de depuración

Finalmente, la figura 5.5.2.3 muestra la ventana de depuración, la cual se trata de un *popup* al igual que la de edición del código y permite ver el nombre y objetivo inicial del programa en la parte superior junto a un botón para descargar la traza de la depuración.

Tras esto encontramos una sección donde se nos mostrará la traza de la depuración, y finalmente encontramos los botones de control y el botón para finalizar la ejecución.

En la siguiente iteración de diseño de la interfaz el tutor nos comunicó los cambios que debíamos implementar al diseño. En primer lugar, eliminamos completamente la ventana de edición de texto, moviendo los botones de carga desde fichero y descarga a fichero a la ventana principal, y permitiendo la edición del código en esta.

El segundo cambio que se realizó fue el de eliminar el campo “Nombre del Programa” de la ventana de depuración, dado que no se vio necesario ya que en ningún momento el usuario decide el nombre del programa.

Ambos cambios podemos verlos en las figuras 5.5.2.4 y 5.5.2.5 que encontraremos a continuación.

En el proceso de desarrollo deberemos basarnos en estos mockups lo máximo posible para crear la interfaz.

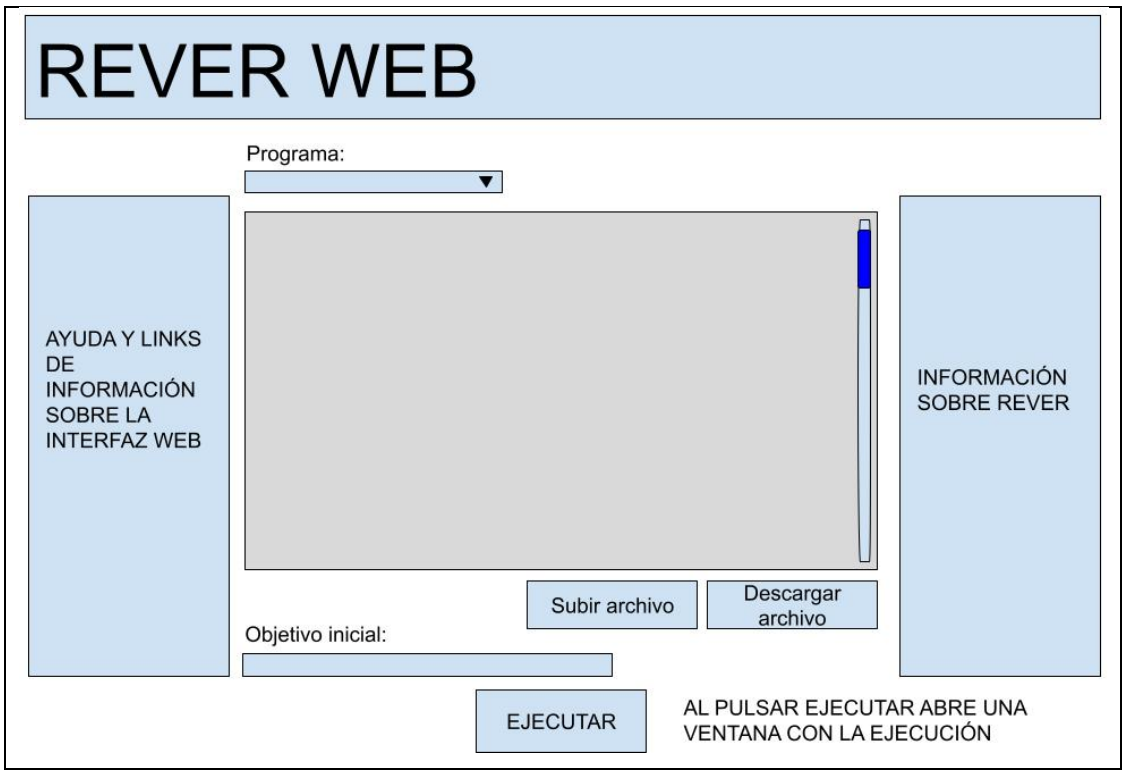

Figura 5.5.2.4 – Mockup v2 ventana principal

Figura 5.5.2.5 – Mockup v2 ventana de depuración

5.5.3 Diseño de la lógica

En cuanto al apartado lógico de la interfaz de usuario, crearemos un archivo JavaScript con todas las funciones necesarias para mantener la lógica separada del aspecto gráfico.

Para la comunicación con el servidor utilizaremos la metodología AJAX (***A**synchronous **J**avaScript **A**nd **X**ML*), mediante la cual enviaremos peticiones al servidor cuando sea necesario, y una vez obtengamos la respuesta la mostraremos por pantalla en la posición correspondiente.

En primer lugar, necesitaremos una función que se encargue de realizar las peticiones mediante AJAX, y tras esto múltiples funciones que muestren la información en función de la acción realizada por el usuario. Por ejemplo, si pulsa el botón "Iniciar depuración" se deberá enviar la petición, abrir la ventana de depuración y mostrar el primer paso de la depuración en la zona central de la ventana. Por otra parte, en caso de pulsar "anterior", se deberá borrar la anterior traza de depuración una vez se ha informado al servidor de ello.

En este último conjunto de funciones encontraremos las que implementan las acciones de iniciar y finalizar la depuración, ejecutar el anterior o siguiente paso de la depuración y omitir la depuración.

Otras funciones que deberemos implementar son las que permiten leer un archivo seleccionado y copiar dicha información en la caja de texto del código a depurar, y las funciones que permiten descargar tanto la traza como el código del programa.

Finalmente, necesitaremos una función que cargue ejemplos sobre la caja de edición del código del programa y sobre la caja del objetivo inicial.

Por otra parte, la acción de cambiar de idioma se implementará mediante PHP, de manera que al cambiar de idioma se realizará una nueva petición de la interfaz y el servidor cargará el idioma correspondiente antes de enviar la interfaz al usuario.

5.6 Tecnología utilizada

Tal y como hemos ido comentando en cada uno de los componentes del sistema, las tecnologías que usaremos para el desarrollo de este proyecto son:

Por parte del servidor REST y la interfaz server-Rever utilizaremos JavaScript mediante el motor NodeJs y llamadas al sistema para manejar archivos y para comunicarnos con los procesos Rever.

En la interfaz de usuario utilizaremos PHP, HTML5, JavaScript, el *framework* Bootstrap5 y peticiones AJAX implementadas mediante JavaScript.

Para servir nuestra interfaz de usuario como página web usaremos el servidor web Apache junto a los módulos que veamos necesario instalar.

Finalmente, aunque no desarrollemos código mediante Prolog, el programa Rever está escrito en dicho lenguaje de programación, así que consideraremos que utilizamos dicha tecnología al intercambiar mensajes con el proceso SWI-Prolog para iniciar Rever.

Todas estas tecnologías han sido analizadas anteriormente en el capítulo 2 – Estado del Arte.

6. Desarrollo

En este capítulo explicaremos en detalle la implementación final de cada uno de los componentes del proyecto, el diseño de los cuales mostramos en el capítulo anterior.

En primer lugar, analizaremos la arquitectura general de cada uno de los componentes, y tras esto comentaremos los problemas con los que nos encontramos y la solución aplicada y finalmente el código utilizado para el desarrollo de cada una de las funciones.

En cuanto al componente de la interfaz de usuario, mostraremos además imágenes del aspecto final y analizaremos los cambios realizados en cuanto al diseño a lo largo del desarrollo.

Además, en un último apartado comentaremos la seguridad del sistema en cuanto a posibles ataques, y las implementaciones realizadas para evitarlos.

Antes que nada, hablaremos de los problemas que hemos tenido que cambiar frente al diseño definido anteriormente.

6.1 Problemas generales de diseño

Encontramos a la hora de la implementación varios problemas generales de diseño que tuvimos que reconsiderar.

En un primer lugar no se tuvo en cuenta el hecho de que un usuario pudiera desconectarse sin llegar a informar al servidor de ello, por lo que el proceso de Rever quedaría ejecutándose hasta el reinicio del servidor.

Para solucionar dicho problema, se implementará un sistema de *timeouts* en los que en caso de no recibir por parte del cliente algún mensaje indicando que no está inactivo en 30 segundos, el proceso se eliminará a sí mismo para liberar memoria en el servidor. Esta solución se implementará en la parte de la interfaz servidor-Rever como veremos en el apartado 6.3 y en la interfaz de usuario como podremos observar en el apartado 6.5.

El segundo problema que encontramos venía relacionado con las teclas de control utilizadas para controlar el proceso Rever sobre SWI-Prolog. Tal y como indicamos en el capítulo 2, Rever espera la pulsación de las teclas del teclado para continuar la depuración. En caso de recibir una pulsación de la tecla con la flecha hacia arriba, Rever mostrará el anterior paso de la depuración, y si recibe la pulsación de la tecla con la flecha hacia abajo, continuará con el siguiente paso de la depuración.

El problema surge al no ser posible simular dichas pulsaciones mediante la tubería de entrada de datos, dado que dichas teclas no tienen un código ASCII asignado y se utiliza otro subsistema del sistema operativo para detectar su pulsación.

Sin embargo, sí podemos simular la pulsación de una tecla como la “A” enviando su código ASCII al programa y el carácter \n como salto de línea.

Para solucionar este problema, deberemos modificar parte del archivo *util.pl* utilizado por el programa Rever. Concretamente, cambiaremos los códigos de pulsación de las teclas *up* y *down* por los códigos de las teclas *d* y *u*. De esta manera, cuando el proceso reciba los códigos ASCII de las teclas d y u continuará la ejecución para la tecla d, y mostrará la traza anterior para la tecla u.

Esta será la única modificación que realizaremos sobre el programa Prolog, dado que como hemos indicado anteriormente, pretendemos realizar este proyecto modificando en lo mínimo posible el programa Rever.

6.2 Servidor web

Dado que como servidor web vamos a usar un producto ya desarrollado, solo necesitaremos realizar su instalación. No será necesario configurarlo dado que por defecto ya es funcional.

Además de su instalación, deberemos descargar e instalar un módulo de Apache para que este sea capaz de ejecutar archivos PHP. El módulo en cuestión es libapache2-mod-php para Apache.

Especificaremos más sobre la instalación de Apache y del módulo PHP en el capítulo 7 – Implantación.

6.3 Interfaz Server-Rever

A continuación, analizaremos la implementación final de la interfaz entre el servidor y los procesos Rever, es decir, el componente encargado de crear, eliminar y comunicarse con dichos procesos.

La hemos realizado sobre un archivo llamado Swi-Node.js. Esto se debe a que comunica SWI-Prolog con el servidor implementado en NodeJs.

En dicho archivo se define la clase SwiplNode, y todas las funciones de esta, además de utilizar al final del archivo la línea de código *module.exports = SwiplNode* para poder importar la clase desde el archivo donde implementaremos el servidor.

6.3.1 Función constructor(cli_id, initial_obj)

Esta función crea el objeto SwiplNode e inicializa sus atributos. Entre ellos encontramos:

- id: El cual se inicializa con el contenido de cli_id.
- process: Para inicializarlo se llama a la función *spawn('swipl')*, la cual inicia un proceso Swi-Prolog mediante llamadas a sistema operativo y devuelve un objeto *child_process*.
- out_buff: Buffer para guardar la salida estándar del proceso.
- err_buff: Buffer para guardar la salida de error del proceso. La implementación de este buffer y del anterior son parte de la solución al problema encontrado con las funciones exec_proc y start_proc, el cual explicaremos más adelante en el apartado 6.3.6.
- io: Se inicializa con el contenido de initial_obj.
- timeout: Contiene un *timeout* que elimina el proceso en caso de estar inactivo. Se trata de la solución a un problema de diseño explicado en el apartado 6.1. Dicho *timeout* ejecutará la función *delete_proc* de esta misma clase en caso de considerarlo inactivo durante 30 segundos.

Además, también inicializaremos dos *listeners* para el atributo process.

El primer *listener* esperará al evento '*data*' de *process.stdout* que se activa cada vez que el proceso recibe datos en su salida estándar. Tras detectar dicho evento, introducirá los datos de la salida estándar al buffer *out_buff* mediante un bucle y la función *push*.

El segundo *listener* también esperará al evento '*data*', pero de la salida de error del proceso (*stderr*) e introducirá los datos de dicha salida en el buffer *err_buff* de la misma manera que el anterior.

6.3.2 Función send_in(mess)

Esta función realiza una llamada a la función *process.stdin.write(mess)*, la cual envía la *String mess* a la entrada del proceso.

6.3.3 Función read_out()

Esta función devolverá todo el contenido del buffer de salida estándar (*out_buff*), y vaciará el buffer. Esta acción se realiza mediante un bucle *while* que para una vez la longitud del buffer es igual a 0. Dentro de dicho bucle se ejecuta la función *shift* sobre dicho buffer, la cual retira y devuelve el primer elemento.

6.3.4 Función read_err()

Devuelve el contenido del buffer de salida de error (*err_buff*) de la misma manera que lo hace la función *read_out()*.

6.3.5 Función delete_proc()

Se realizan 2 llamadas, una a la función *clearTimeout(timeout)*, la cual elimina el temporizador que elimina el proceso en caso de estar inactivo; y una llamada a la función *process.kill()* la cual elimina el proceso SWI-Prolog creado anteriormente.

6.3.6 Función exc_proc(command)

El objetivo de esta función es enviar la *String command* al proceso, esperar a que éste lo procese y devolver el resultado obtenido.

Primero analizaremos el problema con el que nos encontramos para la implementación de esta función, y para ello, antes debemos entender cómo funcionan las llamadas asíncronas en NodeJs.

Las llamadas a funciones asíncronas no devolverán ningún objeto, y tendremos que esperar a la creación de ciertos eventos para recibir los resultados, por lo que la llamada a estas funciones vendrá seguida de *listeners* que esperarán a que se generen dichos eventos.

Para entenderlo mejor analizaremos una función utilizada en esta misma interfaz en la función *send_in(mess)*, la llamada *process.stdin.write(mess)*.

Objeto / Función	Descripción
process	Objeto de tipo <*child_process*> creado mediante la función *spawn(cmd)*.
stdin	*Stream* de datos de entrada de ***process***. Se trata de un objeto de tipo <*stream.writable*>.
write	Función del objeto ***stdin.*** Permite enviar datos a la entrada de ***process***.
mess	Objeto de tipo *String*, parámetro de entrada de la función ***write***. Se trata de la cadena a enviar a la entrada de ***process***.

Tabla 6.2.1.1 – Partes de la función de ejemplo

La llamada anterior no devuelve ningún valor, pero el objeto ***process*** generará un evento cuando el proceso mande datos a ***stdout*** o ***stderr***, las cuales son las tuberías de salida estándar y salida de errores del proceso.

Esto último indica que no tendremos manera de mandar datos al proceso y esperar al resultado para obtener la respuesta, por lo que deberemos implementar una solución que mande la información, espere, y tras esto reciba los datos del proceso.

Para ello, en primer lugar, hemos creado los atributos *out_buff* y *out_err* los cuales almacenarán la salida del proceso como hemos visto anteriormente.

Tras esto, hemos implementado los dos listeners que reaccionarán a cuando el programa manda datos a la salida estándar y la salida de error, y almacenará dicha información en los buffers establecidos.

Ahora podríamos implementar una función que primero enviara los datos, se esperara durante unos cuantos milisegundos a que se procesara la información y finalmente devolviera el resultado del proceso. Sin embargo, JavaScript no implementa ningun método que espere de manera síncrona, y la única manera de hacerlo esperar sería mediante la función asíncrona setTimeout.

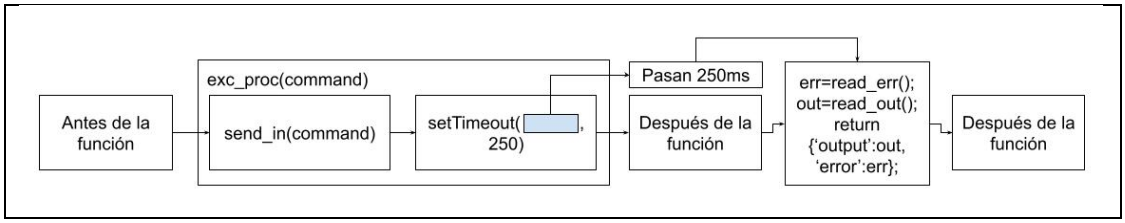

Figura 6.3.6.1 – Traza de la función errónea

Como podemos observar en la figura 6.3.6.1, si ahora mismo implementáramos la función, se devolverían los datos tras 250 milisegundos como queríamos, pero dado que la función *setTimeout()* es asíncrona, durante ese tiempo se ejecutaría código que se encuentra detrás de la función *exc_proc(command)*, lo cual intentamos evitar.

Para ello utilizaremos los objetos *promise* los cuales nos permiten que se espere a que la función se complete para que se ejecute el siguiente código.

En cuanto al uso de *promise* en la implementación de funciones, la función en cuestión deberá devolver un objeto *promise*, el cual contiene el código a ejecutar como un atributo de este. El objeto a devolver por la función en caso de completarse de manera correcta se devolverá mediante *resolve(object)* y en caso de fallo se deberá llamar a *reject(object)*, siendo *object* en ambos casos el objeto a devolver en cada una de las situaciones.

En cuanto a la llamada de la función, se realizará llamándola de la manera habitual (en este caso *process.exc_proc(command)*), y tras esto pasaremos el código a ejecutar una vez se termine la función mediante la función *.then(function)*, o la función a ejecutar en caso de error mediante *.error(function)*.

Para poder visualizarlo de manera mejor presentaremos un ejemplo.

Supongamos que queremos enviar datos a un proceso ya inicializado (utilizando nuestra clase SwiplNode), y tras esto queremos imprimir por pantalla la información devuelta.

Suponemos para ello que la función *exc_proc(command)* ya ha sido implementada (tras el ejemplo analizaremos su implementación final).

El código que necesitamos será el siguiente:

```
exc_proc(command).then(
   (result) => {console.log(result);}
).error(
   (error) => {console.log(error);}
);
```

Figura 6.3.6.2 – Código de ejemplo

Como podemos observar, se ejecutará la función *exc_proc(command)*, y tras esto, en caso de terminar de manera correcta se ejecutará la función contenida en *.then*, y en caso de terminar con un error se ejecutará la función contenida en *.error*.

Finalmente, la implementación de la función *exc_proc(command)* devolverá un objeto *promise*, el cual primero ejecutará la función *send_in(command)*, tras lo cual esperará 250 milisegundos y finalmente leerá ambas salidas y devolverá mediante *resolve()* el resultado.

En caso de que tanto la salida estándar como la salida de error estén vacías se devolverá el *false* mediante *.reject()*.

El funcionamiento final de la función será el que podemos ver en la Figura 6.3.2.3 a continuación.

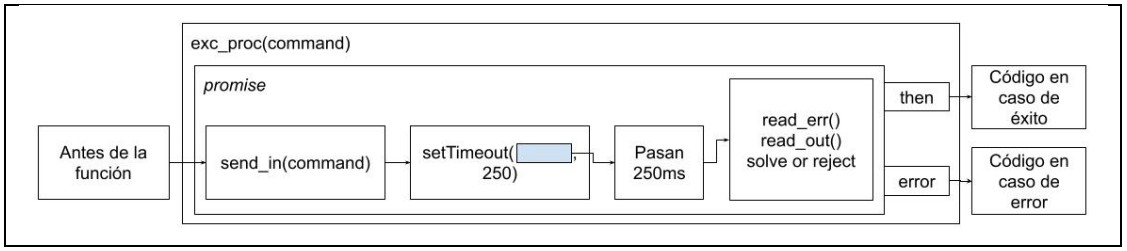

Figura 6.3.2.3 – Traza de la función correcta

6.3.7 Función start_proc()

Esta función tiene como objetivo iniciar el programa Rever sobre el proceso SWI-Prolog al que referenciamos mediante el atributo *process*. Para ello, utiliza la misma estructura que la función anterior. En primer lugar devuelve una *promise* la cual contiene una función que manda múltiples *String* de entrada mediante *send_in()*, se espera 250 milisegundos y vacía los buffers de salida mediante las funciones *read_out* y *read_err*. En caso de que las cadenas estén vacías se devolverá *false* para indicar qué ha fallado. De lo contrario se devolverá *true*.

Los datos que enviaremos al proceso para iniciar Rever son:

- Una cadena con la función '*working_directory*' de Prolog, para indicar que el directorio donde se encuentra Rever es el de la misma carpeta en la que se ejecuta el servidor.
- La cadena '[rever].\n' la cual incluye el archivo del programa Rever.
- La dirección del archivo de usuario en función de la id. Dicho archivo deberá ser creado anteriormente por el servidor.

Mediante dichas entradas el proceso estará preparado para recibir la orden para comenzar la depuración. Dicha orden no se ejecuta en esta función dado que el servidor se encargará de realizar dicha acción.

6.3.8 Función restart_timeout()

Esta función ha sido añadida para solucionar el problema expuesto en el apartado 6.1. La función elimina el timeout mediante la función *clearTimeout(timeout)* y lo vuelve a crear, ampliando así el tiempo límite de inactividad a 30 segundos desde el momento que se ejecuta.

6.3.9 Control de archivos

Para la creación y eliminación de ficheros que contendrán el código del programa a depurar hemos implementado varias funciones en un archivo separado del que contiene la clase de la interfaz server-Rever, al cual hemos llamado cfm.js (*code file manager*). Aun así, consideramos parte de la interfaz dichas funciones ya que son necesarias para el correcto funcionamiento de la interfaz.

Para controlar la creación y eliminación de archivos hemos creado una clase llamada *codeFileManager* la cual contiene dos funciones y el constructor de la clase.

En primer lugar, el constructor inicializa los atributos de la clase, los cuales son una referencia al módulo *fs (file system)* de NodeJs, y la dirección de la carpeta donde se almacenarán dichos archivos. Finalmente, el constructor verifica si existe la carpeta, y si no existe la crea.

Tras esto, la clase implementa la función *set_file(id, data)*, la cual crea (o reemplaza en caso de ya existir) un archivo en la carpeta designada anteriormente, el nombre del cual es el parámetro *id* y el contenido es el parámetro *data*.

Finalmente, la clase implementa una última función llamada *del_file(id)*, la cual elimina el fichero llamado id en caso de existir.

6.4 Servidor REST

Tal y como indicamos en el apartado anterior el servidor REST recibirá peticiones, las analizará, se comunicará con el proceso Rever del usuario que realizó la petición mediante la interfaz server-Rever, y finalmente devolverá el resultado.

Para cumplir dicha función el código del servidor consta de un apartado principal y dos funciones.

6.4.1 Apartado principal

En primer lugar, el apartado principal realiza *requires* para todos aquellos módulos necesarios, en nuestro caso el módulo de NodeJs *http*, y los módulos de la interfaz server-Rever que hemos creado anteriormente (Swi-Node.js y cfm.js).

Tras requerir los módulos, inicializaremos una variable (*fc*) con un objeto de la clase codeFileManager, para poder crear y eliminar archivos; y después inicializaremos la variable que contendrá el array de usuarios (*users*).

Dado que JavaScript permite la indexación de *ítems* en arrays mediante cadenas de caracteres (como si se trataran de diccionarios), en nuestro servidor indexaremos los objetos de tipo SwiplNode usando como clave la id de usuario que se nos manda en las peticiones, de manera que podremos acceder al proceso de un usuario usando su id.

Una vez tenemos las variables necesarias, crearemos e iniciaremos el servidor mediante la función *http.createServer(function(req, res)).listen(port)*, el cual creará un objeto servidor y lo pondrá a la escucha en el puerto *port*. Este servidor ejecutará la función indicada en *function* cada vez que reciba una petición, y le pasará como parámetros los objetos *req* y *res*.

Dichos objetos son de tipo <*IncomingMessage*> y <*ServerResponse*>, los cuales contienen los datos relevantes a la petición y la respuesta respectivamente.

Por ello, todos los datos que necesitemos de la petición los recibiremos del objeto *req*, y modificaremos el objeto *res* a nuestro gusto para crear la respuesta.

La función que ejecutará el servidor cada vez que reciba una petición configurará en primer lugar las cabeceras necesarias para la respuesta mediante el método *res.setHeader("headername", "value")*, donde *headername* es el nombre de la cabecera y *value* el valor.

Tras configurar las cabeceras, el servidor filtrará la petición en función de su método, el cual podemos encontrar en el atributo *method* del objeto *req*. Dado que anteriormente comentamos que solo utilizaríamos el método POST, todas las cabeceras con dicho método ejecutarán la función *server_post(req, res)*, y todas las que no sean de tipo POST ejecutarán el método *method_not_allowed(req, res)*.

Aunque el servidor no vaya a utilizar ningún otro tipo de método, hemos creado su arquitectura permitiendo añadir otros métodos en cualquier momento con pequeños cambios, ya que los demás métodos se encuentran en el código pero están comentados, de manera que solo se necesitaría eliminar los caracteres de comentado y añadir el funcionamiento necesario en dicho método.

6.4.2 Función method_not_allowed(req, res)

Este método devolverá al cliente que realizo la petición una respuesta con el código de error HTTP 405. El error 405 indica al cliente que el servidor no permite ninguna petición con el método utilizado, y indica mediante la cabecera *Allow* los métodos permitidos por el servidor.

En primer lugar, se definirá el código de estado 405 en la respuesta mediante la modificación del atributo *statusCode* del objeto *res*, y se modificará mediante el atributo *statusMessage* el mensaje de estado, cambiándolo por la cadena de texto '*Method Not Allowed*' indicado según el estándar de HTTP.

Finalmente se creará la cabecera *Allow* con el valor *POST* mediante la función *setHeadder('Allow', 'POST')*, y se enviará el mensaje mediante *res.end()*.

6.4.3 Función server_post(req, res)

Este método se encargará de procesar las peticiones de los clientes, se comunicará con el proceso correspondiente y contestará con la respuesta.

En primer lugar, utilizaremos dos *listeners* sobre el objeto *req* para recibir los datos de la petición. Uno de ellos escuchará el evento *data* que se genera cada vez que nuevos datos de dicha petición llegan al servidor. Este *listener* cogerá dichos datos y los concatenará en la variable *data*.

El segundo *listener* esperará al evento *end* que se generará una vez la petición haya sido recibida por completo. En este momento el objeto *data* contendrá el mensaje de petición enviado por la interfaz de usuario en formato JSON, por lo que extraeremos dicho objeto mediante la función *JSON.parse(data)*. Esta función nos devolverá un objeto como el que definimos en la Tabla 5.3.2.

Una vez obtenida la petición como objeto, buscaremos el proceso del usuario en el array *users* utilizando como índice del array el objeto *data.id*. En caso de existir, recibiremos el objeto SwiplNode del usuario encuestión; en caso de ser un usuario nuevo, recibiremos un objeto de tipo *undefined*.

A continuación, se filtrará el tipo de comando a ejecutar mediante un *switch* y el atributo *cmd* de la petición.

Cada uno de estos comandos intercambiará distinta información con el servidor. Entre los comandos posibles encontramos:

- *start*: Su objetivo es crear e inicializar un nuevo proceso para la ejecución de Rever. Primero, se creará un nuevo objeto de tipo SwiplNode sobre la variable *user*, y una vez finalizada su creación (aseguraremos la espera mediante los objetos *promise* y la función .then() tal y como explicamos anteriormente) le transmitiremos la cadena "*rtrace("+user.io+").\n"*, donde *user.io* es el objetivo inicial mediante la función *exc_proc()*.
- Esperaremos a que la función retorne el resultado con el primer paso de la depuración y se lo enviaremos al usuario con el formato de respuesta acordado en la tabla 5.3.1. Finalmente, asignaremos en el array *users* el nuevo proceso de usuario indexado por la id del cliente.
- *end*: Este comando finalizará el proceso del cliente indicado y borrará el archivo con el código a depurar de dicho cliente en caso de existir. Enviará una respuesta vacía al cliente.
- *d*: Este comando indica que el cliente quiere avanzar un paso en la depuración. El servidor enviará la cadena de caracteres *"\x64\n"* mediante la función *exc_proc()* y enviaremos el resultado al cliente.

- *u*: Este comando indica que el cliente quiere ir al paso anterior de la depuración. El servidor enviará la cadena de caracteres *"\x75\n"* mediante la función *exc_proc()* y enviaremos el resultado al cliente.
- *rtm*: Este comando es utilizado por la interfaz del cliente para indicar que sigue activo. Este comando reiniciará el *timeout* de su proceso mediante la función *restart_timeout()* y devolverá un mensaje vacío al cliente.

6.4.4 Función internal_server_error(req, res)

Este método devolverá al cliente que realizo la petición una respuesta con el código de error HTTP 500. Este error es el designado por el estándar para indicar un error en la ejecución del servidor e indicará a nuestra interfaz que debe terminar la depuración y mostrar un mensaje de error.

Esta función se implementa al igual que la función *method_not_allowed* con la diferencia de que el código de estado será el 500, el mensaje de estado será un identificador de error que nosotros designemos, y no se creará ninguna cabecera adicional.

Nuestros códigos de error para el mensaje de estado serán:

- *General_ error*: Indicará un error general en el servidor. No debería darse en ningún momento, pero lo estableceremos por si en algún momento ocurre.
- *Incorrect_io*: Se dará cuando el objetivo inicial sea incorrecto según la sintaxis de Prolog.

6.5 Interfaz de usuario

Para mostrar el desarrollo final de la interfaz comenzaremos por la parte gráfica de esta, y tras ello definiremos el apartado lógico, separando este último en la lógica escrita en JavaScript y la escrita en PHP.

6.5.1 Apartado gráfico

La implementación final ha variado ligeramente de la presentada anteriormente por cuestiones de diseño.

En primer lugar, hemos substituido los cuadros laterales de información por un pie de página y un botón en la zona superior derecha. Estos dos elementos suplirán toda la información que aportaban las columnas laterales dando además un acabado más limpio a la interfaz y permitiendo que la zona central ocupe toda la pantalla en pantallas pequeñas.

Además de lo anterior, hemos añadido un selector de idiomas en la esquina superior derecha junto un botón de ayuda, y un botón junto al selector de ejemplos para cargar el ejemplo que seleccionemos.

Finalmente hemos añadido iconos de información que despliegan dicha información al colocar sobre ellos el cursor. Estos muestran la información relevante sobre el apartado donde se encuentran.

Podemos ver en las figuras 6.5.1.1, 6.5.1.2 y 6.5.1.3 como la pantalla principal se adapta a los distintos tipos de pantalla donde se carga la página. Dichas capturas han sido tomadas sobre la página completa, por lo que a la hora de ejecutarse sobre un navegador este solo mostrará el máximo en altura que la pantalla permita, y el usuario podrá desplazar la pantalla mediante las herramientas que el navegador web ofrezca para ello.

Finalmente, en la figura 6.5.1.4, podemos ver el selector de idioma, y en la figura 6.5.1.5 la página principal en inglés. Todos los textos se traducen al inglés excepto el nombre de los ejemplos.

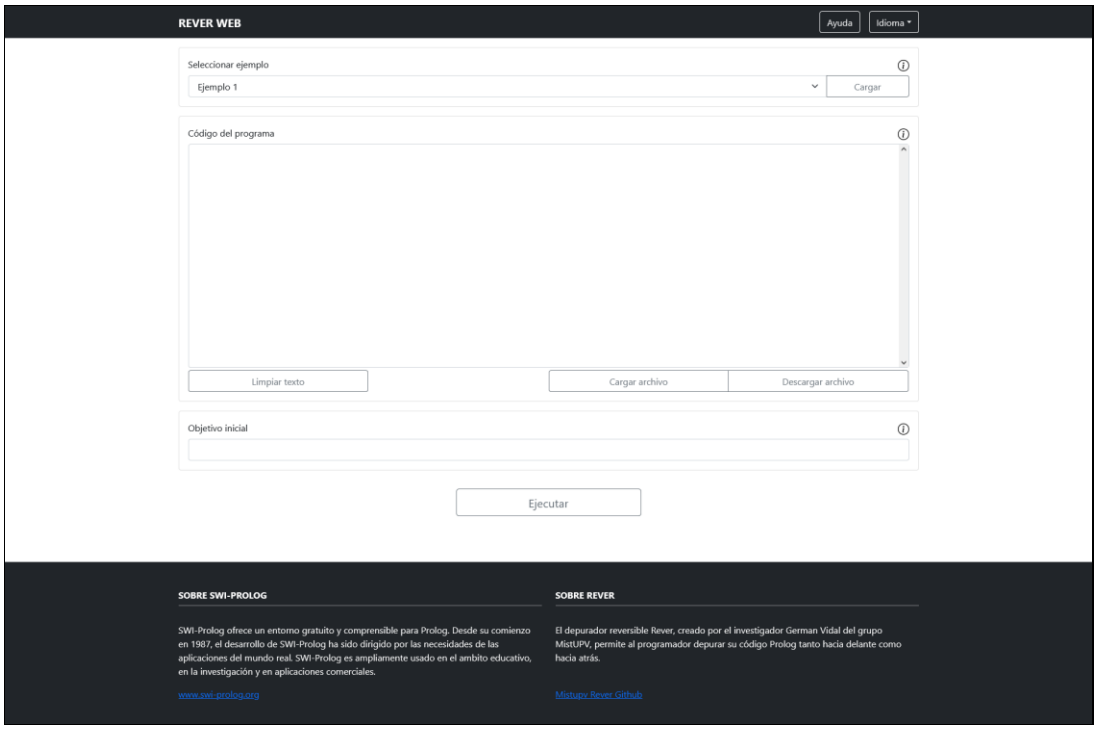

Figura 6.5.1.1 – Página principal en tamaño XXL (más de 1400px de ancho)

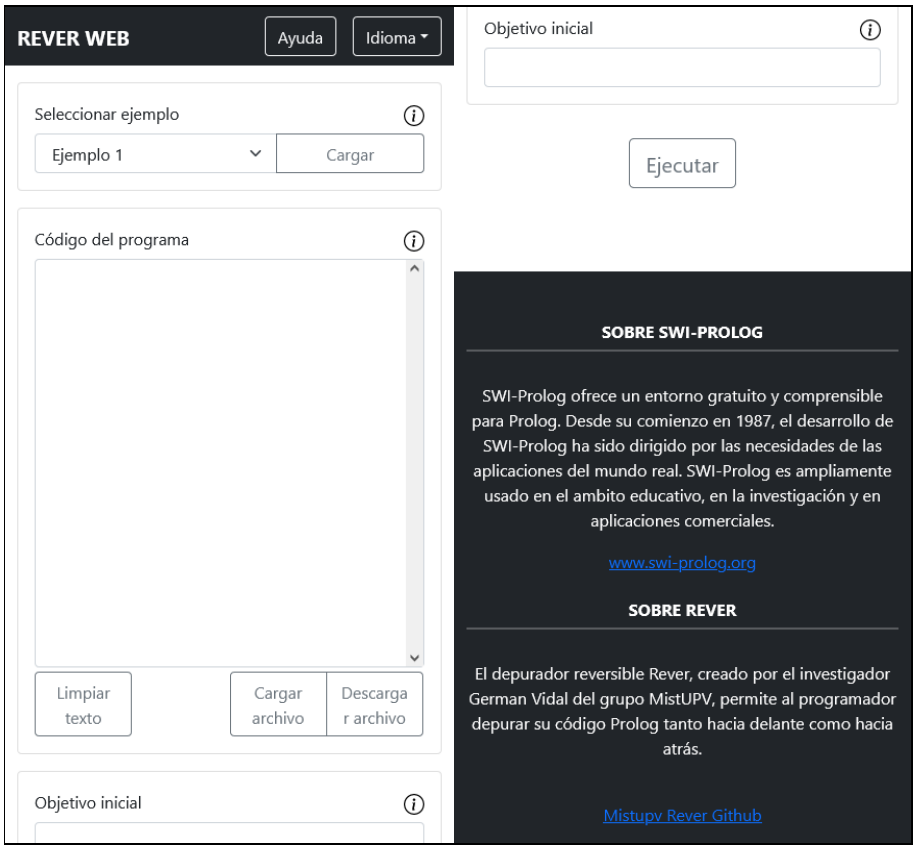

Figura 6.5.1.2 – Página principal en tamaño XS (menos de 576px de ancho)

La figura 6.5.1.2 muestra la página en formato XS en dos partes separadas para evitar dejar espacios en blanco. El aspecto real de la página web es la imagen de la izquierda seguida de la imagen de la derecha.

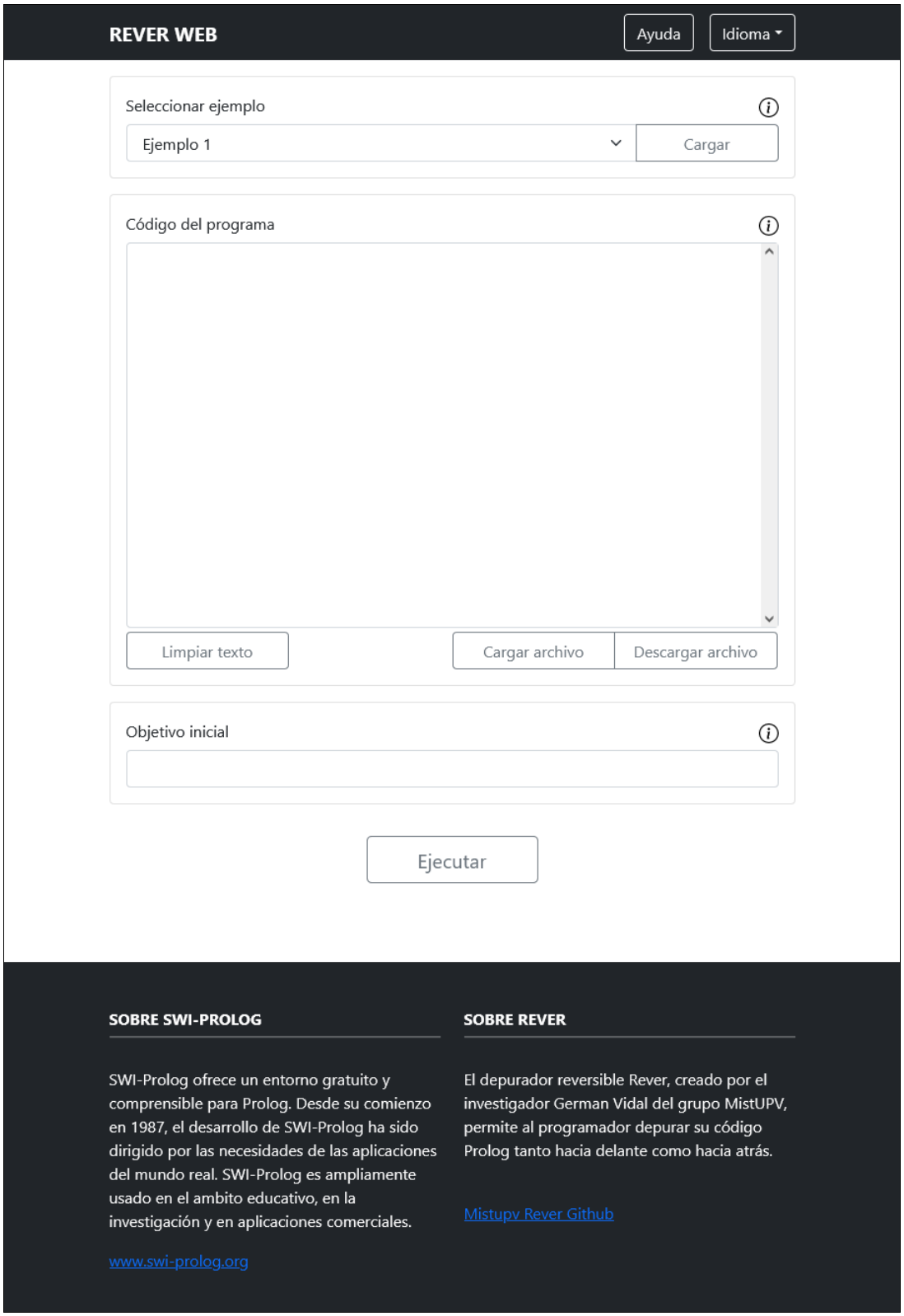

Figura 6.5.1.3 – Página principal en tamaño LG (entre 768 y 992px de ancho)

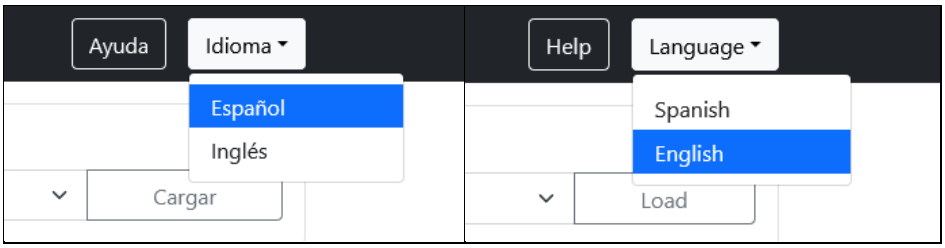

Figura 6.5.1.4 – Selector de idiomas

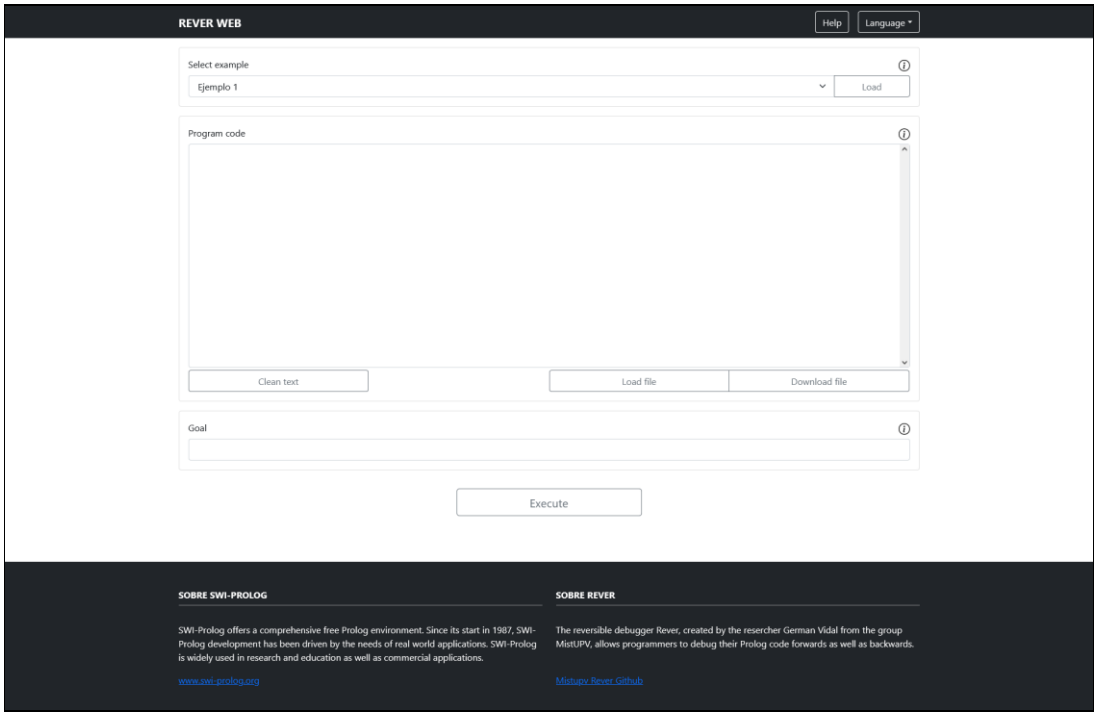

Figura 6.5.1.5 – Página principal en inglés

En cuanto a la ventana de depuración, esta cambia su tamaño en función de la pantalla al igual que la ventana principal. Dado que ya hemos mostrado cómo se consigue dicha función en la página principal al igual que cómo cambia el idioma, únicamente presentaremos la ventana en tamaño XXL (más de 1400 pixeles de ancho). Dicha ventana no contiene ninguna diferencia relevante en cuando al diseño de la figura 5.5.2.5, y podemos verla en la figura 6.5.1.6.

Figura 6.5.1.6 – Ventana de depuración

A continuación, analizaremos cada uno de los elementos de ambas ventanas y explicaremos brevemente cómo se han implementado (sin entrar a nivel de código).

- Cabecera: La cabecera de la página es la parte superior de esta. Para su diseño hemos decidido utilizar el color negro, que resaltará con el cuerpo de la página el cual contendrá un fondo de color blanco. La cabecera contiene los siguientes elementos:
    - Título de la página.
    - Botón de ayuda: En caso de pulsar dicho botón se abrirá una ventana como la que podemos ver en la figura 6.5.1.7, donde se muestra cómo utilizar el depurador.
    - Selector de idiomas: Este último se despliega al hacer clic sobre él, mostrando las opciones posibles.
- Cuerpo: El cuerpo de la página contiene todos los elementos para introducir los datos necesarios para iniciar la depuración. Separaremos el cuerpo de la página en cuatro secciones:
    - Seleccionar un ejemplo: Esta sección contiene la etiqueta 'Seleccionar un ejemplo', un selector de lista desplegable donde podemos ver todos los ejemplos, y un botón para cargar el ejemplo seleccionado.
    - Código del programa: Este apartado contiene la etiqueta 'Código del programa', junto a una caja de texto donde introducir dicho código, y 3 botones. Uno para borrar el texto de la caja de texto, uno para cargar un archivo, y otro para descargar el texto escrito a un fichero. Además, la caja de texto contiene una barra de desplazamiento para escribir textos que ocupen más espacio del que muestra dicha caja.
    - Objetivo inicial: Esta sección contiene una etiqueta con el texto 'Objetivo inicial' junto a una caja de texto donde introducir dicho objetivo.
    - Ejecutar: Este apartado únicamente contiene un botón con el que ejecutar la depuración.
- Pie de página: Contiene dos elementos con la misma estructura. Cada uno de ellos está compuesto por un título, un párrafo de texto y un enlace. El primero muestra información sobre SWI-Prolog, mientras que el segundo muestra información sobre Rever.

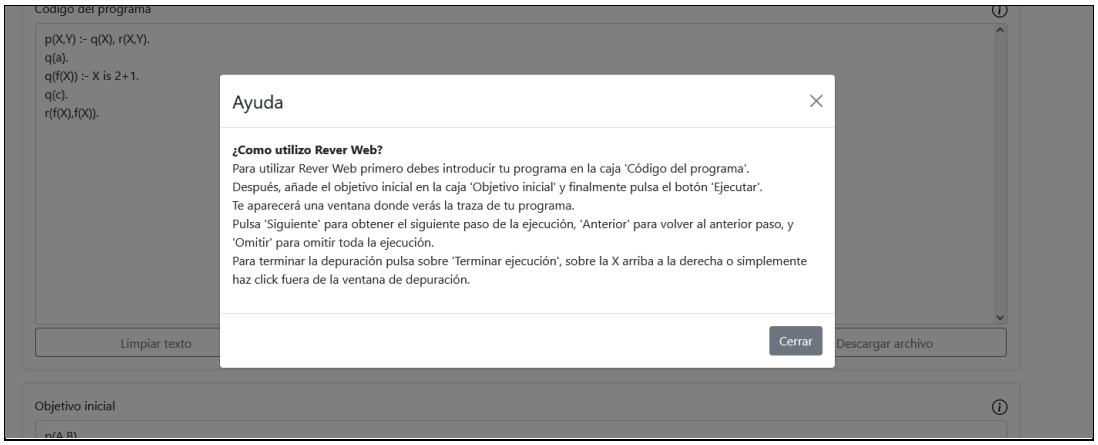

Figura 6.5.1.7 – Ventana de ayuda

Finalmente, la ventana principal contiene diversos iconos de información, y al poner el cursor sobre estos se muestra un mensaje de descripción como el que se puede ver en la figura 6.5.1.8.

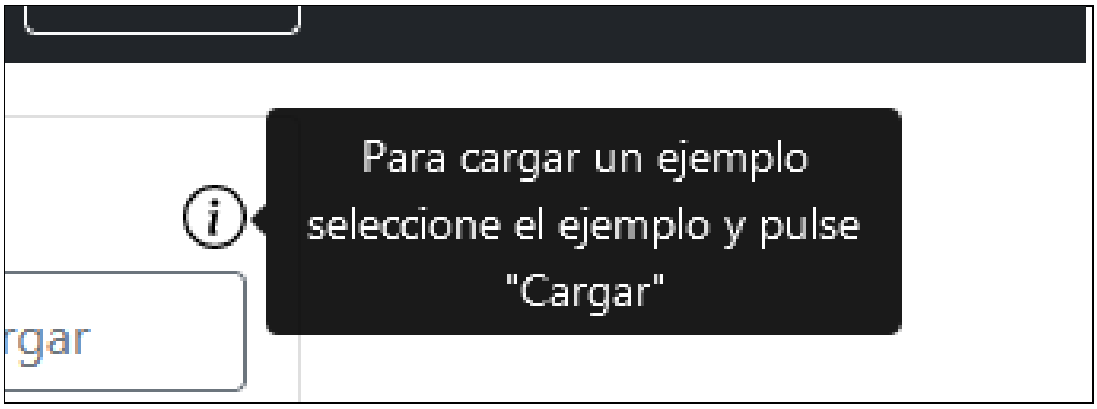

Figura 6.5.1.7 – Texto de ayuda

En cuanto a la ventana de depuración, esta está compuesta por una ventana posicionada sobre la ventana principal la cual contiene cuatro bloques. En la parte superior de dicha ventana podemos ver el título 'Ejecución' y un botón de cerrado. Los demás elementos que componen esta ventana son los siguientes:

- Objetivo inicial: Este apartado muestra un título con el texto 'Objetivo inicial' seguido de el objetivo inicial introducido en la caja de texto de la ventana principal.
- Descarga de la traza: Este bloque contiene un botón que al ser pulsado descargará la traza del programa, es decir, descargará en un archivo de texto el contenido de la ventana donde se muestra la traza.
- Zona de control: En esta sección se muestra un contenedor amplio con una barra para desplazarse verticalmente en caso de estar lleno. En dicho contenedor irá apareciendo la traza de la depuración que estemos realizando. Bajo este contenedor encontramos los botones de control, uno para volver al paso anterior, uno para avanzar al siguiente paso y uno para omitir la depuración.
- Cierre de la ejecución: Se trata de un único botón alineado a la derecha en la parte más baja de la ventana. Al pulsar dicho botón se cerrará la depuración.

6.5.2 Apartado lógico - PHP

En este subapartado mostraremos el apartado PHP de la implementación final de la lógica de la aplicación.

En nuestro proyecto, PHP se encargará de controlar el idioma de la página y de crear e insertar las ID de los usuarios.

Primero debemos tener en cuenta que PHP se ejecuta en el lado del servidor, antes de enviar al usuario la página, por lo que podrá insertar o modificar partes de la página web las cuales serán enviadas al usuario a posteriori.

Para el control del idioma, se ha creado un archivo php llamado *lang.php* el cual contiene dos arrays con un par clave=>valor por cada fragmento de texto traducible.

En la página principal se introducirán los fragmentos *<?php echo lang["ID"];?>* donde *ID* es el identificador del fragmento de texto que queremos colocar en ese lugar.

Al inicio de la página principal se incluirá el archivo *lang.php* y se asignará a la variable *lang* el array correspondiente al idioma seleccionado.

Para establecer qué idioma se ha seleccionado se añadirá a la URL de la página el parámetro *lang*.

La figura 6.5.2.1 muestra un esquema del funcionamiento del sistema de control del idioma de la interfaz de usuario.

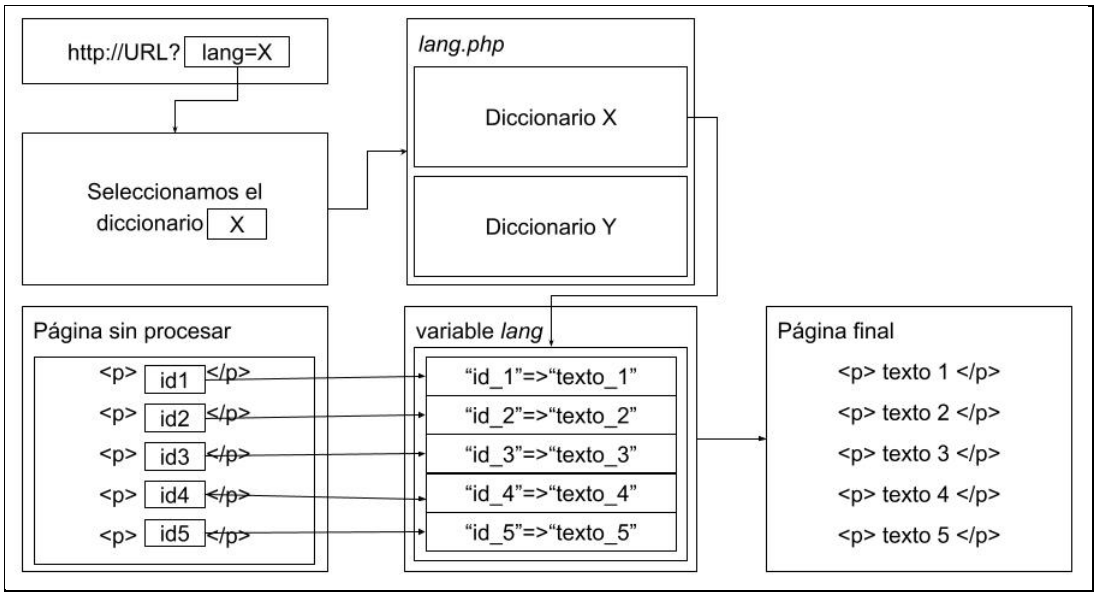

Figura 6.5.2.1 – Sistema de control del idioma

El segundo sistema que implementamos en PHP es el que crea e inserta la ID de usuario en la lógica de la página.

Para ello, utilizaremos el control de sesiones de PHP, el cual creará de forma automática una ID única para cada usuario que se haya conectado a nuestro servidor Apache. Este sistema inyecta una *cookie* de sesión en el navegador del cliente para poder identificarlo.

Para crear la ID de usuario primero utilizaremos la función *session_start()* y tras esto usaremos una mezcla de código JavaScript y PHP de la siguiente manera *<script>var user_id = "<?php echo session_id()?>"</script>*, de manera que la lógica en JavaScript de la página podrá utilizar la variable *user_id* que contendrá la id del usuario.

Además de lo anterior, también asignaremos de la misma manera a la variable *lang* el valor obtenido anteriormente de la URL, para poder acceder a dicho valor desde JavaScript posteriormente.

6.5.3 Apartado lógico - JavaScript

En cuanto al código implementado en JavaScript, encontramos múltiples funciones definidas en el archivo *script.js*, las cuales nos permiten realizar las siguientes acciones:

- Inicializar la página: En primer lugar, deberemos inicializar múltiples elementos en la página, y todo el código relacionado con dicha inicialización está contenido en una única función que se ejecuta una vez que el esqueleto de la página se ha cargado correctamente. Los elementos por inicializar son:
    - Los mensajes de información: Estos mensajes son objetos de tipo *tooltip* de Bootstrap, y deben ser inicializados para que se detecte el ratón sobre estos y se muestre el texto de información correspondiente. Para ello, se utilizará la función *new bootstrap.Tooltip(obj, options)*, donde *obj* es el objeto a inicializar, y *options* son las posibles opciones que aplicar a dichos objetos.
    - Marcar el idioma seleccionado: Detectaremos el idioma actual de la página y resaltaremos la opción de dicho idioma en el selector de idiomas mediante la función *object.classList.add('active')*, siendo *object* el objeto HTML del idioma actual.
    - *Listeners*: Añadiremos múltiples *listeners* para controlar cuándo se pulsa el botón 'Cargar archivo', para detectar cuando se abre o se cierra la ventana de depuración, y para detectar cuándo se pulsa el botón Ejecutar.

- Limpiar texto: Al pulsar dicho botón, se lanzará una función llamada *clean_code()*, la cual buscará el elemento de la caja de texto donde se encuentra el código del programa y la vaciará.
- Cargar archivo: Al pulsar el botón correspondiente, se creará temporalmente un objeto HTML de tipo *input type="file"* el cual lanzará una ventana al usuario para seleccionar el archivo a cargar. Y tras haber sido seleccionado, se leerá el archivo y se copiará su contenido a la caja del código de programa.
- Descargar archivo: Al pulsar el botón para descargar archivos, se lanzará la función *downloadText(text, name)*, la cual creará un archivo temporal con el contenido de la caja de texto. Después se creará un enlace temporal para la descarga de dicho archivo, y se simulará un clic sobre dicho enlace, iniciando así el sistema de descarga del navegador. Finalmente, se eliminará dicho enlace y el archivo temporal.
- Cargar ejemplo: Al pulsar el botón 'Cargar', se llamará a la función *load_example()*, la cual buscará el ejemplo seleccionado en el array *EJEMPLOS* y el array *OI*, y cargará el ejemplo sobre la caja de texto del código del programa y el objetivo inicial sobre su caja de texto correspondiente.
- Iniciar depuración: Al pulsar el botón 'Ejecutar' se verificará en primer lugar que las cajas de texto del código del programa y del objetivo inicial no estén vacías, y entonces llama a la función *start_process()*, la cual realiza los siguientes pasos:
    - Inicia la variable *num_comm* a 0. Esta variable es la encargada de detectar cuantos pasos de la depuración existen en pantalla, para evitar las pulsaciones sobre el botón 'Atras' una vez solo quede un paso en pantalla.
    - Bloquear el botón 'Atras', dado que en un primer lugar solo habrá un paso de la depuración en pantalla.
    - Monta la petición y la envía al servidor mediante una petición AJAX, la cual se realiza mediante un objeto *HMLHttpRequest*. En dicha petición se mandará el código del programa, el objetivo inicial, la id de usuario y el comando *start*.
    - Tras recibir la respuesta, mostrará dicha respuesta en la zona designada.
    - Finalmente, iniciará un intervalo que se ejecutará cada 20segundos, el cual mandará una petición al servidor con el comando *rtm* y la id del usuario, para informar al servidor de que la depuración no está inactiva.
- Siguiente paso: Al pulsar el botón 'Siguiente', se enviará una petición al servidor con el comando 'd' y la id de usuario. Tras recibir la respuesta se escribirá el paso de la depuración en la zona designada. En caso de que el mensaje recibido comience por '**Solution: ' o 'No more solutions…', se bloqueará el botón 'Siguiente'. Finalmente se actualizará el contador *num_comm*.
- Paso anterior: Al pulsar el botón 'Anterior', se enviará una petición AJAX al servidor con el comando 'u' y la id del cliente. Tras recibir un mensaje vacío confirmaremos que el servidor ha realizado la acción, se seleccionará la última entrada en la traza de la depuración y se eliminará. Finalmente se actualizará el contador *num_comm* y en caso de que el contador sea igual a 1 se bloqueará el botón 'Anterior'.
- Omitir depuración: Al pulsar el botón siguiente paso se simularán pulsaciones sobre el botón siguiente hasta que el mensaje recibido comience por '**Solution:' o 'No more solutions…'.
- Terminar depuración: Esta acción se realizará cada vez que se cierre la ventana de depuración. En este caso se enviará una petición AJAX al servidor con el comando '*end*' y la id del usuario.

- Control de errores: En cualquier caso, en el que las peticiones AJAX reciban un código de error, se cerrará la ventana de depuración y se mostrará un texto al usuario mediante la función *alert(text)*, la cual mostrará al usuario una ventana con el texto *text*. Los textos a mostrar podrán ser o bien 'El objetivo inicial no es válido, por favor compruebe la sintaxis utilizada.', 'A ocurrido un error en el servidor, por favor inténtelo de nuevo en unos segundos.' o 'Servidor inactivo, por favor inténtelo más tarde.'.

6.6 Seguridad del sistema

En este último apartado hablaremos de la seguridad del sistema, que vectores de ataque posibles hemos detectado, y como hemos reforzado dichos puntos para evitar ataques.

En primer lugar, analizaremos los puntos de entrada de datos en nuestra interfaz, y seguiremos su ejecución hasta el servidor.

6.6.1 Código del programa

En cuanto al código del programa, solo se envía una única vez al iniciar la depuración, y se introduce directamente en un fichero. No parece un gran problema, dado que el nombre del fichero viene dado por la ID del cliente, y dicho fichero simplemente se acabará incluyendo junto al programa Rever al entorno del proceso SWI-Prolog para la depuración.

El verdadero problema es que el usuario puede controlar el nombre del fichero que está escribiendo mediante la edición de su ID, y con ello puede controlar su ubicación.

Por ejemplo, en caso de que la id de usuario contuviera la cadena "../", por ejemplo "../AAA", al alcanzar la función que crea el archivo se ejecutaría la siguiente línea de código: "*fs.writeFileSync('/tmp/rever_server/../AAA')*", lo cual generaría un archivo sobre la carpeta "*/tmp"*, y no sobre "*/tmp/rever_server",* ya que la cadena "*../*" en Linux hace referencia a la carpeta padre.

De esta manera, en caso de que el proceso del servidor fuera iniciado con permisos elevados (por ejemplo en el caso de ejecutarlo con sudo, o ejecutarlo de manera automática con root desde algún proceso del sistema), se podría reescribir cualquier fichero del sistema, mandando el contenido de dicho fichero en el campo "Código del programa" y modificando la ID de usuario.

Para evitar dicho ataque, tanto en el lado de la interfaz de usuario como en el lado del servidor REST, se implementará un sistema por el cual se eliminarán aquellas coincidencias de la cadena "*../*" dentro del parámetro ID de usuario. Por ejemplo, la ID "*asf3S89adff../../foahgs*" será acortada a "*asf3S89adfffoahgs*".

6.6.2 Objetivo inicial

El objetivo inicial del programa acabará siendo pasado al proceso Rever en la función "*user.exc_proc('rtrace('+user.io+').\n)"*", siendo user.io el objetivo inicial. Primero explicaremos la cadena en secciones para entender el objetivo de esta entrada de datos al proceso.

En primer lugar encontramos la función *rtrace()*. Esta marca el inicio de la depuración en Rever, usando como parámetro el objetivo inicial del programa que se pretende depurar.

En segundo lugar encontramos la variable *user.io*. Esta contiene el objetivo inicial introducido en la interfaz web por el usuario.

Finalmente encontramos la cadena ".\n". Esto se debe a que en la sintaxis de Prolog el punto indica el final de una cláusula, y el carácter "\n" es interpretado como un salto de línea, de manera que simula una pulsación de la tecla "enter" en el proceso.

Sabiendo lo anterior, podríamos pensar que se puede tomar el control del programa mediante los caracteres "." y "\n" utilizándolos como una especie de inyección SQL. Por

ejemplo, “*A). \n halt. \n*” se enviaría al programa como “*rtrace(A).\n halt.).\n*”, finalizando la función “*rtrace()”* y cerrando el programa con la palabra de control “*halt.”*.

Sin embargo, tras varias pruebas, llegamos a la conclusión de que la función rtrace bloquea el programa dentro del proceso de depuración, y cualquier carácter enviado que no sea uno de los designados para el control del programa no hará más que avanzar la depuración.

Por otra parte, para evitar el funcionamiento incorrecto del proceso, filtraremos los caracteres “.” y “\n” de la variable “user.io”.

6.6.3 Botones de control

En cuanto a los botones de control, estos simplemente envían comandos del tipo “d” o “u” al servidor. Por lo que aunque se consideran un punto de interacción, no suponen ningún peligro.

6.6.4 Peticiones al servidor

Uno de los puntos de entrada peligrosos es el servidor REST en sí dado que aceptará cualquier tipo de petición que este estructurada como las que envía la interfaz de usuario.

Para evitar que un usuario malicioso colapse el servidor enviando mensajes que arranquen múltiples procesos Rever, en primer lugar tenemos el sistema por el cual cualquier proceso inactivo más de 30 segundos se borrará.

Pero además, otra opción que dejaremos como posible ampliación del proyecto es añadir una capa que solo permita un número limitado de procesos por IP, y que detecte si una IP esta mandando demasiados mensajes para bloquearla y así evitar que el servidor acabe colapsando al recibir demasiados mensajes y tener demasiados procesos Rever activos.

Hablaremos de dicha posible ampliación en el capítulo 9 en el apartado “Posibles ampliaciones”.

7. Implantación

A continuación, realizaremos una guía de instalación del lado del servidor para el correcto funcionamiento de nuestro proyecto y. tras esta guía. encontraremos un tutorial de uso de nuestra interfaz web.

7.1 Guía de instalación

Esta guía mostrará la instalación de nuestro proyecto sobre un entorno con Ubuntu 20.04 recién instalado.

Comenzaremos con la instalación de todos los programas necesarios mediante el instalador de paquetes de Debian *APT (Advance Packaging Tool)*. Tras esto, mostraremos la disposición de carpetas de nuestro proyecto y definiremos dónde debemos colocar cada uno de los componentes. Finalmente, indicaremos cómo iniciar los componentes necesarios para poder empezar a utilizar nuestra interfaz web.

7.1.1 Instalación de paquetes para Ubuntu

En primer lugar, abriremos una consola del sistema y verificaremos si existen actualizaciones de los paquetes del sistema ejecutando el comando 'sudo apt update'.

Una vez haya finalizado, utilizaremos el comando 'sudo apt upgrade' para descargar e instalar las actualizaciones encontradas.

Tras esto, instalaremos el servidor web Apache mediante el comando 'sudo apt-get install apache2'.

Ahora, instalaremos PHP y el complemento php para Apache. Para ello ejecutaremos en la consola 'sudo apt install php libapache2-mod-php'.

Una vez termine podremos comprobar si la instalación es funcional ejecutando el comando 'sudo echo "<?php phpinfo();?>" | tee /var/www/html/info.php' y accediendo mediante nuestro navegador a la URL 'localhost/info.php'. Si todo funciona correctamente, deberíamos obtener una página con información de nuestra configuración php.

Dado que no necesitaremos la página anterior, podemos ejecutar 'sudo rm /var/www/html/info.php' para eliminarla.

A continuación, instalaremos NodeJs mediante el comando 'sudo apt install nodejs'.

Podremos comprobar que se ha instalado correctamente si al ejecutar 'nodejs -v' en el terminal recibimos la información sobre la versión de NodeJs.

Finalmente, para acabar la instalación de paquetes, instalaremos SWI-Prolog. En primer lugar, añadiremos el repositorio oficial de SWI-Prolog mediante la ejecución del comando 'sudo add-apt-repository ppa:swi-prolog/stable'. Una vez finalice el comando, actualizaremos la lista de repositorios con 'sudo apt-get update' y, finalmente, instalaremos SWI-Prolog con 'sudo apt-get install swi-prolog'.

En este momento habremos acabado la instalación de los paquetes necesarios.

7.1.2 Carpetas y ficheros del proyecto

El archivo con el contenido del proyecto contendrá dos carpetas en su interior, una con el nombre 'Rever_server', la cual contendrá el servidor REST, y otra con el nombre 'Rever_web', la cual contiene la página web y todos los componentes necesarios para su funcionamiento.

El servidor contendrá los archivos para el funcionamiento de Rever (ansi_termx.pl, iso_predicates.pl, prolog_code.pl, util.pl y rever.pl) y los archivos para el funcionamiento del servidor REST (cfm.js, Swi-Prolog.js y server_rever.js).

Por parte de la aplicación web, encontraremos las siguientes carpetas y ficheros:

- bootstrap: Contiene el *framework* Bootstrap5.
- css: Ficheros de configuración css.
- img: Imágenes e iconos.
- js: Contiene los ficheros *'scripts.js'* y *'ejemplos.js'*. En el primero encontramos toda la lógica de la página explicada anteriormente, y en el segundo encontramos los ejemplos disponibles. Podemos añadir otros nuevos en caso de ser necesario.
- php: Contiene el fichero de idiomas *lang.php*.
- index.php: Se trata de la página web en sí. En caso de acceder a la carpeta /Rever_web Apache intentará servir el archivo llamado index.php dentro de esta misma carpeta.

7.1.3 Configuración e instalación de la interfaz web

En primer lugar, configuraremos el puerto y la dirección que utilizarán los clientes para conectarse con el servidor. Para ello, abriremos el archivo '/Rever_web/js/scripts.js' y modificaremos la variable *server_dir* situada al principio del archivo.

Deberemos escribir una cadena con la siguiente estructura "*IP:PUERTO*", donde IP será la ip sobre la que tendremos el servidor REST, y PUERTO un puerto disponible. Recomendamos el uso de un puerto superior al 10001 dado que un gran número de aplicaciones conocidas usan puertos inferiores a este.

Si queremos verificar qué puertos están siendo utilizados en el computador podemos usar el comando 'sudo lsof -i -P -n', y para verificar nuestra IP el comando 'ifconfig'.

Tras esto abrimos el archivo' /Rever_server/server_rever.js' y modificamos la variable *PORT* que se puede encontrar en la parte superior del archivo. Este valor debe coincidir con el configurado en el archivo '/Rever_web/js/scripts.js'

A continuación, copiaremos la carpeta 'Rever_web' a la carpeta '/var/www/html/'. Esto se debe a que el servidor web Apache distribuirá los archivos que se encuentren en dicha carpeta.

Para iniciar el servidor REST, accederemos con un terminal a la carpeta 'Rever_server' y ejecutaremos el comando 'node server_rever.js', tras lo cual el terminal quedará bloqueado. No se debe el terminal o se cerrará el servidor REST.

La interfaz web quedará accesible y funcional bajo la dirección URL '*IP*/Rever_web/main_page.php', donde *IP* es la ip del computador donde se esté ejecutando el servidor Apache.

Finalmente, para permitir la conexión de otros computadores a nuestro servidor REST, deberemos configurar en el cortafuegos el puerto elegido anteriormente. Para ello, utilizaremos el comando 'sudo ufw allow *port*/tcp' siendo *port* el puerto donde escucha nuestro servidor REST.

7.2 Guía de uso

En la siguiente guía mostraremos cómo utilizar la interfaz implementada en este proyecto desde el punto de vista del cliente. Para ello, supondremos que tanto el servidor web como el servidor REST han sido instalados e iniciados sobre un computador con la IP 192.168.0.8.

En primer lugar, abrimos el navegador web y accedemos a la URL donde se encuentra la interfaz web tal y como podemos ver en la figura 7.2.1.

Además, como podemos ver en la figura 7.2.2, también podemos acceder directamente a la carpeta donde se encuentra nuestra interfaz, en cuyo caso Apache nos redireccionará al archivo "*index.php"*.

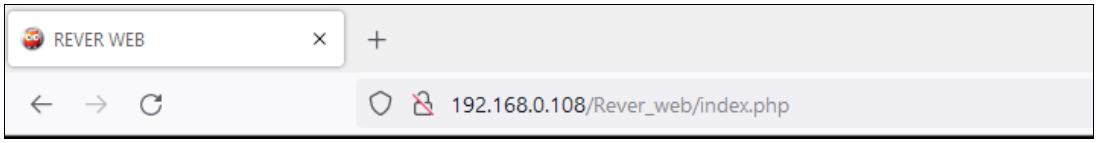

Figura 7.2.1 – URL de la interfaz web

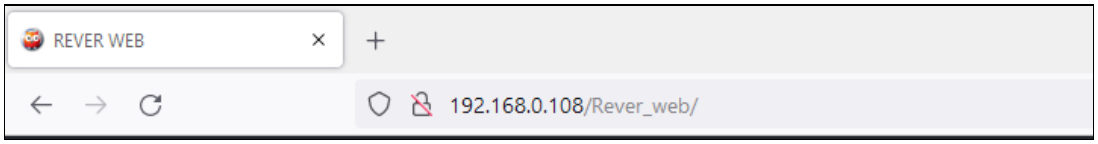

Figura 7.2.2 – URL de la carpeta donde se encuentra la interfaz

Una vez accedemos a la interfaz, tenemos tres opciones:

- Escribir el código y el objetivo inicial:
  En este caso simplemente añadiremos el código a la caja etiquetada como "Código del programa" y el objetivo inicial en la caja "Objetivo inicial".
- Cargar un ejemplo:
  Para ello, seleccionaremos el ejemplo que queramos cargar en la lista desplegable de ejemplos y pulsaremos el botón "Cargar" situado a su derecha.
- Cargar el código desde un archivo e introducir el objetivo inicial:
  Para esta opción, seleccionaremos el botón "Cargar archivo" y, tras ello, nos aparecerá una ventana como la de la figura 7.2.3, en la cual seleccionaremos el archivo que decidamos cargar. Finalmente, escribiremos en la caja etiquetada como "Objetivo inicial" nuestro objetivo para el programa.

Las tres opciones anteriores nos llevarán a la misma situación, en la cual habremos rellenado los campos "Código del programa" y "Objetivo inicial".

Para esta guía utilizaremos un código Prolog inspirado en una escena de la película "Los Monty Python y los caballeros de la mesa cuadrada" de 1975, el cual podemos ver en la figura 7.2.4.

A continuación, pulsaremos el botón "Ejecutar", con lo cual se abrirá la ventana de depuración y nos aparecerá la primera traza de la depuración del programa, como podemos observar en la figura 7.2.5.

Mediante los botones inferiores ("Anterior", "Omitir" y "Siguiente") podremos controlar la ejecución de la depuración (figura 7.2.6).

Al pulsar el botón "Anterior", volveremos al paso anterior de la ejecución.

El botón "Omitir" nos permitirá avanzar la ejecución automáticamente hasta el último paso de ésta.

Al hacer clic en "Siguiente", se nos mostrará el siguiente paso de la ejecución.

Finalmente, para terminar la ejecución podemos pulsar el botón “Terminar ejecución”, la cruz en la esquina superior derecha o simplemente hacer clic en cualquier espacio fuera de la ventana de depuración (figura 7.2.7).

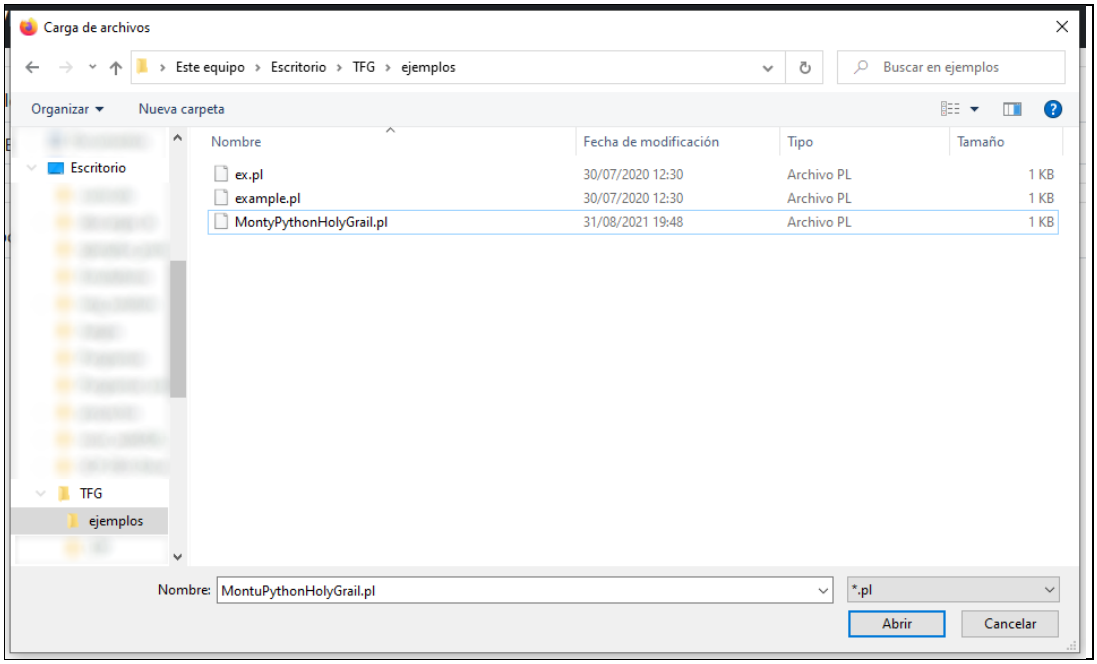

Figura 7.2.3 – Ventana de carga de archivos

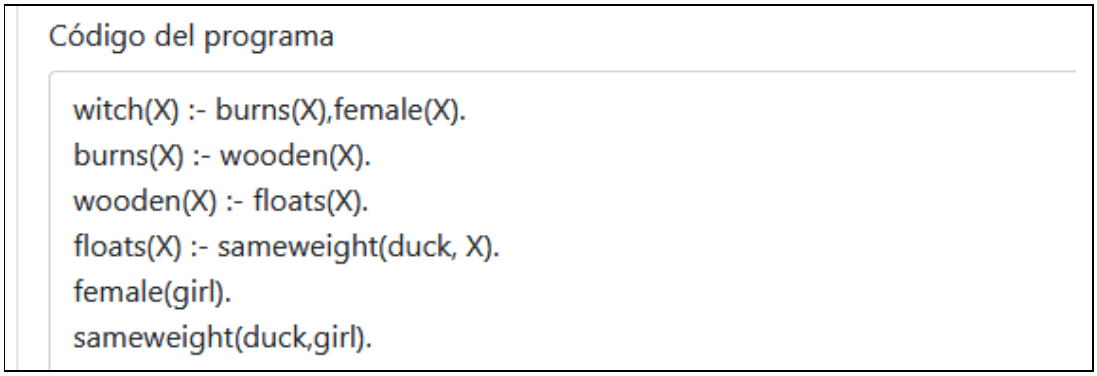

Figura 7.2.4 – Código del programa

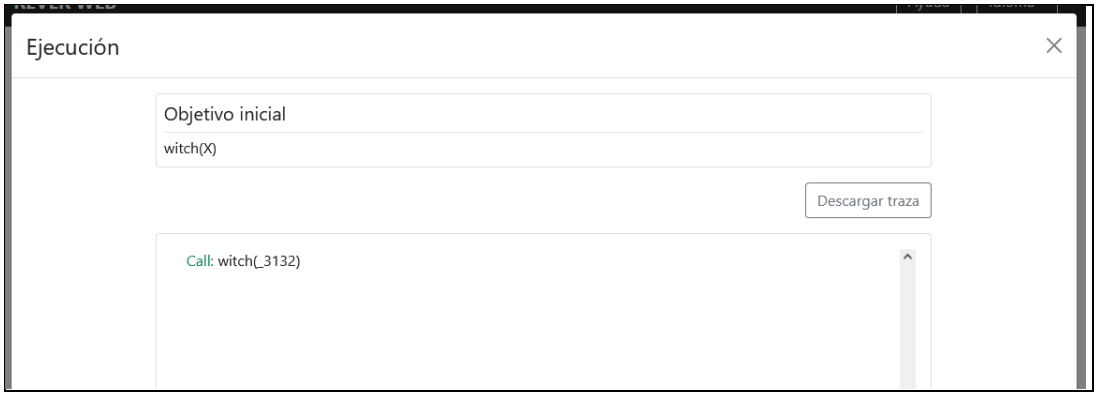

Figura 7.2.5 – Código del programa

Figura 7.2.6 – Botones de control

Figura 7.2.7 – Botones de terminación

Otras acciones que nos permite realizar la interfaz son:

- Cambiar el idioma: Para ello seleccionaremos el idioma pulsando sobre la lista desplegable "Idioma" en la esquina superior derecha.
- Descargar el código del programa: Haremos clic en el botón descargar y el navegador descargará el contenido en un fichero.
- Limpiar la caja "Código del programa": Para ello haremos clic en el botón "Limpiar texto".
- Descargar la traza de la depuración: Dentro de la ventana de depuración, haremos clic en "Descargar traza", tras lo cual el navegador descargará un fichero de texto con dicha traza.

8. Pruebas

En el siguiente capítulo diseñaremos y usaremos métodos de prueba que verifiquen el correcto funcionamiento de nuestro proyecto.

Separaremos dichas pruebas en 2 apartados distintos. En el primero, realizaremos pruebas de instalación, con lo que aseguraremos que nuestro proyecto funciona en distintos dispositivos con Ubuntu 20.04.

Finalmente, en el segundo apartado, ejecutaremos múltiples depuraciones sobre el programa Rever y analizaremos los resultados respecto a los dados por una depuración en consola.

8.1 Pruebas de instalación

Instalaremos nuestro proyecto en 3 entornos distintos que estén ejecutando el sistema operativo Ubuntu 20.04.

8.1.1 Computador 1 - Portatil *Slimbook*

La primera instalación la realizaremos sobre un portátil de la marca Slimbook con las especificaciones que podemos encontrar en la tabla 8.1.1.1.

Especificaciones técnicas	
Procesador	AMD Ryzen 7 4800H, 8 núcleos, 16 hilos
Memoria	DDR4 64GBytes
Disco duro	2TBytes SSD Intel 660p series

Tabla 8.1.1.1 – Especificaciones computador 1

Siguiendo la guía del capítulo 7, instalamos los componentes necesarios, conectamos el servidor y accedemos a la interfaz mediante el navegador.

Esta primera instalación ha funcionado correctamente y no ha producido ningún fallo.

8.1.2 Computador 2 – Máquina Virtual

Esta segunda instalación la realizaremos sobre una máquina virtual con imagen Ubuntu 20.04, ejecutándose sobre el programa VirtualBox de Oracle, en el mismo computador definido en el apartado 8.1.1 pero sobre un entorno Windows10. Las especificaciones técnicas de la máquina virtual las podemos ver en la tabla 8.1.1.2.

Especificaciones técnicas	
Procesador	AMD Ryzen 7 4800H, 1 núcleo
Memoria	DDR4 4GBytes
Disco duro	25GBytes SSD Intel 660p series

Tabla 8.1.1.2 – Especificaciones computador 2

Al igual que en el apartado anterior, no hemos encontrado ningún fallo. Tras seguir la guía de instalación el proyecto ha funcionado perfectamente.

Sin embargo, debemos tener en cuenta que al tratarse de una máquina virtual deberemos realizar configuraciones extra desde VirtualBox para poder acceder al servicio desde otro computador de la misma red, además de configurar el cortafuegos de Windows para permitir la entrada de comunicaciones TCP sobre el puerto 80 y sobre el puerto configurado en el servidor REST.

8.1.3 Computador 3 – Raspberry PI 3B

En este caso, instalaremos nuestro proyecto sobre una Raspberry con el sistema operativo Ubuntu Server 20.04.3 LTS. Esto se debe a que este modelo no tiene suficiente potencia para ejecutar la interfaz gráfica, por lo que hemos instalado la versión servidor de Ubuntu 20.04.

Las especificaciones técnicas de la Raspberry 3B las podemos ver en la tabla 8.1.1.3.

Especificaciones técnicas	
Procesador	Quad Core Broadcom BCM2837
Memoria	1GByte
Disco duro	MicroSD 32GBytes

Tabla 8.1.1.3 – Especificaciones computador 3

La instalación en este entorno ha sido exitosa, aunque un poco más complicada en ciertos pasos dado que no dispone de interfaz gráfica.

Por desgracia, al realizar las pruebas de funcionamiento sobre este entorno, hemos notado que aparecía texto incorrecto en las respuestas de la depuración.

Tras investigarlo, hemos descubierto que esto se debe a que el procesador de la Raspberry no es capaz de procesar las respuestas en los 250 milisegundos que habíamos fijado como máximo de tiempo.

Ampliando el tiempo máximo de respuesta del proceso Rever hemos conseguido obtener el resultado correcto que esperábamos.

Además, hemos descubierto que al ejecutar Rever sobre el terminal se generaban excepciones sobre el programa. Sin embargo, al utilizar nuestra interfaz web no se generaban dichas excepciones.

Por ello, nuestro proyecto puede ser además una posible solución a entornos que como éste no permiten la ejecución del programa de manera correcta.

8.2 Pruebas de funcionamiento

En este apartado probaremos a depurar 3 programas Prolog con la interfaz de usuario. Tras esto, guardaremos los resultados de la depuración y los compararemos con los resultados de la depuración del mismo programa sobre una ejecución en consola.

De esta manera determinaremos si la interfaz está mostrando la información correcta.

8.2.1 Programa 1 - ex.pl

Este primer programa es uno de los ejemplos que podemos encontrar al descargar Rever desde repositorio de mistupv en *github*.

El programa es el siguiente:

```
p(X,Y) :- q(X), r(X,Y).
q(a).
q(b).
q(c).
r(b,b).
r(b,c).
r(c,c).
```

Y se inicia con el objetivo inicial "*p(A,B)*"
Tras ejecutar dicho programa en ambos entornos se generan las respuestas de depuración como podemos ver en la figura 8.2.1.1.

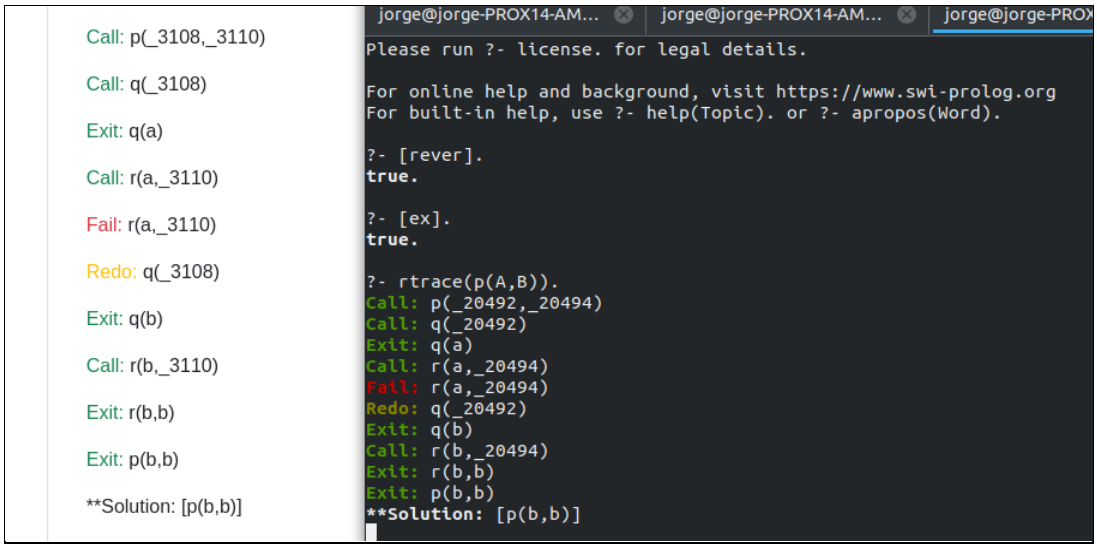

Figura 8.2.1.1 – Prueba de la depuración del programa 1

8.2.2 Programa 2 - example.pl

Este programa, al igual que el anterior, proviene del repositorio de mistupv en *github.*

Su código es el siguiente:

p(X,Y) :- q(X), r(X,Y).
q(a).
q(f(X)) :- X is 2+1.
q(c).
r(f(X),f(X)).

Y su objetivo inicial es le mismo que el del programa anterior: "*p(A,B)*"

El resultado tanto de la consola y de la interfaz web vuelven a ser el mismo. Podemos comprobarlo en la figura 8.2.2.1

Figura 8.2.2.1 – Prueba de la depuración del programa 2

8.2.1 Programa 3 - mphg.pl

Finalmente, depuraremos el programa 3, el cual ya vimos anteriormente en la figura 7.2.4.

Se trata del programa inspirado en la película “Los Monty Python y los caballeros de la mesa cuadrada”.

Ejecutaremos dicho código con el objetivo inicial “*witch(X)*”.

Tal y como podemos ver en la figura 8.2.3.1 ambos resultados son idénticos, por lo que el programa se ejecutó correctamente.

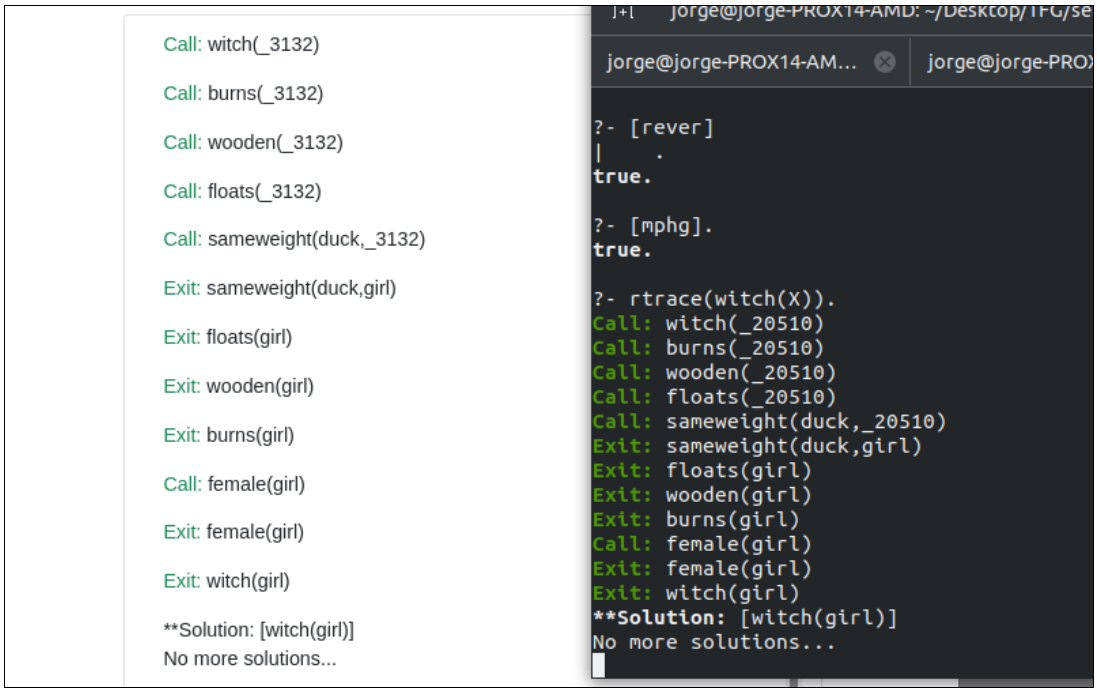

Figura 8.2.3.1 – Prueba de la depuración del programa 3

En cuanto a la diferencia entre los valores numéricos comenzados por “_” en ambas depuraciones, es necesario explicar que dichos valores son variables temporales que usa Prolog en su ejecución, y el valor mostrado no es más que un nombre temporal que el mismo Prolog asigna a dicha variable.

Por lo tanto, no es necesario que coincidan, pero sí es necesario comprobar que se encuentran en las llamadas correctas. Es decir, si en la figura 8.2.1.1 cambiáramos los valores de la interfaz “_3108” y “_3110” por “A” y “B” respectivamente, y a la vez cambiáramos los valores de la consola “_20442” y “_20494” por los mismos valores “A” y “B”, ambas ejecuciones deberían ser idénticas.

En conclusión, hemos comprobado que el programa se ejecuta correctamente sobre la interfaz, y que el proyecto puede ser instalado en diversos computadores siempre y cuando tengan como sistema operativo Ubuntu 20.04 o alguna otra versión de Linux equivalente.

9. Conclusiones

En primer lugar, comentaremos qué objetivos se ha cumplido en el desarrollo final de la interfaz web, qué errores han surgido y cómo los hemos resuelto.

A continuación, hablaremos de posibles mejoras y desarrollos futuros que se podrían implementar en dicha interfaz.

Finalmente, compararemos las tecnologías utilizadas con los estudios cursados, destacando qué conocimientos de dichos estudios hemos podido aplicar y qué otros conocimientos hemos aprendido mediante el desarrollo de este proyecto.

9.1 Objetivos alcanzados

Como conclusión, podemos observar que el desarrollo del proyecto ha resultado en un producto funcional que cumple todos y cada uno de los objetivos definidos en el primer capítulo.

En primer lugar, hemos conseguido diseñar e implementar una interfaz de usuario que es capaz de ajustarse a cualquier dispositivo donde se reproduzca, además de permitir al usuario hacer uso de todas las funciones que ofrece el depurador Rever en consola, pero de manera más sencilla e intuitiva.

Además de las funciones comentadas anteriormente, también nos permite realizar acciones como cargar o descargar ficheros con el código del programa, elegir entre inglés o español, descargar la traza de la depuración y elegir entre múltiples ejemplos de programas.

Cabe destacar que no se ha llegado a implementar el modo *debug* de Rever explicado en el capítulo 3. Esto se debe a que mediante el uso del botón “Omitir” permitimos al usuario reproducir la traza del programa de forma automática, sin tener que pulsar una vez por cada paso. Esta era la finalidad del modo *debug*, por lo que podemos afirmar que aunque no se llegue a usar en el desarrollo final, seguimos implementando la funcionalidad completa de Rever.

Desde la fase de diseño, en la estructura gráfica de la interfaz hemos implementado únicamente los elementos necesarios para su uso con el objetivo de no sobrecargar la interfaz. Gracias a ello, a la distribución de dichos elementos y a los elementos de información incluidos junto a las etiquetas descriptivas utilizadas, la interfaz final es intuitiva y sencilla.

En cuanto a la parte del servidor, hemos conseguido comunicar el servidor con la interfaz mediante una estructura de servidor REST y peticiones AJAX, que nos permitirá recibir las peticiones necesarias y contestarlas. Para ello, hemos conectado el servidor con los procesos Rever mediante una interfaz a la que hemos llamado “*SwiplNode*”.

Para poder controlar la creación de archivos, hemos implementado una clase llamada “*codeFileManager*”, la cual nos permite crear ficheros sobre una carpeta temporal de manera rápida y segura.

Además, gracias a nuestro sistema de identificación de usuarios y a la implementación por parte del servidor que permite acceder al proceso de cada usuario en cuestión, hemos conseguido que varios usuarios se encuentren conectados a la vez utilizando la interfaz sin ningún problema.

Para terminar con los objetivos cumplidos, gracias a la instalación del servidor web Apache, podremos distribuir nuestra interfaz mediante el puerto 80 a todos aquellos usuarios que lo requieran.

Por otra parte, nos encontramos con dos grandes problemas tras la fase de diseño. Por un lado, no pudimos simular las pulsaciones de las teclas “Flecha arriba” y “Flecha abajo” en la comunicación con el proceso, por lo que para solucionarlo tuvimos que modificar Rever para cambiar dichas teclas por “u” y “d”, respectivamente, dado que sí podíamos simular la pulsación de teclas con caracteres ASCII.

El segundo problema tuvo lugar en el diseño de la lógica de la interfaz y del servidor. En el caso de que un usuario saliera de la página sin cerrar el proceso, este quedaba abierto para siempre, por lo que consumía recursos del servidor. Para solucionarlo añadimos un sistema de mensajes entre la interfaz y el servidor REST, en el cual si el servidor no recibe un mensaje con el comando “*rtm*” en más de 30 segundos, el proceso se cierra.

9.2 Mejoras y futuros desarrollos

A lo largo del diseño y desarrollo del proyecto, se nos han ido ocurriendo ideas que podrían ampliar la usabilidad y seguridad del sistema, pero que no entraban dentro del alcance de nuestro proyecto o eran demasiado complejas para llevarlas a cabo en poco tiempo. A continuación describiremos algunas de ellas.

En cuanto a la seguridad del sistema, tal y como se comentó en el capítulo 6, nuestro desarrollo cubre aspectos de seguridad, pero aún quedan ciertas partes del programa en las que se podría reforzar. Más en concreto, cualquiera podría mandar al servidor REST peticiones de manera maliciosa para generar muchos procesos y que el servidor consuma todos sus recursos.

Una posible solución sería añadir al servidor una manera de limitar los procesos abiertos en función a la IP del usuario que los requiere. De esta manera un único usuario no podría ocupar todos los recursos del sistema.

Otra posible solución sería detectar el número de peticiones que realiza una determinada IP en un corto periodo de tiempo, y bloquear dicha IP en caso de ser necesario.

En cuanto a la comunicación entre el servidor y los procesos Rever, tal y como hemos podido ver en el apartado 8.1.3, el servidor podría esperar demasiado poco tiempo a recibir los datos del proceso. Por ello, se podría implementar tanto en el programa Rever como en el servidor algún tipo de palabra de control, el cual informara al proceso de que el mensaje ha terminado.

9.3 Conocimientos estudiados y aplicados

Mi paso por la universidad y más en concreto los estudios cursados en ésta me han permitido obtener los conocimientos necesarios para el desarrollo de un proyecto como este. De todos modos a lo largo de éste también he aprendido sobre tecnologías no estudiadas durante los estudios.

Entre los conocimientos estudiados, encontramos campos como:

- El funcionamiento de sistemas operativos y su interacción con los procesos de usuario, estudiados en la asignatura de Fundamentos de sistemas operativos (FSO).
- El diseño de interfaces de usuario, estudiado en las asignaturas Interfaces Persona Computador (IPC) y Desarrollo Centrado en el Usuario (DCU).
- La organización del proyecto y diseño general de este, estudiados en Ingeniería del Software (ISW) y Gestión de Proyectos (GPR).

- Todas aquellas asignaturas que me han enseñado las bases de la programación como Introducción a la Programación (IPC), Programación (PRG), Lenguajes, tecnologías y paradigmas de la programación (LTP) y Estructuras de datos y Algoritmos (EDA).
- La seguridad del sistema, estudiados en Seguridad en redes y sistemas informáticos (SSR) y Seguridad web (SEW).
- Todas aquellas asignaturas orientadas a las Tecnologías de la Información, la cual fue mi especialidad cursada, como Redes de computadores (RED), Concurrencia y sistemas distribuidos (CSD), Tecnologías y sistemas de información en la red (TSR), Desarrollo web (DEW) y Diseño de sitios web (DSW).

Además de estos conocimientos obtenidos a lo largo de los estudios, también he aprendido sobre múltiples tecnologías o he ampliado mis conocimientos en tecnologías ya estudiadas.

Entre otros, he aprendido sobre desarrollo de páginas web mediante el *framework* Bootstrap, el cual aunque había sido mencionado y utilizado en alguna de las asignaturas, se estudiaban solo ciertos aspectos de este y, en realidad, es muy amplio, con lo que he podido investigar más acerca de su funcionamiento.

También he aprendido sobre la programación lógica, mediante el estudio del intérprete SWI-Prolog, la depuración reversible y Rever.

Finalmente, he ampliado mis conocimientos sobre nodeJS, más en concreto sobre sus módulos *spawn*, *http* y *fs*; y sobre PHP y su sistema de sesiones.

Bibliografía

1. Apt, K. R. (1997). *From logic programming to Prolog.* Prentice Hall International series in computer science, Prentice Hall.

2. Byrd, L. (1980). *Understanding the Control Flow of Prolog Programs.* Tarnlund, S.A.

3. *Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification*. (7 de Junio de 2011). Obtenido de W3C: https://www.w3.org/TR/CSS21/

4. Clocksin, W. F., & Nellish, C. S. (1994). *Programming in Prolog (4. ed.).* Springer.

5. *Documentación Bootstrap v5.1*. (s.f.). Obtenido de Bootstrap: https://getbootstrap.com/docs/5.1/getting-started/introduction/

6. *Documentación NodeJs v16.8.0*. (s.f.). Obtenido de NodeJs: https://nodejs.org/api/

7. Flanagan, D. (2006). *JavaScript the definitive guide.* O'REILLY.

8. *HTML Standard*. (1 de Septiembre de 2021). Obtenido de WhatWg: https://html.spec.whatwg.org/multipage/

9. Josh, L. (2015). *Modern PHP New features and good practices.* O'REILLY.

10. Kurose, J. F., & Ross, K. W. (2017). *Redes de computadoras Un enfoque descendente.* Pearson.

11. *Página oficial SWI-Prolog*. (s.f.). Obtenido de SWI-Prolog: https://www.swi-prolog.org/

12. Pressman, R. S. (2005). *Ingeniería del software: un enfoque práctico.* McGraw-Hill.

13. *World Wide Web Consortium*. (s.f.). Obtenido de https://www.w3.org/

14. Vidal G. (2020) Reversible Computations in Logic Programming. In: Lanese I., Rawski M. (eds) Reversible Computation. RC 2020. Lecture Notes in Computer Science, vol 12227. Springer, Cham. https://doi.org/10.1007/978-3-030-52482-1_15

15. Vidal G. Reversible Debugging in Logic Programming. *Submitted for publication*. arXiv:2007.16171