Universidad Carlos III de Madrid
Escuela Politécnica Superior
Bachelor's Degree in Computer Science and Engineering

Bachelor Thesis

WepSIM: Simulador de procesador elemental con unidad de control microprogramada

Author: Javier Prieto Cepeda
Supervisor: Félix García Carballeira

Leganés, Madrid, Spain
November 2016

Insanity: doing the same thing over and over again and expecting different results.

Albert Einstein

Agradecimientos

Por agradecer ...

WepSIM: Simulador de procesador elemental con unidad de control microprogramada

by

Javier Prieto Cepeda

Abstract

Keywords: MIPS · Simulation · Assembler · Microprogramming

Supervisor: Félix García Carballeira
Title: Full Professor

Contents

List of Figures

List of Tables

Chapter 1

Introducción

El primer capítulo introduce brevemente el objetivo del proyecto, incluyendo las características clave del proyecto y su motivación (Section 1.1, *Motivación*), los objetivos del proyecto (Section 1.2, *Objetivos*), y toda la estructura del documento(Section 1.3, *Estructura del documento*).

1.1 Motivación

La enseñanza de la arquitectura de un computador es una parte básica y fundamental en la formación de los estudiantes de Ingeniería Informática mediante la cual los alumnos logran obtener una visión y comprensión del comportamiento a bajo nivel de la máquina. Para lograr que los estudiantes comprendan y asienten correctamente los fundamentos teóricos, es necesario el uso de clases prácticas, en donde el alumno pueda ser capaz de interactuar con un computador de arquitectura igual o similar a la explicada en teoría y logre extrapolar los fundamentos teóricos al comportamiento real de la máquina.

Uno de los principales problemas a la hora de diseñar estas clases prácticas es lograr obtener los medios necesarios para que los alumnos puedan hacer uso de un computador similar al visto en las clases teóricas, debido al coste que supone tener un número suficiente de computadores para el número de alumnos que deben hacer uso de ellos y su mantenimiento, la limitación de movilidad a la hora de realizar las practicas debido a la necesidad física del computador, etc. Para evitar estos problemas, actualmente se hace uso de simuladores y emuladores que proporcionan las funciones necesarias para las clases prácticas, evitando los problemas anteriormente comentados.

Un emulador es un software que se encarga de imitar el comportamiento de una computa-

dora de forma que programas pensados para una arquitectura concreta, puedan ser ejecutados en otra diferente. Por otro lado, un simulador es un software que trata reproducir el comportamiento de un computador, pero con un menor nivel de realismo, puesto que el emulador se encarga de modelar de forma precisa el dispositivo de manera que los programas ejecutados sobre él funcionen como si estuvieran siendo ejecutados en el dispositivo original.

Hay distintos simuladores que se pueden utilizar para trabajar con los principales aspectos que se tratan en las asignaturas de Estructura y Arquitectura de Computadores: ensamblador, caché, etc. Aunque la idea de usar distintos simuladores cae dentro de la estrategia de "divide y vencerás", hay dos principales problemas con estos simuladores: cuanto más realistas son más compleja se hace la enseñanza (tanto del simulador como de la tarea simulada), y cuantos más simuladores se usan más se pierde la visión de conjunto.

Hay otro problema no menos importante: la mayoría de los simuladores están pensados para PC. Uno de los objetivos que nos planteamos con WepSIM es que pudiera ser utilizado en dispositivos móviles (smartphones o tablets), para ofrecer al estudiante una mayor flexibilidad en su uso.

Además de tener un simulador portable a distintas plataformas, el simulador ha de ser lo más autocontenido posible de manera que integre la ayuda principal para su uso (no como un documento separado que sirva de manual de uso para ser impreso) permitiendo al usuario hacer un uso completo de la aplicación sin la necesidad de salir de ella.

Por todo ello, nos hemos planteado cómo ofrecer un simulador que sea simple y modular, y que permita integrar la enseñanza de la microprogramación con la programación en ensamblador. En concreto, puede utilizarse para microprogramar un juego de instrucciones y ver el funcionamiento básico de un procesador, y para crear programas en ensamblador basados en el ensamblador definido por el anterior microcódigo. Esto es de gran ayuda, por ejemplo, para la programación de sistemas dado que es posible ver cómo interactúa el software en ensamblador con el hardware en el tratamiento de interrupciones. La idea es ofrecer un simulador que ofrezca una visión global de lo que pasa en hardware y software, evitando además el tiempo extra que supone el aprendizaje de distintas herramientas.

1.2 Objetivos

El objetivo principal de este proyecto, es desarrollar un simulador, que a diferencia de los existentes, pueda simular de forma completa el comportamiento de un procesador elemental permitiendo comprobar el estado de los componentes en cada ciclo de reloj, de manera que ayude a los alumnos a comprender y asimilar de forma sencilla y visual el funcionamiento de un procesador. Los objetivos secundarios son:

- Diseñar la especificación del juego de instrucciones que permita la creación de un lenguaje ensamblador adaptado a la arquitectura del simulador.
- Diseñar e implementar el compilador del juego de instrucciones para la generación del firmware del simulador.
- Diseñar e implementar el compilador genérico de ensamblador que permita la generación del binario correspondiente al juego de instrucciones diseñado.
- Diseñar e implementar el motor del simulador permitiendo ejecuciones reales del código ensamblador correspondiente al juego de instrucciones compilado.
- Diseñar una interfaz que proporcione en todo momento la información necesaria en relación al estado de la ejecución del código, de forma sencilla y visual.
- Permitir que los usuarios puedan importar/exportar tanto la especificación del juego de instrucciones como el código ensamblador.
- Crear un mecanismo de "modificación en caliente" que permita en mitad de una ejecución modificar el juego de instrucciones, de forma que se puedan realizar pruebas sin necesidad de reiniciar la ejecución.

1.3 Estructura del documento

El documento contiene los siguientes capítulos:

- Capítulo 1, *Introducción*, presenta una breve descripción del contenido del documento. También incluye la motivación y los objetivos del proyecto.

- Capítulo 2, *Estado del arte*, incluye una descripción de los diferentes tipos de simuladores de lenguaje ensamblador y simuladores de microcódigo y presenta el trabajo relacionado.
- Capítulo 3, *Análisis*, describe brevemente el proyecto, explica la solución elegida, establece los requisitos y presenta el marco regulador del proyecto.
- Capítulo 4, *Diseño*, detalla el diseño del sistema, incluyendo todos sus componentes.
- Capítulo 5, *Implementación y despliegue*, incluye los detalles de implementación de las partes principales del software desarrollado y las características necesarias para la implementación de la aplicación.
- Capítulo 6, *Verificación, validación y evaluación*, detalla una verificación y validación completa del proyecto. También muestra una evaluación de diferentes casos de prueba utilizando el simulador.
- Capítulo 7, *Planificación y presupuesto*, presenta los conceptos relacionados con la planificación seguida, descompone todos los costes del proyecto y describe el entorno socioeconómico.
- Capítulo 8, *Conclusiones y trabajos futuros*, incluye las contribuciones del proyecto, explica las principales conclusiones del proyecto y presenta los trabajos futuros.
- Appendix A, *User Manual*, incluye un manual de usuario completo para la aplicación. Contiene un tutorial que guía al usuario desde la creación de un nuevo juego de instrucciones hasta una ejecución completa paso a paso, y una serie de ejemplos educativos para aprender el funcionamiento de un procesador mediante el uso de simulaciones utilizando el software desarrollado.

Chapter 2

Estado del arte

Este capítulo presenta el estado del arte, la última y más avanzada etapa de las tecnologías relacionadas con nuestra aplicación. Primero, se presentan los diferentes simuladores existentes para microprogramación (Section 2.1). Después, se presentan los diferentes simuladores existentes para la programación en código ensamblador (Section 2.2). Por último, realizamos una comparación de nuestro trabajo con el contexto actual de los distintos simuladores expuestos previamente (Section 2.3).

2.1 Simuladores para microprogramación

En esta sección, se explican los diferentes simuladores existentes para la microprogramación. Cabe destacar, que actualmente existen pocos simuladores que nos proporcionen esta funcionalidad, debido a que una gran parte de ellos, son utilizados de forma interna por las empresas para verificar el correcto funcionamiento de sus unidades en la fase de diseño y desarrollo. En primer lugar, vamos a centrarnos en los simuladores que están más enfocados a una labor docente.

P8080E [1] es un simulador desarrollado en el DATSI en la Facultad de Informática de la Universidad Politécnica de Madrid. Este simulador, no consta de una interfaz gráfica portable e interactiva que se ajuste a las tecnologías actuales. Para realizar poder realizar el desarrollo del microcódigo, el simulador requiere el binario completo de cada ciclo, de forma que resulta un tanto complejo su uso. Pese a ser un simulador bastante completo, no está orientado para la enseñanza de ensamblador y microprogramación con la misma herramienta, puesto que no permite la definición del juego de instrucciones con un formato diferente al definido por

defecto.

En [2], se describe el desarrollo y la implementación de un simulador para una arquitectura de microprogramación específica. Fue publicado en el año 1986, y estaba basado en la arquitectura definida en el entonces popular libro de ingeniería de computadores [3]. El simulador, requería de la definición de cada uno de los ciclos de ejecución de las intrucciones para la posterior generación de la memoria de control. Esta definición, se realizaba mediante la sintaxis definida en el libro, lo que hacía de este simulador una herramienta bastante completa al para aprender junto con el libro.

```
                SOURCE LINE
 0:  mar := pc; rd;
 1:  pc  := pc + 1; rd;
 2:  ir  := mbr;
 3:  tir := 1shift(ir + ir);
 4:  tir := 1shift(ir);
 5:  alu := tir; if n then goto 9;

 6:  mar := ir ; rd;
 7:  rd;
 8:  ac  := mbr; goto 0;

 9:  mar := ir ; mbr := ac; wr;
10:  wr; goto 0;
```

Figure 2-1: *Ejemplo de definición de microcódigo en simulador Tanenbaum.*

2.2 Simuladores para programación en ensamblador

En esta sección, se explican los diferentes simuladores existentes para la programación en ensamblador. Los simuladores más conocidos para labores docentes, son SPIM, MARS y WebMIPS.

SPIM [4], es un simulador de un procesador MIPS de 32 bits desarrollado a principios de 1990. En un primer momento implementaba el juego de instrucciones MIPS-1, utilizado por los computadores MIPS R2000/R3000. Actualmente, implementa la arquitectura MIPS32 más reciente y sus instrucciones adicionales. Permite ejecutar programas en ensamblador para esta arquitectura. Además, también permite depurar el código implementado, de forma que el alumno pueda corregir con mayor facilidad los errores cometidos. Este simulador, fue creado

por James R. Larus y tiene versiones compiladas para Windows, Mac OS X y Unix/Linux e incluso tiene una versión básica para Android, aunque su diseño no está pensado para dispositivos móviles. Puesto que el simulador está totalmente enfocado al procesador MIPS, posee el juego completo de instrucciones para la versión de 32 bits del procesador y todas las directivas de las que consta el lenguaje. Otra característica que hace de SPIM un potente simulador, es proveer de un pequeño sistema operativo que soporta las principales llamadas al sistema mediante la instrucción syscall. SPIM, es un simulador que permite múltiples configuraciones mediante su interfaz de usuario, de manera que se puede indicar:

- Activación del uso de pseudoinstrucciones.
- Simulación de predicción de saltos y accesos a memoria, con la latencia correspondiente.
- Activación del uso del manejador de la señal trap y la carga del manejador personalizado.
- Activación de la visualización de los mensajes en caso de ocurrir excepciones.
- Activación del uso de "memory-mapped IO".

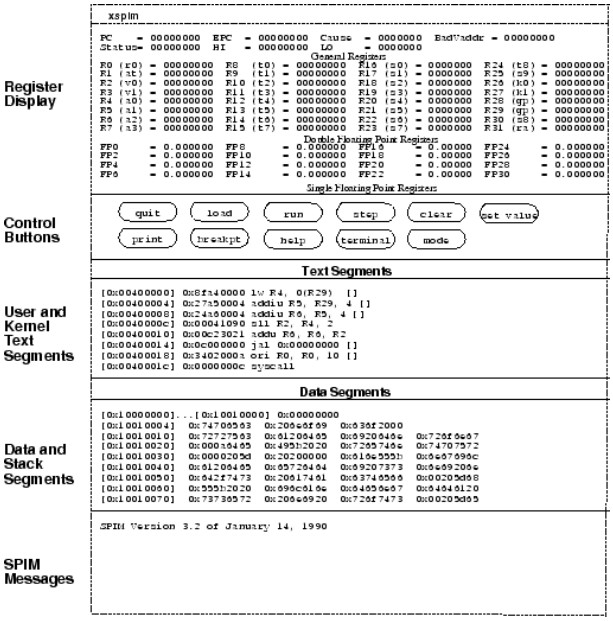


Figure 2-2: *Interfaz SPIM.*

SPIM, además cuenta con una versión más actualizada del simulador, que posee una interfaz gráfica más potente. Éste simulador se llama QtSPIM [5], y cuenta con todas las características anteriormente mencionadas de SPIM.

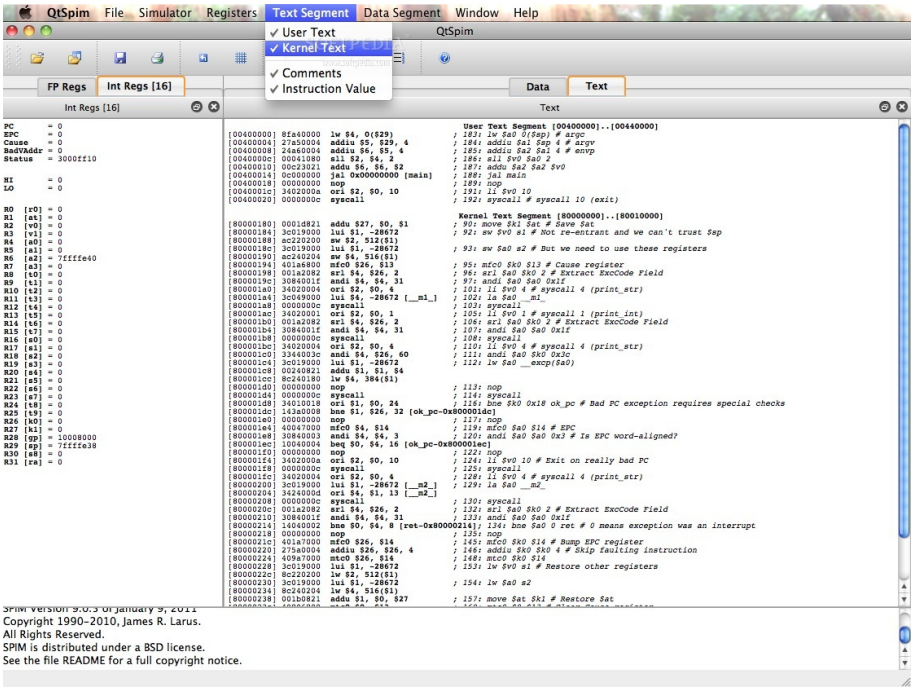


Figure 2-3: *Interfaz QtSPIM.*

MARS [6], es un simulador con interfaz gráfica desarrollado en JAVA para el lenguaje ensamblador MIPS. Fue creado en el año 2005 como una alternativa del simulador SPIM y diseñado específicamente para las necesidades limitadas de cusos de pregrado. Está basado en la arquitectura MIPS RISC, que tiene un número pequeño de elementos del lenguaje e instrucciones. Consta de operaciones de entrada/salida, saltos, condiciones y operaciones aritmético-lógicas tanto enteras como en coma flotante. Alguna de las mejoras que incorpora MARS con respecto a SPIM son [7]:

- Depuración más sencilla y cómoda, debido constar de GUI. Ésto se debe, a que para añadir un punto de ruptura, únicamente hay que clickar sobre un checkbox que tiene la instrucción.

- Los programas puedes ejecutarse de forma inversa, lo que permite volver a un estado anterior de la máquina en la ejecución de un programa.

- La velocidad de ejecución puede modificarse, de forma que si se ralentiza el usuario puede observar como cada instrucción se resalta cuando está siendo ejecutada y ver los cambios que se producen en los valores del banco de registros y de las ubicaciones de memoria.

- La memoria y los registros se pueden modificar de una manera WYSIWYG.
- Incorpora un editor de texto, eliminando dependencias externas para el uso del simulador.
- Permite que los desarrolladores puedan suministrar nuevos complementos, que por ejemplo, sirvan para extender la arquitectura o mostrar los datos de forma intuitiva.

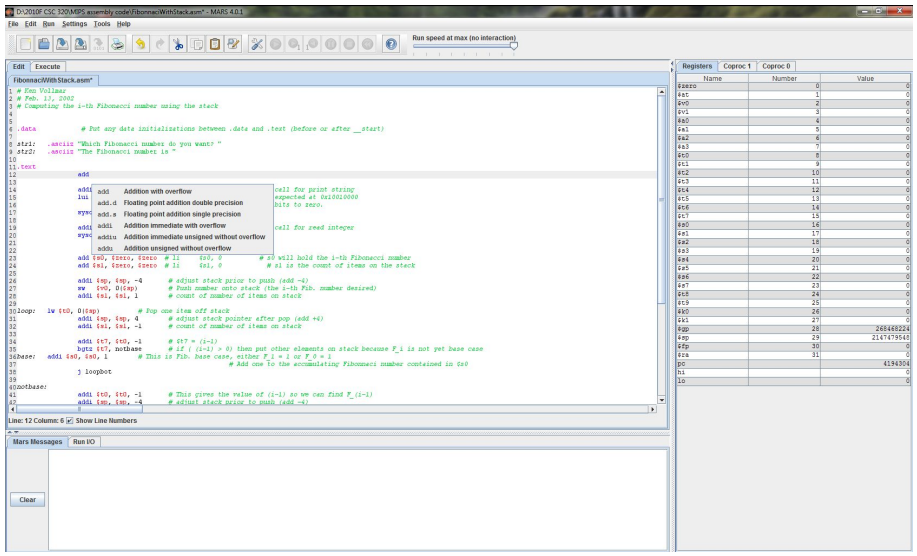


Figure 2-4: *Interfaz MARS.*

P88110, es emulador descrito en [8] que se basa en la emulación de un procesador superescalar. El propósito fundamental de este emulador, es mostrar el funcionamiento de este tipo de procesadores, haciendo visible el funcionamiento del pipeline y las dependencias existentes. Está basado en una arquitectura específica (MC88110) y pese a integrar el efecto de un procesador superescalar no integra la microprogramación, que haría de él un emulador completo.

WebMIPS [9], es un simulador desarrollado en la Facultad de Ingeniería de la Información de Sienna, Italia. Está escrito en el lenguaje de programación ASP.NET y se utiliza mediante una página web, de forma que no requiere de su instalación en local. No soporta el juego completo de instrucciones de MIPS, sino que consta de un juego reducido de instrucciones que sirve de intruducción al estudio de arquitectura de computador. Con respecto a los anteriores simuladores mencionados, difiere al mostrar el esquema arquitectónico, indicando las 5 etapas de pipeline de las que consta el computador que simula. Otra característica, es la indicación del número de ciclos de reloj que consume la ejecución del programa. Tiene como objetivo

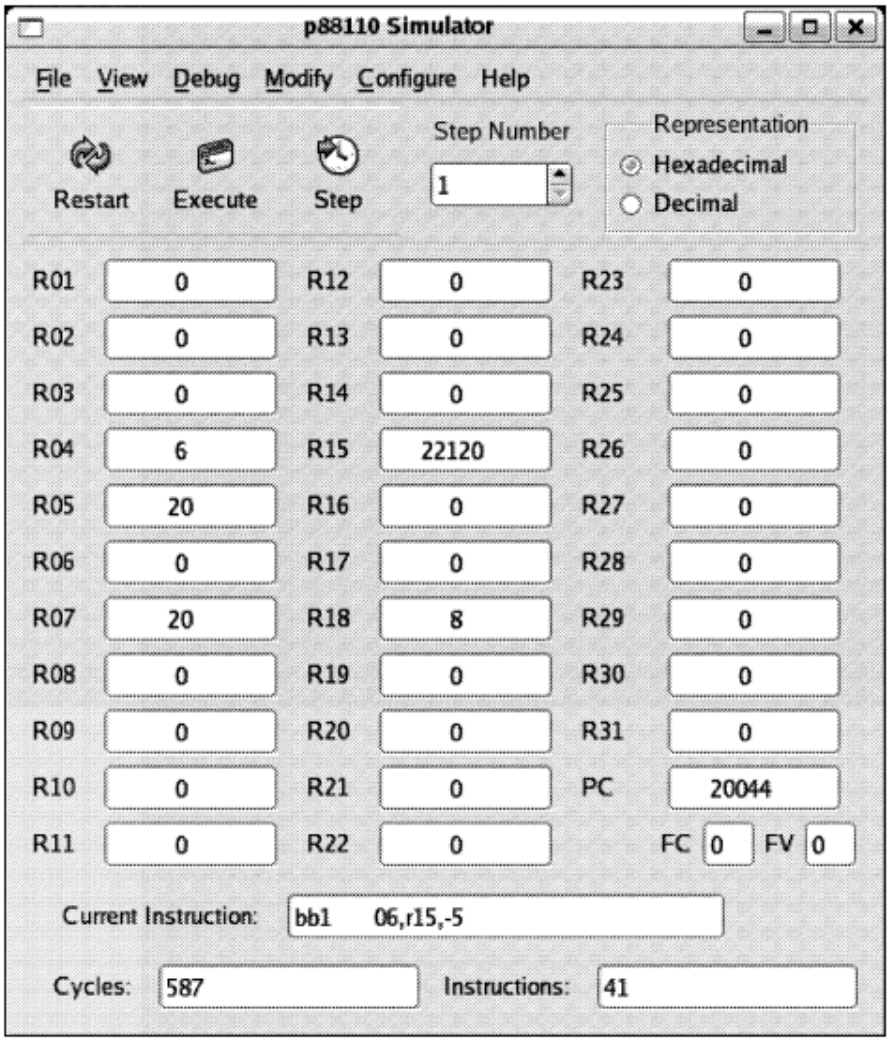


Figure 2-5: *Interfaz em88110.*

principal la demostración de la ejecución del juego de instrucciones básico explicado durante la asignatura "Arquitectura de Computadores" impartida por los creadores del simulador.

Entre las diferentes características a destacar de este simulador, se encuentran:

- Verificación del código ensamblador, comprobando que no existen errores en la definición el código.
- Dispone de códigos simples de ejemplo ("load-and-play") que facilitan la comprensión del funcionamiento del procesador.
- Permite diferentes visualizaciones del estado de la memoria y los registros del procesador.
- Consta de dos modos de ejecución: paso a paso y ejecución completa del programa.

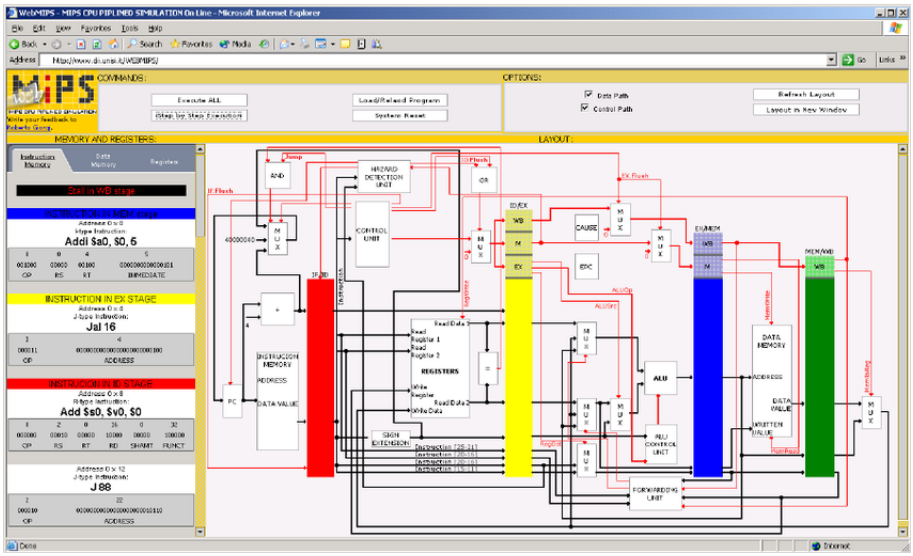


Figure 2-6: *Interfaz WebMIPS.*

2.3 Propuesta de simulación unificada

Hay distintas herramientas muy útiles para labores docentes relacionadas con la asignatura Estructura de Computadores, aunque como se comento previamente, no existe una herramienta que nos proporcione el desarrollo y simulación tanto a nivel de microcódigo como a nivel de ensamblador. WepSIM, une ambos aspectos ofreciendo:

- Visión interrelacionada de la microprogramación y la programación en ensamblador.
- Flexibilidad en la plataforma usada, pensando el dispositivos móviles.
- Herramienta autocontenida, de forma que incorpora en sí misma tanto la ayuda como distintos ejemplos que favorecen la comprensión del simulador.
- Consta de un modelo hardware que puede ser modificado o ampliado en caso de ser necesario.
- Permite definir un amplio conjunto de instrucciones máquina.
- Puede ser usado como herramienta de microprogramación o de programación en ensamblador.

De este modo, con WepSIM se pretende crear un simulador unificado, abarcando los aspectos más relevantes de la asignatura "Estructura de Computadores" y permitiendo que todas las prácticas de la asignatura puedan realizarse en la misma plataforma, evitando la pérdida de tiempo y dificultad que supone para el alumno el habituarse a diferentes entornos para cada una de las prácticas.

WepSIM, se puede definir por tanto como un simulador de programación en ensamblador y microprogramación, flexible puesto que permite la personalización del juego de instrucciones de la máquina, multiplataforma puesto que permite su uso desde diferentes tipos de dispositivos, e interactivo al poder modificar la configuración de la máquina en tiempo de ejecución.

2.4 Tecnologías web

Actualmente, gran parte de los recursos disponibles en el día a día son ofrecidos a través de Internet y su distribución es posible gracias a las tecnologías web. Estas tecnologías, permiten

Simulador	SPIM	MARS	PC88110	WebMIPS	WepSIM	P8080E	MicMac
Ensamblador	✓	✓	✓	✓	✓		
Microprogramación					✓	✓	✓
Multiplataforma	✓			✓	✓		
Interactivo					✓		
Juego de instrucciones personalizable					✓		

Table 2.1: *Comparación de simuladores de ensamblador y microcódigo.*

la construcción de las páginas web haciendo uso de las tecnologías para el desarrollo de páginas web y las tecnologías de interconexión de computadores permitiendo a los usuarios el intercambio en formato de hipertexto de todo tipo de datos y de aplicaciones software.

Cuando se realiza el desarrollo de una página web, son tres las tecnologías que se están empleando con mayor frecuencia: HTML, CSS y JavaScript. Todas estas tecnologías vienen definidas y estandarizadas por el organismo internacional W3C y cada una de ellas, tiene una función concreta en el funcionamiento de la página web:

- HTML: Es un lenguaje de marcado con el que se realiza la estructuración de la página web. Es un estándar que sirve de referencia del software que conecta con la elaboración de páginas web en sus diferentes versiones, define una estructura básica y un código (denominado código HTML) para la definición de contenido de una página web, como texto, imágenes, vídeos, juegos, entre otros.

  El lenguaje HTML basa su filosofía de desarrollo en la diferenciación. Para añadir un elemento externo a la página (imagen, vídeo, script, entre otros.), este no se incrusta directamente en el código de la página, sino que se hace una referencia a la ubicación de dicho elemento mediante texto. De este modo, la página web contiene solamente texto mientras que recae en el navegador web (interpretador del código) la tarea de unir todos los elementos y visualizar la página final. Al ser un estándar, HTML busca ser un lenguaje que permita que cualquier página web escrita en una determinada versión, pueda ser interpretada de la misma forma (estándar) por cualquier navegador web actualizado.

  En cambio, a lo largo de las diferentes versiones, se han añadido y eliminado diferentes características, con la finalidad de hacer el lenguaje más eficiente y facilitar el desarrollo de páginas web con diferentes plataformas y navegadores. Un navegador web desactual-

izado, no será capaz de interpretar correctamente una página web escrita en una versión superior de HTML a la que pueda interpretar.

- CSS: Es un lenguaje de diseño gráfico utilizado para definir y crear la presentación de un documento estructurado escrito en un lenguaje de marcado, como puede ser, HTML. Su principal finalidad, es establecer el diseño visual de las páginas web e interfaces de usuario escritas en los lenguajes HTML o XHTML. Este lenguaje además, permite aplicar estilos no visuales como pueden ser las hojas de estilo auditivas.

  Esta tecnología web, es usada por la mayoría de sitios web para crear páginas visualmente atractivas, interfaces de usuario para aplicaciones web, y GUIs para muchas de las aplicaciones móviles existentes en el mercado. Con ella, se pretende separar el contenido de la página web de la presentación, buscando mejorar la accesibilidad del documento y dando una mayor flexibilidad y control en la especificación de las características de la presentación, permitiendo que varios docuementos HTML compartan un mismo estilo usando una única hoja de estilos reduciendo la complejidad y la repetición de código en la estructura del documento.

- JavaScript: Es un lenguaje de programación interpretado orientado a las páginas web que surgió de la necesidad de ampliar las posibilidades del HTML. Su principal función en las páginas web, es realizar tareas y operaciones en el marco de la aplicación cliente como la mejora de la interfaz de usuario y la generación de páginas web dinámicas, aunque también es utilizado en el lado del servidor .

  Este lenguaje, fue diseñado con una sintaxis similar a la del lenguaje de programación C, aunque adopta nombres y convenciones del lenguaje de programación Java. En cambio, Java y Javascript tienen propósitos y semánticas distintos.

  En la actualidad, todos los navegadores realizan la interpretación del código JavaScript integrado en las páginas web. Para poder interactuar con una página web el lenguaje está provisto de una implementación del DOM que facilita el control y manejo de las interacciones del usuario.

Chapter 3

Análisis

El objetivo principal de este capitulo, es describir el proyecto mediante la obtención y especificación de los requisitos del simulador, que puede proporcionar información suficiente para un análisis detallado que, por lo tanto, puede servir para continuar diseñando e implementando (Capítulos 4, *Diseño*; and 5, *Implementación y despliegue*) un software que cumpla con esos requisitos.

Con el fin de obtener los requisitos del sistema, el tutor ha desempeñado el papel del cliente en diferentes reuniones, mientras que el alumno ha desempeñado los roles de analista, diseñador, programador y probador.

La sección 3.1 resume brevemente la descripción del proyecto. La sección 3.2 especifica los requisitos del sistema, empezando con los requisitos de usuario y finalizando con los requisitos funcionales y no-funcionales. La sección 3.2.2 especifica los caso de uso del sistema. Finalmente, la sección 3.3 indica el conjunto de leyes y regulaciones para la gestión del software.

3.1 Descripción del proyecto

El objetivo de este proyecto es construir una herramienta que permita simular con realismo el comportamiento de un procesador basado en la arquitectura indicada por el el tutor del proyecto, de forma que sirva como única herramienta para los alumnos a lo largo de la asignatura Estructura de Computadores.

Los simuladores actuales para la enseñanza de Estructura de Computadores, están focalizados a una función concreta, como puede ser la simulación de cachés, la simulación de código ensamblador o la simulación de microcódigo entre otros, pero a la hora de unificar todas estas

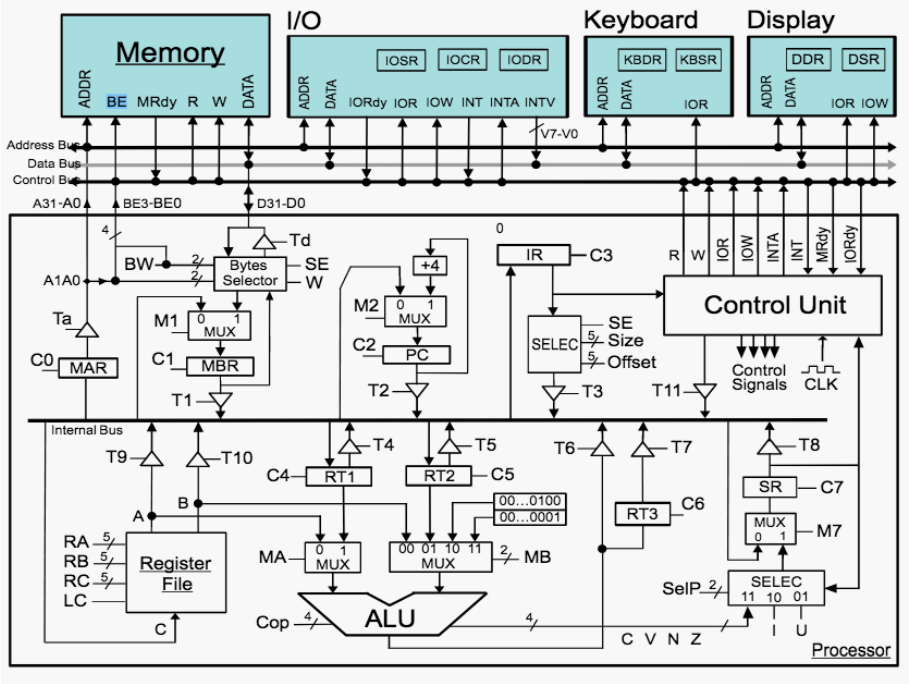


Figure 3-1: *Arquitectura CPU WepSIM .*

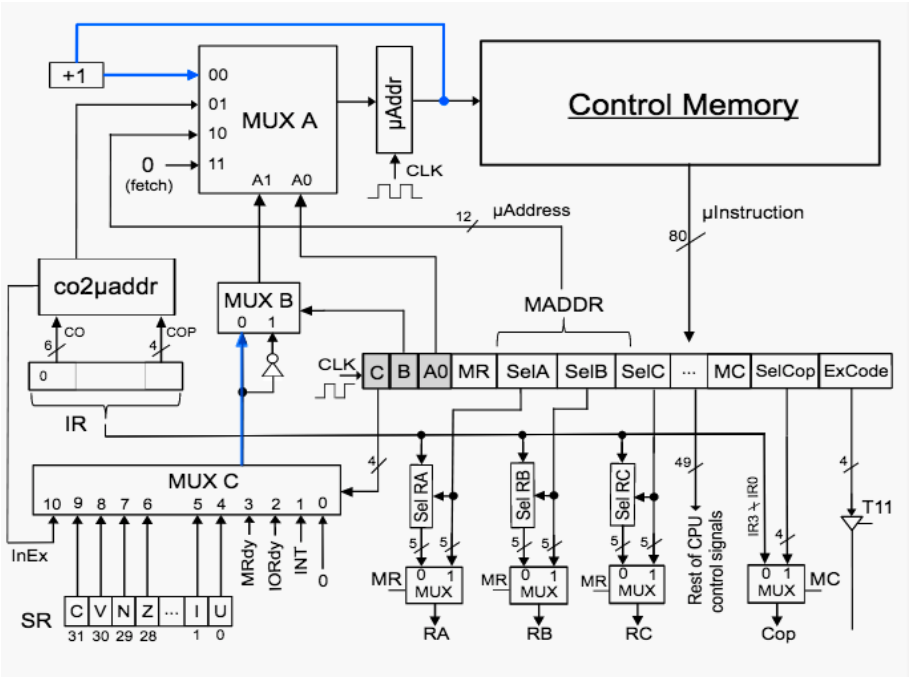


Figure 3-2: *Arquitectura unidad de control WepSIM.*

funcionalidades en una única herramienta, existe un vacío que genera una pérdida de tiempo en el aprendizaje de cada herramienta y la no posibilidad de ver una simulación completa.

El reto al que se enfrente cualquier simulador educativo es el de ser capaz de simular fielmente el funcionamiento de un dispositivo permitiendo al estudiante poder observar con el mayor detalle posible el comportamiento éste.

El sistema que se propone debe ser capaz de simular con realismo el comportamiento del procesador, permitiendo la definición del juego de instrucciones a utilizar para el posterior desarrollo de código ensamblador y su ejecución en el simulador con un alto nivel de detalle en cada uno de los ciclos de ejecución,.De esta forma, los estudiantes podrán comprender fácilmente los contenidos teóricos expuestos en la asignatura y serán capaces de realizar todas las prácticas en una misma herramienta, pudiendo ver de forma incremental el desarrollo de software de bajo nivel sobre un procesador elemental.

3.1.1 Procesador elemental WepSIM

En la figura 3-1, se muestra la estructura del Procesador Elemental WepSIM. WepSIM, consta de un módulo de memoria, un dispositivo teclado, un dispositivo pantalla y un dispositivo de E/S genérico que se puede utilizar para trabajar con interrupciones.

WepSIM es un procesador de 32 bits que direcciona la memoria por bytes, que cuenta con un banco de 32 registros y dos registros adicionales (RT1 y RT2) que no son visibles para el programador de ensamblador, pero que permiten el almacenamiento temporal de datos para la realización de operaciones intermedias. Desde los registros, es posible enviar los valores para operar en una ALU que dispone de las 15 operaciones aritmético-lógicas más comunes. El registro PC tiene su propio operador de sumar cuatro, de forma que no es necesario hacer uso de la ALU para esta operación. El resultado de las operaciones realizadas en la ALU puede ser almacenado en un registro temporal (RT3) que también es invisible para el programador de ensamblador, o ser enviado directamente al bus interno a través del correspondiente triestado.

El registro de estado (SR) puede ser actualizado con los flags resultantes de la última operación de la alu (O, N y Z). Para ello, SELEC/SelP representa un bloque de circuitos que permite indicar qué parte del registro de estado (SR) debe ser actualizado. A la derecha de SELEC/SelP llegan los bits del registro de estado SR como entrada (Input=O N Z I U) y SelP permite seleccionar qué grupo de estos bits se actualizará en el registro de estado: los bits O, N y Z con los valores procedentes de la ALU, el bit I con el valor indicado o el bit U con el valor

indicado para el mismo.

El registro de instrucción (IR) tiene asociado un módulo selector (circuito de más alto nivel que un multiplexor, etc.) que permite seleccionar un segmento del valor binario almacenado en el registro de instrucción que pasará hacia T3.

En concreto, se indica la posición (Offset, donde 0 represente el bit menos significativo del registro IR) inicial y el número de bits (Size) a tomar a partir de dicha posición inicial, así como si se desea hacer extensión de signo (SE) antes de pasar el valor a la entrada de T3.

Los registros MAR y MBR se usan para almacenar la dirección y el contenido asociado a esta dirección en las operaciones de lectura/escritura con la memoria. La memoria está diseñada para un funcionamiento síncrono o asíncrono. Actualmente funciona de forma síncrona, pero dispone de la señal MRdy para en un futuro trabajar de forma asíncrona. El circuito de selección permite indicar qué porción de la palabra de memoria es la que se desea (un byte, dos bytes o una palabra completa de cuatro bytes).

Se dispone también de tres dispositivos de E/S: un teclado, una pantalla y un dispositivo genérico que se puede configurar para generar diversos tipos de interrupciones.

Finalmente, la Unidad de Control genera las señales de control para cada ciclo de reloj. La figura 3-2 muestra la Unidad de Control con mayor detalle. Se trata de una unidad de control microprogramada con secuenciamiento implícito. Las señales de control para el ciclo de reloj actual se almacenan en el registro de micro-instrucción (aquel con los campos A0, B, C, SelA, etc.). El contenido de este registro proviene de la memoria de control, concretamente del contenido en la posición a la que apunta el registro de micro-dirección. La micro-dirección almacenada en este registro puede modificarse usando el multiplexor "MUX A". Hay cuatro opciones: la microdirección actual más uno, una micro-dirección indicada en la propia micro-instrucción (que se solapa con SelA, SelB y parcialmente con SelE), la primera micro-dirección asociada al campo de código de operación de la instrucción del registro IR, y finalmente el valor cero, que es la dirección de la memoria de control donde se almacena la microrrutina correspondiente al fetch. La microdirección se puede seleccionar de forma condicional, para ello se usa el multiplexor "MUX C" que permite seleccionar los bits del registro de estado (SR) o valores de las señales de control de E/S.

Para generar los valores correspondientes a las señales selectoras del banco de registros RA, RB y RE se usan los circuitos selectores SelRA, SelRB y SelRE. Estos selectores toman como entrada los 32 bits del registro de instrucción (IR) por un lado y el campo SelA, SelB y SelE

por otro, de forma que toman SelX como el desplazamiento dentro del registro de instrucción (de 0 a 32) desde donde tomar los siguientes 5 bits correspondientes a las señales RA, RB o RE. Permiten por tanto seleccionar 5 bits consecutivos de los 32 bits del registro de instrucción.

El multiplexor MR permite indicar si las señales RA, RB y RE serán literalmente los valores almacenados en SelA, SelB y SelE (MR=1) o bien si SelA, SelB y SelE indican el desplazamiento dentro de la instrucción donde están los valores a utilizar para RA, RB y RE. Esto último permite que en la instrucción se pueda indicar los registros a usar como operandos en el banco de registros, en lugar de indicarlos desde la micro-instrucción.

Para la señal Cop (código de operación en la ALU) la señal MC permite tomar el valor SelCop de la micro-instrucción (MC=1) o bien los 4 bits menos significativos del registro de instrucción (MC=0), es decir IR3-IR0.

En [10] se puede encontrar una descripción más detallada del procesador descrito anteriormente.

3.2 Requisitos

Esta sección proporciona una descripción detallada de los requisitos de la aplicación. Para la tarea de la especificación de requisitos, se han seguido las prácticas recomendadas por IEEE [11]. De acuerdo con estas prácticas, una buena especificación debe abordar la funcionalidad del software, los problemas de rendimiento, las interfaces externas, otras características no funcionales y las limitaciones de diseño o implementación. Además, la especificación de los requisitos debe ser:

- **Completa:** el documento refleja todos los requisitos de software importantes.
- **Consistente:** los requisitos no deben generar conflictos entre sí.
- **Correcta:** cada requisito debe ser cumplido por el software según las necesidades del usuario.
- **Modificable:** la estructura de la especificación permite cambios en los requisitos de una manera simple, completa y consistente.
- **Clasificación basada en la importancia y la estabilidad:** cada requisito debe indicar su importancia y su estabilidad.

- **Trazable:** el origen de cada requisito es claro y se puede hacer referencia fácilmente en otras etapas.
- **Inequívoco:** cada requisito tiene una sola interpretación.
- **Verificable:** Cada requisito debe ser verificable, es decir, existe algún proceso para verificar que el software cumple con cada requisito.

A partir de los requisitos de los usuarios, que constituyen una referencia informal al comportamiento del producto que el cliente espera, derivamos los requisitos de software (en este caso, requisitos funcionales y no funcionales) que guiaron el proceso de diseño con información específica sobre la funcionalidad del sistema y otras características. Los requisitos recuperados se estructuraron de acuerdo con el siguiente esquema:

1. **Requisitos de Usuario**
   (a) **Capacidad:** el requisito describe la funcionalidad esperada del sistema como en casos de uso.
   (b) **Restricción:** el requisito especifica las restricciones o condiciones que el sistema debe cumplir.
2. **Requisitos de Software**
   (a) **Funcionales**
      i. **Funcional:** el requisito describe la funcionalidad básica y el propósito del sistema mientras se minimiza la ambigüedad.
      ii. **Inverso:** el requisito limita la funcionalidad de la aplicación para aclarar su alcance.
   (b) **No-Funcionales**
      i. **Rendimiento:** el requisito se relaciona con el rendimiento mínimo requerido del sistema resultante.
      ii. **Interfaz:** el requisito está relacionado con la interfaz de usuario de la aplicación.
      iii. **Escalabilidad:** el requisito está relacionado con la capacidad del sistema para adaptarse a cargas de trabajo cada vez mayores.

iv. **Plataforma:** el requisito especifica las plataformas subyacentes de software y hardware en las que funcionará el sistema.

La tabla 3.1 proporciona la plantilla utilizada para la especificación de requisitos. Tenga en cuenta que para los requisitos del usuario, el formato de identificación será UR-XYY, donde X indica el subtipo de requisito: requisitos de capacidad (C) o restricciones (R). YY corresponde al número de requisito en su subcategoría. Para los requisitos de software, se utilizará el formato de identificación SR-X-YZZ, donde X indica si es un requisito funcional (F) o no funcional (NF), e Y representa su subcategoría: funcional (F), inversa (I)), Rendimiento (P), interfaz (UI), escalabilidad (S) o plataforma (PL). ZZ corresponde al número de requisito en su subcategoría.

ID	Requisito ID.
Nombre	Nombre del requisito.
Tipo	Indica la categoría en la que se colocaría el requisito de acuerdo con el esquema descrito anteriormente.
Origen	Constituye la fuente del requisito. Puede ser el usuario, otro requisito u otros actores involucrados en el proyecto.
Prioridad	Indica la prioridad del requisito según su importancia. Un requisito puede ser identificado como *esencial*, *condicional* or *opcional*.
Estabilidad	Indica la variabilidad del requisito a través del proceso de desarrollo, definido como *estable* or *inestable*.
Descripción	Explicación detallada del requisito.

Table 3.1: *Plantilla para la especificación de requisitos.*

3.2.1 Requisitos de Usuario

Esta subsección especifica los requisitos de usuario.

ID	UR-C01
Nombre	Simulación del modelo hardware propuesto
Tipo	Capacidad
Origen	Usuario
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta simulará el comportamiento de la arquitectura definida para la asignatura Estructura de Computadores.

Table 3.2: *Requisito de usuario UR-C01.*

ID	UR-C02
Nombre	Definición de juego de instrucciones del simulador
Tipo	Capacidad
Origen	Usuario
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta debe permitir la definición del juego de instrucciones a utilizar por el usuario siguiendo el formato especificado por el tutor.

Table 3.3: *Requisito de usuario UR-C02.*

ID	UR-C03
Nombre	Operaciones con el juego de instrucciones
Tipo	Capacidad
Origen	Usuario
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta debe permitir al usuario las siguientes operaciones con el juego de instrucciones: cargar desde fichero, exportar a un fichero y generar el firmware.

Table 3.4: *Requisito de usuario UR-C03.*

ID	UR-C04
Nombre	Definición de juego del código ensamblador
Tipo	Capacidad
Origen	Usuario
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta debe permitir la definición del código ensamblador a simular siguiendo el formato utilizado en la asignatura Estructura de Computadores y el juego de instrucciones cargado previamente.

Table 3.5: *Requisito de usuario UR-C04.*

ID	UR-C05
Nombre	Operaciones con el código ensamblador
Tipo	Capacidad
Origen	Usuario
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta debe permitir al usuario las siguientes operaciones con el código ensamblador: cargar desde fichero, exportar a un fichero y generar el binario asociado.

Table 3.6: *Requisito de usuario UR-C05.*

ID	UR-C06
Nombre	Tipos de simulación
Tipo	Capacidad
Origen	Usuario
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta debe permitir tres tipos diferentes de simulación: simulación ciclo a ciclo de reloj, simulación instrucción a instrucción y simulación completa del código.

Table 3.7: *Requisito de usuario UR-C06.*

ID	UR-C07
Nombre	Configuración de la velocidad de las simulaciones
Tipo	Capacidad
Origen	Usuario
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta debe permitir al usuario seleccionar la velocidad de la simulación.

Table 3.8: *Requisito de usuario UR-C07.*

ID	UR-C08
Nombre	Esquemas de la arquitectura
Tipo	Capacidad
Origen	Usuario
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta debe mostrar la arquitectura de la CPU y la Unidad de Control que forman el simulador.

Table 3.9: *Requisito de usuario UR-C08.*

ID	UR-C09
Nombre	Información a mostrar
Tipo	Capacidad
Origen	Usuario
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta debe mostrar al usuario la información del estado del simulador en cada paso de la simulación.

Table 3.10: *Requisito de usuario UR-C09.*

ID	UR-R01
Nombre	Herramienta web
Tipo	Restricción
Origen	Usuario
Prioridad	Esencial
Estabilidad	Estable
Descripción	El simulador debe de ser una herramienta web.

Table 3.11: *Requisito de usuario UR-R01.*

ID	UR-R02
Nombre	Navegadores web
Tipo	Restricción
Origen	Usuario
Prioridad	Esencial
Estabilidad	Estable
Descripción	El simulador debe de funcionar en los siguiente navegadores web: Microsoft edge, Mozilla Firefox, Google Chrome y Safari.

Table 3.12: *Requisito de usuario UR-R02.*

ID	UR-R03
Nombre	Interfaz de usuario
Tipo	Restricción
Origen	Usuario
Prioridad	Esencial
Estabilidad	Estable
Descripción	La interfaz de usuario debe de ser compatible para uso en pc's y smartphones.

Table 3.13: *Requisito de usuario UR-R03.*

ID	UR-R04
Nombre	Tiempo por ciclo de reloj
Tipo	Restricción
Origen	Usuario
Prioridad	Esencial
Estabilidad	Estable
Descripción	El tiempo de ejecución de un ciclo de reloj del simulador no debe de sobrepasar 0,1 segundos.

Table 3.14: *Requisito de usuario UR-R04.*

ID	UR-R05
Nombre	Ejecución en local
Tipo	Restricción
Origen	Usuario
Prioridad	Esencial
Estabilidad	Estable
Descripción	Las simulaciones serán realizadas en la máquina del usuario, siendo únicamente necesario el servidor para el acceso a la herramienta.

Table 3.15: *Requisito de usuario UR-R05.*

ID	UR-R06
Nombre	Conexión a internet
Tipo	Restricción
Origen	Usuario
Prioridad	Esencial
Estabilidad	Estable
Descripción	Para realizar una simulación, el dispositivo del usuario no necesita conexión a internet.

Table 3.16: *Requisito de usuario UR-R06.*

ID	UR-R07
Nombre	Tecnologías de desarrollo
Tipo	Restricción
Origen	Usuario
Prioridad	Esencial
Estabilidad	Estable
Descripción	El simulador debe de ser desarrollado mediante el lenguaje de programación JavaScript, posibilitando migraciones futuras.

Table 3.17: *Requisito de usuario UR-R07.*

3.2.2 Modelo de Casos de Uso

Un caso de uso representa un uso típico que realizará alguno de los futuros usuarios del sistema desarrollado. El diagrama que se muestra a continuación representa los casos de uso de un usuario que utilice esta herramienta desarrollada. Se ha partido el diagrama en dos para hacerlo mucho más legible. A continuación se muestra la primera parte del diagrama. En la página siguiente se mostrará la segunda parte.

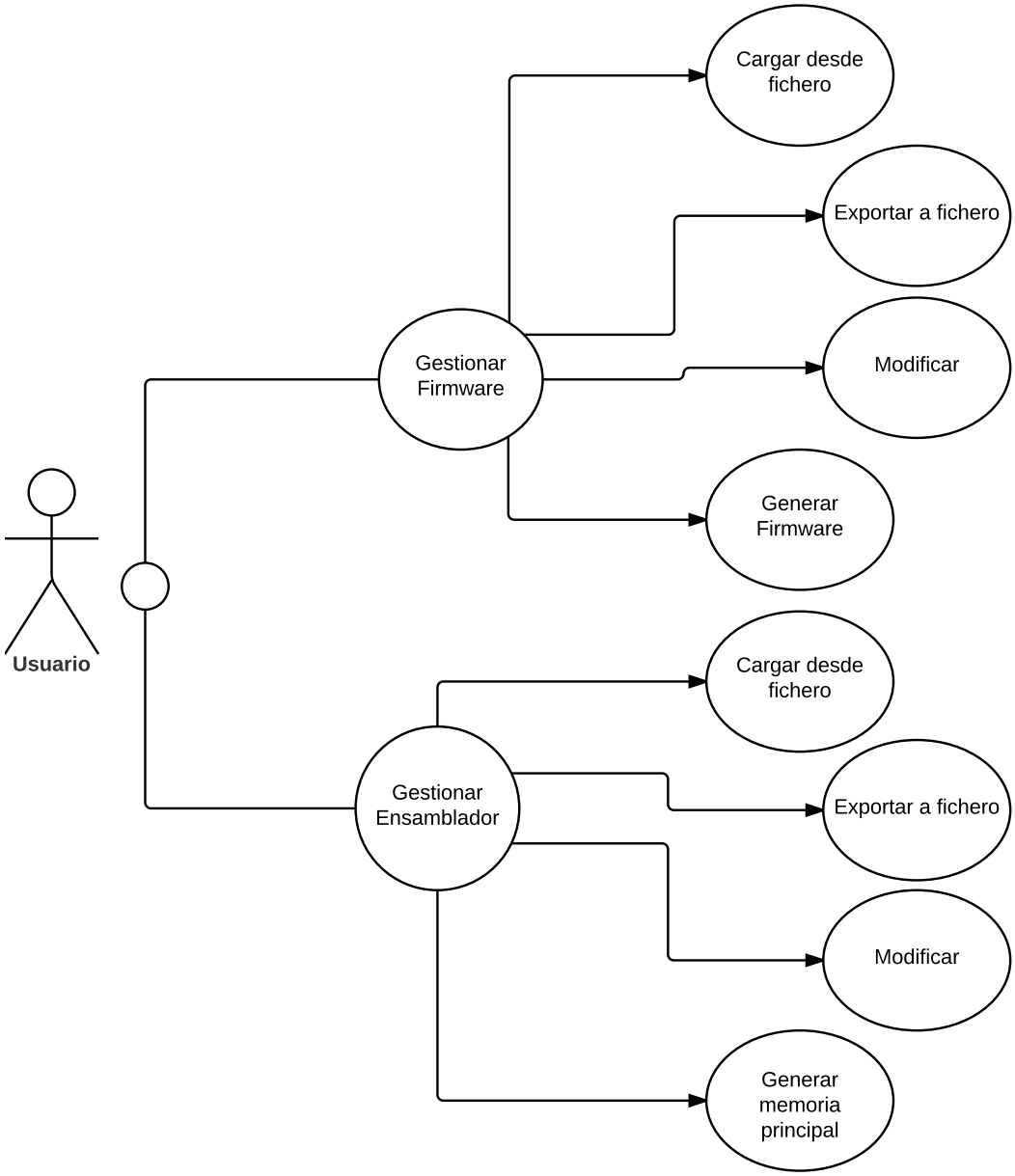


Figure 3-3: *Diagrama Modelo Casos de Uso 1.*

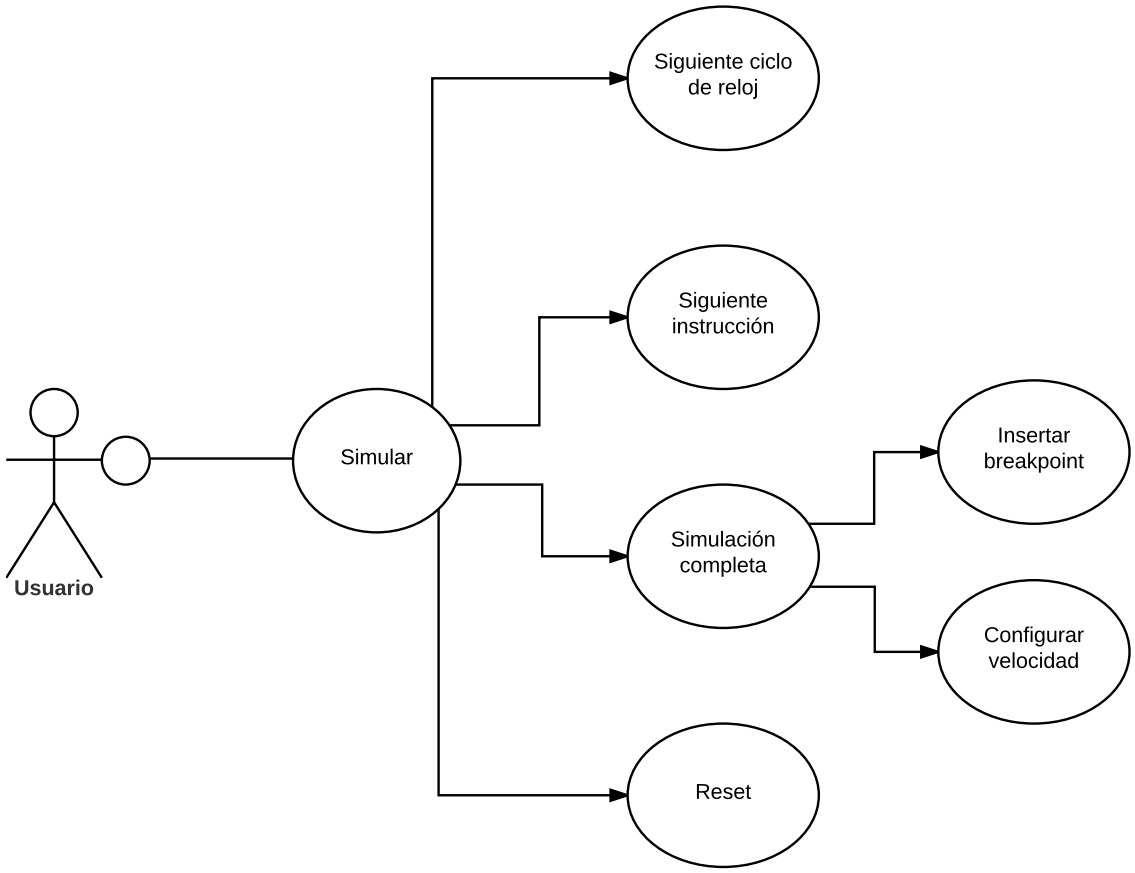


Figure 3-4: *Diagrama Modelo Casos de Uso 2.*

Para realizar la especificación clara, completa y detallada de cada uno de los casos de uso recogidos en el diagrama anterior se utilizará una tabla donde se recogerán los siguientes campos de información.

La tabla 3.18 proporciona la plantilla utilizada para la especificación de casos de uso. Tenga en cuenta que el formato de identificación será UC-XX, donde XX corresponde al número de caso de uso.

ID	Caso de Uso ID.
Nombre	Nombre del Caso de Uso.
Actores	Describe el actor o los actores que intervienen en la realización del caso de uso.
Objetivo	Describe textualmente el cometido concreto del caso de uso.
Precondiciones	Muestra el estado del sistema que debe darse para que se pueda realizar el caso de uso.
Postcondiciones	Presenta el estado del sistema tras la realización del caso de uso.
Escenario Básico	Especifica la secuencia de pasos principales que se efectúan para realizar el caso de uso.

Table 3.18: *Plantilla de caso de uso.*

ID	UC-01
Nombre	Cargar juego de instrucciones
Actores	Usuario
Objetivo	Cargar en la herramienta un nuevo juego de instrucciones desde un fichero que indica el usuario.
Precondiciones	Ninguna.
Postcondiciones	El juego de instrucciones se carga en la herramienta, mostrándose en el editor de texto correspondiente al firmware.
Escenario Básico	• El usuario selecciona la pestaña firmware. • Pulsa el botón cargar firmware. • Selecciona el fichero deseado y acepta.

Table 3.19: *Caso de Uso UC-01.*

ID	UC-02
Nombre	Exportar juego de instrucciones
Actores	Usuario
Objetivo	Exportar a un fichero el juego de instrucciones cargado en la memoria de control.
Precondiciones	La memoria de control ha sido generada.
Postcondiciones	Se genera un fichero en el dispositivo del usuario con el juego de instrucciones cargado previamente en la memoria de control.
Escenario Básico	• El usuario selecciona la pestaña firmware. • Pulsa el botón guardar firmware. • Selecciona el nombre de fichero deseado y acepta.

Table 3.20: *Caso de Uso UC-02*

ID	UC-03
Nombre	Modificar juego de instrucciones
Actores	Usuario
Objetivo	Editar el juego de instrucciones desde la herramienta.
Precondiciones	Ninguna.
Postcondiciones	Las modificaciones realizadas por el usuario en el juego de instrucciones son mostradas en el editor de texto correspondiente al firmware.
Escenario Básico	• El usuario selecciona la pestaña firmware. • Realiza las modificaciones deseadas en el editor de texto.

Table 3.21: *Caso de Uso UC-03.*

ID	UC-04
Nombre	Generar firmware
Actores	Usuario
Objetivo	Generar la memoria de control a partir del juego de instrucciones definido en la herramienta.
Precondiciones	El juego de instrucciones debe estar definido en la herramienta.
Postcondiciones	Se genera la memoria de control asociada al juego de instrucciones definido por el usuario.
Escenario Básico	• El usuario selecciona la pestaña firmware. • Define el juego de instrucciones deseado en la herramienta. • Pulsa el botón generar firmware.

Table 3.22: *Caso de Uso UC-04.*

ID	UC-05
Nombre	Cargar ensamblador
Actores	Usuario
Objetivo	Cargar en la herramienta un nuevo código ensamblador desde un fichero que indica el usuario.
Precondiciones	Ninguna.
Postcondiciones	El código ensamblador se carga en la herramienta, mostrándose en el editor de texto correspondiente al ensamblador.
Escenario Básico	• El usuario selecciona la pestaña ensamblador. • Pulsa el botón cargar ensamblador. • Selecciona el fichero deseado y acepta.

Table 3.23: *Caso de Uso UC-05.*

ID	UC-06
Nombre	Exportar ensamblador
Actores	Usuario
Objetivo	Exportar a un fichero el código ensamblador cargado para la simulación.
Precondiciones	El código ensamblador ha sido cargado en el simulador.
Postcondiciones	Se genera un fichero en el dispositivo del usuario con el código ensamblador cargado previamente en el simulador.
Escenario Básico	• El usuario selecciona la pestaña ensamblador. • Pulsa el botón guardar ensamblador. • Selecciona el nombre de fichero deseado y acepta.

Table 3.24: *Caso de Uso UC-06.*

ID	UC-07
Nombre	Modificar ensamblador
Actores	Usuario
Objetivo	Editar el código ensamblador desde la herramienta.
Precondiciones	Ninguna.
Postcondiciones	Las modificaciones realizadas por el usuario en el juego de instrucciones son mostradas en el editor de texto correspondiente al firmware.
Escenario Básico	• El usuario selecciona la pestaña ensamblador. • Realiza las modificaciones deseadas en el editor de texto.

Table 3.25: *Caso de Uso UC-07.*

ID	UC-08
Nombre	Generar memoria principal
Actores	Usuario
Objetivo	Cargar el código ensamblador definido por el usuario en el simulador, generando el contenido de la memoria principal correspondiente al ensamblador.
Precondiciones	El código ensamblador debe estar definido en la herramienta.
Postcondiciones	Se genera la memoria principal asociada al código ensamblador definido por el usuario.
Escenario Básico	• El usuario selecciona la pestaña ensamblador. • Define el código ensamblador deseado en la herramienta. • Pulsa el botón compilar.

Table 3.26: *Caso de Uso UC-08.*

ID	UC-09
Nombre	Siguiente ciclo de reloj
Actores	Usuario
Objetivo	Avanzar un ciclo de reloj en la simulación.
Precondiciones	La memoria de control y la memoria principal deben de estar generadas.
Postcondiciones	La ejecución de la simulación avanza un ciclo de reloj, siendo modificado el estado del simulador.
Escenario Básico	• El usuario selecciona la simulador. • Pulsa el botón siguiente micro-instrucción.

Table 3.27: *Caso de Uso UC-09.*

ID	UC-10
Nombre	Siguiente instrucción
Actores	Usuario
Objetivo	Avanzar una instrucción en la simulación.
Precondiciones	La memoria de control y la memoria principal deben de estar generadas.
Postcondiciones	La ejecución de la simulación avanza una instrucción, siendo modificado el estado del simulador.
Escenario Básico	• El usuario selecciona la simulador. • Pulsa el botón siguiente instrucción.

Table 3.28: *Caso de Uso UC-10.*

ID	UC-11
Nombre	Insertar breakpoint
Actores	Usuario
Objetivo	Insertar un punto de detención en una instrucción del código ensamblador a simular.
Precondiciones	La memoria de control y la memoria principal deben de estar generadas.
Postcondiciones	Se añade un breakpoint en la simulación.
Escenario Básico	• El usuario selecciona la pestaña simulador. • Selecciona la pestaña Assembly Debugger. • Pulsa sobre la instrucción en la que añadir el breakpoint.

Table 3.29: *Caso de Uso UC-11.*

ID	UC-12
Nombre	Configurar velocidad
Actores	Usuario
Objetivo	Configurar la velocidad de la simulación.
Precondiciones	La memoria de control y la memoria principal deben de estar generadas.
Postcondiciones	Se modifica la velocidad de la simulación.
Escenario Básico	• El usuario selecciona la pestaña simulador. • Modifica la velocidad mediante la barra deslizable de velocidad.

Table 3.30: *Caso de Uso UC-12.*

ID	UC-13
Nombre	Reinicio del simulador
Actores	Usuario
Objetivo	Reiniciar la simulación.
Precondiciones	La memoria de control y la memoria principal deben de estar generadas.
Postcondiciones	El simulador queda reiniciado y preparado para el comienzo de la simulación.
Escenario Básico	• El usuario selecciona la pestaña simulador. • Pulsa sobre el botón reset.

Table 3.31: *Caso de Uso UC-13.*

3.2.3 Requisitos Funcionales

Esta subsección especifica los requisitos funcionales.

ID	SR-F-F01
Nombre	Arquitectura de 32 bits
Tipo	Funcional
Origen	UR-C01
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta debe simular una arquitectura de 32 bits.

Table 3.32: *Requisito Funcional SR-F-F01.*

ID	SR-F-F02
Nombre	Unidad de control microprogramable
Tipo	Funcional
Origen	UR-C01
Prioridad	Esencial
Estabilidad	Estable
Descripción	La unidad de control debe de ser micoprogramable.

Table 3.33: *Requisito Funcional SR-F-F02.*

ID	SR-F-F03
Nombre	Datos en complemento a2
Tipo	Funcional
Origen	UR-C01
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta operará con números enteros en complemento a2.

Table 3.34: *Requisito Funcional SR-F-F03.*

ID	SR-F-F04
Nombre	Definición de juego de instrucciones
Tipo	Funcional
Origen	UR-C02
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta debe permitir al usuario la definición del juego de instrucciones a utilizar.

Table 3.35: *Requisito Funcional SR-F-F04.*

ID	SR-F-F05
Nombre	Formato del juego de instrucciones
Tipo	Funcional
Origen	UR-C02
Prioridad	Esencial
Estabilidad	Estable
Descripción	El juego de instrucciones debe seguir el formato utilizado en la asignatura Estructura de Computadores, definiendo instrucciones del lenguaje ensamblador, nombrado de los registros y pseudoinstrucciones.

Table 3.36: *Requisito Funcional SR-F-F05.*

ID	SR-F-F06
Nombre	Modificación del juego de instrucciones
Tipo	Funcional
Origen	UR-C03
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta debe de permitir la modificación del juego de instrucciones a utilizar por el usuario.

Table 3.37: *Requisito Funcional SR-F-F06.*

ID	SR-F-F07
Nombre	Carga de juego de instrucciones
Tipo	Funcional
Origen	UR-C03
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta debe permitir la carga de un fichero que contenga el juego de instrucciones.

Table 3.38: *Requisito Funcional SR-F-F07.*

ID	SR-F-F08
Nombre	Exportar juego de instrucciones
Tipo	Funcional
Origen	UR-C03
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta debe permitir exportar el juego de instrucciones en un fichero.

Table 3.39: *Requisito Funcional SR-F-F08.*

ID	SR-F-F09
Nombre	Generación firmware del simulador
Tipo	Funcional
Origen	UR-C03
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta debe de generar el firmware simulador a partir del juego de instrucciones definido y cargado por el usuario.

Table 3.40: *Requisito Funcional SR-F-F09.*

ID	SR-F-F10
Nombre	Definición del código ensamblador
Tipo	Funcional
Origen	UR-C04
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta debe permitir al usuario la definición del código ensamblador a utilizar en la simulación.

Table 3.41: *Requisito Funcional SR-F-F10.*

ID	SR-F-F11
Nombre	Formato del código ensamblador
Tipo	Funcional
Origen	UR-C04
Prioridad	Esencial
Estabilidad	Estable
Descripción	El código ensamblador a simular debe de seguir el formato utilizado en la asignatura Estructura de Computadores, utilizando el lenguaje definido en el juego de instrucciones.

Table 3.42: *Requisito Funcional SR-F-F11.*

ID	SR-F-F12
Nombre	Edición del código ensamblador
Tipo	Funcional
Origen	UR-C05
Prioridad	Esencial
Estabilidad	Estable
Descripción	La heramienta debe permitir modificar el código ensamblador.

Table 3.43: *Requisito Funcional SR-F-F12.*

ID	SR-F-F13
Nombre	Carga del código ensamblador
Tipo	Funcional
Origen	UR-C05
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta debe permitir la carga de un fichero que contenga el código ensamblador.

Table 3.44: *Requisito Funcional SR-F-F13.*

ID	SR-F-F14
Nombre	Generación ejecutable del codigo ensamblador
Tipo	Funcional
Origen	UR-C05
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta debe de generar el binario ejecutable asociado al código ensamblador definido para la simulación.

Table 3.45: *Requisito Funcional SR-F-F14.*

ID	SR-F-F15
Nombre	Simulación ciclo a ciclo de reloj
Tipo	Funcional
Origen	UR-C06
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta debe de permitir la simulación ciclo a ciclo de reloj del código ensamblador cargado por el usuario.

Table 3.46: *Requisito Funcional SR-F-F15.*

ID	SR-F-F16
Nombre	Simulación instrucción a instrucción
Tipo	Funcional
Origen	UR-C06
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta debe de permitir la simulación instrucción a instrucción del código ensamblador cargado por el usuario.

Table 3.47: *Requisito Funcional SR-F-F16.*

ID	SR-F-F17
Nombre	Simulación completa
Tipo	Funcional
Origen	UR-C06
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta debe de permitir la simulación completa del código ensamblador cargado por el usuario.

Table 3.48: *Requisito Funcional SR-F-F17.*

ID	SR-F-F18
Nombre	Velocidad de simulación
Tipo	Funcional
Origen	UR-C07
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta debe de permitir al usuario elegir la velocidad de simulación del código ensamblador cargado.

Table 3.49: *Requisito Funcional SR-F-F18.*

3.2.4 Requisitos No-Funcionales

Esta subsección especifica los requisitos no-funcionales.

ID	SR-F-F19
Nombre	Esquema CPU
Tipo	Funcional
Origen	UR-C08
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta debe de mostrar el esquema de la CPU del simulador, modificando el color de las señales activas y buses de datos actualizados en cada ciclo de reloj.

Table 3.50: *Requisito Funcional SR-F-F19.*

ID	SR-F-F20
Nombre	Esquema Unidad de Control
Tipo	Funcional
Origen	UR-C08
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta debe de mostrar el esquema de la Unidad de Control del simulador, modificando el color de las señales activas y buses de datos actualizados en cada ciclo de reloj.

Table 3.51: *Requisito Funcional SR-F-F20.*

ID	SR-F-F21
Nombre	Información de la simulación
Tipo	Funcional
Origen	UR-C09
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta debe de mostrar durante la ejecución el estado de los registros del simulador.

Table 3.52: *Requisito Funcional SR-F-F21.*

ID	SR-NF-PL01
Nombre	Herramienta web
Tipo	Interfaz
Origen	UR-R01
Prioridad	Esencial
Estabilidad	Estable
Descripción	El simulador debe de ser diseñado como una herramienta web.

Table 3.53: *Requisito Funcional SR-NF-PL01.*

ID	SR-NF-PL02
Nombre	Herramienta web
Tipo	Interfaz
Origen	UR-R02
Prioridad	Esencial
Estabilidad	Estable
Descripción	El simulador debe poder ser ejecutado en los navegadores: Microsoft Edge, Mozilla Firefox, Google Chrome y Safari.

Table 3.54: *Requisito Funcional SR-NF-PL02.*

ID	SR-NF-PL03
Nombre	Plataforma de ejecución
Tipo	Interfaz
Origen	UR-R05
Prioridad	Esencial
Estabilidad	Estable
Descripción	Las ejecuciones de la herramienta deben ser realizadas en el dispositivo del usuario.

Table 3.55: *Requisito Funcional SR-NF-PL03.*

ID	SR-NF-PL04
Nombre	Internet
Tipo	Interfaz
Origen	UR-R06
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta únicamente necesitará conexión a internet para el acceso a la herramienta.

Table 3.56: *Requisito Funcional SR-NF-PL04.*

ID	SR-NF-PL05
Nombre	Lenguaje de programación JavaScript
Tipo	Interfaz
Origen	UR-R06
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta debe ser desarrollada en el lenguaje de programación JavaScript.

Table 3.57: *Requisito Funcional SR-NF-PL05.*

ID	SR-NF-UI01
Nombre	Interfaz de usuario
Tipo	Interfaz
Origen	UR-R03
Prioridad	Esencial
Estabilidad	Estable
Descripción	La interfaz de usuario del simulador debe de ser compatible tanto con pc's y smartphones.

Table 3.58: *Requisito Funcional SR-NF-UI01.*

ID	SR-NF-P01
Nombre	Tiempo por ciclo de reloj
Tipo	Interfaz
Origen	UR-R03
Prioridad	Esencial
Estabilidad	Estable
Descripción	El tiempo de ejecución de un ciclo de reloj no debe exceder 0,1 segundos.

Table 3.59: *Requisito Funcional SR-NF-P01.*

3.3 Marco Regulador

Esta sección discute las restricciones necesarias teniendo en cuenta el marco regulador. En concreto, se especifican las restricciones legales aplicables al simulador.

3.3.1 Restricciones Legales

Para el uso de la mayoría de herramientas web, los usuarios deben de registrarse, y las bases de datos de las mismas manejan información confidencial de los usuarios, por lo que es necesario garantizar que terceros no puedan acceder a esa información. Una solución es cifrar la información transmitida mediante algún protocol criptografico. En España, este requisito es especificado en el artículo 104 del RD 1720/2007 [12], que se ocupa de la Ley Española de Protección de Datos.

En contraste, la aplicación desarrollada no utiliza datos privados de los usuarios y tampoco transmite información confidencial a terceros, ya que es un simulador que únicamente utiliza los códigos generados por el usuario para ejecutarlos de forma local en la máquina del usuario.

Por otro lado, es crucial que nuestro simulador esté disponible como un software de código abierto. Queremos que sea tal que cualquiera pueda redistribuir el código o modificarlo por los términos de la Licencia Pública General Menor de GNU (LGPL) [13]. Para ello, nuestro simulador está disponible en el siguiente sitio web:

`https://www.arcos.inf.uc3m.es/~wepsim/`.

Chapter 4

Diseño

En este capítulo se realiza una descripción completa del simulador desarrollado, incluyendo la arquitectura interna y los diferentes componentes software, que componen la herramienta descritos con anterioridad en [14].

La sección 4.1 discute la solución elegida y la compara con las alternativas consideradas. La sección 4.2 describe cada uno de los componentes que componen el simulador.

4.1 Solución elegida

Para que los profesores de la asignatura Estructura de Computadores puedan hacer uso de una herramienta que sirva de ayuda para la explicación de los conceptos teóricos de la asignatura, y los alumnos puedan utilizarla para comprender estos conceptos y realizar posteriormente las prácticas de la asignatura, se propone el diseño e implementación de una herramienta web que simule con realismo en funcionamiento de un procesador elemental con unidad de control microprogramable.

Este simulador, será desarrollado como una herramienta web debido a la portabilidad que proporciona, ya que podrá ser ejecutado sobre un gran número de diferentes dispositivos independientemente del sistema operativo que utilice, puesto que únicamente necesita un navegador web para su correcto funcionamiento. De esta forma, los profesores y alumnos podrán hacer uso de la herramienta sin depender de su instalación en el dispositivo a utilizar, incluso pudiendo los alumnos realizar las prácticas sobre dispositivos móviles.

Para lograr dicha portabilidad, el simulador ha sido desarrollado en HTML5 (HTML + JavaScript + CSS) haciendo posible su ejecución en cualquier plataforma (smartphones, tablet,

PC, etc.) que pueden ejecutar Microsoft Edge, Mozilla Firefox, Google Chrome o Safari. Además, la herramienta depende de los siguientes frameworks/bibliotecas: JQuery, JQueryUI, JQuery Mobile, Knockout y BootStrap.

Por tanto, la solución elegida es capaz de unificar en una misma herramienta todas las funcionalidades requeridas para la enseñanza de Estructura de computadores con un alto nivel de detalle, con alta disponibilidad al facilitarse su como una herramienta web, y con una gran portabilidad puesto que podrá ser ejecutada sobre un gran número de diversos dispositivos.

4.2 Arquitectura de WepSIM

La arquitectura de la solución presentada en este trabajo consta de tres elementos principales:

- Modelo hardware: permite definir el hardware a usar.
- Modelo software: permite definir el juego de instrucciones a utilizar.
- Motor de simulación: simula el funcionamiento del hardware ejecutando el microcódigo/lenguaje máquina definido con anterioridad.

El modelo hardware permite definir los distintos elementos típicos de un computador (memoria principal, procesador, etc.) de una forma modular. La forma de definir estos elementos equilibra dos objetivos contrapuestos: es suficientemente completa como para imitar los principales aspectos de la realidad, pero es lo suficientemente mínima para facilitar su uso. Ante todo se persigue que sea una herramienta didáctica.

El modelo software permite definir el microcódigo y el ensamblador basado en este microcódigo de la forma tan intuitiva posible. El ensamblador a usar viene dado por un conjunto de instrucciones que puede ser definido por el usuario e intenta ser lo suficientemente flexible como para poder definir diferentes tipos y juegos de instrucciones, como por ejemplo MIPS o ARM.

El tercer elemento de la arquitectura propuesta es un motor que toma como entrada el modelo hardware descrito y el modelo software de trabajo, y se encarga de mostrar el funcionamiento del hardware con el software dado.

La figura 4-1 resume la arquitectura de WepSIM. El punto de inicio es el modelo hardware que describe el procesador a ser simulado. Ello incluye el procesador, la memoria y algunos

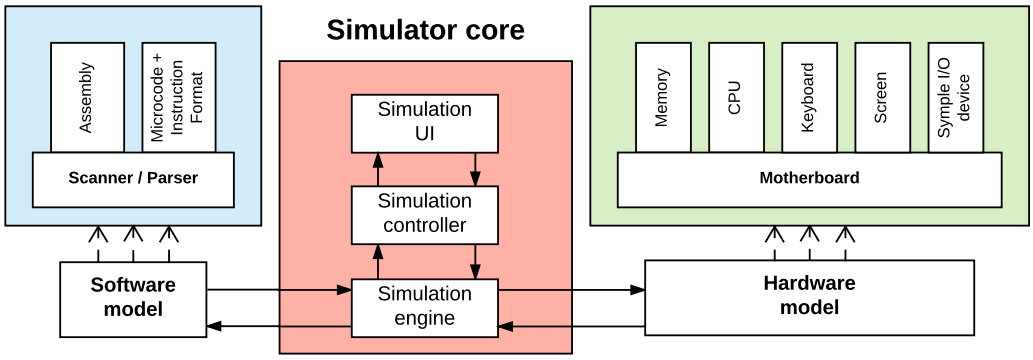


Figure 4-1: *Arquitectura de WepSIM.*

dispositivos de E/S: teclado, pantalla y un dispositivo de E/S simple que genera interrupciones. El modelo hardware describe el estado global del procesador. A partir del estado global del procesador, el motor de simulación actualiza el estado en cada ciclo de reloj.

La unidad de control simulada almacena las señales de control de cada ciclo en una memoria de control. La memoria de control tiene todos los microprogramas para las instrucciones con las que trabaja el procesador, y el fetch para leer la instrucción de memoria y decodificarla.

El microcódigo (el contenido de la memoria de control) junto con el formato de cada instrucción (campos de la instrucción y su longitud) se describe en un fichero de texto. El modelo software lee este fichero, lo traduce a binario y lo carga en el procesador. La definición del lenguaje ensamblador a utilizar se describe junto con el microcódigo, y el modelo software permite traducir a binario programas escritos en dicho ensamblador.

El motor de simulación pregunta al subsistema del modelo software por el microcódigo definido, la descripción del formato de instrucción y el contenido de la memoria principal. Los binarios se cargan en los elementos del modelo hardware, y a continuación el motor de simulación actualiza el estado global en cada ciclo de reloj.

WepSIM dispone de un controlador de simulación que se encarga de actualizar el ciclo de reloj y mostrar el estado global. El subsistema de interfaz de simulación actualiza la interfaz de usuario. Cuando el usuario usa la interfaz de usuario para solicitar una operación, el subsistema de interfaz de simulación traslada la petición al controlador de simulación. Como se puede ver, se usa un Modelo- Vista-Controlador (MVC) básico para la arquitectura de WepSIM.

4.2.1 Modelo hardware

El modelo hardware que usa WepSIM permite definir los distintos elementos típicos de un computador (memoria principal, procesador, etc.) de una forma modular y de manera que sea posible añadir, quitar o modificar estos elementos.

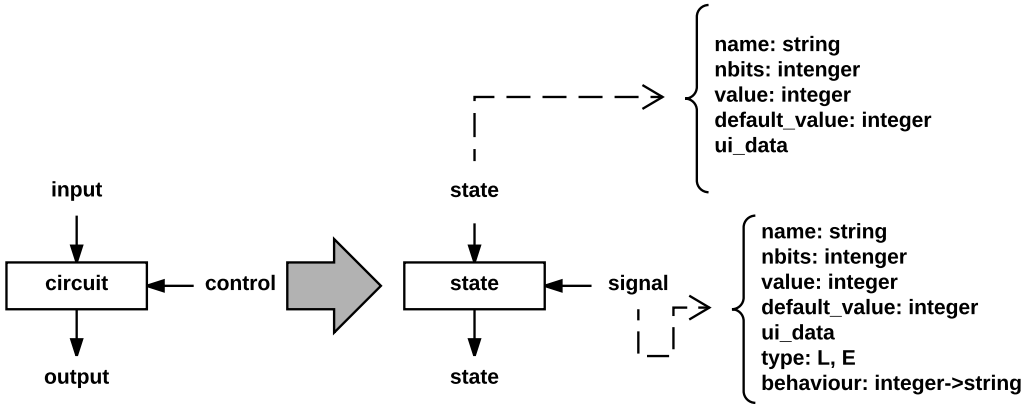


Figure 4-2: *Modelado del hardware.*

La figura 4-2 introduce el modelo propuesto. Cada elemento del circuito se describe como una caja negra con posibles entradas, posibles salidas y señales de control (que controlan las posibles transformaciones de las entradas a las salidas). El subsistema del modelo hardware transforma esta caja negra en dos conjuntos de objetos: estados y señales. Un estado tiene un identificador (el nombre), el valor (un valor entero) y un valor inicial (el valor por defecto). Los valores que puede tomar son valores naturales dentro de un rango, dado por el número de bits con los que se representa el estado. Una señal es un estado especial que controla el valor de otros estados o señales. Hay dos atributos asociados a las señales (y no a los estados): el tipo de señal (por nivel o por flanco) y su comportamiento. Para cada valor de señal una cadena de caracteres describe en un Lenguaje Simple lo que la señal mueve o transforma. Este Lenguaje Simple se compone principalmente de instrucciones que representan las operaciones elementales.

Las figuras 4-3 y 4-4 muestran dos ejemplos: un triestado y un registro. El triestado controla dos estados: el estado del bus al que se conecta BUS IB y el estado del registro de entrada, REG RT1 en este caso. Ambos representan el valor a la salida de la puerta (BUS IB) y el valor del registro RT1 (REG RT1). La señal T4 se encarga de indicar cuándo el valor del registro RT1 se envía a la salida. Esta señal T4 es una señal por nivel (tipo: L), con valor cero no tiene efecto

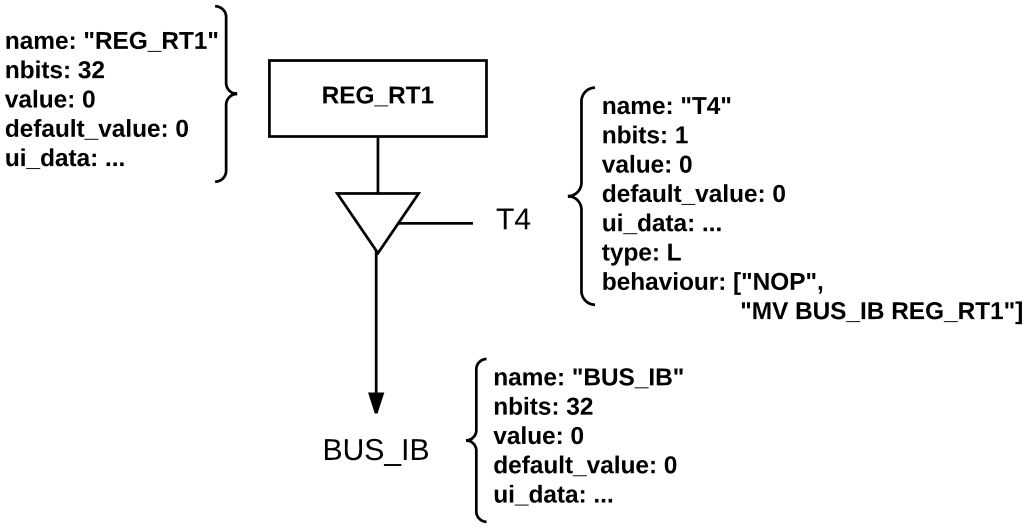


Figure 4-3: *Ejemplo de modelado de una puerta triestado.*

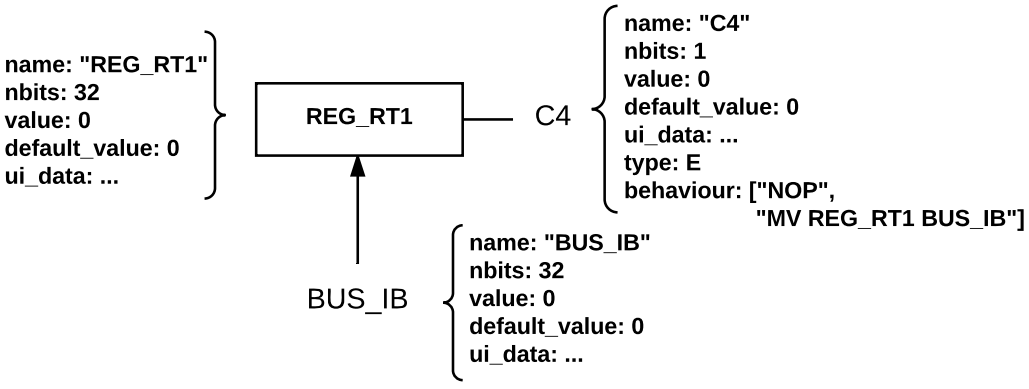


Figure 4-4: *Ejemplo de modelado de un registro.*

(comportamiento "NOP"). Cuando el valor de la señal es uno entonces el comportamiento es el de copiar el valor del registro RT1 a la salida (comportamiento "MV BUS IB REG RT1").

El ejemplo con el registro (figura 4-4) es similar. En este caso trabaja con dos estados: el contenido del registro RT1 y el contenido situado a la entrada (BUS IB). La señal C4 controla cuándo se almacena en el registro RT1 el valor que hay en la entrada. La diferencia está en el tipo de señal: C4 es una señal por flanco de bajada (tipo: E), por lo que al final del ciclo de reloj (pasa de uno a cero) si la señal vale uno entonces el comportamiento es el de copiar el valor situado a la entrada al registro (comportamiento "MV REG RT1 BUS IB").

El Lenguaje Simple usado para definir los comportamientos añade a las operaciones elementales otras operaciones necesarias. Por ejemplo disparar una señal ("FIRE C4") que ayuda a propagar el efecto de una señal al revaluar la señal inmediata que podría verse afectada. Otro ejemplo lo encontramos en dos operaciones que pueden ser muy útiles a la hora de depurar: imprimir el valor de un estado ("PRINT E BUS IB") e imprimir el valor de una señal ("PRINT S C4").

4.2.2 Modelo software

Una vez definido el procesador elemental usando el modelo hardware propuesto, toca describir el conjunto de instrucciones que es capaz de ejecutar así como el microcódigo que lo orquesta. En un fichero de texto se define el formato de las instrucciones máquina junto con el cronograma asociado a la ejecución de cada una de las instrucciones máquina. La figura 4-5 muestra un ejemplo de definición para la instrucción li (load inmmediate), que almacena un valor inmediato en un registro.

```
li reg val
{
    co=000010,
    nwords=1,
    reg=reg(25,21),
    val=inm(15,0),
    {
        (SE=0, OFFSET=0, SIZE=10000, SE=1, T3=1,
         LE=1, MR=0, SELE=10101, A0=1, B=1, C=0)
    }
}
```

Figure 4-5: *Ejemplo de formato de instrucción.*

El fichero con el cronograma de fetch y todos los cronogramas de las instrucciones define el microcódigo para la plataforma WepSIM. El simulador permite la definición de diferentes juegos y formatos de instrucciones. Inicialmente se ha implementado un subconjunto de las instrucciones del MIPS, pero es posible definir instrucciones de otros conjuntos de forma similar. En este fichero se pueden asignar códigos simbólicos a los registros del banco de registros, lo que permite que en los programas escritos en ensamblador se puedan usar dichos símbolos (por ejemplo, registro $t3 en la figura 4-6).

```
.text
main:   li    $t3 8
        li    $t5 10
        add $t6  $t3 $t5
```

Figure 4-6: *Ejemplo de código fuente en ensamblador.*

El campo “co” identifica el código de instrucción máquina, que es un número binario de 6 bits. Esto permite definir hasta 64 instrucciones distintas. Dado que los últimos 4 bits de la instrucción pueden usarse para seleccionar la operación en la ALU, es posible seleccionar hasta 16 operaciones aritmético-lógicas con un mismo código de instrucción, por lo que se podrían tener 79 (63+16) instrucciones en total.

Cuando WepSIM carga el microcódigo, cada código de instrucción tiene asociado una dirección de comienzo en la memoria de control donde se almacena el cronograma asociado. Esta tabla con dos columnas (el código de instrucción y su dirección de comienzo asociada en la memoria de control) se carga en la ROM co2microAddr mostrada en la figura 3-2.

El campo "nwords" define cuantas palabras precisa la instrucción para su definición y carga en memoria. Una palabra en WepSIM son 4 bytes.

Para cada campo de la instrucción se define el bit inicial, el bit final (ambos incluidos) y el tipo de campo (registro, valor inmediato, dirección absoluta y dirección relativa a PC). Una vez definido el formato, se definen todas las microinstrucciones que necesita la instrucción máquina definida para su ejecución. Todas las microinstrucciones se encuentran encerradas entre llaves y cada microinstrucción está formada por una lista de tuplas (señal, valor) encerradas entre paréntesis. Para la instrucción definida en la figura 4-5 se precisa de una sola microinstrucción, en la que se indican qué señales se activan durante un ciclo de reloj. Para las señales no indicadas se asume que su valor es 0 durante el ciclo de reloj correspondiente.

Una vez cargado el microcódigo en WepSIM, es posible cargar cualquier fichero ensamblador que haya sido codificado usando las instrucciones máquina definidas anteriormente en el microcódigo.

En la figura 4-6 se muestra un ejemplo de código fuente en ensamblador que se puede usar en WepSIM. Este ejemplo en particular muestra un código estilo MIPS. Para que un programa en ensamblador pueda utilizar la instrucción de carga inmediata li (load inmmediate) y de suma add (addition), deben haber sido definidas previamente en el microcódigo. WepSIM puede comprobar los errores de sintaxis y construir el binario mediante el rellenado de los campos descritos en la definición del microcódigo correspondiente a la instrucción. La figura 4-7 muestra un ejemplo de traducción a binario para la instrucción li $2 5 en función del formato definido en la figura 4-5. También se debe haber definido en el fichero de microcódigo el valor del registro asociado a la etiqueta $2 (00100 en este caso).

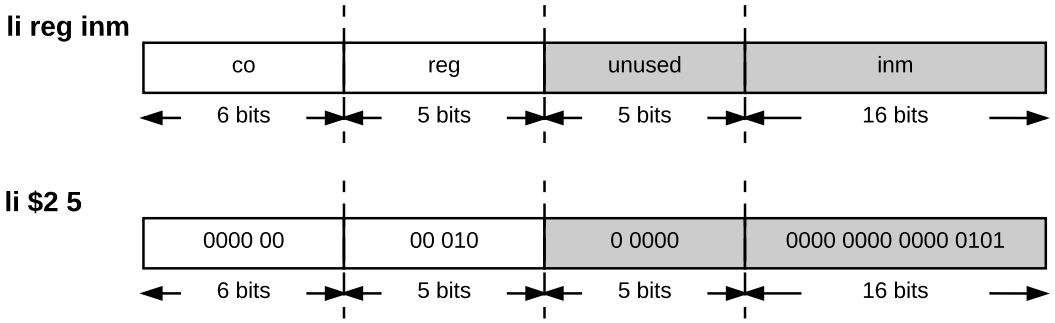


Figure 4-7: *Formato de instrucción descrito en el microcódigo y ejemplo de su traducción en binario.*

Una de las grandes ventajas del simulador WepSIM es que no está limitado a un conjunto de instrucciones concreto. Se puede definir un amplio conjunto de instrucciones de procesadores reales o inventados. Se puede usar para añadir por ejemplo, a un conjunto de instrucciones MIPS, otras instrucciones diferentes no incluidas en dicho conjunto de instrucciones.

4.2.3 Motor del simulador

El motor de simulación diseñado en WepSIM, es el componente que sirve de unión entre el modelo hardware y software. Su función principal reside en la simulación del computador utilizando la definición del modelo hardware y el juego de instrucciones definido, realizando la ejecución de los códigos ensamblador, siendo el controlador de la herramienta.

Además de realizar el proceso de simulación, este componente a su vez realiza la actual-

ización de la interfaz de usuario del simulador, en donde se muestra el valor de cada uno de los registros de la máquina y el estado de los buses y señales que conforman el modelo hardware.

Chapter 5

Implementación y despliegue

This chapter deals with the implementation and deployment of the software. Regarding the implementation of the system, the more complicated parts of the code are explained (Section 5.1, *Implementación*). On the other hand, we explain the steps required to deploy the final system (Section 5.2, *Deployment*).

5.1 Implementación

As we explained in Chapter 3, *Análisis*, we have implemented the simulator using the C programming language and the tools provided by SimGrid. The SimGrid core is responsible for planning the different processes, but the developer is the one that must use synchronization mechanisms to avoid race conditions. He have used several mutexes and condition variables in order to solve the problem of reading-writing in shared structures.

Furthermore, we have worked so that the project servers simulated in ComBoS have a realistic behavior. To do this, we have divided the functioning of each server (both scheduling and data servers) into two main processes: requests and dispatcher (see Figure 5-1).

The *requests* process is in charge of receiving all client requests through the corresponding mailbox. It receives messages asynchronously; that is, the process does not wait to finish receiving a message from a client in order to start receiving the next one. Each time a request is received, it is inserted into a queue that is shared with the *dispatcher* process. The dispatcher process is responsible for dealing with requests and answering to the volunteer clients if necessary. Response messages are sent asynchronously as well.

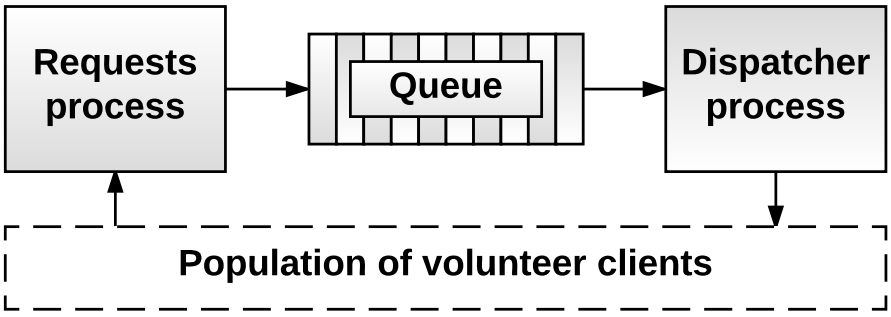


Figure 5-1: *Server main processes.*

On the other hand, each client is implemented with at least three different processes: the client main process (Pseudocode 5.1), which updates the client parameters every scheduling interval; the work fetch process (Pseudocode 5.2), which selects the project to ask for work; and the execution processes, one per attached project, that execute the tasks. We have not included the pseudocode of the execution process because it only dequeues tasks and executes them. Our simulator is complemented with the most important features of the real scheduler (deadline scheduling, long term debt, fair sharing and exponential back-off) as explained in Section **??**, **??** (Chapter 4, *Diseño*).

Pseudocode 5.1 Client main process

```
1: function CLIENT_MAIN( )
2:     while time < max_time do
3:         increase wall_cpu_time to the running project
4:         UPDATE_DEBT
5:         UPDATE_DEADLINE_MISSED
6:         CPU_SCHEDULING
7:         SIGNAL Work fetch process
8:         WAIT scheduling_interval
9:     end while
10:    Return
11: end function
```

Pseudocode 5.2 Work fetch process

```
1: function WORK_FETCH( )
2:     project = null
3:     while time < max_time do
4:         for each project p in projects do
5:             if p meets the requirements then
6:                 project = p
7:             end if
8:         end for
9:         if project and not deadlines_missed then
10:            ASK_FOR_WORK(project)
11:        end if
12:        WAIT work_fetch_period
13:    end while
14:    SIGNAL Client main process
15:    Return
16: end function
```

5.2 Deployment

This section presents the deployment of the system. The technical specifications recommended for the final user to obtain the best experience from the application are:

- **Operating System**: Ubuntu 14.04.4 LTS (Linux distribution) or higher.
- **Processor**: Intel(R) Core(TM) i7 CPU 920 @2.67GHz or higher.
- **Random-Access Memory (RAM)**: 8 GB or higher.
- **Storage**: 1 GB of free space in the Hard Disk Drive.
- **Network**: Internet connection is not required.
- **Software**: The following software must be installed in order to run the application:
    1. GCC (GNU Compiler) 5.1 or higher.
    2. SimGrid toolkit 3.10 or higher.

To make life easier for the end user, we have organized the application files in the simplest way possible. Figure 5-2 shows the initial structure of the application files.

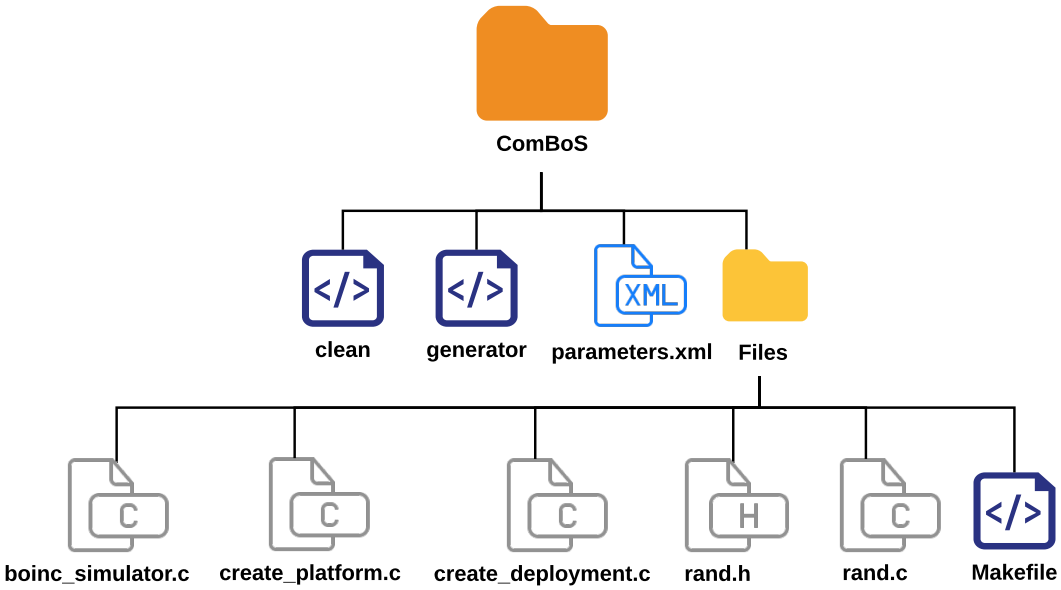


Figure 5-2: *Initial folder structure.*

The files located inside the Complete BOINC Simulator (ComBoS) main folder are detailed below:

- **clean:** it is a script that removes the files created by the generator script.
- **generator:** it is a script that generates all the files needed for the simulation based on the parameters specified in the parameters.xml file. The files created by this script are:
  - **execute:** it is a script that executes the simulation (runs the file boinc_simulator).
  - **Files/create_platform:** it is the executable file that results from compiling the create_platform.c file.
  - **Files/create_deployment:** it is the executable file that results from compiling the create_deployment.c file.
  - **Files/platform.xml:** platform file created by create_platform.
  - **Files/deployment.xml:** deployment file created by create_deployment.
  - **Files/boinc_simulator:** it is the executable file that results from compiling the boinc_simulator.c and rand.c files.

- **parameters.xml:** to create a simulation, ComBoS requires the specification of all the parameters needed in an Extensible Markup Language (XML) file, such as the simulation time in hours. All other parameters required by the XML file are detailed in Tables **??** and **??** (presented in Chapter 4, *Diseño*)
- **Files:** folder that includes:
  - **boinc_simulator.c:** simulator main source code.
  - **create_platform.c:** source code that contains the generation of the platform file.
  - **create_deployment.c:** source code that contains the generation of the deployment file.
  - **rand.h:** header file for random functions.
  - **rand.c:** source code that contains the random functions.
  - **Makefile:** script that compiles and links the code files.

The Appendix A presents a complete user manual of the simulator. It includes a tutorial for the installation of the SimGrid toolkit, and a number of practical and educational examples to learn how to perform simulations. In a basic way, in order to deploy the application, the user only needs to follow the following steps:

- Download the main folder of the ComBoS application (of which the structure is presented in Figure 5-2).
- Indicate the simulation parameters in the parameters XML file. This file must include all the parameters specified in Tables **??** and **??** (presented in Chapter 4, *Diseño*).
- Generate all the simulation files required, by just running the generator script.
- Run the execution script.

Chapter 6

Verificación, validación y evaluación

This chapter details the verification, validation and evaluation of the project. First, we present the verification and validation of the simulator (Section 6.1, *Verificación y validación*), and we detail a series of tests that allowed us to verify that we had met all the requirements set in Chapter 3 (*Análisis*). After this, we show the validation of the outputs of the simulations, demonstrating that the simulator performs accurate and realistic simulations. We also display a study of the performance of the simulator (Section 6.2, *Performance Study*), in which we show that ComBoS is efficient and scalable. Finally, we present several case studies of the simulator usage (Section 6.3, *Case Studies*), with the corresponding analysis and evaluation of the results.

We have used the drand48 Linux functions [15] as random number generator in our simulations. These functions generate pseudo-random numbers using the linear congruential algorithm and 48-bit integer arithmetic. Each simulation result presented in this chapter is based on the average of 20 runs. For a 95% interval, the error is less than $\pm$ 3% for all values.

6.1 Verificación y validación

The main objective of this section is to verify that all the requirements set out in Chapter 3 (*Análisis*) have been fulfilled. In addition, we validate the results provided by ComBoS, comparing them to the results of SimBOINC and to the statistical results of the official Berkeley Open Infrastructure for Network Computing (BOINC) webpage.

In software engineering, verification and validation are the processes of checking that a software system meets specifications and that it fulfills its intended purpose. As explained in Chapter 3 (*Análisis*), the customer initially sets the requirements desired for the final product

(user requirements). From there, analysts specify software requirements (functional and non-functional requirements). In order to verify that the project requirements are met, verification and validation processes are needed (see Figure 6-1).

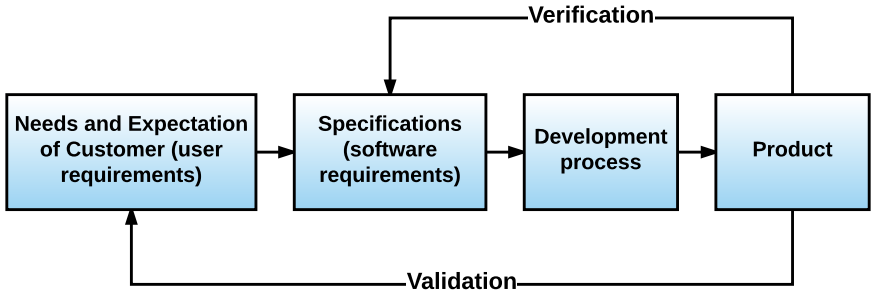


Figure 6-1: *Software verification and validation.*

Software verification is the process of evaluating work-products (not the actual final product) of a development phase to determine whether they meet the specified requirements for that phase (the software requirements). *Software validation* is the process of evaluating the final product at the end of the development process to determine whether it satisfies the requirements specified by the user at the beginning of the project [16].

6.1.1 Pruebas de verificación

In order to perform the verification tests, we have followed a dynamic process during the development phase of the software. With these tests we wanted to answer the question: "Are we building the product right?". Table 6.1 provides the template used for the verification tests. Note that the ID format is VET-XX, where XX indicates the verification test number.

ID	Test ID.
Name	Test name.
Requirements	Software requirements fulfilled with this test.
Description	Test description.
Preconditions	Predicates that must always be true before performing the test.
Procedure	A fixed, step-by-step sequence of activities performed by the test.
Postconditions	Predicates that must always be true just after performing the test.
Evaluation	*Passed* or *Failed*.

Table 6.1: *Template for verification tests.*

Then, we specify the verification tests.

ID	VET-01
Name	Platform.
Requirements	SR-NF-PL01, SR-NF-PL02, SR-NF-PL03, SR-NF-UI02.
Description	Verify that the software can be used on the platform and is developed with the tools specified in the requirements.
Preconditions	1. Use a machine with Ubuntu 14.04 operating system. 2. GCC (GNU Compiler) 5.1 or higher must be installed on the machine. 3. The user must be located in the main directory of the ComBoS application.
Procedure	1. Check the code of the simulator source files (all these files are inside the /Files folder). 2. Run the generator script with the default parameters to create the simulation files. 3. Run the simulation.
Postconditions	1. All source files must be written in C programming language. 2. The implementation, setup and control of the simulations must be carried out using the MSG API of the SimGrid toolkit. 3. Simulations run while a progress bar indicates the percentage of execution. 4. The simulator must successfully finish its execution in the specified operating system.
Evaluation	Passed

Table 6.2: *Verification test VET-01.*

ID	VET-02
Name	Realistic BOINC elements in simulations.
Requirements	SR-F-F06, SR-NF-UI01, SR-NF-UI02.
Description	Verify that the simulator allows to simulate all the BOINC actual elements.
Preconditions	1. The user must be located in the main directory of the ComBoS application.
Procedure	1. Specify the following elements in the simulation parameters: tasks, volunteer hosts, servers, data servers, networks, and hosts availability. 2. Run the generator script to create the simulation files. 3. Run the simulation.
Postconditions	1. Simulations run while a progress bar indicates the percentage of execution. 2. The simulator must successfully finish its execution.
Evaluation	Passed

Table 6.3: *Verification test VET-02.*

ID	VET-03
Name	Statistics of BOINC projects.
Requirements	SR-F-F02, SR-NF-UI01, SR-NF-UI02.
Description	Verify that the outputs of the simulator are the same as those published by BOINCstats [17]. The outputs are: credits, hosts, active hosts, and FLOPS.
Preconditions	1. The user must be located in the main directory of the ComBoS application.
Procedure	1. Specify the simulation parameters. 2. Run the generator script to create the simulation files. 3. Run the simulation.
Postconditions	1. Simulations run while a progress bar indicates the percentage of execution. 2. The simulator must successfully finish its execution. 3. The outputs of the simulator contain at least: credits, hosts, active hosts, and FLOPS (The same as those published by BOINCstats [17]).
Evaluation	Passed

Table 6.4: *Verification test VET-03.*

ID	VET-04
Name	Multiple BOINC proyects simultaneously.
Requirements	SR-F-F04, SR-NF-UI01, SR-NF-UI02.
Description	Verify that the simulator allows multiple project simulations simultaneously.
Preconditions	1. The user must be located in the main directory of the ComBoS application.
Procedure	1. Specify the simulation parameters of three different projects (for example, the SETI@home, Einstein@home, and LHC@home projects). 2. Run the generator script to create the simulation files. 3. Run the simulation.
Postconditions	1. Simulations run while a progress bar indicates the percentage of execution. 2. The simulator must successfully finish its execution.
Evaluation	Passed

Table 6.5: *Verification test VET-04.*

ID	VET-05
Name	BOINC client scheduler.
Requirements	SR-F-F05, SR-NF-UI01, SR-NF-UI02.
Description	Verify that the client scheduler implemented produces the same results as the actual BOINC scheduler.
Preconditions	1. The user must be located in the main directory of the ComBoS application.
Procedure	1. Specify the client side simulation parameters of three different projects: SETI@home, Einstein@home, and LHC@home. 2. Run the generator script to create the simulation files. 3. Run the simulation.
Postconditions	1. Simulations run while a progress bar indicates the percentage of execution. 2. The simulator must successfully finish its execution. 3. The outputs of the simulation are the same as the real BOINC client scheduler (it is detailed in 6.1.3, *Validation of the Client Scheduler*).
Evaluation	Passed

Table 6.6: *Verification test VET-05.*

ID	VET-06
Name	Accurate simulations of BOINC projects.
Requirements	SR-F-F01, SR-F-F03, SR-NF-UI01, SR-NF-UI02.
Description	Verify that the outputs of the simulator for existing projects (SETI@home, Einstein@home, and LHC@home) should be almost identical to those published in BOINCstats [17].
Preconditions	1. The user must be located in the main directory of the ComBoS application.
Procedure	1. Specify the simulation parameters of three different projects: SETI@home, Einstein@home, and LHC@home. 2. Run the generator script to create the simulation files. 3. Run the simulation.
Postconditions	1. Simulations run while a progress bar indicates the percentage of execution. 2. The simulator must successfully finish its execution. 3. The outputs of the simulation are the same as the actual BOINC projects, in terms of FLOPS and credit (it is detailed in 6.1.4, *Validation of Whole Simulator*).
Evaluation	Passed

Table 6.7: *Verification test VET-06.*

ID	VET-07
Name	Large simulations.
Requirements	SR-NF-S01, SR-NF-UI01, SR-NF-UI02.
Description	Verify the application is able to perform simulations with more than 100,000 hosts in a machine with at least 8 GB of RAM.
Preconditions	1. Use a machine with at least 8GB or RAM. 2. The user must be located in the main directory of the ComBoS application.
Procedure	1. Specify the simulation parameters with more than 100,000 hosts. 2. Run the generator script to create the simulation files. 3. Run the simulation.
Postconditions	1. Simulations run while a progress bar indicates the percentage of execution. 2. The simulator must successfully finish its execution.
Evaluation	Passed

Table 6.8: *Verification test VET-07.*

ID	VET-08
Name	Execution time.
Requirements	SR-NF-P01, SR-NF-UI01, SR-NF-UI02.
Description	Check that simulations follow a linear execution time.
Preconditions	1. The user must be located in the main directory of the ComBoS application.
Procedure	1. Specify the simulation parameters with different workloads. 2. Run the generator script to create the simulation files. 3. Run the simulation. 4. Go to 2 specifying different simulation parameters.
Postconditions	1. Simulations run while a progress bar indicates the percentage of execution. 2. The simulator must successfully finish its execution. 3. Check that executions follow a linear execution time when increasing the workload (it is detailed in 6.2, *Performance Study*).
Evaluation	Passed

Table 6.9: *Verification test VET-08.*

The verification test traceability matrix (Table 6.10) determines that all the software requirements have been verified during the development phase of the project.

Requirements	**VET-01**	**VET-02**	**VET-03**	**VET-04**	**VET-05**	**VET-06**	**VET-07**	**VET-08**
SR-F-F01						✓		
SR-F-F02			✓					
SR-F-F03						✓		
SR-F-F04				✓				
SR-F-F05					✓			
SR-F-F06		✓						
SR-NF-PL01	✓							
SR-NF-PL02	✓							
SR-NF-PL03	✓							
SR-NF-S01							✓	
SR-NF-P01								✓
SR-NF-UI01		✓	✓	✓	✓	✓	✓	✓
SR-NF-UI02	✓	✓	✓	✓	✓	✓	✓	✓

Table 6.10: *Verification test traceability matrix.*

6.1.2 Validation Tests

To perform the validation tests, we have checked the final software, comparing it with the user needs specified in Chapter 3 (*Análisis*). With these tests we want to answer the question: "Have we built the right product?". Table 6.11 provides the template used for the validation tests. Note that the ID format is VAT-XX, where XX indicates the validation test number.

ID	Test ID.
Name	Test name.
Requirements	User requirements fulfilled with this test.
Verification tests	Verification tests that help us to validate this test.
Description	Test description.
Preconditions	Predicates that must always be true before performing the test.
Procedure	A fixed, step-by-step sequence of activities performed by the test.
Postconditions	Predicates that must always be true just after performing the test.
Evaluation	*Passed* or *Failed*.

Table 6.11: *Template for validation tests.*

Then, we specify the validation tests.

ID	VAT-01
Name	BOINC projects simulation.
Requirements	UR-C01.
Verification tests	VET-03, VET-06.
Description	Validate that the simulator is able to simulate the behavior of BOINC projects.
Preconditions	1. The user must be located in the main directory of the ComBoS application.
Procedure	1. Specify the simulation parameters of three different BOINC projects: SETI@home, Einstein@home, and LHC@home. 2. Run the generator script to create the simulation files. 3. Run the simulation.
Postconditions	1. The simulator must successfully finish its execution. 2. The outputs of the simulation are the same as the actual BOINC projects, in terms of FLOPS and credit (it is detailed in 6.1.4, *Validation of Whole Simulator*).
Evaluation	Passed.

Table 6.12: *Validation test VAT-01.*

ID	VAT-02
Name	Client scheduling.
Requirements	UR-C02.
Verification tests	VET-05.
Description	Validate that the client scheduler implemented produces the same results as the actual BOINC scheduler.
Preconditions	1. The user must be located in the main directory of the ComBoS application.
Procedure	1. Specify the client side simulation parameters of three different projects: SETI@home, Einstein@home, and LHC@home. 2. Run the generator script to create the simulation files. 3. Run the simulation.
Postconditions	1. The simulator must successfully finish its execution. 2. The outputs of the simulation are the same as the real BOINC client scheduler (it is detailed in 6.1.3, *Validation of the Client Scheduler*).
Evaluation	Passed.

Table 6.13: *Validation test VAT-02.*

ID	VAT-03
Name	Simulation components.
Requirements	UR-C03.
Verification tests	VET-02.
Description	Validate that the simulations cover all the elements of the BOINC infrastructure.
Preconditions	1. The user must be located in the main directory of the ComBoS application.
Procedure	1. Specify the following elements in the simulation parameters: tasks, volunteer hosts, servers, data servers, networks, and hosts availability. 2. Run the generator script to create the simulation files. 3. Run the simulation.
Postconditions	1. The simulator must successfully finish its execution.
Evaluation	Passed.

Table 6.14: *Validation test VAT-03.*

ID	VAT-04
Name	Platform.
Requirements	UR-R01, UR-R02.
Verification tests	VET-01.
Description	Validate that the software can be used on the platform and is developed with the tools specified in the requirements.
Preconditions	1. Use a machine with a Linux operating system. 2. The user must be located in the main directory of the ComBoS application.
Procedure	1. Check the code of the simulator source files (all these files are inside the /Files folder). 2. Run the generator script with the default parameters to create the simulation files. 3. Run the simulation.
Postconditions	1. The simulator must successfully finish its execution. 2. The implementation, setup and control of the simulations must be carried out using the MSG API of the SimGrid toolkit.
Evaluation	Passed.

Table 6.15: *Validation test VAT-04.*

ID	VAT-05
Name	Scalability.
Requirements	UR-R03.
Verification tests	VET-07.
Description	Validate that the application is able to perform large simulations.
Preconditions	1. The user must be located in the main directory of the ComBoS application.
Procedure	1. Specify the simulation parameters with more than 100,000 hosts. 2. Run the generator script to create the simulation files. 3. Run the simulation.
Postconditions	1. The simulator must successfully finish its execution.
Evaluation	Passed.

Table 6.16: *Validation test VAT-05.*

The validation test traceability matrix (Table 6.17) determines that all the user needs have been validated in the final product.

Requirements	**VAT-01**	**VAT-02**	**VAT-03**	**VAT-04**	**VAT-05**
UR-C01	✓				
UR-C02		✓			
UR-C03			✓		
UR-R01				✓	
UR-R02				✓	
UR-R03					✓

Table 6.17: *Validation test traceability matrix.*

6.1.3 Validation of the Client Scheduler

To validate the client scheduler of ComBoS, we have compared the results of different executions of the simulator with the equivalent SimBOINC simulations. Of course, as we only want to validate the client scheduler (individually), we have simulated scenarios with no delay caused by network or servers.

All scenarios considered are based on a single client host with three associated projects (Einstein@home, SETI@home and LHC@home). Through the different tests we have varied the priorities of the projects and the time of each simulation. When using hosts with the same power, our goal is to compare the number of tasks executed in each simulator.

As explained in Section **??**, **??** (Chapter 2, *Estado del arte*), SimBOINC simulates the BOINC client scheduler and its simulations are highly accurate, because it uses almost exactly the BOINC client's CPU scheduler source code. Tables 6.18, 6.19 and 6.20 show different test cases:

- Table 6.18 presents the number of tasks executed by a client host of $1.4 \cdot 10^9$ FLOPS on simulations of 100, 500, 1,000, 5,000, and 10,000 hours. The priorities of the three projects are the same, so that each project uses the same runtime (33% CPU). The results of ComBoS and SimBOINC are almost identical.

	SimBOINC			*ComBoS*		
Time in hours	Einstein@home (33%)	SETI@home (33%)	LHC@home (33%)	Einstein@home (33%)	SETI@home (33%)	LHC@home (33%)
100	1	21	33	1	22	28
500	7	108	166	7	112	163
1,000	14	220	331	13	223	333
5,000	70	1,103	1,652	70	1,106	1,659
10,000	139	2,214	3,319	139	2,221	3,331

Table 6.18: *Executed tasks (three projects running on a single host of 1.4 GigaFLOPS).*

- Table 6.19 proposes a case similar to the previous test. In this case, the host has a power of $5.5 \cdot 10^9$ FLOPS and the priorities of the projects differ. The tasks of the LHC@home project consume 50% of CPU usage, while the tasks of the Einstein@home and SETI@home projects consume 25% of the CPU usage each. As in the previous case, the number of tasks executed in ComBoS is practically the same as in the case of SimBOINC.

	SimBOINC			*ComBoS*		
Time in hours	Einstein@home (25%)	SETI@home (25%)	LHC@home (50%)	Einstein@home (25%)	SETI@home (25%)	LHC@home (50%)
100	4	67	181	4	64	182
500	21	332	975	21	333	975
1,000	42	662	1,955	40	662	1,981
5,000	208	3,297	9,831	206	3,297	9,889
10,000	416	6,581	19,637	413	6,593	19,784

Table 6.19: *Executed tasks (three projects running on a single host of 5.5 GigaFLOPS).*

- Table 6.20 includes three different test cases. In each test case, a host of FLOPS $5.5 \cdot 10^9$ runs a unique project (100% of the CPU time). In the first case, the host performs tasks of Einstein@home project and the results are exactly the same in both simulators. In the case of SETI@home and LHC@home projects the results vary minimally.

	Einstein@home(100%)		*SETI@home*(100%)		*LHC@home*(100%)	
Time in hours	SimBOINC	ComBoS	SimBOINC	ComBoS	SimBOINC	ComBoS
100	16	16	263	263	395	394
500	82	82	1,318	1,319	1,978	1,972
1,000	164	164	2,637	2,639	3,956	3,945
5,000	824	824	13,177	13,195	19,780	19,728
10,000	1,649	1,649	26,315	26,390	39,473	39,457

Table 6.20: *Executed tasks (single project running on a single host of 5.5 GigaFLOPS).*

If we consider only the client scheduler, ComBoS results match those of SimBOINC, demonstrating the proper functioning of the simulator in this regard.

6.1.4 Validation of Whole Simulator

To validate the complete simulator, we have relied on data from the BOINCstats website [17], which provides official statistical results of BOINC projects. In this section, we analyze the behavior of ComBoS considering the simulation results of the SETI@home, Einstein@home and LHC@home projects.

We have used the CPU power traces of the client hosts that make up the VN of each project [18, 19, 20]. We have not used any other traces. In order to model the availability and unavail-

ability of the hosts, we used the results obtained in [21]. This research analyzed about 230,000 hosts' availability traces obtained from the SETI@home project. According to this paper, 21% of the hosts exhibit truly random availability intervals, and it also measured the goodness of fit of the resulting distributions using standard probability-probability (PP) plots. For availability, the authors saw that in most cases the Weibull distribution is a good fit. For unavailability, the distribution that offers the best fit is the log-normal. The parameters used for the Weibull distribution are $shape = 0.393$ and $scale = 2.964$. For the log-normal, the parameters obtained and used in ComBoS are a distribution with mean $\mu = -0.586$ and standard deviation $\sigma = 2.844$. All these parameters were obtained from [21] too. For the network parameters, we have used the bandwidth and latency values of current ADSL networks, and 10 Gbps for the network backbone. SimGrid's models allow us to adjust this network values. We have obtained all the other parameters of the simulations from the official websites of the SETI@home, Einstein@home, and LHC@home projects.

			BOINCstats		*ComBoS*	
Project	Total hosts	Active hosts	*GigaFLOPS*	*Credit/day*	*GigaFLOPS*	*Credit/day*
SETI@home	3,970,427	175,220	864,711	171,785,234	865,001	168,057,478
Einstein@home	1,496,566	68,338	1,044,515	208,902,921	1,028,172	205,634,486
LHC@home	356,942	15,814	7,521	1,504,214	7,392	1,393,931

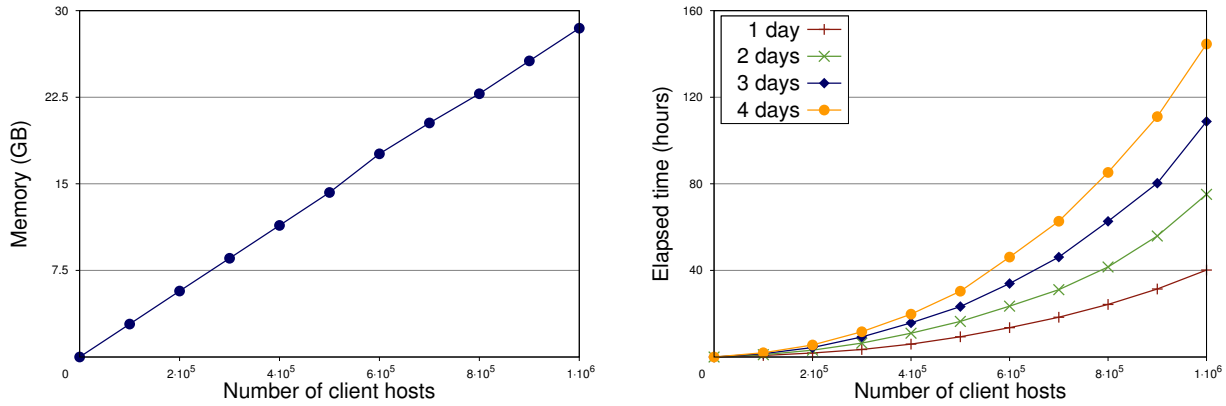
Table 6.21: *Validation of the whole simulator.*

Table 6.21 compares the actual results of the SETI@home, Einstein@home and LHC@home projects with those obtained with ComBoS in terms of GigaFLOPS and credits. The error obtained is 2.2% for credit/day and 0.03% for GigaFLOPS compared to the SETI@home project; 1.6% for credit/day and for GigaFLOPS compared to the Einstein@home project; and 7.3% for credit/day and 1.7% for GigaFLOPS compared to the LHC@home project. We consider that these results allow us to validate the whole simulator.

6.2 Performance Study

In this section we analyze the performance of the simulator in terms of memory usage and execution time. All measurements were made on a computer with 32 GB of RAM and 8 Intel Core i7 processors running at 2.67 Ghz each. The server runs the Linux 3.13.0-85-generic kernel. It runs the 3.10 version of the SimGrid toolkit. In spite of the computer has eight cores, each simulation was performed individually in a single core.

Figure 6-2a shows the memory usage of the simulator and Figure 6-2b shows the execution time by increasing the number of client hosts in each simulation. Note that the tests have been carried out up to 1 million hosts, the same number of active hosts of all BOINC projects together. Figure 6-2a shows a linear ($O(n)$) memory footprint. Figure 6-2b shows the execution time of the simulator for four different simulation times: 1 day, 2 days, 3 days and 4 days. Both metrics demonstrate that the simulator is highly scalable. This has been possible due to the high performance [22] of the SimGrid toolkit.



(a) *Memory usage.* (b) *Execution time.*

Figure 6-2: *Performance study.*

6.3 Case Studies

6.3.1 Combined Results

Chapter 7

Planificación y presupuesto

This chapter presents a detailed planning of the project (Section 7.1, *Planificación*). Then, we explain the project costs (Section 7.2, *Presupuesto*). At the end of the chapter, we comment on the socio-economic environment of the project (Section 7.3, *Entorno Socio-Económico*).

7.1 Planificación

This section includes the complete project planning. First, we describe the software development methodology used. After that, we detail the time duration of each phase of the project, collecting all times in a Gantt chart.

7.1.1 Justificación de la Metodología

Due to its characteristics, we have divided our project into three iterations:

- **Basic functionality:** the first iteration has been to achieve the simulation of a simple distributed computing system. The aim of this phase has been to simulate client machines that exchange messages with a server through the a network.

- **Client side:** this phase has been to incorporate all the necessary functionality on the client side (described in Chapter 4, *Diseño*).

- **Server side:** this phase has been to incorporate all the necessary functionality on the server side (described in Chapter 4, *Diseño*).

It was necessary to have an iterative methodology used to develop each of the phases independently to join all together in the last stage and obtain the final product. For this purpose, we have analyzed three different software development methodologies: Software prototyping [23], the Waterfall model [24] and the Spiral model [25]. Software prototyping did not fit well because it requires building a prototype of the software in a short time. The Waterfall model is a sequential design process, used in software development processes, in which progress is seen as flowing steadily downwards (like a waterfall) through different phases. The problem with this methodology is that it does not allow iterations within the software development. Finally, the Spiral model allowed fragmenting the project into different iterations. The model combines the strengths of the other two models (simplicity and flexibility), and uses an iterative process. Although this model is slower than the other two, it allowed us to apply different iterations so we decided to apply it to the whole process.

7.1.2 Ciclo de Vida

The life cycle development process of the project has followed the Spiral lifecycle model [25]. Figure 7-1 shows the Spiral model using a scheme.

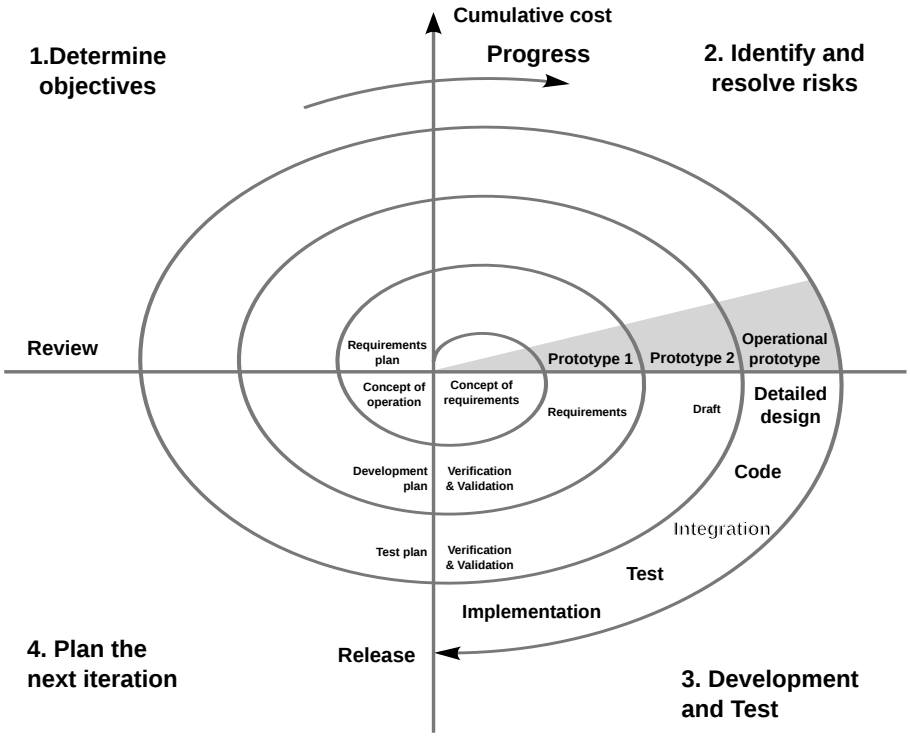


Figure 7-1: *Spiral model (Boehm, 2000).*

The Spiral model has four phases, which are repeated during the different iterations of the model. These phases are:

- **Planning** (Determine objectives in Figure 7-1): the user requirements are gathered, a feasibility study of the system is performed, and the iteration objectives are determined.
- **Analysis** (Identify and resolve risks in Figure 7-1): a full analysis of requirements is done and the potential risks are identified. This phase ends with a basic design.
- **Development and Test**: Code implementation is done. Test cases and test results are performed.
- **Evaluation** (Plan the next iteration in Figure 7-1): Customers evaluate the software and provide their feedback. In this case, the student tries to get the supervisor's approval. This is the *critical task* of the life cycle, since we can only move on to the next iteration of the Spiral lifecycle model if this task is approved.

Each phase starts with a design goal and ends with the customer (the supervisor) reviewing the progress so far. As previously explained, we have divided the software development into three iterations: basic functionality, client side, and server side. In the last iteration, the complete software must undergo extensive testing in order to validate the simulator.

7.1.3 Tiempo Estimado

The Gantt chart (Figure 7-2) shows all the tasks carried out during the project development. This project has been developed within a Collaboration in University Departments Scholarship [26], funded by the Spanish Ministry of Education, Culture, and Sport. The project began on November 2st, 2015, and ended on June 22, 2016, making a total of almost eight months of work. During this time, I have worked from Monday to Friday, four hours a day.

The Gantt chart shows all the tasks performed in each iteration of the spiral lifecycle model. Recall that the three iterations were: Basic functionality, Client side and Server side. In addition to the tasks (phases) mentioned above (Planning, Analysis, Development and Test, and Evaluation), we have included the Documentation task at the end of each iteration. The Documentation task has consisted mainly in drafting this bachelor thesis.

ID	Task name	Start	Finish	Duration
1	**Iteration 1: Basic functionality**	**Mon 11/2/15**	**Fri 12/11/15**	**30 days**
2	Planning	Mon 11/2/15	Tue 11/3/15	2 days
3	Analysis	Wed 11/4/15	Mon 11/9/15	4 days
4	Development and Test	Tue 11/10/15	Mon 11/30/15	15 days
5	Evaluation	Tue 12/1/15	Mon 12/7/15	5 days
6	Documentation	Tue 12/8/15	Fri 12/11/15	4 days
7	**Iteration 2: Client side**	**Mon 12/14/15**	**Tue 3/15/16**	**67 days**
8	Planning	Mon 12/14/15	Tue 12/15/15	2 days
9	Analysis	Wed 12/16/15	Tue 12/22/15	5 days
10	Development and Test	Wed 12/23/15	Tue 2/23/16	45 days
11	Evaluation	Wed 2/24/16	Wed 3/2/16	6 days
12	Documentation	Thu 3/3/16	Tue 3/15/16	9 days
13	**Iteration 3: Server side**	**Wed 3/16/16**	**Wed 6/22/16**	**71 days**
14	Planning	Wed 3/16/16	Thu 3/17/16	2 days
15	Analysis	Fri 3/18/16	Thu 3/31/16	10 days
16	Development and Test	Fri 4/1/16	Tue 5/24/16	38 days
17	Evaluation	Wed 5/25/16	Wed 6/1/16	6 days
18	Documentation	Thu 6/2/16	Wed 6/22/16	15 days

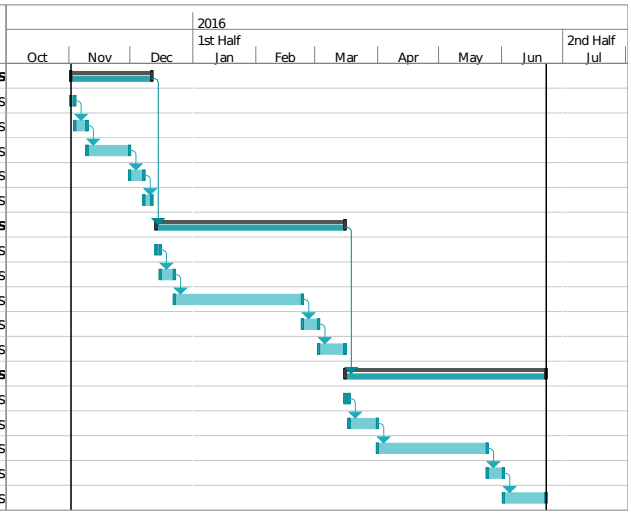


Figure 7-2: *Gantt chart.*

7.2 Presupuesto

This section details the overall project budget. On the one hand, we present the project costs and, on the other hand, we disclose the offer presented to the customer.

7.2.1 Coste del Proyecto

Table 7.1 summarizes the main features of the project including the total budget.

Project Information	
Title	A Complete Simulator for Volunteer Computing Environments
Author	Saúl Alonso Monsalve
Department	Computer Science and Engineering Department
Start date	2nd of November of 2015
End date	22nd of June of 2016
Duration	8 months
Indirect costs ratio	20 %
Total budget	30,526.49

Table 7.1: *Project Information.*

Then the total budget of the project is broken down below.

Costes Directos

In this part, the direct costs of the project are presented. Table 7.2 shows the direct costs caused by personnel costs, based on the planning presented in the previous section. The supervisor and the student have played the following roles:

- **Supervisor:** Project manager.
- **Student:** Analyst, Developer, Tester.

Category	Cost per hour (€)	Hours	Total (€)
Project manager	60	56	3,360
Analyst	35	188	6,580
Developer	35	316	11,060
Tester	25	112	2,800
Total			**23,800.00**

Table 7.2: *Human resources costs.*

Table 7.3 shows the direct costs caused by equipment acquisition and usage. The chargeable cost, C, is calculated using the following formula:

$$C = \frac{d \cdot c \cdot u}{D} \tag{7.1}$$

Where:

- **C:** Chargeable cost. It is equivalent to the depreciated value.
- **d:** Time the equipment has been used.
- **c:** Equipment cost.
- **u:** Project dedication. Percentage of time the equipment has been used.
- **D:** Equipment depreciation period.

Concept	Cost, c (€)	Dedication, u (%)	Dedication, d (months)	Depreciation, D (months)	Chargeable cost, C (€)
Desktop PC	799.99	100	8	36	177.78
Laptop	529.99	25	8	36	29.44
ARCOS Tucan	89,501.60	10	6	60	895.02
ARCOS Mirlo	2,469.99	70	6	60	172.90
Printer	399.24	5	3	60	1.00
Total					**1,276.14**

Table 7.3: *Equipment costs.*

Furthermore, the equipment presented in Table 7.3 is detailed below:

- **Desktop PC:** All in One - Asus Z220ICUK, 21.5", i5-6400T, 8GB, 1TB)
- **Laptop:** Toshiba L50D-C-19D, A10-8700P, 8GB RAM and 1TB.
- **ARCOS Tucan:** Cluster used by the research group ARCOS.
- **ARCOS Mirlo:** Server used by the research group ARCOS. 32GB RAM and eight i7 processors of 2.67GHz each.
- **Printer:** HP LaserJet Enterprise P3015.

Other direct costs are shown in Table 7.4. These costs consist of office material, a toner for the printer, and the monthly travel pass. Office material includes: pencils, pens, notebooks, paper, tipex, and markers.

Concept	Cost (€)
Office material	112.98
Toner (x1)	89.62
Monthly travel pass (x8)	160
Total	**362.60**

Table 7.4: *Other direct costs.*

Resumen de Costes

Table 7.5 shows the complete summary of the project costs. Indirect costs (20% of direct costs) consist of the electricity and water bills, telephone, Internet access, etc.

Costs summary	
Human resources	23,800.00
Equipment	1,276.14
Other direct costs	362.60
Indirect costs	5,087.75
Total budget	**30,526.49**

Table 7.5: *Costs summary.*

The total budget for this project amounts to **30,526.49 € (thirty thousand five hundred twenty-six euro and forty-nine cent)**.

7.2.2 Oferta de Proyecto Propuesta

Table 7.6 shows a detailed offer proposal. This offer includes the estimated risks (20%), the expected benefits (15%), and the Value Added Tax (Spanish Impuesto Sobre el Valor Añadido (IVA)), which corresponds to 21% [27]. After applying all theses concepts, the final amount for this project in case of sale to a third-party client is **50,973.14 € (fifty thousand nine hundred seventy-three euro and fourteen cent).**

Offer proposal			
Concept	**Increment (%)**	**Partial value (€)**	**Aggregated cost (€)**
Project costs	-	30,526.49	30,526.49
Risk	20	6,105.30	36,631.79
Benefits	15	5,494.77	42,126.56
IVA	21	8,846.58	50,973.14
Total			**50,973.14**

Table 7.6: *Offer proposal.*

7.3 Entorno Socio-Económico

As commented in previous chapters, ComBoS can guide the design of BOINC projects. This means that BOINC project designers can perform accurate simulations using ComBoS before deploying the system. Thanks to this, designers can save money and resources, because they will know the performance of the system before deploying it. In addition, it can also save energy because designers will not need to perform tests using the original infrastructure, as they will only need to use ComBoS in order to analyze the functioning of different alternatives.

Moreover, BOINC operates as a platform for distributed applications in areas as diverse as mathematics, medicine, molecular biology, climatology, environmental science, and astrophysics. On the one hand, there are projects that help the scientific community, such as the SETI@home project [28], of which the purpose is to analyze radio signals, searching for signs of extraterrestrial intelligence; or the Citizen Science Grid project [29], which is dedicated to supporting a wide range of research and educational projects. On the other hand, there are projects dedicated to the environmental care, such as the Climateprediction.net project [30], which studies climate. Therefore, our simulator indirectly contributes to both science and the environment.

Chapter 8

Conclusiones y trabajos futuros

In this chapter we discuss the main contributions of our work. In addition, we present the conclusions of the work, revise the objectives set at the beginning of this document, and include some personal conclusions. Finally, we discuss future work.

8.1 Contribuciones

8.2 Conclusiones

8.3 Trabajos Futuros

Appendix A

User Manual

This appendix presents a detailed user manual of ComBoS. First we indicate the basic requirements to deploy the application and a detailed tutorial for the installation of SimGrid. Finally, we present an example of the simulator usage.

A.1 Basic Requirements

The technical specifications recommended for the final user to obtain the best experience from the application are:

- **Operating System**: Ubuntu 14.04.4 LTS (Linux distribution) or higher.
- **Processor**: Intel(R) Core(TM) i7 CPU 920 @2.67GHz or higher.
- **RAM**: 8 GB or higher.
- **Storage**: 1 GB of free space in the Hard Disk Drive.
- **Network**: Internet connection is not required.
- **Software**: The following software must be installed in order to run the application:

    1. GCC (GNU Compiler) 5.1 or higher.
    2. SimGrid toolkit 3.10 or higher.

A.2 SimGrid Installation

We will present a tutorial for the installation of the SimGrid toolkit (version 3.10). First, you have to download the official binary package from the *download page* (`http://simgrid.gforge.inria.fr/download.php`). In this case you will download the file *SimGrid-3.10.tar.gz*.

Then, you have to recompile de archive. This should be done in a few lines:

```
$ tar xf SimGrid-3.10.tar.gz
$ cd SimGrid-3.10
$ cmake -DCMAKE_INSTALL_PREFIX=/opt/simgrid $HOME
$ make
$ make install
```

After following these steps, you will have the SimGrid toolkit installed in your computer.

A.3 Usage Example

In order to use ComBoS, you must download the corresponding files from the following website: `https://www.arcos.inf.uc3m.es/~combos/`. After unzipping the downloaded file, the unzipped files will follow the folder structure presented in Figure 5-2 (Section 5.2, *Deployment* (Chapter 5, *Implementación y despliegue*)). To perform simulations using ComBoS, it is necessary to model the platform to be simulated. Once you know the environment to simulate, you must specify all simulation parameters in the parameters XML file.

Figure A-1 shows an example of a potential simulation that can be carried out by ComBoS. The figure shows a simplified platform with two BOINC projects and 350,000 clients. The first project is represented by two scheduling servers (SS0 and SS1) and two data servers (DS0 and DS1). The second project consists of a single scheduling server (SS2) and three data servers (DS2, DS3 and DS4). Clients are grouped into three sets. The first group (G0) consists of 100,000 hosts and has a route to the first project. The second group (G1), has 200,000 hosts and a route to both projects. The third group (G2) consists of 50,000 computers and has route to the second project. The rest of the figure shows the links among the elements of the environment (from L0 to L7). In each of the links, latency and bandwidth are indicated.

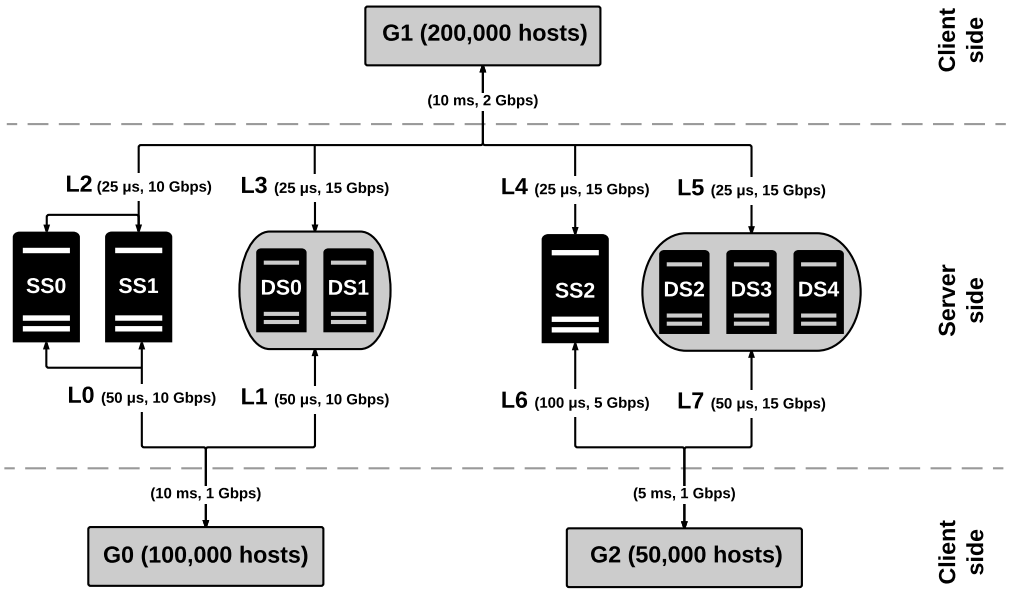
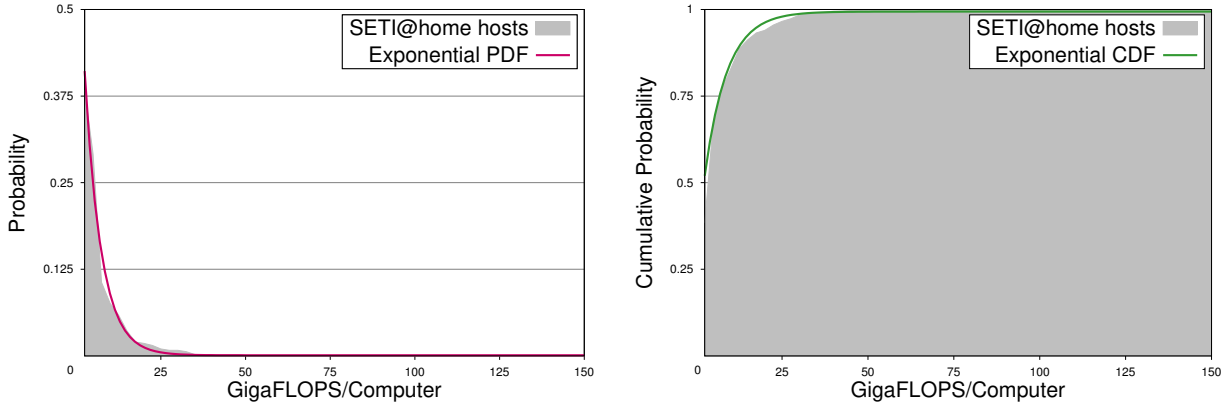


Figure A-1: *Simulator platform example.*

To create a simulation, ComBoS requires to specify all the parameters described in Tables **??** and **??** in the parameters.xml file (see Listing A.1). Users can define the power and availability of the volunteer hosts via either a traces file or distribution functions. For example, in the case of the SETI@home project, we have analyzed the 3,900,000 hosts that participate in this project. The CPU performance of the hosts can be modeled according to an exponential function, as shown Figure A-2, which has a mean of 5.871 GigaFLOPS per host.



(a) *Probability density function of SETI@home hosts power.*

(b) *Cumulative distribution function of SETI@home hosts power.*

Figure A-2: *CPU performance modeling for SETI@home hosts.*

Our software first processes the XML file, creating the necessary deployment and platform files for subsequent simulations. In ComBoS, all this is transparent to the user. The user only has to specify all the parameters of the simulation in the XML file (Listing A.1), and run the generator script using the following command:

```
$ ./generator
```

The above command generates the platform and deployment files. The platform file contemplates all the necessary elements in the simulation: hosts, clusters, links, etc. The deployment file indicates the processes that should be created during the simulation. In addition, the generator script also compiles all source files needed for the simulation and generates the executable file. Finally, to run the simulation you just need to run the execution script:

```
$ ./execute
```

The execution results are composed by multiple statistical results (see Listings A.2): the execution time, the memory usage of the simulator, the load of the scheduling and data servers, the total number of work requests received in the scheduling servers, the job statistics (number of jobs created, sent, received, analyzed, success, fail, too late, etc), the credit granted to the clients, the number of FLOPS, the average power of the volunteer nodes, and the percentage of time the volunteer nodes were available during the simulation.

```
<simulation_time>100</simulation_time>

<!-- Server side -->
<server_side>
    <n_projects>2</n_projects>
    <sproject>
        <snumber>0</snumber>
        <name>PROJECT1</name>
        <nscheduling_servers>2</nscheduling_servers>
        <ndata_servers>2</ndata_servers>
        <server_pw>12000000000.0</server_pw>
        <disk_bw>80000000</disk_bw>
        <ifgl_percentage>100<ifgl_percentage>
        <ifcd_percentage>100<ifcd_percentage>
        <input_file_size>368640</input_file_size>
        <task_fpops>7560000000000</task_fpops>
        <output_file_size>65536</output_file_size>
        <min_quorum>2</min_quorum>
        <target_nresults>2</target_nresults>
        <max_error_results>2</max_error_results>
        <max_total_results>4</max_total_results>
        <max_success_results>3</max_success_results>
        <delay_bound>100000000</delay_bound>
        <success_percentage>95</success_percentage>
        <canonical_percentage>95</canonical_percentage>
        <replication>2</replication>
    <sproject/>
    <sproject>
        <snumber>1</snumber>
        <name>PROJECT2</name>
        <nscheduling_servers>1</nscheduling_servers>
        <ndata_servers>3</ndata_servers>
        <server_pw>12000000000.0</server_pw>
        <disk_bw>60000000</disk_bw>
        <ifgl_percentage>100<ifgl_percentage>
        <ifcd_percentage>100<ifcd_percentage>
        <input_file_size>52428800</input_file_size>
        <task_fpops>20800000000000</task_fpops>
        <output_file_size>16777216</output_file_size>
```

```
        <min_quorum>2</min_quorum>
        <target_nresults>2</target_nresults>
        <max_error_results>2</max_error_results>
        <max_total_results>4</max_total_results>
        <max_success_results>3</max_success_results>
        <delay_bound>100000000</delay_bound>
        <success_percentage>95</success_percentage>
        <canonical_percentage>95</canonical_percentage>
        <replication>2</replication>
    <sproject/>
</server_side>

<!-- Client side -->
<client_side>
    <n_groups>3</n_groups>
    <group>
        <n_clients>100000</n_clients>
        <connection_interval>1</connection_interval>
        <scheduling_interval>3600</connection_interval>
        <gbw>1Gbps</gbw>
        <glatency>10ms</glatency>
        <traces_file>NULL</traces_file>
        <max_speed>117.71</max_speed>
        <min_speed>0.07</min_speed>
        <pv_distri>5</pv_distri>
        <pa_param>0.1734</pa_param>
        <pb_param>-1</pb_param>
        <av_distri>0</av_distri>
        <aa_param>0.393</aa_param>
        <ab_param>2.964</ab_param>
        <nv_distri>2</nv_distri>
        <na_param>2.844</na_param>
        <nb_param>-0.586</nb_param>
        <att_projs>1</att_projs>
        <gproject>
            <pnumber>0</pnumber>
            <priority>1</priority>
            <lsbw>10Gbps</lsbw>
            <lslatency>50us</lslatency>
```

```
79          <ldbw>10Gbps</ldbw>
80          <ldlatency>50us</latency>
81      </gproject>
82  </group>
83  <group>
84      <n_clients>200000</n_clients>
85      <connection_interval>1</connection_interval>
86      <scheduling_interval>3600</connection_interval>
87      <gbw>2Gbps</gbw>
88      <glatency>10ms</glatency>
89      <traces_file>NULL</traces_file>
90      <max_speed>117.71</max_speed>
91      <min_speed>0.07</min_speed>
92      <pv_distri>5</pv_distri>
93      <pa_param>0.1734</pa_param>
94      <pb_param>−1</pb_param>
95      <av_distri>0</av_distri>
96      <aa_param>0.393</aa_param>
97      <ab_param>2.964</ab_param>
98      <nv_distri>2</nv_distri>
99      <na_param>2.844</na_param>
100     <nb_param>−0.586</nb_param>
101     <att_projs>2</att_projs>
102     <gproject>
103         <pnumber>0</pnumber>
104         <priority>1</priority>
105         <lsbw>10Gbps</lsbw>
106         <lslatency>25us</lslatency>
107         <ldbw>15Gbps</ldbw>
108         <ldlatency>25us</latency>
109     </gproject>
110     <gproject>
111         <pnumber>1</pnumber>
112         <priority>1</priority>
113         <lsbw>15Gbps</lsbw>
114         <lslatency>25us</lslatency>
115         <ldbw>15Gbps</ldbw>
116         <ldlatency>25us</latency>
117     </gproject>
```

```
118     </group>
119     <group>
120         <n_clients>50000</n_clients>
121         <connection_interval>1</connection_interval>
122         <scheduling_interval>3600</connection_interval>
123         <gbw>1Gbps</gbw>
124         <glatency>5ms</glatency>
125         <traces_file>NULL</traces_file>
126         <max_speed>117.71</max_speed>
127         <min_speed>0.07</min_speed>
128         <pv_distri>5</pv_distri>
129         <pa_param>0.1734</pa_param>
130         <pb_param>−1</pb_param>
131         <av_distri>0</av_distri>
132         <aa_param>0.393</aa_param>
133         <ab_param>2.964</ab_param>
134         <nv_distri>2</nv_distri>
135         <na_param>2.844</na_param>
136         <nb_param>−0.586</nb_param>
137         <att_projs>1</att_projs>
138         <gproject>
139             <pnumber>1</pnumber>
140             <priority>1</priority>
141             <lsbw>5Gbps</lsbw>
142             <lslatency>100us</lslatency>
143             <ldbw>15Gbps</ldbw>
144             <ldlatency>50us</latency>
145         </gproject>
146     </group>
147 </client_side>
```

Listing A.1: *parameters.xml file filled with the parameters of the example.*

```
1  Memory usage: 11,570,812 KB
2
3  Total number of clients: 350,000
4
5  ##################### PROJECT1 ####################
6
7  Simulation ends in 100 h (360,000 sec)
8
9  Scheduling server 0: Busy: 13.4%
10 Scheduling server 1: Busy: 13.4%
11 Data server 0:  Busy: 12.2%
12 Data server 1:  Busy: 12.2%
13
14  Number of clients: 300,000
15  Messages received:   32,227,795 (work requests received + results received)
16  Work requests received:  16,179,688
17  Results created:   16,179,689 (100.0%)
18  Results sent:   16,179,688 (100.0%)
19  Results received:   16,048,107 (99.2%)
20  Results analyzed:   16,048,107 (100.0%)
21  Results success:   15,245,637 (95.0%)
22  Results failed:   802,470 (5.0%)
23  Results too late:   0 (0.0%)
24  Results valid:   13,616,636 (84.8%)
25  Workunits total:   7,716,639
26  Workunits completed:   6,863,142 (88.9%)
27  Workunits not completed:  853,497 (11.1%)
28  Workunits valid:   6,808,318 (88.2%)
29  Workunits error:   54,824 (0.7%)
30  Throughput:   89.5 mens/s
31  Credit granted:   231,482,812 credits
32  FLOPS average:   285,949 GFLOPS
33
34 ##################### PROJECT2 ####################
35
36 Simulation ends in 100 h (360,000 sec)
37
38 Scheduling server 0: Busy: 1.7%
39 Data server 0:  Busy: 100.0%
```

```
40 Data server 1: Busy: 100.0%
41 Data server 2: Busy: 100.0%
42
43  Number of clients: 250,000
44  Messages received:  2,070,120 (work requests received + results received)
45  Work requests received:  1,161,080
46  Results created:  1,161,081 (100.0%)
47  Results sent:  1,161,080 (100.0%)
48  Results received:  909,040 (78.3%)
49  Results analyzed:  909,040 (100.0%)
50  Results success:  863,710 (95.0%)
51  Results failed:  45,330 (5.0%)
52  Results too late:  0 (0.0%)
53  Results valid:  743,688 (81.8%)
54  Workunits total:  559,384
55  Workunits completed:  374,713 (67.0%)
56  Workunits not completed:  184,671 (33.0%)
57  Workunits valid:  371,844 (66.5%)
58  Workunits error:  2,869 (0.5%)
59  Throughput:  5.8 mens/s
60  Credit granted:  35,697,024 credits
61  FLOPS average:  42,968 GFLOPS
62
63 Group 0. Average speed: 6.704465 GFLOPS. Available: 62.1% Not available 37.9%
64 Group 1. Average speed: 5.455870 GFLOPS. Available: 55.3% Not available 44.7%
65 Group 2. Average speed: 6.110686 GFLOPS. Available: 57.0% Not available 43.0%
66
67 Clients. Average speed: 5.906157 GFLOPS. Available: 57.5% Not available 42.5%
68
69 Execution time:
70 0 days 7 hours 51 min 49 s
```

Listing A.2: *Simulation execution results.*

Bibliography

[1] "P8080e simulator." https://www.datsi.fi.upm.es/docencia/Estructura/U_Control/#HERRAMIENTAS. Accessed: 2017-04-26.

[2] R.-F. Yen and Y. Kim, "Development and implementation of an educational simulator software package for a specific microprogramming architecture," *IEEE Transactions on Education*, no. 1, pp. 1–11, 1986.

[3] A. S. Tanenbaum, *Structured Computer Organization 2nd*. ACM, 1984.

[4] J. R. Larus, *Spim s20: A mips r2000 simulator*. Center for Parallel Optimization, Computer Sciences Department, University of Wisconsin, 1990.

[5] I. Aguilar Juárez and J. R. Heredia Alonso, "Simuladores y laboratorios virtuales para ingenierÍa en computación," 2013.

[6] K. Vollmar and P. Sanderson, "Mars: an education-oriented mips assembly language simulator," in *ACM SIGCSE Bulletin*, vol. 38, pp. 239–243, ACM, 2006.

[7] S. R. Vegdahl, "Mipspilot: A compiler-oriented mips simulator," *Journal of Computing Sciences in Colleges*, vol. 24, no. 2, pp. 32–39, 2008.

[8] M. I. Garcia, S. Rodríguez, A. Pérez, and A. García, "p88110: A graphical simulator for computer architecture and organization courses," *IEEE Transactions on Education*, vol. 52, no. 2, pp. 248–256, 2009.

[9] I. Branovic, R. Giorgi, and E. Martinelli, "Webmips: a new web-based mips simulation environment for computer architecture education," in *Proceedings of the 2004 workshop on Computer architecture education: held in conjunction with the 31st International Symposium on Computer Architecture*, p. 19, ACM, 2004.

[10] J. C. PEREZ, F. G. CARBALLEIRA, J. D. G. Sánchez, and D. E. Singh, *Problemas resueltos de estructura de computadores*. Ediciones Paraninfo, SA, 2015.

[11] Institute for Electrical and Electronics Engineers, "IEEE Recommended Practice for Software Requirements Specifications," *IEEE*, pp. 830–1998, 1998.

[12] BOE, 19 de enero de 2008, "Real Decreto 1720/2007, de 21 de diciembre, por el que se aprueba el Reglamento de desarrollo de la Ley Orgánica 15/1999, de 12 de diciembre, de protección de datos de carácter personal.," *Boletín Oficial del Estado, 17:4103-4136*.

[13] "The GNU Lesser General Public License." http://www.gnu.org/licenses/licenses.html, Last visited, June 2016.

[14] A. C. Mateos, F. G. Carballeira, and J. P. Cepeda, "Wepsim: Simulador modular e interactivo de un procesador elemental para facilitar una visión integrada de la microprogramación y la programación en ensamblador," *Enseñanza y aprendizaje de ingeniería de computadores: Revista de Experiencias Docentes en Ingeniería de Computadores*, no. 6, pp. 35–53, 2016.

[15] UNIX, "drand48." `http://pubs.opengroup.org/onlinepubs/7908799/xsh/drand48.html`, Last visited, June 2016.

[16] Software Testing Fundamentals, "Verification vs Validation." `http://softwaretestingfundamentals.com/verification-vs-validation/`, Last visited, Mar. 2016.

[17] "BOINCstats." `http://boincstats.com/en/stats`, Last visited, Feb. 2016.

[18] SETI@home, "CPU performance of SETI@home volunteer computers." `http://setiathome.berkeley.edu/cpu_list.php`, Last visited, Mar. 2016.

[19] EINSTEIN@home, "CPU performance of EINSTEIN@home volunteer computers." `https://www.einsteinathome.org/cpu_list.php`, Last visited, Mar. 2016.

[20] LHC@home classic, "CPU performance of LHC@home volunteer computers." `http://lhcathomeclassic.cern.ch/sixtrack/cpu_list.php`, Last visited, Mar. 2016.

[21] Bahman Javadi, Derrick Kondo, Jean-Marc Vincent, David P. Anderson, "Discovering Statistical Models of Availability in Large Distributed Systems: An Empirical Study of SETI@home," *Parallel and Distributed Systems, IEEE Transactions*, vol. 22, pp. 1896–1903, 2011.

[22] Arnaud Legrand, "Scheduling for Large Scale Distributed Computing Systems: Approaches and Performance Evaluation Issues," tech. rep., Université Grenoble Alpes, 2015.

[23] Accelerated Technologies, Inc, "The Human Condition: A Justification for Rapid Prototyping," *Time Compression Technologies, vol. 3 no. 3*, p. 1, May 1998.

[24] Benington, Herbert D., "Production of Large Computer Programs," *IEEE Annals of the History of Computing (IEEE Educational Activities Department)*, p. 350–361, Oct. 1983.

[25] B. Boehm, "A Spiral Model of Software Development and Enhancement," *IEEE Computer 21, 5*, pp. 61–72, 1988.

[26] BOE, 30 de junio de 2015, "Resolución de 17 de junio de 2015, de la Secretaría de Estado de Educación, Formación Profesional y Universidades, por la que se convocan becas de colaboración de estudiantes en departamentos universitarios para el curso académico 2015-2016," *Boletín Oficial del Estado, 155:53607-53616*.

[27] BOE, 6 de agosto de 2012, "Resolución de 2 de agosto de 2012, de la Dirección General de Tributos, sobre el tipo impositivo aplicable a determinadas entregas de bienes y prestaciones de servicios en el Impuesto sobre el Valor Añadido," *Boletín Oficial del Estado, 187:56055-56060*.

[28] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer, "SETI@home: an experiment in public-resource computing," *Commun. ACM*, vol. 45, no. 11, pp. 56–61, 2002.

[29] "Citizen Science Grid." http://csgrid.org/csg/, Last visited, Feb. 2016.

[30] Oxford University, "Climateprediction.net." http://climateprediction.net, Last visited, Apr. 2016.