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".

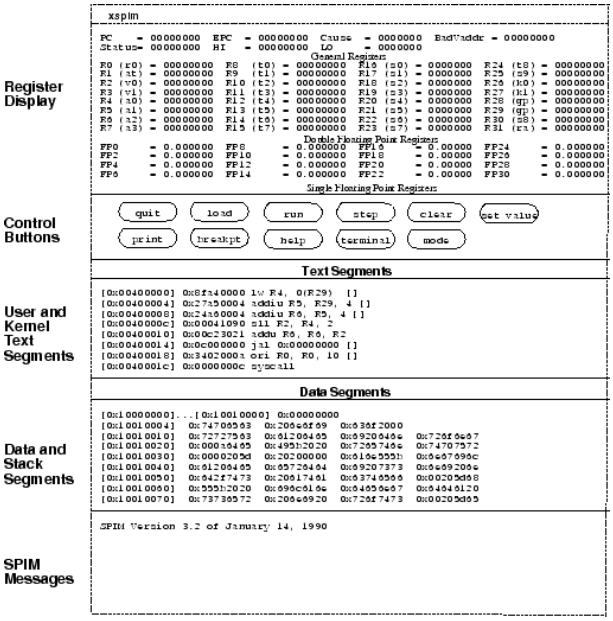



Figure 2-2: *Interfaz SPIM.*

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.

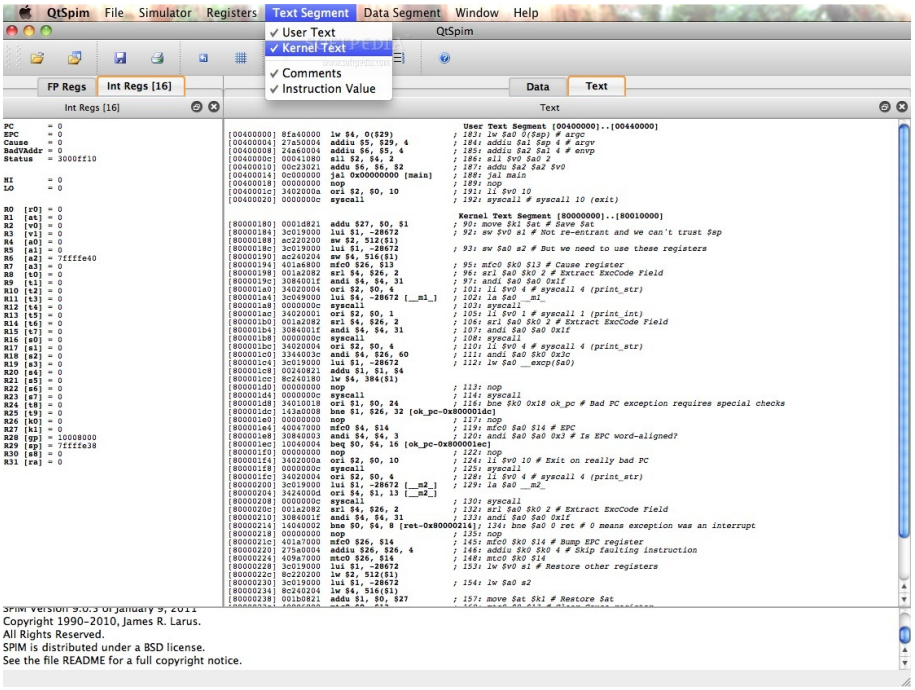



Figure 2-3: *Interfaz QtSPIM.*

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.

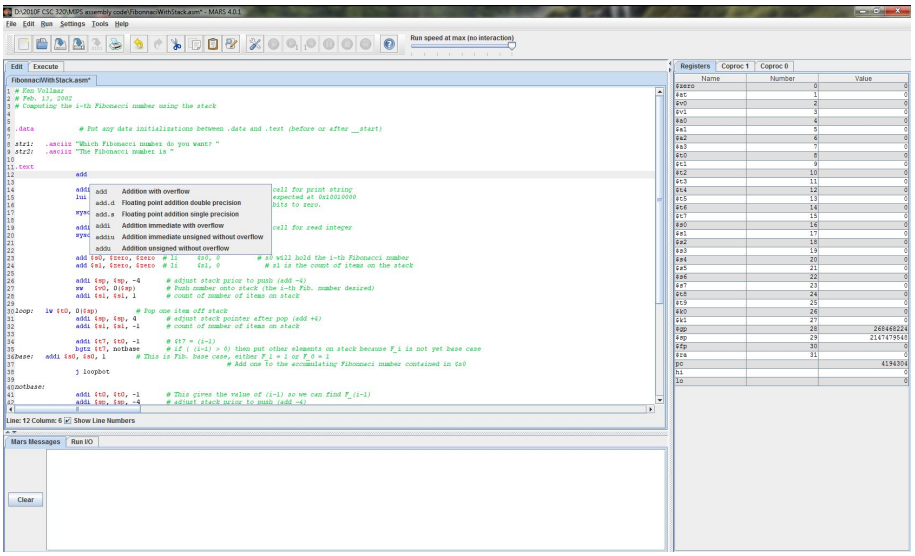


Figure 2-4: *Interfaz MARS.*

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

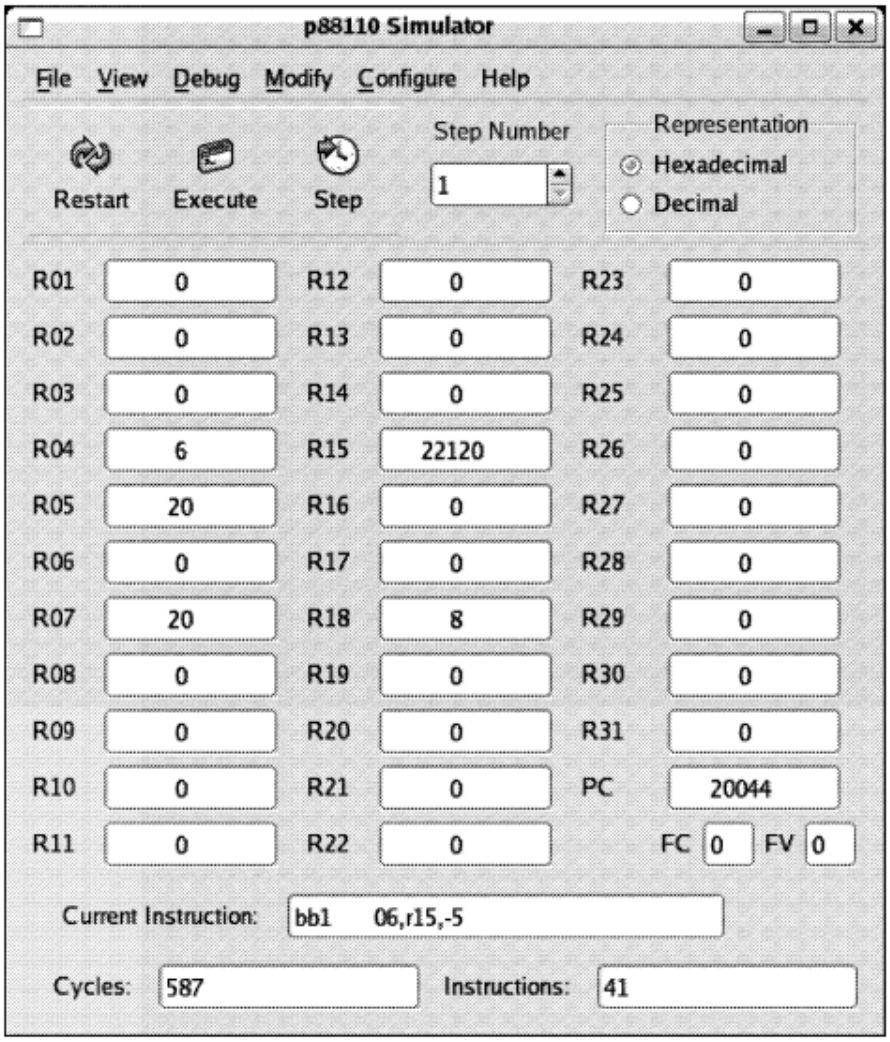



Figure 2-5: *Interfaz em88110.*

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.

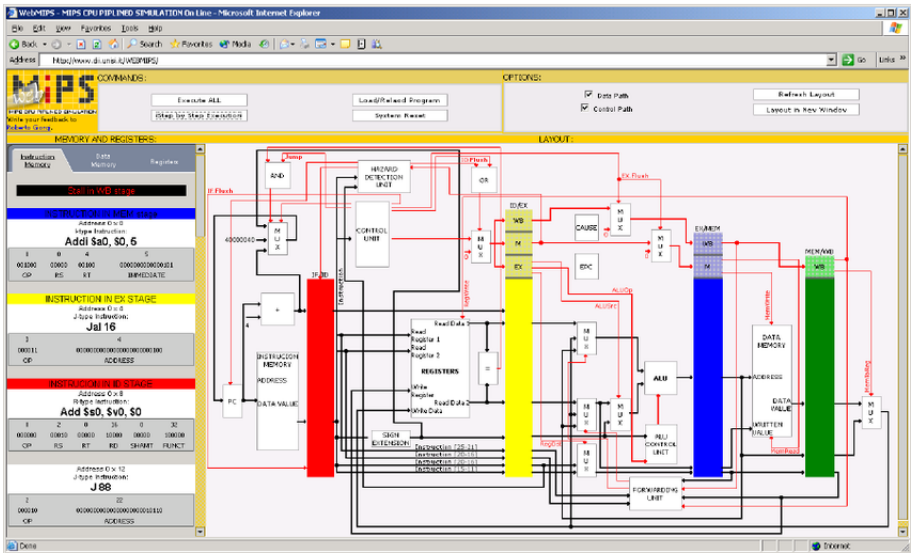



Figure 2-6: *Interfaz WebMIPS.*

## 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.

| 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.*

# 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 discute la solución elegida y la compara con las alternativas consideradas. La sección 3.3 especifica los requisitos del sistema, empezando con los requisitos de usuario y finalizando con los requisitos funcionales y no-funcionales. Finalmente, la sección 3.4 indica el conjunto de leyes y regulaciones para la gestión del software.

## 3.1 Descripción del proyecto

Explicamos lo que se pretende obtener del simulador.

## 3.2 Solución elegida

Explicamos el porque hemos elegido esta solución, y nos comparamos.

| Features | SimGrid | PVMsim | Virtual-GEMS | MDCSim | SPECI-2 |
|---|---|---|---|---|---|
| Languages | C/C++/Java/Ruby | C | C/C++/Ruby | C++/Java | Java |
| Open Source | ✓ | ✓ | ✓ | ✗ | ✓ |
| Models | | | | | |
| Communication | ✓ | ✗ | ✗ | ✓ | ✓ |
| Energy | ✓ | ✗ | ✗ | ✓ | ✗ |
| Hardware | ✓ | ✓ | ✓ | ✓ | ✓ |
| Scheduling | ✓ | ✓ | ✗ | ✗ | ✓ |
| Users | ✓ | ✗ | ✗ | ✓ | ✗ |

Table 3.1: *Comparison of simulation frameworks for distributed computing systems.*

## 3.3 Requisitos

This section provides a detailed description of the application requirements. For the requirement specification task, the IEEE recommended practices [10] were followed. According to these practices, a good specification must address the software functionality, performance issues, the external interfaces, other non-functional features and design or implementation constraints. Moreover, the requirements specification must be:

- **Complete:** the document reflects all significant software requirements.
- **Consistent:** requirements must not generate conflicts with each other.
- **Correct:** every requirement is one that the software shall meet according to the user needs.
- **Modifiable:** the structure of the specification allows changes to the requirements in a simple, complete and consistent way.
- **Ranked based on importance and stability:** every requirement must indicate its importance and its stability.
- **Traceable:** the origin of every requirement is clear and it can be easily referenced in further stages.
- **Unambiguous:** every requirement has a single interpretation.
- **Verifiable:** every requirement must be verifiable, that is, there exists some process to verify that the software complies with every single requirement.

Starting from the user requirements, which constitute an informal reference to the product performance that the client expects, we derived the software requirements (in this case, functional requirements and non-functional requirements) that guided the design process with specific information on the functionality of the system and other characteristics. The retrieved requirements were structured according with the following schema:

1. **User Requirements**
   (a) **Capacity:** the requirement describes the expected system functionality as in use cases.
   (b) **Restriction:** the requirement specifies constraints or conditions the system must fulfil.
2. **Software Requirements**
   (a) **Functional**
      i. **Functional:** the requirement describes the basic system functionality and purpose while minimizing ambiguity.
      ii. **Inverse:** the requirement limits the functionality of the application to clarify its scope.
   (b) **Non-Functional**
      i. **Performance:** the requirement is related to the minimum required performance of the resulting system.
      ii. **Interface:** the requirement is related to the user interface of the application.
      iii. **Scalability:** the requirement is related to the ability of the system to adapt to increasing workloads.
      iv. **Platform:** the requirement specifies the underlying software and hardware platforms in which the system will operate.

Table 3.2 provides the template used for requirements specification. Note that for user requirements, the ID format will be UR-XYY, where X indicates the requirement subtype: capacity requirements (C), or restrictions (R). YY corresponds to the requirement number under its subcategory. For software requirements, the ID format SR-X-YZZ will be used, where X indicates if it is a functional (F) or non-functional (NF) requirement, and Y represents its subcategory:

functional (F), inverse (I), performance (P), interface (UI), scalability (S), or platform (PL). ZZ corresponds to the requirement number under its subcategory.

| | |
|---|---|
| **ID** | Requirement ID. |
| **Name** | Requirement name. |
| **Type** | Indicates the category in which the requirement would be placed according to the previously described schema. |
| **Origin** | Constitutes the requirement source. It might be the user, another requirement or other stakeholders involved in the project. |
| **Priority** | Indicates the requirement priority according to its importance. A requirement can be identified either as *essential*, *conditional* or *optional*. |
| **Stability** | Indicates the requirement variability through the development process, defined as *stable* or *unstable*. |
| **Description** | Detailed explanation of the requirement. |

Table 3.2: *Template for requirements specification.*

### 3.3.1 Requisitos de Usuario

This subsection specifies the user requirements.

| | |
|---|---|
| **ID** | UR-C01 |
| **Name** | BOINC projects simulation |
| **Type** | Capacity |
| **Origin** | User |
| **Priority** | Essential |
| **Stability** | Stable |
| **Description** | The application shall simulate real BOINC projects. |

Table 3.3: *User requirement UR-C01.*

| | |
|---|---|
| **ID** | UR-C02 |
| **Name** | Client scheduling |
| **Type** | Capacity |
| **Origin** | User |
| **Priority** | Essential |
| **Stability** | Stable |
| **Description** | The client scheduler of the simulator shall follow the actual BOINC client scheduling. |

Table 3.4: *User requirement UR-C02.*

| | |
|---|---|
| **ID** | UR-C03 |
| **Name** | Simulation components |
| **Type** | Capacity |
| **Origin** | User |
| **Priority** | Essential |
| **Stability** | Stable |
| **Description** | The simulations shall cover all the elements present in the BOINC infrastructure. |

Table 3.5: *User requirement UR-C03.*

| | |
|---|---|
| **ID** | UR-R01 |
| **Name** | Linux as underlying OS |
| **Type** | Restriction |
| **Origin** | User |
| **Priority** | Essential |
| **Stability** | Stable |
| **Description** | The simulator shall be designed for Linux operating systems. |

Table 3.6: *User requirement UR-R01.*

| | |
|---|---|
| **ID** | UR-R02 |
| **Name** | SimGrid toolkit |
| **Type** | Restriction |
| **Origin** | User |
| **Priority** | Essential |
| **Stability** | Stable |
| **Description** | The application shall use the SimGrid toolkit in order to implement the distributed computing functionalities. |

Table 3.7: *User requirement UR-R02.*

| | |
|---|---|
| **ID** | UR-R03 |
| **Name** | Scalability |
| **Type** | Restriction |
| **Origin** | User |
| **Priority** | Essential |
| **Stability** | Stable |
| **Description** | The simulator shall be scalable (carry out executions by simulating a large number of client hosts). |

Table 3.8: *User requirement UR-R03.*

### 3.3.2 Requisitos Funcionales

This subsection specifies the functional requirements.

| | |
|---|---|
| **ID** | SR-F-F01 |
| **Name** | Credit calculation |
| **Type** | Functional |
| **Origin** | UR-C01 |
| **Priority** | Essential |
| **Stability** | Stable |
| **Description** | The simulator shall calculate the number of credits granted to each volunteer client analogously to actual BOINC projects. |

Table 3.9: *Functional requirement SR-F-F01.*

| | |
|---|---|
| **ID** | SR-F-F02 |
| **Name** | Collection of statistics |
| **Type** | Functional |
| **Origin** | UR-C01 |
| **Priority** | Essential |
| **Stability** | Stable |
| **Description** | The simulator shall collect, for each project, the same statistics that actual BOINC projects (published in BOINCstats [11]). |

Table 3.10: *Functional requirement SR-F-F02.*

| | |
|---|---|
| **ID** | SR-F-F03 |
| **Name** | Almost identical outputs |
| **Type** | Functional |
| **Origin** | UR-C01 |
| **Priority** | Essential |
| **Stability** | Stable |
| **Description** | The outputs of the simulator for existing projects should be almost identical to those published in BOINCstats [11]. |

Table 3.11: *Functional requirement SR-F-F03.*

| | |
|---|---|
| **ID** | SR-F-F04 |
| **Name** | Multiple BOINC projects |
| **Type** | Functional |
| **Origin** | UR-C01 |
| **Priority** | Essential |
| **Stability** | Stable |
| **Description** | The simulator shall allow the simulation of different projects simultaneously. |

Table 3.12: *Functional requirement SR-F-F04.*

| | |
|---|---|
| **ID** | SR-F-F05 |
| **Name** | Client scheduler |
| **Type** | Functional |
| **Origin** | UR-C02 |
| **Priority** | Essential |
| **Stability** | Stable |
| **Description** | The client scheduler shall follow the actual BOINC client scheduling (described in [12]). |

Table 3.13: *Functional requirement SR-F-F05.*

| | |
|---|---|
| **ID** | SR-F-F06 |
| **Name** | Realistic simulation elements |
| **Type** | Functional |
| **Origin** | UR-C03 |
| **Priority** | Essential |
| **Stability** | Stable |
| **Description** | All simulations shall include the following elements: tasks, volunteer hosts, servers, data servers, networks, and hosts availability. |

Table 3.14: *Functional requirement SR-F-F06.*

### 3.3.3 Requisitos No-Funcionales

This subsection specifies the non-functional requirements.

| | |
|---|---|
| **ID** | SR-NF-PL01 |
| **Name** | Ubuntu 14.04 |
| **Type** | Platform |
| **Origin** | UR-R01 |
| **Priority** | Essential |
| **Stability** | Stable |
| **Description** | The simulator shall work on the Ubuntu Linux distribution, version 14.04. |

Table 3.15: *Non-functional requirement SR-NF-PL01.*

| | |
|---|---|
| **ID** | SR-NF-PL02 |
| **Name** | SimGrid MSG API |
| **Type** | Platform |
| **Origin** | UR-R02 |
| **Priority** | Essential |
| **Stability** | Stable |
| **Description** | The implementation, setup and control of the simulations shall be carried out using the MSG API of the SimGrid toolkit. |

Table 3.16: *Non-functional requirement SR-NF-PL02.*

| | |
|---|---|
| **ID** | SR-NF-PL03 |
| **Name** | C programming language |
| **Type** | Platform |
| **Origin** | UR-R03 |
| **Priority** | Essential |
| **Stability** | Stable |
| **Description** | The simulator shall be written in the C programming language. |

Table 3.17: *Non-functional requirement SR-NF-PL03.*

| | |
|---|---|
| **ID** | SR-NF-S01 |
| **Name** | Large simulations |
| **Type** | Scalability |
| **Origin** | UR-R03 |
| **Priority** | Essential |
| **Stability** | Stable |
| **Description** | The application must be able to perform simulations with more than 100,000 hosts in a machine with at least 8 GB of Random-Access Memory (RAM). |

Table 3.18: *Non-functional requirement SR-NF-S01.*

| | |
|---|---|
| **ID** | SR-NF-P01 |
| **Name** | Linear-time execution |
| **Type** | Performance |
| **Origin** | UR-R03 |
| **Priority** | Conditional |
| **Stability** | Stable |
| **Description** | Runtime of the simulator must be linear (approximately) in the number of hosts. |

Table 3.19: *Non-functional requirement SR-NF-P01.*

| | |
|---|---|
| **ID** | SR-NF-UI01 |
| **Name** | Simulation parameters |
| **Type** | Interface |
| **Origin** | Analyst |
| **Priority** | Essential |
| **Stability** | Stable |
| **Description** | To perform simulations, users only need to specify the simulation parameters in an Extensible Markup Language (XML) file. |

Table 3.20: *Non-functional requirement SR-NF-UI01.*

| | |
|---|---|
| **ID** | SR-NF-UI02 |
| **Name** | Progress bar |
| **Type** | Interface |
| **Origin** | Analyst |
| **Priority** | Conditional |
| **Stability** | Stable |
| **Description** | The simulations should include a progress bar. |

Table 3.21: *Non-functional requirement SR-NF-UI02.*

## 3.4 Marco Regulador

This section discusses the necessary constraints taking into account the regulatory framework. Specifically, the legal restrictions applicable to the simulator are specified.

### 3.4.1 Restricciones Legales

In the real BOINC system, users must be registered, and Berkeley Open Infrastructure for Network Computing (BOINC) databases handle confidential information from users, so it is necessary to ensure that third parties can not access that information. One solution is to encrypt the information transmitted following some cryptographic protocol. In Spain, this requirement is specified in the article 104 of the RD 1720/2007 [13], which deals with the Spanish Data Protection Law.

In contrast, the developed application does not use private data from users, and neither transmits any confidential information to third-parties, because it is just a simulator that does not even require Internet access.

On the other hand, it is crucial that our simulator be available as an open-source software. We want it to be such that anyone can redistribute the code or modify it by the terms of the GNU Lesser General Public License (LGPL) [14]. To do this, our simulator is available on the following website: `https://www.arcos.inf.uc3m.es/~combos/`.

# Chapter 4

# Diseño

This chapter provides a complete description of the developed simulator, including the internal architecture and the different software components. Complete BOINC Simulator (ComBoS) is a complete simulator of BOINC infrastructures that simulates the behavior of all componentes involved: projects, servers, network, scheduling, redundant computing, and volunteer nodes. In this chapter we describe all the simulator components (Section 4.1, *Componentes del Simulador*) and present the policies of the client scheduler (Section **??**, **??**).

## 4.1 Componentes del Simulador

### 4.1.1 Interfaz Gráfica

### 4.1.2 Core

### 4.1.3 Compilador

# 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.

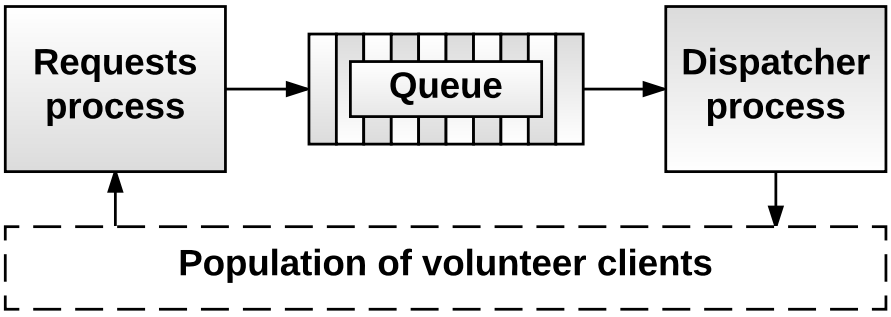



Figure 5-1: *Server main processes.*

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.
- **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.

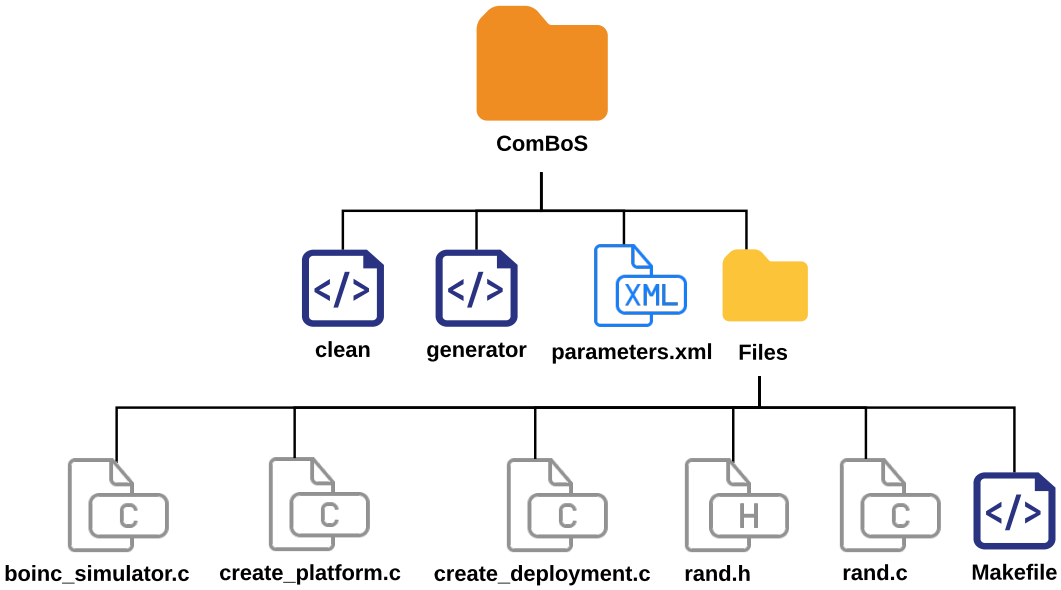



Figure 5-2: *Initial folder structure.*

The files located inside the 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 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 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).

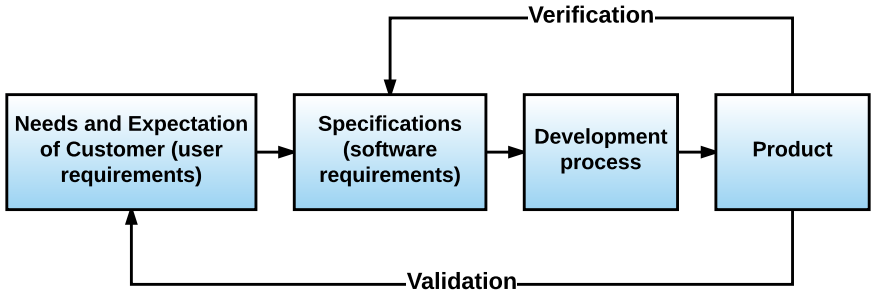



Figure 6-1: *Software verification and validation.*

*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.<br>2. GCC (GNU Compiler) 5.1 or higher must be installed on the machine.<br>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).<br>2. Run the generator script with the default parameters to create the simulation files.<br>3. Run the simulation. |
| **Postconditions** | 1. All source files must be written in C programming language.<br>2. The implementation, setup and control of the simulations must be carried out using the MSG API of the SimGrid toolkit.<br>3. Simulations run while a progress bar indicates the percentage of execution.<br>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.<br>2. Run the generator script to create the simulation files.<br>3. Run the simulation. |
| **Postconditions** | 1. Simulations run while a progress bar indicates the percentage of execution.<br>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 [11]. 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.<br>2. Run the generator script to create the simulation files.<br>3. Run the simulation. |
| **Postconditions** | 1. Simulations run while a progress bar indicates the percentage of execution.<br>2. The simulator must successfully finish its execution.<br>3. The outputs of the simulator contain at least: credits, hosts, active hosts, and FLOPS (The same as those published by BOINCstats [11]). |
| **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).<br>2. Run the generator script to create the simulation files.<br>3. Run the simulation. |
| **Postconditions** | 1. Simulations run while a progress bar indicates the percentage of execution.<br>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.<br>2. Run the generator script to create the simulation files.<br>3. Run the simulation. |
| **Postconditions** | 1. Simulations run while a progress bar indicates the percentage of execution.<br>2. The simulator must successfully finish its execution.<br>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 [11]. |
| **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.<br>2. Run the generator script to create the simulation files.<br>3. Run the simulation. |
| **Postconditions** | 1. Simulations run while a progress bar indicates the percentage of execution.<br>2. The simulator must successfully finish its execution.<br>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.<br>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.<br>2. Run the generator script to create the simulation files.<br>3. Run the simulation. |
| **Postconditions** | 1. Simulations run while a progress bar indicates the percentage of execution.<br>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.<br>2. Run the generator script to create the simulation files.<br>3. Run the simulation.<br>4. Go to 2 specifying different simulation parameters. |
| **Postconditions** | 1. Simulations run while a progress bar indicates the percentage of execution.<br>2. The simulator must successfully finish its execution.<br>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.<br>2. Run the generator script to create the simulation files.<br>3. Run the simulation. |
| **Postconditions** | 1. The simulator must successfully finish its execution.<br>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.<br>2. Run the generator script to create the simulation files.<br>3. Run the simulation. |
| **Postconditions** | 1. The simulator must successfully finish its execution.<br>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.<br>2. Run the generator script to create the simulation files.<br>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.<br>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).<br>2. Run the generator script with the default parameters to create the simulation files.<br>3. Run the simulation. |
| **Postconditions** | 1. The simulator must successfully finish its execution.<br>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.<br>2. Run the generator script to create the simulation files.<br>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 [11], 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 [17, 18, 19]. 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 [20]. 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 [20] 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 |

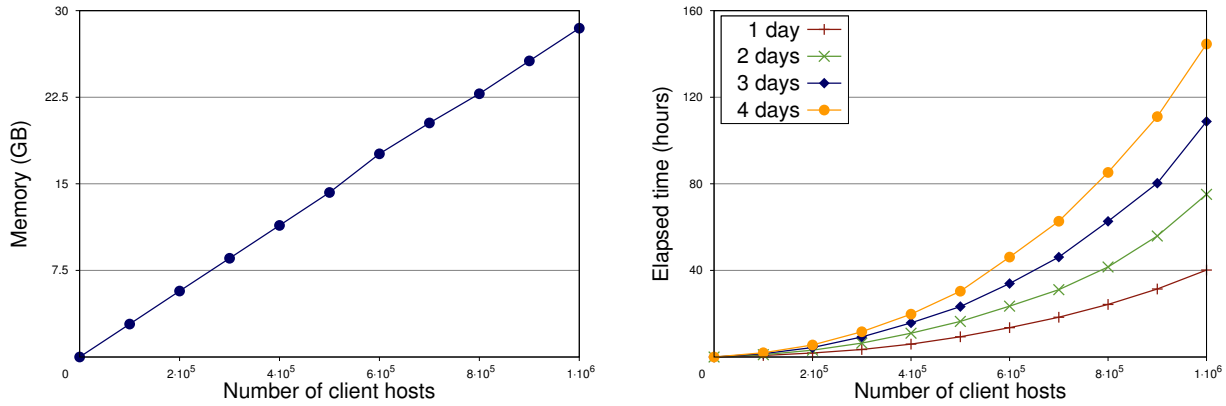
Table 6.21: *Validation of the whole simulator.*

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 [21] 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 [22], the Waterfall model [23] and the Spiral model [24]. 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 [24]. Figure 7-1 shows the Spiral model using a scheme.

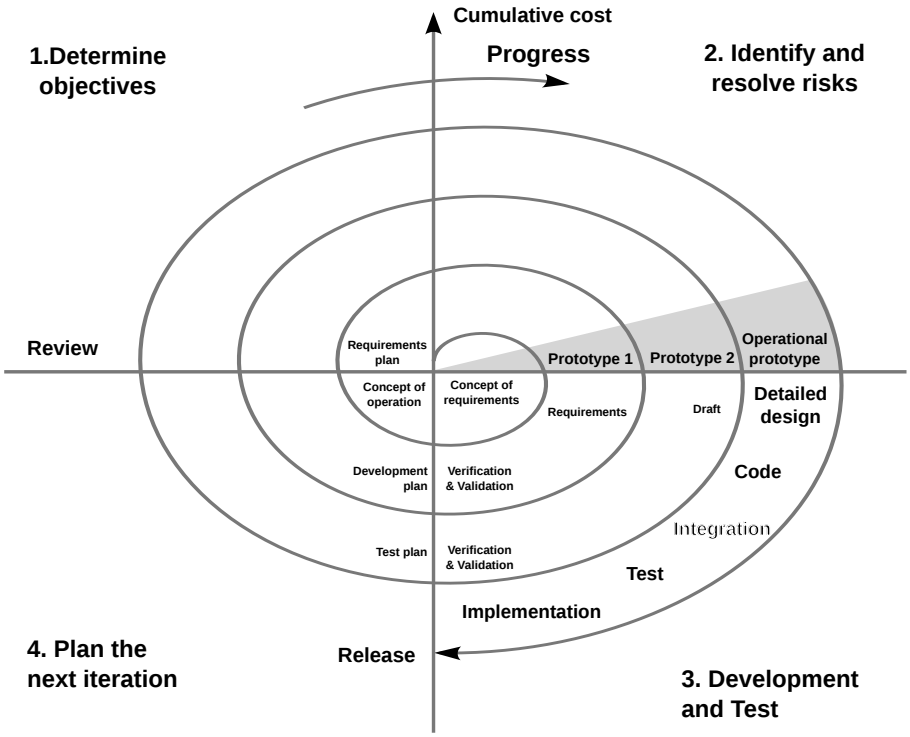



Figure 7-1: *Spiral model (Boehm, 2000).*

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 [25], 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 |

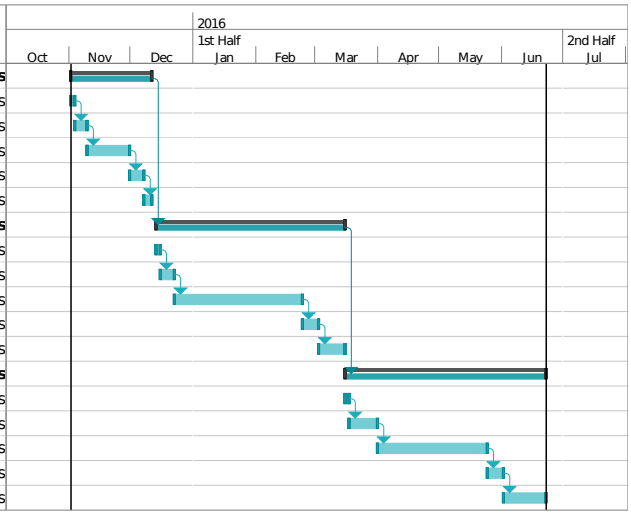



Figure 7-2: *Gantt chart.*

## 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% [26]. 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 [27], of which the purpose is to analyze radio signals, searching for signs of extraterrestrial intelligence; or the Citizen Science Grid project [28], 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 [29], 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.

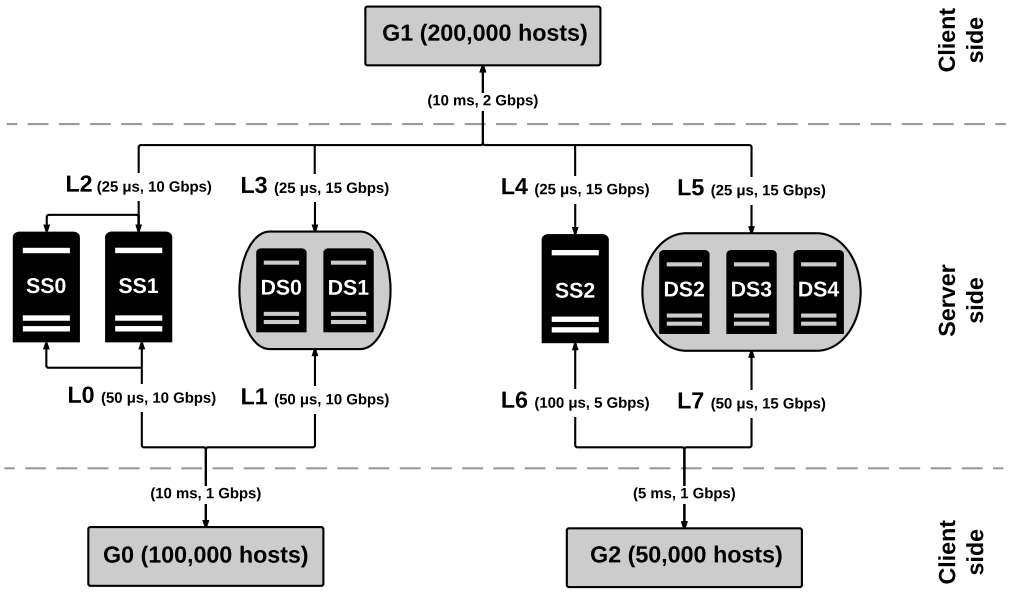
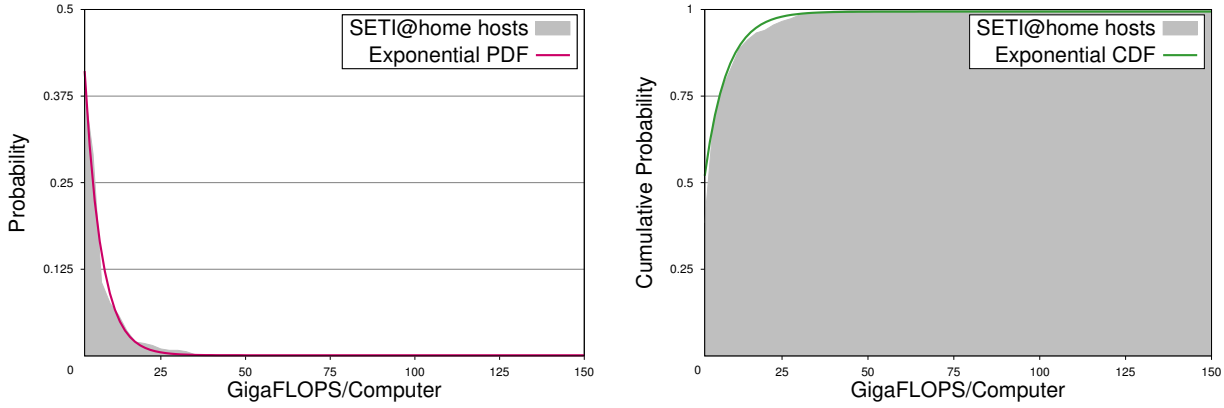



Figure A-1: *Simulator platform example.*

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.*

```
Memory usage: 11,570,812 KB

Total number of clients: 350,000

#################### PROJECT1 ####################

Simulation ends in 100 h (360,000 sec)

Scheduling server 0: Busy: 13.4%
Scheduling server 1: Busy: 13.4%
Data server 0:  Busy: 12.2%
Data server 1:  Busy: 12.2%

 Number of clients: 300,000
 Messages received:   32,227,795 (work requests received + results received)
 Work requests received:  16,179,688
 Results created:   16,179,689 (100.0%)
 Results sent:   16,179,688 (100.0%)
 Results received:   16,048,107 (99.2%)
 Results analyzed:   16,048,107 (100.0%)
 Results success:   15,245,637 (95.0%)
 Results failed:   802,470 (5.0%)
 Results too late:   0 (0.0%)
 Results valid:   13,616,636 (84.8%)
 Workunits total:   7,716,639
 Workunits completed:   6,863,142 (88.9%)
 Workunits not completed:  853,497 (11.1%)
 Workunits valid:   6,808,318 (88.2%)
 Workunits error:   54,824 (0.7%)
 Throughput:   89.5 mens/s
 Credit granted:   231,482,812 credits
 FLOPS average:   285,949 GFLOPS

#################### PROJECT2 ####################

Simulation ends in 100 h (360,000 sec)

Scheduling server 0: Busy: 1.7%
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] Institute for Electrical and Electronics Engineers, "IEEE Recommended Practice for Software Requirements Specifications," *IEEE*, pp. 830–1998, 1998.

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

[12] D. P. Anderson, "Local scheduling for volunteer computing," in *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pp. 1–8, IEEE, 2007.

[13] 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*.

[14] "The GNU Lesser General Public License." http://www.gnu.org/licenses/licenses.html, Last visited, June 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] SETI@home, "CPU performance of SETI@home volunteer computers." http://setiathome.berkeley.edu/cpu_list.php, Last visited, Mar. 2016.

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

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

[20] 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.

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

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

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

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

[25] 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*.

[26] 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*.

[27] 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.

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

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