Universidad de las Ciencias Informáticas

Facultad 1

Título: Plataforma de procesamiento distribuido en entornos colaborativos

Trabajo de diploma para optar por el título de INGENIERO EN CIENCIAS INFORMÁTICAS

Autores:

Heidy Luis Díaz

Sael José Berrillo Borrero

Tutores:

Ing. Enrique Almeida Maldonado

MsC. Jorge Landrian García

Cotutor:

Ing. Félix Alejandro Prieto Carratala

La Habana, 17 de junio del 2013

Declaración de autoría

Declaramos que somos los únicos autores de este trabajo y autorizamos al Centro de Identificación y Seguridad Digital de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio. Para que así conste firmamos la presente a los ____ días del mes de _____ del año _____.

_____	_____
Heidy Luis Díaz	Sael José Berrillo Borrero
_____	_____
Ing. Enrique Almeida Maldonado	MsC. Jorge Landrian García

Resumen

A medida que la tecnología avanza los problemas científicos a resolver se vuelven cada vez más complejos y requieren de mayor capacidad de cómputo. Un ordenador en solitario no es capaz de resolver este tipo de problemas, sin embargo, la unión de varios ordenadores trabajando sobre partes estructuradas del problema puede generar mejores resultados; con este objetivo surgen los sistemas de computación distribuida.

El departamento de Seguridad Digital del CISED[1] de la UCI[2], a partir del surgimiento de problemas complejos en la rama de la criptografía, ha identificado la necesidad de desarrollar una plataforma de procesamiento distribuido que permita resolver problemas de alto costo computacional, utilizando la capacidad de procesamiento disponible en las computadoras de la universidad. Debido a que por políticas de seguridad, la mayoría de estas computadoras son utilizadas por usuarios con privilegios restringidos, y sumado a que gran parte de las actividades que se realizan sobre las mismas utilizan el navegador web, se propone el uso de los mismos mediante extensiones, que no requieran de privilegios administrativos en su instalación, para servir como nodos de procesamiento.

En el presente trabajo se realiza un estudio de los principales sistemas de computación distribuida existentes, adquiriendo mayor visión sobre sus ventajas y desventajas para posteriormente elaborar una propuesta de solución.

El documento recoge los resultados de la investigación realizada, describiendo las principales características de los sistemas analizados, así como la arquitectura y diseño del sistema propuesto, las herramientas y tecnologías utilizadas, y los artefactos generados en el proceso de desarrollo.

Palabras claves: Computación distribuida, computación voluntaria, navegador web, *plugin*, procesamiento distribuido.

[1]CISED: Centro de Identificación y Seguridad Digital.
[2]UCI: Universidad de las Ciencias Informáticas.

Índice

Índice de figuras

Índice de tablas

Introducción

Desde el surgimiento de las primeras computadoras a mediados del siglo XX [1] la informática ha estado en constante desarrollo. Con el transcurso de los años, se ha ido perfeccionando, evolucionando desde las gigantescas máquinas que permitían realizar tareas muy simples y de uso restringido para organizaciones muy selectas, hasta los más modernos ordenadores con mucha mayor capacidad que los primeros y de uso muy cotidiano.

Un evento muy importante en el desarrollo de las Tecnologías de la Información y la Comunicación (TIC) fue la creación de las redes de computadoras en el año 1969 [2]. Gracias a las redes de ordenadores, y a la creación y constante progreso de Internet, se hizo posible el intercambio de información, la integración de aplicaciones en diferentes ubicaciones físicas, así como la colaboración y optimización de recursos. El uso de esta nueva tecnología estuvo en constante aumento llegando a incorporarse a todos los sectores de la vida económica y social, siendo común en la actualidad el uso de una red en una escuela, una fábrica o una oficina.

El desarrollo de la computación distribuida vino de la mano de las redes locales de alta velocidad, permitiendo el uso de varias computadoras conectadas en red para conseguir un objetivo común. Dentro de la computación distribuida existen diferentes tipos de sistemas como son: los sistemas de computación clusters, los sistemas de computación grid y los sistemas de computación voluntaria, siendo estos últimos los más utilizados para lograr ambientes colaborativos en el procesamiento masivo de datos. En la computación voluntaria, también conocida como computación de ciclos redundantes, los propietarios de ordenadores, habitualmente domésticos, donan sus recursos informáticos, tanto de almacenamiento como de procesamiento, para ejecutar tareas de uno o más proyectos [3].

Cuba, a pesar de ser un país subdesarrollado, no se ha quedado al margen de este desarrollo tecnológico. Algunas de las acciones llevadas a cabo con el fin de fomentar el desarrollo de la sociedad cubana a través del uso de las tecnologías informáticas fueron la creación del Ministerio de Informática y Comunicaciones (MIC), el programa de los Joven Club de Computación, la introducción de la computación en todos los niveles de la educación, el desarrollo de multimedias educativas para la enseñanza primaria y secundaria, la creación de la carrera de Ingeniería Informática en todas las provincias y la creación en el 2002 [4] de la Universidad de las Ciencias Informáticas (UCI).

La UCI tiene como misión formar ingenieros en Ciencias Informáticas altamente calificados, capaces de impulsar la industria cubana del *software*. Con el objetivo de cumplir con esta importante misión, cuenta con un amplio y riguroso programa de estudios que incluye la vinculación de los estudiantes a los proyectos productivos. En la mayoría de estos proyectos surgen frecuentemente tareas cuyas soluciones implican una elevada demanda de cómputo y su respuesta debe darse en un tiempo limitado. Específicamente en el departamento Seguridad Digital del Centro de Identificación y Seguridad Digital

(CISED), en algunas de las tareas que realiza, se deben desarrollar operaciones matemáticas relacionadas con criptografía. Para resolver dichas tareas existen en el mundo ordenadores de alto rendimiento a los cuales no se puede acceder debido a su elevado precio económico en el mercado internacional. Por el contrario, la universidad cuenta con una gran cantidad de computadoras conectadas, formando una de las mayores redes del país, sin embargo, ninguna de ellas es suficientemente potente para resolverlas o demoraría demasiado en obtener un resultado, por lo que el uso de una sola computadora no resulta factible. La capacidad de procesamiento de estas computadoras en muchas ocasiones no es empleada al máximo ya que la mayoría del tiempo se encuentran realizando tareas con un bajo costo sobre los recursos de *hardware*, principalmente la ejecución de aplicaciones web, debido a que en la actualidad todos los ordenadores están dotados de un navegador y en la universidad, como en el resto del mundo, la mayoría de los servicios se brindan mediante la web.
Adicionalmente, por políticas de seguridad, la mayoría de estas computadoras son utilizadas por usuarios con privilegios restringidos, por lo que resulta imposible para ellos instalar programas que requieran para su instalación de permisos sobre carpetas restringidas del sistema operativo (SO). En esta investigación, a este tipo de programas se les denominará programas invasivos, puesto que implican cambios en carpetas pertenecientes a los sistemas de archivos del SO sobre los que se quieran instalar.

Esta situación conlleva a plantear el siguiente **problema de investigación**: ¿Cómo facilitar la resolución de tareas de alto costo computacional en el CISED?

En este contexto se tiene como **objeto de estudio**: La computación distribuida.

Para ello se plantea como **objetivo general** de la investigación: Desarrollar una plataforma de procesamiento distribuido no invasiva, que permita resolver tareas de alto costo computacional en el CISED, a partir del aprovechamiento de la capacidad de procesamiento disponible en las diferentes computadoras de la red.

Desglosándose en los siguientes **objetivos específicos**:

- Analizar los principales conceptos asociados a la computación distribuida.
- Realizar un estudio de las soluciones existentes para la resolución de tareas complejas.
- Definir las herramientas y tecnologías que se utilizarán para el desarrollo de la solución.
- Implementar una plataforma de procesamiento distribuido no invasiva.
- Validar la solución desarrollada.

Enmarcándose en el **campo de acción**: El procesamiento distribuido mediante la computación voluntaria.

Para guiar la investigación se plantea la siguiente **idea a defender**: El desarrollo de una plataforma de procesamiento distribuido no invasiva, que permita utilizar la capacidad de procesamiento disponible en las diferentes estaciones de trabajo de la red UCI, facilitará la resolución de tareas que requieren de alto costo computacional en el CISED.

La investigación se desarrollará a través de las siguientes **tareas de investigación**:

- Elaboración del marco teórico de la investigación. (Heidy Luis Díaz)
- Estudio de las estrategias existentes para la implementación de plataformas de procesamiento distribuido. (Sael José Berrillo Borrero)
- Selección de la estrategia adecuada para resolver tareas de alto costo computacional. (Heidy Luis Díaz y Sael José Berrillo Borrero)
- Definición de la metodología, las herramientas y tecnologías a utilizar para el desarrollo de la aplicación. (Heidy Luis Díaz y Sael José Berrillo Borrero)
- Diseño del sistema. (Heidy Luis Díaz y Sael José Berrillo Borrero)
- Implementación del sistema propuesto. (Heidy Luis Díaz y Sael José Berrillo Borrero)
- Realización de pruebas de *software* al sistema desarrollado. (Heidy Luis Díaz y Sael José Berrillo Borrero)

La investigación estará sustentada en los siguientes **métodos de investigación**:

Métodos teóricos:

- Analítico-sintético: permitirá consultar diversas fuentes bibliográficas y extraer los elementos más importantes que se relacionan con el objeto de estudio, siendo de gran importancia en el estudio del estado del arte.
- Análisis histórico-lógico: posibilitará la compresión lógica del objeto de estudio haciendo un análisis riguroso de sus antecedentes y el proceso evolutivo por el cual han transitado todas las tecnologías relacionadas con el procesamiento distribuido.

Métodos empíricos:

- Entrevista: permitirá obtener información, experiencias, ideas, puntos de vistas, que contribuyan al desarrollo de la investigación y aporten conocimiento específico del tema.

Justificación de la investigación

El presente trabajo propone la elaboración de una plataforma de procesamiento distribuido que permita la resolución de tareas de alto costo computacional en el CISED, reduciendo considerablemente su tiempo de respuesta. Esta plataforma constituye una propuesta novedosa ya que plantea la creación de un *plugin* como *software* para los trabajadores de la misma, aspecto poco abordado en los sistemas distribuidos. Dicho aporte evita la necesidad de una instalación previa en los trabajadores, posibilitando que tanto los

usuarios que no poseen permisos para ejecutar programas invasivos como los que poseen poco conocimiento sobre informática puedan fácilmente donar sus recursos para colaborar con determinados proyectos, lo que contribuye a alcanzar una mayor cantidad de colaboradores.

Estructura del documento

El documento ha sido estructurado en 3 capítulos, los cuales se describen a continuación:

Capítulo I: Fundamentación teórica: en este capítulo se identifican y describen los conceptos asociados a la computación distribuida. Se realiza un análisis sobre las principales plataformas de procesamiento distribuido existentes en el mundo. Además, se realiza la selección y descripción de las tecnologías, herramientas y metodología de desarrollo de *software* a utilizar para el desarrollo de la solución.

Capítulo II: Diseño de la solución: este capítulo recoge la descripción y análisis de la solución propuesta, así como la arquitectura y los patrones de diseño seleccionados para el desarrollo de la misma. Se describen las clases que modelan la solución y sus principales funcionalidades, y se definen las características que debe tener la plataforma mediante la especificación de los requerimientos funcionales y no funcionales.

Capítulo III: Implementación y prueba: en este capítulo se traducen los elementos del diseño a implementación. Se describe la implementación del sistema en términos de componentes y la manera en que estos componentes serán desplegados. Además, recoge los estándares de codificación utilizados, las diferentes interfaces y funcionalidades de la plataforma implementada, así como los resultados de las pruebas realizadas a la misma.

Capítulo I: Fundamentación teórica

1.1 Introducción

En el presente capítulo se describen los elementos fundamentales que constituyen el soporte teórico de la investigación. Se definen los conceptos fundamentales relacionados con la computación distribuida. Se realiza un estudio del estado del arte, a nivel nacional e internacional, de las principales plataformas de procesamiento distribuido que puedan dar solución al problema de investigación. Además, se describen las tecnologías, metodologías de desarrollo de *software* y herramientas que se utilizarán en el desarrollo de la solución.

1.2 Computación distribuida

La computación distribuida surge para resolver problemas de computación masiva utilizando un gran número de computadoras conectadas en red [5]. La capacidad de procesamiento, la memoria y el almacenamiento de datos son recursos compartidos por todas las computadoras del sistema distribuido, permitiendo a una sola utilizar la potencia del resto, convirtiendo una red de computadoras en un potente superordenador [6]. Es por esto que la computación distribuida ha sido diseñada para resolver problemas demasiado grandes para cualquier supercomputadora, dividiéndolos en múltiples problemas más pequeños, denominados sub-problemas. Cada uno de ellos se resuelve de manera paralela e independiente por un ordenador diferente. Las soluciones de los sub-problemas, se recogen y se combinan por otra máquina denominada servidor, la cual también es la encargada de descomponer el problema original en los sub-problemas [7].

“Un sistema distribuido es una colección de ordenadores independientes que se muestra ante sus usuarios como un único sistema coherente” [8].

“Definimos un sistema distribuido como aquel en el que los componentes *hardware* o *software*, localizados en computadores unidos mediante red, comunican y coordinan sus acciones solo mediante paso de mensajes” [9].

Los sistemas distribuidos pueden ser tan simple como un grupo de máquinas similares funcionando con el mismo SO, o tan complejo como una red heterogénea formada por cualquier plataforma informática. Su tamaño puede variar desde decenas de hosts (LAN), centenas de hosts (MAN), o miles o millones de hosts (WAN) [6].

1.2.1 Principios de la computación distribuida

Los sistemas distribuidos tienen un conjunto de características que constituyen principios que deben tenerse en cuenta para su diseño y construcción.

1.2.1.1 Extensibilidad

La extensibilidad de un sistema distribuido es la característica que determina si puede ser extendido y re-implementado respecto a diferentes aspectos. Pueden ser extendidos a nivel de *hardware*, mediante la inclusión de computadoras a la red, y a nivel de *software*, por la introducción de nuevos servicios y la re-implementación de los ya existentes. La extensibilidad de *software* se logra especificando y documentando las interfaces de *software* claves del sistema y poniéndolas a disposición de los desarrolladores [9].

1.2.1.2 Escalabilidad

Un sistema distribuido se considera escalable si conserva su efectividad cuando ocurre un incremento significativo del número de usuarios y recursos. El *software* de los sistemas distribuidos debe seguir funcionando, sin necesidad de realizar cambios, cuando la escala del sistema se incrementa [9].

1.2.1.3 Tratamiento de fallos

Los sistemas distribuidos deben conocer las posibles formas en que pueden fallar los componentes de los que depende y estar diseñados para tratar apropiadamente cada uno de estos fallos [9].

1.2.1.4 Concurrencia

La concurrencia es una de las características más importantes de los sistemas distribuidos, consiste en la ejecución de varios procesos de manera simultánea. En una computadora equipada con un único procesador central, la concurrencia se lleva a cabo intercalando la ejecución de los distintos procesos, mientras que en los sistemas distribuidos la cantidad de procesos que pueden ejecutarse en paralelo es proporcional a la cantidad de computadoras conectadas al sistema [10].

1.2.1.5 Transparencia

La transparencia consiste en ocultar al usuario y al programador de aplicaciones la separación de los componentes de un sistema distribuido, de manera que el sistema se perciba como un todo y no como una colección de componentes independientes. Existen ocho tipos o grados de transparencia: acceso, ubicación, concurrencia, replicación, frente a fallos, movilidad, prestaciones y escalado. De ellos los dos más importantes son la transparencia de acceso y la transparencia de ubicación [9].

La transparencia de acceso permite acceder a los recursos remotos de la misma manera que se accede a los recursos locales. La transparencia de ubicación permite ocultar la localización de los recursos de forma tal que puedan ser accedidos sin conocer su localización.

1.2.2 Tipos de computación distribuida

La computación distribuida puede ser clasificada de acuerdo a sus características en computación cluster, computación grid y computación voluntaria. A continuación se describen detalladamente cada una de ellas.

1.2.2.1 Computación cluster

Los clusters están formados por un conjunto de ordenadores conectados entre sí mediante una red de alta velocidad, trabajando como un solo recurso de computación integrado. Suelen disponer de un *software* que realiza la distribución de la carga de trabajo y establece las comunicaciones entre las diferentes computadoras o nodos del cluster. Los nodos poseen la misma ubicación física y disponen del mismo SO. Comúnmente, este tipo de sistemas distribuidos cuenta con un único centro de almacenamiento de datos compartido. Los clusters utilizan todos los ordenadores en busca de lograr un mejor rendimiento de cada uno de ellos [3].

1.2.2.2 Computación grid

En un sistema grid diferentes organizaciones unen sus recursos para lograr la colaboración entre un grupo de personas o instituciones. Esta colaboración se realiza en la forma de una organización virtual (OV), ofreciendo a las personas que pertenecen a una misma OV derechos de acceso a los recursos proporcionados por la misma [11].

Los sistemas grid cuentan con un número indeterminado de ordenadores dedicados, generalmente distribuidos geográficamente y pertenecientes a diferentes organizaciones. Estas computadoras se encuentran unidas de manera transparente, generando a la vista del usuario un único recurso. Permiten compartir una amplia variedad de recursos entre los que se encuentran: computadoras, supercomputadoras, sistemas de almacenamiento, bases de datos y dispositivos especializados, tales como telescopios, sensores, etc. Además, poseen un elevado nivel de heterogeneidad ya que en ellos se pueden encontrar diferentes clases de sistemas operativos [3].

1.2.2.3 Computación voluntaria

La computación voluntaria es un acuerdo en el que personas o entidades donan sus recursos computacionales a proyectos investigativos de su interés, los cuales utilizan dichos recursos para el procesamiento y almacenamiento distribuido. Consiste en uno o varios servidores que distribuyen tareas para ser procesadas por un grupo de ordenadores voluntarios que colaboran en la ejecución de un proyecto. Fundamentalmente, cuando la computadora permanece encendida, pero sin utilizarse, la capacidad de procesamiento se desperdicia generalmente en algún protector de pantalla. Este tipo de computación distribuida utiliza las computadoras voluntarias cuando no están siendo utilizadas, de esta

manera, se dona a la ejecución de los proyectos los ciclos redundantes, aprovechando al máximo la capacidad de procesamiento de las computadoras. Este tipo de computación es ampliamente utilizada ya que reúne gran cantidad de ordenadores voluntarios lo que permite desarrollar una gran capacidad de cálculo [3].

1.2.2.4 Fundamentación de la selección del tipo de computación distribuida

La potencia computacional que se obtiene con la unión de los ordenadores domésticos y portátiles del mundo excede por mucho lo que puede proporcionar un laboratorio o incluso un país completo, y esa potencia cada día es mayor. La computación voluntaria o de ciclos redundantes permite aprovechar mediante Internet toda esta potencia, utilizando las computadoras de todo el mundo como nodos de procesamiento para la resolución de problemas de alto costo computacional.

Actualmente, la cantidad de ordenadores portátiles existentes en la UCI es considerable, por lo que su uso contribuiría a elevar la capacidad de cómputo en la misma. Esto sería posible mediante la utilización de la computación voluntaria ya que no estaría limitado solamente al uso de las computadoras ubicadas en los laboratorios docentes o de producción, sino que todos los interesados podrían donar sus recursos a la ejecución de tareas.

1.3 Aspectos problemáticos de la computación voluntaria

Los sistemas de computación voluntaria deben controlar algunos aspectos problemáticos [3]:

- La heterogeneidad de los sistemas que participan.
- La disponibilidad esporádica de los recursos participantes.
- La necesidad de la no interferencia del ordenador cuando sea el usuario quien realiza el uso.
- El anonimato de los usuarios.

1.4 Análisis de soluciones anteriores

El primer proyecto de computación voluntaria conocido fue GIMPS (acrónico en inglés de Great Internet Mersenne Prime Search) [12], creado en enero de 1996 y encargado de buscar números primos de Mersenne. En años sucesivos surgieron otros proyectos como Superweb [13], Popcorn [14], Charlotte [15] y Bayanihan [16]. El desarrollador de este último proyecto, Luis F.G. Sarmenta, fue quien acuñó el término “computación voluntaria”. En 1999 fueron lanzados al público proyectos como SETI@home [17] y Folding@home [18], recibiendo muy buena acogida por cientos de miles de voluntarios alrededor del mundo.

El *software* cliente de los primeros proyectos de computación voluntaria consistía en un simple programa que combinaba la computación distribuida y la infraestructura del sistema. Esta arquitectura monolítica no era flexible, resultando difícil actualizar las versiones sin modificar la infraestructura. Recientemente se

han desarrollado sistemas *middleware* que proveen una infraestructura de computación distribuida independientemente de la computación científica [3]. Entre ellos se encuentra BOINC (acrónico en inglés de Berkely Open Infrastructure for Network Computing) [19], desarrollado en la universidad de Berkely, siendo el más utilizado. Otros ejemplos son XtremWeb [20] desarrollado por la universidad Paris-Sud, y sus extensiones XtremWeb-CH [21] y XtremWeb-HEP [22], además de T-arenal desarrollado por el departamento de Bioinformática de la UCI.

A continuación se realiza el análisis de cada uno de estos sistemas, especificando sus características principales.

1.4.1 BOINC

BOINC es una plataforma de computación distribuida que utiliza recursos de computación voluntarios. Fue desarrollada originalmente para el proyecto SETI@home, pero actualmente es utilizada por diversos proyectos en campos como la física, la medicina, la biología molecular y la climatología. En la actualidad, es desarrollada por el Laboratorio de Ciencias Espaciales de la Universidad de California en Berkeley, por el equipo que desarrolló y continúa desarrollando el proyecto SETI@home, dirigido por David Anderson, recibiendo contribuciones de desarrolladores de todo el mundo. Su principal objetivo es favorecer la creación de muchos proyectos, por lo que alienta a una gran parte de los propietarios de ordenadores del mundo a participar en uno o más proyectos. Esta plataforma es considerada como un casi-superordenador, disponiendo de aproximadamente 560.000 ordenadores activos en todo el mundo y con un rendimiento medio de 1 PFLOPS[3], superando el superordenador Blue Gene. BOINC ha sido desarrollada para múltiples plataformas como Microsoft Windows, GNU/Linux y diversos sistemas Unix (Solaris, BSD, Mac OS X); es de código abierto y está protegida por la licencia GNU LGPL. Además, está diseñada como una estructura libre que permite a cualquier usuario convertirse en voluntario y participar en la ejecución de tareas de un proyecto [3].

BOINC es utilizada por una amplia variedad de proyectos, que requieren de una gran capacidad de cálculo, por lo que la intención de utilizar esta infraestructura es obtener una enorme capacidad de cómputo utilizando ordenadores personales. Estos proyectos son en su gran mayoría sin ánimo de lucro y suelen estar dirigidos por universidades o entidades públicas. Algunos de estos proyectos son [3]: SETI@home [17], Predictor@home [23], Folding@home [18], Rosetta@home [24], y Einstein@home [25].

[3]PFLOPS: es una medida del rendimiento de una computadora en cuanto a cálculos por segundo, por ejemplo: 1 PFLOP equivale a 1 000 000 000 000 000 cálculos por segundo.

1.4.2 XtremWeb

XtremWeb es un proyecto *Peer-to-Peer*[4] desarrollado en la Universidad de Paris-Sud, Francia. Fue diseñado originalmente para estudiar modelos de ejecución en el ámbito de la computación global, y actualmente, es una plataforma de código abierto que permite construir redes de computadoras recolectando recursos remotos (computadoras personales, estaciones de trabajo, servidores) conectados a Internet. Su objetivo principal es computar aplicaciones distribuidas utilizando el tiempo de inactividad de las máquinas voluntarias [26]. Entre sus principales características está el ser multiusuarios, multiaplicaciones y permitir despliegues a través de distintos dominios [20]. Además, es una plataforma de *software* libre y soporta diversos sistemas operativos como Linux, Windows y Mac OS X.

XtremWeb implementa tres entidades distintas, el coordinador, los trabajadores y los clientes. El coordinador controla todo el proceso de gestión de las tareas (programación de las tareas y almacenamiento de los resultados) y es el único que se encuentra bajo el completo control del administrador de la red XtremWeb. Los clientes son usuarios autorizados a enviar tareas al coordinador en forma de transacciones. Estos envían las tareas al coordinador proporcionando archivos binarios y parámetros opcionales, y permiten al usuario recoger los resultados. Por último, los trabajadores son entidades distribuidas (máquinas voluntarias) encargadas de computar las tareas previstas por el coordinador [26].

1.4.3 XtremWeb-CH

XtremWeb-CH (XW-CH) es un *middleware* de computación voluntaria que facilita la implementación y ejecución de aplicaciones paralelas y distribuidas. XW-CH es una versión mejorada de XtremWeb. A diferencia de XtremWeb, que solo soporta aplicaciones mono-módulo, XW-CH permite la ejecución de aplicaciones compuestas por múltiples módulos comunicados. La comunicación entre los módulos de XW-CH se hace directamente entre los nodos (trabajadores), sin tener que pasar por el servidor, mientras que en XtremWeb la comunicación entre los trabajadores no puede tener lugar sino es mediante el servidor. Esta nueva funcionalidad permite reducir la carga de comunicación al servidor y se acerca aún más al concepto *Peer-to-Peer*, pues todos los nodos tienen las mismas funciones y el mismo poder. XW-CH está compuesto por tres componentes principales: el coordinador, el trabajador, y el almacén. Los trabajos o las solicitudes son enviados mediante páginas web o utilizando diferentes API [27].

El coordinador es el encargado de aceptar las peticiones de ejecución que vienen de los clientes, asignar las tareas a los trabajadores, supervisar la ejecución de cada una de estas, detectar la caída o desconexión del trabajador y relanzar las tareas a cualquier otro trabajador disponible en caso de ser necesario. El trabajador extrae la tarea asignada por el coordinador y luego de computar la tarea devuelve

[4] *Peer-to-Peer*: en una arquitectura *Peer-to-Peer* los nodos actúan simultáneamente como clientes y servidores respecto a los demás nodos de la red.

los resultados. Los almacenes son utilizados por los trabajadores para transferir los datos de entrada necesarios para la ejecución de las tareas. Un nodo almacén actúa como un depósito o servidor de entrada y salida de archivos requeridos por las tareas.

1.4.4 XtremWeb-HEP

XtremWeb for High Energy Physics (XW-HEP) es una plataforma de computación global que apunta a conglomerar recursos computacionales de toda la Internet. Contrario a lo que su nombre podría indicar, este es un *framework* de uso general, es decir, no está específicamente dedicado a la comunidad de física de alta energía (HEP). Es un *software* libre que está bajo la licencia GPL, de código abierto y sin ánimos de lucro. Este proyecto, basado en XtremWeb, es una iniciativa multidisciplinaria, donde el Laboratorio del Acelerador Lineal (LAL) y el Laboratorio de Investigaciones Informáticas (LRI) juntaron sus esfuerzos para realizar una plataforma de investigación, cuyo objetivo es el estudio de sistemas distribuidos a gran escala. XW-HEP extiende el servidor de XtremWeb introduciendo la noción de permisos de acceso a datos (librerías, ejecutables, archivos de entrada, archivos de salida, etc.). Este proyecto está formado por los clientes, los voluntarios y el servidor. Los clientes y los voluntarios son implantados en las computadoras personales de los usuarios esparcidos por toda Internet, por lo que representan la parte distribuida del sistema, mientras que el servidor es la parte centralizada, encargada de la gestión de la plataforma [27].

1.4.5 T-Arenal

T-Arenal es un sistema distribuido que utiliza varios recursos de cómputo como estaciones de trabajo, que pueden ser heterogéneos en cuanto a *hardware* y *software*, para dar solución a grandes problemas que puedan ser descompuestos en tareas más pequeñas e independientes [28].

Esta plataforma fue desarrollada por la línea investigativa de Bioinformática de la UCI a partir del sistema *Java Heterogeneous Distributed System* [29], incorporando el enfoque de varios servidores para aumentar el ancho de banda y la potencia de los recursos computacionales ofrecidos por un solo servidor cuando el número de estaciones de trabajo crece. Este diseño multi-servidor permite organizar y utilizar gran cantidad de puestos de trabajo ubicados en una institución, sin presentar limitaciones en el tamaño del sistema. T-Arenal combina la arquitectura *Peer-to-Peer* y Cliente-Servidor en un solo modelo. Sus estaciones de trabajo están jerárquicamente organizadas como un árbol de tres niveles. La raíz y los nodos internos son los servidores, los cuales colaboran entre ellos para compartir el poder de cómputo a través de una comunicación *Peer-to-Peer*, mientras que los clientes son las hojas y calculan las tareas [30].

Está dividido en tres componentes esenciales: servidor central, servidor de peticiones y cliente.

Servidor central: Su función principal es la de chequear y planificar la asignación de las diferentes tareas a

los servidores de peticiones. También se encarga de controlar información sobre los usuarios que pueden acceder e interactuar con el sistema, así como, almacenar los problemas a partir de los cuales se crearán las diferentes ejecuciones. Además, realiza el seguimiento de todos los eventos que ocurren durante su funcionamiento.

Servidor de peticiones: Es el responsable de solicitar las tareas al servidor central, y de atender y controlar la realización de las mismas. La tarea a ser atendida, será dividida en pequeñas sub-tareas denominadas unidades de trabajo. El principal aspecto del servidor de peticiones es repartir las diferentes unidades de trabajos entre los clientes, siempre y cuando estos estén autorizados a interactuar con el mismo. El servidor de peticiones procesa el resultado de cada una de las sub-tareas generadas, para finalmente construir el resultado de la tarea original.

Cliente: Es el que realiza una solicitud de una unidad de trabajo al servidor de peticiones correspondiente. Realiza el procesamiento de la unidad de trabajo y posteriormente retorna el resultado obtenido, y así sucesivamente.

1.5 Valoración del estado del arte

Después del análisis realizado se puede llegar a la conclusión de que estos sistemas son capaces de resolver problemas complejos, que pueden ser descompuestos en tareas más pequeñas e independientes, reduciendo considerablemente el tiempo de respuesta. Sin embargo, todos ellos requieren de una instalación previa en las máquinas voluntarias que donan sus recursos para la ejecución de las tareas. Esto constituye un inconveniente, debido a que por cuestiones de seguridad informática, la mayoría de los usuarios que acceden a las computadoras de la universidad no tienen privilegios administrativos sobre las mismas, por lo que resulta imposible para ellos la instalación de programas invasivos. A pesar de no poder ser utilizados como solución al problema de investigación, puesto que no se adecuan completamente a la situación problémica existente, muchos de sus elementos serán utilizados para la implementación de la solución.

1.6 Metodologías de desarrollo de *software*

“Una metodología es una colección de procedimientos, técnicas, herramientas y documentos auxiliares que ayudan a los desarrolladores de *software* en sus esfuerzos por implementar nuevos sistemas de información. Una metodología está formada por fases, cada una de las cuales se puede dividir en sub-fases, que guiarán a los desarrolladores de sistemas a elegir las técnicas más apropiadas en cada momento del proyecto y también a planificarlo, gestionarlo, controlarlo y evaluarlo” [31].

Las metodologías de desarrollo de *software* tienen como objetivo estructurar, planificar y controlar dicho proceso, por lo que indican paso a paso todas las actividades que se deben efectuar para alcanzar el producto deseado, indicando además quién debe participar en el desarrollo de cada una de las

actividades y qué papel deben tener en las mismas [32]. Además, detallan la información que se debe producir como resultado de una actividad y la información necesaria para comenzarla.

Existe una gran variedad de metodologías para la creación del *software*, las cuales se pueden clasificar en dos grandes grupos [33]:

- Las denominadas metodologías tradicionales o pesadas, orientadas al control de los procesos, estableciendo rigurosamente las actividades a desarrollar, herramientas a utilizar y notaciones que se usarán.
- Las llamadas metodologías ágiles o ligeras, orientadas a la interacción con el cliente y al desarrollo incremental del *software*, mostrando versiones parcialmente funcionales del *software* al cliente en intervalos cortos de tiempo.

1.6.1 Metodologías tradicionales

Las metodologías tradicionales tienen como objetivo alcanzar un *software* más eficiente y predecible, por lo que destacan la importancia de llevar a cabo la planificación detallada a largo plazo de todo el trabajo, y una vez que se tiene todo detallado, comienza el ciclo de desarrollo del producto *software* [34].

Resultan de gran utilidad en proyectos de gran tamaño (respecto a tiempo y recursos), donde se requiere de un alto grado de organización y un numeroso equipo de desarrollo, además, generan una gran cantidad de documentos, resultando inconveniente su utilización en proyectos sencillos, con pequeños equipos de desarrollo. Estas metodologías, no se adaptan fácilmente a los cambios, por lo que no son adecuadas cuando se trabaja en un entorno donde los requisitos pueden ser cambiantes, y en donde se exige reducir drásticamente los tiempos de desarrollo pero manteniendo una alta calidad [35].

1.6.2 Metodologías ágiles

Las metodologías ágiles surgen como alternativa a los métodos muy estructurados, estrictos y lentos, extraídos del modelo de desarrollo en cascada. Tienen sus inicios en 1999 cuando Kent Beck crea la metodología eXtreme Programming (XP) iniciando el movimiento de metodologías ágiles al que se unirían muchas otras metodologías surgidas anteriormente. Estas eran llamadas inicialmente “metodologías livianas”. Es en febrero de 2001, luego de una reunión efectuada en Utah-EEUU, en la cual participaron 17 expertos de la industria del *software*, que nace el término “ágil” vinculado al desarrollo de *software*. Tras la reunión se crea La Alianza Ágil (en inglés *The Agile Alliance*), organización dedicada a promover los conceptos asociados al desarrollo ágil de *software* y contribuir a la adopción de estos conceptos por parte de las empresas, teniendo como punto de partida el Manifiesto Ágil, documento que resume la filosofía ágil [35].

1.6.3 Metodologías Ágiles versus Metodologías Tradicionales

Con el objetivo de seleccionar la metodología adecuada para guiar el proceso de desarrollo del presente trabajo se realizó una comparación entre las metodologías ágiles y las metodologías tradicionales, recogiendo esquemáticamente sus principales diferencias. (Ver Anexo I)

Debido a todo lo antes planteado y teniendo en cuenta que se dispone de poco tiempo para la implementación y el equipo de trabajo es pequeño, contando con tan solo dos integrantes, se decide la utilización de una metodología de desarrollo de *software* ágil.

1.6.4 Análisis de las principales metodologías ágiles

Las metodologías ágiles más destacadas hasta el momento en el proceso de desarrollo de *software* son XP (eXtreme Programming), Scrum y Crystal Clear [35]. A continuación se describen cada una de ellas.

1.6.4.1 *eXtreme Programming* o XP

XP es la metodología que le dio conciencia al movimiento actual de metodologías ágiles. Fue creada por Kent Beck, autor del primer libro sobre la materia, titulado *Extreme Programming Explained: Embrace Change* (1999) [35].

XP está centrada en potenciar las relaciones interpersonales como clave para el éxito en el desarrollo de *software*, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. Está basada en la retroalimentación continua entre el cliente y el equipo de desarrollo, la comunicación fluida entre todos los participantes, la simplicidad en las soluciones implementadas y el coraje para enfrentar los cambios. Es especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, donde existe un alto riesgo técnico y el equipo de desarrollo es pequeño o mediano. Es una metodología altamente orientada a la implementación y genera poca documentación. Su ciclo de vida consiste de seis fases: Exploración, Planificación de la Entrega (Release), Iteraciones, Producción, Mantenimiento y Muerte del proyecto. En todas las interacciones de este ciclo tanto el cliente como el programador aprenden, existiendo una retroalimentación entre ellos. Esta metodología utiliza las historias de usuario para especificar los requisitos del *software*, y el tratamiento de las mismas es muy dinámico y flexible. Además, cuenta con pocos roles entre los que se destacan el programador y el cliente [36].

La metodología XP es la metodología ágil de más renombre en la actualidad y se diferencia de las demás metodologías de este tipo en el alto nivel de disciplina de las personas que participan en el proyecto [35].

1.6.4.2 Scrum

Scrum surge en 1986, de un artículo de la Harvard Business Review titulado “The New Product Development Game”, que introducía las mejores prácticas más utilizadas en 10 compañías japonesas

altamente innovadoras. A partir de ahí Ken Scwaber y Jeff Sutherland formalizan el proceso conocido como Scrum en el año 1995.

Scrum es un método iterativo e incremental que define el desarrollo de *software* como un proceso de naturaleza caótica, inseguro y complejo, por lo que está especialmente indicado para proyectos que trabajan con requisitos inestables. Su intención es maximizar la retroalimentación sobre el desarrollo, pudiendo corregir problemas y mitigar riesgos de forma temprana. En Scrum el desarrollo de *software* se divide en tres fases: Planificación, Iteraciones (*Sprints*) y Cierre. La primera y la última fase están bien definidas, sin embargo el *Sprint* es un proceso empírico, pues varias de las tareas en esta fase no pueden ser identificadas ni controladas. Durante la Planificación se define el "*Product Backlog"*, que contiene todos los requerimientos funcionales y no funcionales que deberá satisfacer el sistema a construir, y a partir de ahí se definirán las iteraciones o *Sprint*, que tienen una duración recomendada de un mes. Cada *Sprint* es una iteración del producto final, y en cada iteración se agrega más funcionalidad al producto, siendo el resultado de cada *Sprint* un incremento ejecutable que se muestra al cliente. Dentro de cada *Sprint* el Scrum Master (equivalente al Líder de Proyecto) llevará a cabo la gestión de la iteración, convocando diariamente al "*Scrum Daily*", una reunión que dura no más de 15 minutos, con el propósito de lograr la retroalimentación sobre las tareas realizadas, los obstáculos que se presentan y las tareas a desarrollar.

Resulta muy habitual que Scrum se complemente con XP; en estos casos, Scrum suministra un marco de administración basado en patrones organizacionales, mientras XP constituye la práctica de programación, usualmente orientada a objetos y con fuerte uso de patrones de diseño [35].

1.6.4.3 Crystal Clear

La familia de metodologías Crystal, desarrollada por el investigador de IBM el Dr. Alistair Cockburn, es un conjunto de diferentes metodologías que pueden ser seleccionadas en función de las características del proyecto que se está desarrollando [35].

Crystal Clear es la menor de esta familia de metodologías, y está diseñada para ser utilizada por equipos de hasta ocho integrantes en el desarrollo de sistemas cuyos posibles errores puedan causar una pérdida prudencial de dinero o de confort. Esta metodología se centra en tres propiedades claves, que constituyen su núcleo: efectuar entregas frecuentes, que consiste en liberar código ejecutable y testeado a usuarios reales cada pocas semanas o meses; realizar una mejora reflexiva consistente en tomarse un pequeño tiempo (unas pocas horas cada semana o una vez al mes) para pensar, reflexionar y discutir bien qué se está haciendo; y mantener una comunicación osmótica mediante la ubicación física del equipo de trabajo en un mismo local, permitiendo que la información esté "flotando" en el ambiente, así el equipo puede obtenerla con mayor facilidad. Otras propiedades importantes que pueden ser añadidas para incrementar la seguridad del proyecto son: crear seguridad personal entre los miembros del equipo, mantener el foco

en la tarea, tener acceso fácil a usuarios expertos y trabajar en un ambiente técnico con pruebas automáticas, administración de configuración e integración frecuente [37].

Los métodos Crystal no prescriben las prácticas de desarrollo, las herramientas o los productos que pueden usarse, pudiendo combinarse con otros métodos como Scrum y XP. Según el propio Cockburn, Crystal Clear debería proporcionar un método ligero, sin embargo, comparado con XP resulto muy pesado. A pesar de ser más fácil de aprender e implementar, XP es más disciplinado, por lo que si un equipo ligero no puede tolerar sus rigores, lo mejor será que se mude a XP [35].

1.6.5 Fundamentación de la selección de la metodología de desarrollo de *software*

Después del análisis realizado a las principales metodologías ágiles de desarrollo de *software*, se decide la utilización de XP. El principal motivo para esta selección es que XP es un proceso altamente orientado a la implementación, genera poca documentación y fomenta la programación en parejas. Otro elemento a destacar es que el CISED cuenta con varios profesionales con experiencia y resultados positivos en el uso de esta metodología, sirviendo como apoyo y fuente de consulta.

1.7 Herramientas, lenguajes y tecnologías

A continuación se describen las herramientas, lenguajes y tecnologías usadas para dar solución al problema planteado teniendo en cuenta las necesidades existentes y el entorno donde se desplegará el sistema.

1.7.1 Lenguaje de modelado

El modelado constituye un mecanismo para facilitar la comprensión de las soluciones informáticas que cada vez tienen un nivel de complejidad más elevado. Los lenguajes de modelado permiten trabajar con sistemas mayores y más complejos, facilitando el proceso de codificación e implementación del sistema de forma distribuida y en distintas plataformas [38].

1.7.1.1 BPMN

BPMN (acrónico en inglés de *Business Process Modeling Notation*) tiene como principal objetivo proporcionar una forma de notación gráfica para expresar los procesos de negocio en un diagrama de procesos de negocio que sea fácilmente comprensible tanto para los usuarios, analistas y los desarrolladores encargados de la aplicación de la tecnología que llevará a cabo esos procesos, como para los trabajadores del negocio que van a administrar y supervisar esos procesos. Por lo tanto, BPMN crea un estándar entre el diseño del proceso de negocio y el proceso de implementación [39].

1.7.1.2 UML

UML (acrónico en inglés de *Unified Modeling Language*) es el lenguaje de modelado de sistemas de *software* más conocido y utilizado en la actualidad. Permite la modelación de sistemas con tecnología orientada a objetos. Es un lenguaje gráfico para visualizar, especificar, construir y documentar los artefactos de un sistema con gran cantidad de *software*. UML proporciona una forma estándar de escribir los planos de un sistema, cubriendo los aspectos conceptuales, tales como procesos del negocio y funciones del sistema, y los aspectos concretos, tales como las clases escritas en un lenguaje de programación específico, esquemas de bases de datos, componentes y *software* reutilizable [40].

1.7.1.3 Fundamentación de la selección del lenguaje de modelado

En el presente trabajo no será necesario realizar la modelación de procesos del negocio por lo que se descarta la utilización de BPMN. La plataforma que se desea desarrollar será implementada con tecnología orientada a objetos, razón principal por lo que se decide utilizar UML como lenguaje de modelado, además, es el lenguaje de modelado de sistemas de *software* más conocido y utilizado en la actualidad.

1.7.2 Herramienta CASE

Las herramientas de Ingeniería de Software Asistida por Computadora (CASE, acrónico en inglés de *Computer Aided Software Engineering*) son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de *software* reduciendo el costo de las mismas en términos de tiempo y de dinero. Estas herramientas pueden ayudar en todos los aspectos del ciclo de vida de desarrollo del *software* en tareas como el proceso de realizar un diseño del proyecto, cálculo de costos, implementación de parte del código automáticamente con el diseño dado, documentación o detección de errores [41].

1.7.2.1 Rational Rose Enterprise

Rational Rose Enterprise proporciona un lenguaje de modelado común para simplificar el entorno de trabajo y permitir una creación más rápida de *software* de calidad. Soporta la notación UML y permite la generación de código Ada, ANSI C++, C++, CORBA, Java y Visual Basic. Ha sido desarrolla para la familia de sistemas operativos de Windows [42].

1.7.2.2 Altova UModel

Altova UModel diseña visualmente modelos de aplicaciones en UML, permite realizar ingeniería inversa y generación de código en lenguaje Java, C#, o Visual Basic .NET. La línea de productos de Altova requiere el uso de Windows XP, Windows Vista, Windows 7 o Windows Server 2003-2008 [43].

1.7.2.3 Visual Paradigm for UML

Visual Paradigm for UML es una herramienta ampliamente utilizada en el mundo del *software* que permite a los profesionales modelar sus diseños. Esta herramienta fue desarrollada para una amplia gama de usuarios, incluyendo ingenieros de *software*, analistas de sistemas, analistas del negocio y arquitectos de sistemas. Es un *software* privativo, pero brinda una versión libre para uso no comercial. Proporciona un diseño centrado en casos de uso y enfocado al negocio que genera un *software* de mayor calidad. Es capaz de generar código e ingeniería inversa para varios lenguajes de programación. Permite manejar grandes estructuras de manera eficiente. Soporta el ciclo de vida completo del desarrollo de *software*: análisis y diseño orientado a objetos, construcción, pruebas y despliegues. Visual Paradigm for UML ha sido desarrolla para todos los sistemas operativos compatibles con Java, incluyendo Windows, Linux y Mac OS X [44].

1.7.2.4 Fundamentación de la selección del lenguaje de modelado

Tanto Rational Rose Enterprise como Altova UModel tienen como inconveniente que solo son compatibles con la familia de sistemas operativos de Windows por lo que se descarta su utilización como herramienta de modelado. Por el contrario, Visual Paradigm for UML ha sido desarrolla para varios sistemas operativos como Windows, Linux y Mac OS X por lo que se decide su utilización. Además, el equipo de desarrollo cuenta con experiencia suficiente en el trabajo con la misma, lo que evita la necesidad de emplear tiempo en el estudio de otra herramienta de este tipo.

1.7.3 Lenguajes de programación

Un lenguaje de programación es un idioma diseñado para expresar procesos que pueden ser llevados a cabo por máquinas como computadoras. Pueden usarse para crear programas que controlen el comportamiento físico y lógico de una máquina o para expresar algoritmos con precisión. En la implementación de un sistema informático la selección y utilización de uno o varios lenguajes de programación es un elemento fundamental, pues es, a través de estos lenguajes, que el hombre se comunica con la computadora, transmitiéndole lo que se desea que realice.

Para la implementación de la solución se decide utilizar el lenguaje Java, para la programación del servidor. La razón principal de esta selección es que es independiente de la plataforma y la arquitectura de red, lo que garantiza que pueda ejecutarse en cualquier SO. Además, Java proporciona una colección de clases para su uso en aplicaciones de red, que facilitan la creación de aplicaciones distribuidas [45]. Para la programación del trabajador se decide utilizar el lenguaje JavaScript principalmente porque es interpretado de manera nativa por todos los navegadores web modernos, y permite realizar operaciones y cálculos matemáticos [46]; además, se utilizará HTML para incluir código JavaScript en el navegador web.

1.7.3.1 Java

Java constituye un lenguaje “simple, orientado a objetos, distribuido, interpretado, robusto, seguro, de arquitectura neutra, portable, de altas prestaciones, multitarea y dinámico” [47]. Una de las ventajas de este lenguaje es que es independiente de la plataforma y de la arquitectura de red, por lo que una aplicación escrita en Java puede ejecutarse en cualquier SO [48]. Java proporciona una colección de clases para su uso en aplicaciones de red, que permiten abrir sockets, y establecer y aceptar conexiones con servidores o clientes remotos, facilitando así la creación de aplicaciones distribuidas [45].

1.7.3.2 JavaScript

JavaScript es un lenguaje de programación interpretado, que se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico. Permite crear funciones con variables y objetos, y realizar operaciones y cálculos matemáticos. Una de las principales ventajas de este lenguaje es que es interpretado por todos los navegadores modernos [46].

1.7.3.3 HTML

HTML (acrónimo en inglés de *Hypertext Markup Language*) es el lenguaje predominante para la construcción de páginas web. Es usado para describir la estructura y el contenido que contendrá la página web. HTML basa su sintaxis en un elemento de base denominado etiqueta, a través de estas se van describiendo los elementos del documento, como enlaces, párrafos e imágenes. Puede incluir un *script* (por ejemplo *JavaScript*), el cual puede afectar el comportamiento de navegadores web [49].

1.7.4 Entorno de Desarrollo Integrado (IDE)

Un IDE (acrónico en inglés de *Integrated Development Environment*) es un ambiente de programación donde confluyen un editor de código, un compilador, un depurador y un constructor de interfaz gráfica de usuario, ofreciéndole al desarrollador un marco amigable para la mayoría de los lenguajes de programación.

Existen diferentes IDEs basados en Java entre los que se encuentran Eclipse y NetBeans, seleccionándose este último para el desarrollo de la solución ya que facilita la creación de interfaces gráficas de usuario para aplicaciones web y de escritorio, permitiendo ahorrar tiempo de desarrollo.

1.7.4.1 NetBeans

NetBeans es un IDE creado principalmente para el lenguaje de programación Java, aunque puede servir para otros lenguajes de programación. Esta herramienta ayuda a los programadores a escribir, compilar, depurar y ejecutar programas. NetBeans contiene todos los módulos necesarios para el desarrollo de aplicaciones Java en una sola descarga, facilitándole al usuario comenzar a trabajar inmediatamente.

Además, ofrece servicios comunes permitiéndole al desarrollador enfocarse en la lógica específica de su aplicación. Soporta el desarrollo de todos los tipos de aplicación Java (J2SE, web, EJB y aplicaciones móviles), y es un producto libre y gratuito sin restricciones de uso [50].

1.7.5 Sistema de mensajería o *Middleware* Orientado a Mensajes

Las capacidades de mensajería son típicamente provistas por sistemas de *software* denominados sistema de mensajería o *Middleware* Orientado a Mensajes (MOM, acrónico en inglés de *Message Oriented Middleware*). Los MOMs son componentes especializados en el manejo de mensajes. Su principal cometido es participar como intermediario entre la comunicación de aplicaciones distribuidas, logrando desacoplarlas. Los MOMs establecen una comunicación asíncrona entre el emisor y el receptor, por lo que en ningún momento están directamente conectados. El emisor envía el mensaje y no se queda a la espera de recibir confirmación de recepción de su mensaje, sino que sigue trabajando normalmente, siendo el MOM quien se encarga de que todos los mensajes lleguen siempre a su destino. No obstante, también hay soluciones que trabajan con conexiones síncronas o pseudo-síncronas. Por lo general los sistemas de mensajería soportan alguno de estos modelos de comunicación [51]:

- Punto-a-Punto: este modelo es basado en canales punto a punto. Cuando una aplicación produce un mensaje, lo coloca en un canal de este tipo y un receptor recibirá el mensaje. A este tipo de canales se pueden suscribir varios interesados en recibir mensajes, pero solo uno obtendrá cada mensaje. El sistema de mensajería se encargara de determinar cuál de los destinatarios subscritos obtendrá el mensaje.
- Publicación-Suscripción: este modelo permite que una aplicación pueda enviar mensajes a varios destinatarios simultáneamente. Para esto, las aplicaciones colocan los mensajes en un canal que corresponde a cierto tópico definido, el cual tiene el siguiente comportamiento: los interesados en recibir mensajes del tópico se suscriben al mismo, el emisor coloca el mensaje en el canal, y una copia del mismo será entregada a cada uno de los subscriptores.

1.7.5.1 ActiveMQ

ActiveMQ es un sistema de mensajería basado en Java que se ajusta a la API JSM (acrónico en inglés de *Java Message Service*). Es uno de los muchos proyectos ofrecidos por la Fundación de *Software* Apache (Apache). Soporta varios clientes entre ellos C, C++, .NET, C#, Erlang y Perl. No solo soporta el protocolo STOMP, sino que también proporciona su propio protocolo de interoperabilidad llamado *OpenWire*, el cual es un protocolo binario encaminado a lograr un mayor rendimiento. Este protocolo tiene clientes en Java, C, C++ y C#, pero no resuelve todos los problemas relacionados con la interoperabilidad. ActiveMQ soporta ambos modelos de mensajería, punto a punto y publicación-suscripción. Presenta módulos para establecer la comunicación a través de una serie de protocolos de transporte tales como TCP, SSL, NIO,

UDP, Multicast, JGroups y JXTA. Este sistema de mensajería no presenta la capacidad de realizar enrutamiento, pero puede ser añadida mediante la utilización de otro proyecto Apache llamado Camel [52].

1.7.5.2 ZeroMQ

ZeroMQ (también conocido como 0MQ o ZMQ) es una librería de mensajería desarrollada por iMatix Corporation junto a una gran comunidad de colaboradores. En un principio se presentó como "*middleware* de mensajería" y en este momento se define como una "nueva capa en la pila de red". ZeroMQ no es un sistema completo, sino que es una biblioteca simple de usar mediante programación. Básicamente ofrece una interfaz *socket* que permite construir rápidamente un sistema de mensajería. Puede hacer uso de los protocolos de transporte IPC y Multicast, y de la dosificación inteligente de mensajes, lo que le permite utilizar eficientemente una conexión TCP/IP reduciendo al mínimo la sobrecarga de protocolo y las llamadas al sistema [53].

1.7.5.3 RabbitMQ

RabbitMQ es un *software* de negociación de mensajes, que entra dentro de la categoría de *middleware* de mensajería, desarrollado y mantenido por Rabbit Technology Ltd. Implementa el estándar AMQP lo que proporciona que tenga un esquema de enrutamiento flexible. El servidor RabbitMQ está escrito en Erlang y utiliza el *framework* OTP para construir sus capacidades de ejecución distribuida y conmutación ante errores. RabbitMQ es compatible con varios clientes, siendo tres de ellos considerados como oficiales, estos son Java, .NET/C# y Erlang. Sin embargo, debido al apoyo sustancial de la comunidad hay una serie de otros clientes en Python, Perl, Ruby entre otros. Ofrece varios *plugins* para ampliar su funcionalidad entre los que se encuentra un *plugin* para el protocolo STOMP. Posee soporte para SSL y asume una conexión TCP. Al igual que ActiveMQ soporta ambos modelos de mensajería Punto-a-Punto y Publicación-Suscripción. Es un *software* de código abierto y está liberado bajo la licencia MPL. Está basado en una plataforma comprobada, ofreciendo alta confiabilidad, disponibilidad y escalabilidad, junto con un rendimiento y latencia predecibles y consistentes. Tiene un código fuente compacto y fácil de mantener, que permite una rápida adaptabilidad [52]. Además, ha sido utilizado por grandes organizaciones en algunos de sus proyectos, como por ejemplo: VMware hace un uso extensivo de RabbitMQ en sus productos de virtualización y servicios de nube; Google lo utiliza en su proyecto de código abierto Rocksteady; también es ampliamente utilizado en el proveedor de búsqueda local AT&T Interactive; y OpenStack, que es una iniciativa de código abierto que proporciona un sistema operativo en nube masivamente escalable, usa RabbitMQ para la mensajería [54].

1.7.5.4 Fundamentación de la selección del sistema de mensajería

ZeroMQ es un sistema de mensajería muy ligero, especialmente diseñado para escenarios de alto rendimiento y baja latencia como la que se puede encontrar en el mundo financiero por lo que se descarta su utilización en el desarrollo de la solución. Por el contrario, ActiveMQ es una tecnología madura que ha existido durante muchos años. Es altamente configurable y existen una serie de otros proyectos ofrecidos por la Fundación de *Software* Apache que pueden mejorar sus capacidades. El sistema de mensajería RabbitMQ, por otra parte, está destinado a adherirse al protocolo AMQP en lugar de la API de JMS, como lo hace ActiveMQ, y se ha creado mucho más tarde que este. Ambos están bien establecidos en el mercado, tienen un excelente soporte, son fiables, proporcionan un alto rendimiento, son de código abierto y cuentan con el apoyo de toda una comunidad, por lo que ambos podrían ser usados como sistema de mensajería en la solución que se desea desarrollar. Sin embargo, se decide la utilización de RabbitMQ debido a que varios profesores del CISED tienen experiencia en el trabajado con este *middleware* de mensajería, lo que resulta de gran ayuda porque pueden servir de apoyo y fuente de consulta. Además, RabbitMQ cuenta con un alto prestigio ya que ha sido utilizado por proyectos de gran importancia.

1.7.6 Navegador web

Un navegador web es una aplicación encargada de interpretar la información de archivos y sitios web. Su funcionalidad básica es permitir la visualización de documentos de texto, que posiblemente contengan recursos multimedia incrustados.

Actualmente existe una gran cantidad de navegadores en el mundo, pero según StatCounter, un sitio dedicado al estudio del comportamiento del uso de navegadores para el acceso a internet, los preferidos por los usuarios son: Google Chrome, Internet Explorer y Mozilla Firefox [55].

En el caso de Cuba, el sitio StatCounter arrojó los siguientes resultados (Ver Figura 1.1):

- Mozilla Firefox: 72.72%
- Internet Explorer: 14.64%
- Google Chrome: 8.86 %

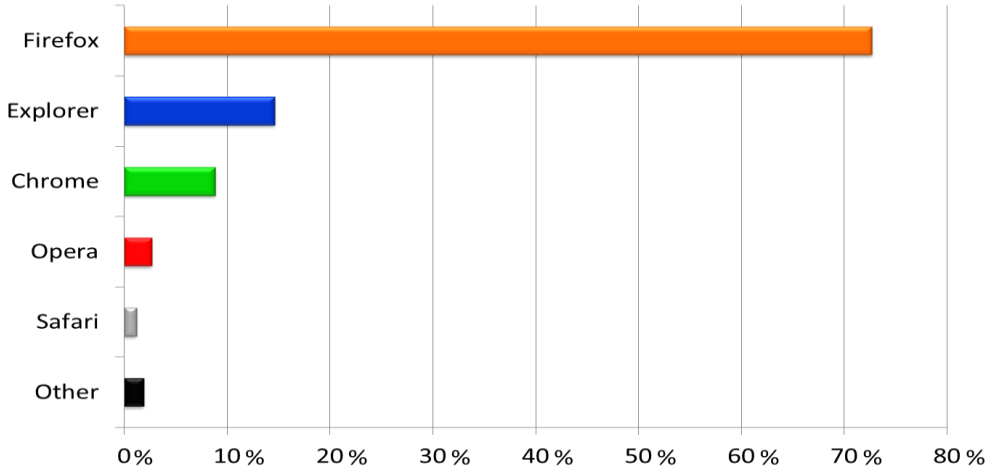


Figura 1.1: Uso de navegadores en Cuba desde abril del 2012 hasta abril del 2013.

Como se puede observar, en nuestro país el uso del Mozilla Firefox es el más extendido, además, es un navegador web multiplataforma por lo que se decide su utilización en el sistema que se desea desarrollar.

1.7.6.1 Mozilla Firefox

Mozilla Firefox es un navegador web libre y de código abierto descendiente de *Mozilla Application Suite* y desarrollado por la Fundación Mozilla. Su código fuente está publicado bajo una triple licencia GNU GPL, GNU LGPL o MPL [56]. Es compatible con varios lenguajes web, incluyendo HTML, XML, XHTML, CSS 1, 2 y 3, JavaScript y DOM, y sus versión 21.0 incluye HTML5. Está disponible para varios sistemas operativos como Microsoft Windows, GNU/Linux y Mac OS X [57].

1.7.7 Herramienta para la creación de *plugins*

Un *plugin* es un *software* que gestiona contenido que no está diseñado para ser procesado por el navegador. Esto incluye normalmente formatos de archivos como: vídeos, audio, juegos, presentaciones y otros. Existen dos herramientas para el desarrollo de *plugins* para el navegador web Mozilla Firefox: el Add-on Builder y el Add-on SDK [58].

1.7.7.1 Add-on Builder

Add-on Builder es un entorno de desarrollo basado en la web que permite a los desarrolladores crear complementos para Firefox usando HTML, CSS y JavaScript. Proporciona un editor de código robusto, herramientas para la creación de *plugins* y herramientas de prueba [58].

1.7.7.2 Add-on SDK

Add-on SDK es un *software* utilizado para la implementación de complementos de Firefox empleando tecnologías web estándares: JavaScript, HTML y CSS. El SDK incluye las APIs de JavaScript que pueden ser utilizadas para crear complementos, además de herramientas para crear, ejecutar, probar y

empaquetar complementos [59]. El Add-on SDK es una alternativa descargable para trabajar a nivel local [58].

1.7.7.3 Fundamentación de la selección de la herramienta para la creación de *plugins*

Tanto Add-on Builder como Add-on SDK tienen la ventaja de que ninguno de los complementos creados con ellas necesita reiniciar Firefox para instalarse, lo que significa que los usuarios no tienen que interrumpir su navegación para comenzar a utilizar el *plugins* de inmediato. Además, con la utilización de estas herramientas la creación de complementos para Firefox es más rápida y sencilla [58]. El Add-on Builder es un entorno de generación en línea, es decir, la programación de los complementos se realiza desde Internet, lo que constituye un inconveniente debido a que para el equipo de desarrollo el acceso a internet limitado, por lo que se descarta su utilización. Se decide utilizar la herramienta Add-on SDK ya que, a diferencia de Add-on Builder, es un *software* de escritorio.

1.7.8 Servidor web

Un servidor web es un programa que tiene la función de satisfacer las peticiones de clientes que recibe mediante el protocolo HTTP [60].

1.7.8.1 Apache

El servidor Apache es un servidor web de código abierto para uso comercial desarrollado por la *Apache Software Foundation* que da la posibilidad de construir sistemas confiables a individuos e instituciones [61]. Apache está diseñado para ser un servidor web potente y flexible que pueda funcionar en la más amplia variedad de plataformas y entornos. Tiene un diseño modular que permite a los administradores de sitios web elegir qué características van a ser incluidas en el servidor seleccionando qué módulos se van a cargar, ya sea al compilar o al ejecutar el servidor [62]. Apache ha sido el servidor web más popular en Internet desde abril de 1996 [61].

1.7.8.2 IIS

IIS (acrónico en inglés de *Internet Information Server*) es una serie de servicios para los ordenadores que funcionan con Windows [63]. Contiene varios componentes con diferentes responsabilidades, como la escucha de las peticiones realizadas al servidor, la gestión de procesos, y la lectura de los archivos de configuración. Estos componentes incluyen los servicios de publicación en Internet y los servicios de activación de los procesos de Windows [64].

1.7.8.3 Fundamentación de la selección del servidor web

Se decide utilizar el servidor web Apache por ser multiplataforma, a diferencia de su homólogo el IIS que solo es desarrollado para Windows. Apache es una tecnología gratuita de código fuente abierto, esto le da

una transparencia a este *software* de manera que si se desea ver qué es lo que se está instalando como servidor, se puede conocer, sin ningún secreto, sin ninguna puerta trasera; y por último es altamente configurable de diseño modular por lo que es muy sencillo ampliar sus capacidades.

1.8 Conclusiones parciales

En este capítulo de realizó un estudio de los principales conceptos relacionados con la computación distribuida, a partir del cual se concluye que la computación voluntaria constituye una fuerte alternativa para resolver problemas que requieren una eleva demanda de cómputo, ya que logra reunir gran capacidad de procesamiento mediante la utilización de ordenadores personales distribuidos por todo el mundo.

Además, se realizó un análisis de los principales sistemas de procesamiento distribuido enfocados a la computación voluntaria, que existen tanto a nivel nacional como internacional, analizando las características, funcionalidades, ventajas y desventajas de cada uno de ellos. Basado en dicho análisis, se pudo concluir que existen sistemas capaces de realizar las funcionalidades deseadas pero requieren de la instalación previa de un *software* en las computadoras que donan sus recursos a la ejecución de problemas, por lo que su utilización no resulta factible, haciéndose necesario desarrollar una plataforma de procesamiento distribuido no invasiva que facilite la resolución de problemas complejos en el CISED.

También se realizó un estudio de las herramientas y tecnologías relacionadas con el desarrollo de sistemas distribuidos, concluyendo que la metodología de desarrollo de *software* que guiará el proceso de desarrollo será XP, utilizándose como lenguaje de modelado UML y como herramienta CASE para el modelado del sistema Visual Paradigm for UML. Además, se utilizarán como lenguajes de programación para la implementación del trabajador JavaScript y HTML, y para el servidor Java, utilizándose como IDE NetBeans y como herramienta para la creación de complementos Add-on SDK. También se decidió utilizar como *middleware* de mensajería RabbitMQ, como navegador web Mozilla Firefox y como servidor web Apache.

Capítulo II: Diseño de la solución

2.1 Introducción

En el presente capítulo se realiza una descripción de la propuesta de solución y se muestran las principales características que tendrá el sistema. Se exponen los requisitos funcionales y no funcionales identificados, especificándose los funcionales mediante las historias de usuario, tal como propone XP. Además, se describe la arquitectura que tendrá el sistema, haciendo énfasis en los estilos arquitectónicos utilizados, así como en los patrones de diseño más importantes.

2.2 Modelo de dominio

Teniendo en cuenta que el problema de investigación que la plataforma pretende resolver no está determinado a partir de procesos bien definidos que permitan modelar el funcionamiento de la misma, se ha procedido a crear un modelo de domino. (Ver Figura 2.1)

El modelo de dominio también conocido como Modelo Conceptual, es una representación visual de los conceptos u objetos del mundo real que son significativos para un problema o área de interés [65].

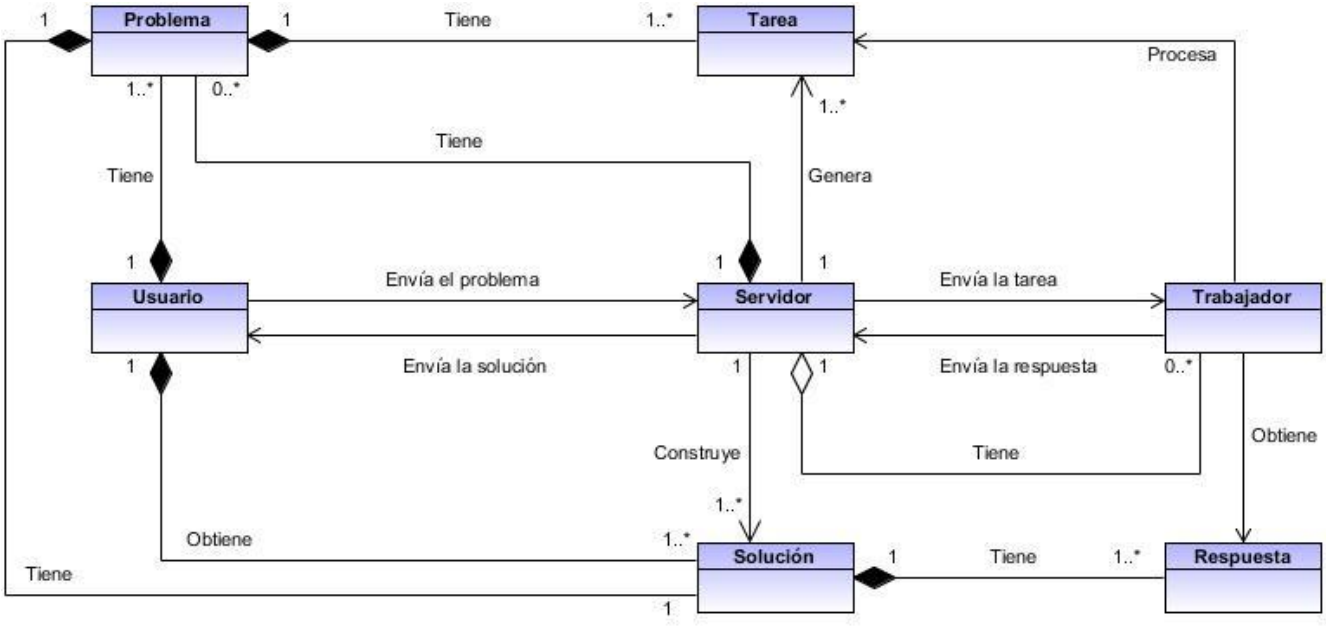


Figura 2.1: Modelo de dominio.

2.2.1 Glosario de conceptos del modelo de dominio

Con el objetivo de lograr una mejor comprensión, se describen a continuación cada uno de los conceptos que intervienen en el modelo de dominio mostrado:

Usuario: representa la persona que interactúa con el sistema para resolver un problema.

Problema: representa la tarea de alto costo computacional que posee un usuario y que se va a resolver de manera distribuida.

Servidor: representa la entidad encargada de dividir el problema en pequeñas tareas, asignar estas tareas a los trabajadores, construir la solución final del problema a partir de las respuestas enviadas por los trabajadores y devolverla al usuario.

Tarea: representa una de las pequeñas partes en que es dividido el problema.

Trabajador: representa la entidad encargada de procesar la tarea que le fue asignada y obtiene una respuesta correspondiente a la misma, la cual retorna al servidor.

Respuesta: representa la entidad obtenida a partir del procesamiento de una tarea realizado por un trabajador y es utilizada por el servidor para conformar la solución.

Solución: es la entidad final que se le devuelve al usuario y que es conformada por el servidor a partir de las respuestas retornadas por cada trabajador.

2.3 Propuesta de solución

En la UCI, como en el resto del mundo, la mayoría de los servicios son brindados mediante aplicaciones web, a las cuales se accede a través de navegadores, por lo que la frecuencia con que se utilizan estos programas es relativamente alta.

Para dar solución al problema de investigación, teniendo en cuenta que los navegadores modernos son capaces de ejecutar de forma nativa código JavaScript y que para incluirles funcionalidades adicionales a través de *plugins* (o extensiones) no es necesario ningún permiso especial por parte de los usuarios, se propone el desarrollo de una plataforma de procesamiento distribuido utilizando un *plugin* en el navegador como *software* para los trabajadores del sistema distribuido.

2.3.1 Descripción de la propuesta de solución

La plataforma de procesamiento distribuido propuesta estará compuesta por tres componentes fundamentales:

Servidor: es el encargado de dividir un problema en múltiples tareas más pequeñas, que puedan ser ejecutadas en los trabajadores, y conformar la solución final del problema a partir de la unión de las respuestas a las tareas. Además, tiene la responsabilidad de situar las tareas en la cola de tareas del *middleware* de mensajería. Es una aplicación de escritorio que proporciona una interfaz gráfica para el usuario del sistema, haciendo fácil y sencilla la interacción con el mismo.

***Middleware* de mensajería**: puede estar situado en el servidor o en otra computadora conectada a este. Es el encargado de asignar las tareas a los trabajadores y enviar la respuesta de cada tarea realizada al servidor, garantizando que si un trabajador no puede por alguna razón realizar la tarea asignada, esta sea colocada nuevamente en la cola de tareas.

Plugin: es el encargado de procesar a través del navegador las tareas que le son asignadas. Además, envía de vuelta la respuesta situándola en la cola de respuestas del *middleware* de mensajería.
El sistema también contará con un servidor web donde se publican las librerías que sean necesarias para la resolución de las tareas. Los datos pertenecientes al problema son guardados de forma persistente en ficheros, así como las salvas de seguridad realizadas por la aplicación para evitar la pérdida de información.

2.3.1.1 Problema

En la plataforma de procesamiento distribuido propuesta, el problema consiste en una clase Java que implementa una interfaz en la cual se definen un conjunto de funcionalidades que deben ser implementadas por el usuario. Estas funcionalidades detallan el problema, ofreciendo una descripción de sus parámetros y de cada uno de los tipos de tareas en que se debe dividir, y definen cómo este será dividido y cómo se conforma la solución final del mismo a partir de las respuestas a las tareas.
El problema para su complementación utilizará un conjunto de ficheros JavaScript. Estos ficheros se dividen en dos grupos: los que contienen el código de cada uno de los tipos de tareas que deben generarse y las librerías necesarias para el procesamiento de las mismas, las cuales deben ser incluidas por el trabajador.

2.3.1.2 Tarea

Las tareas constituyen las pequeñas partes en que puede ser dividido un problema y son resueltas de manera paralela e independiente por los trabajadores. Están compuestas por un conjunto de parámetros, el código JavaScript y las librerías necesarias para su procesamiento.

2.3.1.3 Salvas de seguridad

Con el objetivo de evitar que se produzcan pérdidas de información, la plataforma propuesta realizará una serie de salvas de seguridad, garantizando que en caso de ocurrir algún incidente, como afectaciones con el fluido eléctrico, pueda ser reiniciada y volver al estado en que se encontraba antes de ocurrir dicho incidente. Estas salvas se efectuarán mediante la realización de salvas totales y salvas incrementales.
Salvas totales: consisten en almacenar periódicamente toda la información gestionada por el sistema, quedando guardado el estado en que se encuentra el servidor. Cada vez que se realiza una salva total, todas las salvas diferenciales así como la salva total anterior son eliminadas.
Salvas incrementales: cada vez que se envía una tarea para ser procesada por un trabajador, la información del problema al cual pertenece varía, por lo que se realiza una salva almacenando la información actualizada, es decir, la cantidad de tareas enviadas. En el momento en que una respuesta es recibida por el servidor, se realiza una salva almacenando toda su información.

El sistema, una vez reiniciado luego de ocurrir un incidente, carga la información almacenada en la salva total, y la actualiza con la información guardada en las salvas diferenciales.

2.4 Historias de usuario

Las historias de usuario son utilizadas en XP para especificar los requisitos funcionales del *software* desde el punto de vista del cliente [66].

Durante la fase de Exploración se identificaron las siguientes historias de usuario:

- Crear tipo de problema.
- Mostrar listado de tipos de problemas.
- Eliminar tipo de problema.
- Resolver problema.
- Mostrar listado de problemas.
- Eliminar problema.
- Mostrar listado de problemas resueltos.
- Mostrar detalles de un problema resuelto.
- Monitorear problemas en proceso de resolución.
- Detener ejecución de un problema.
- Reanudar ejecución de un problema.
- Salvar servidor.
- Restaurar servidor.

A continuación se muestran las historias de usuario del sistema que tienen prioridad alta para el cliente. (Ver desde Tabla 2.1 hasta Tabla 2.5) (El resto de las historias de usuario pueden ser consultadas en el Anexo II)

Historia de Usuario	
Código: HU_1	**Nombre historia de usuario:** Crear tipo de problema.
Modificación de historia de usuario número: ninguna.	
Programador: Heidy Luis Díaz.	**Iteración asignada:** 1.
Prioridad: Alta.	**Puntos estimados:** 1 (semanas).
Riesgo en desarrollo: Alto.	**Puntos reales:** 1 (semanas).
Descripción: Se le solicita al usuario que cargue el fichero comprimido que contiene toda la información referente al tipo de problema que desea crear. Una vez cargado el fichero, el sistema comprueba la existencia de toda la información necesaria y conforma el tipo de problema, el cual es almacenado en el disco duro de la computadora.	

Observaciones: En caso de que el tipo de problema ya exista, se le notifica al usuario. En caso de no existir toda la información necesaria para crear el tipo de problema, se le notifica al usuario. En caso de crearse satisfactoriamente el tipo de problema, se le notifica al usuario.

Tabla 2.1: HU_1 Crear tipo de problema.

Historia de Usuario	
Código: HU_4	**Nombre historia de usuario:** Resolver problema.
Modificación de historia de usuario número: ninguna.	
Programador: Sael José Berrillo Borrero.	**Iteración asignada:** 1.
Prioridad: Alta.	**Puntos estimados:** 5 (semanas).
Riesgo en desarrollo: Alto.	**Puntos reales:** 4 (semanas).
Descripción: Se muestra un listado con todos los tipos de problemas existentes, dándole la posibilidad al usuario de seleccionar de qué tipo es el problema que desea resolver. Una vez que seleccionado, se le solicita que introduzca los valores de los parámetros requeridos para crear un problema de ese tipo. Después de creado el problema el sistema comienza su ejecución.	
Observaciones: En caso de no existir ningún tipo de problema, se le notifica al usuario. En caso de no seleccionar algún tipo de problema, se le notifica al usuario. En caso de no introducir los parámetros requeridos en el formato correcto, se le notifica al usuario.	

Tabla 2.2: HU_4 Resolver problema.

Historia de Usuario	
Código: HU_7	**Nombre historia de usuario:** Mostrar listado de problemas resueltos.
Modificación de historia de usuario número: ninguna.	
Programador: Sael José Berrillo Borrero.	**Iteración asignada:** 1.
Prioridad: Alta.	**Puntos estimados:** 1 (semanas).
Riesgo en desarrollo: Alto.	**Puntos reales:** 1 (semanas).
Descripción: Se muestra un listado con todos los problemas resueltos.	
Observaciones: En caso de no existir ningún problema resuelto, se le notifica al usuario.	

Tabla 2.3: HU_7 Mostrar listado de problemas resueltos.

Historia de Usuario	
Código: HU_8	**Nombre historia de usuario:** Mostrar detalles de un problema resuelto.
Modificación de historia de usuario número: ninguna.	
Programador: Sael José Berrillo Borrero.	**Iteración asignada:** 1.
Prioridad: Alta.	**Puntos estimados:** 1 (semanas).
Riesgo en desarrollo: Alto.	**Puntos reales:** 1 (semanas).
Descripción: Una vez mostrado el listado de problemas resueltos, el usuario tiene la posibilidad de obtener los detalles de cada uno de ellos, seleccionando del listado el que desea detallar. Después de seleccionado y marcada la opción de detallar, el sistema muestra todos los detalles del mismo.	
Observaciones: En caso de no existir ningún problema resuelto, se le notifica al usuario. En caso de marcar la opción de detallar sin seleccionar algún problema resuelto, se le notifica al usuario.	

Tabla 2.4: HU_8 Mostrar detalles de un problema resuelto.

Historia de Usuario	
Código: HU_9	**Nombre historia de usuario:** Monitorear problemas en proceso de resolución.
Modificación de historia de usuario número: ninguna.	
Programador: Heidy Luis Díaz.	**Iteración asignada:** 1.
Prioridad: Alta.	**Puntos estimados:** 2 (semanas).
Riesgo en desarrollo: Alto.	**Puntos reales:** 1 (semanas).
Descripción: Se muestra un listado con todos los problemas en proceso de resolución, permitiendo observar cómo evoluciona dicho proceso para cada uno de los problemas.	
Observaciones:	

Tabla 2.5: HU_9 Monitorear problemas en proceso de resolución.

2.5 Requisitos no funcionales

Los requerimientos no funcionales son las propiedades o cualidades que el producto debe tener. Debe pensarse en estas propiedades como las características que hacen al producto atractivo, usable, rápido o confiable [67].

Requerimientos de *software*:

Servidor:

- Debe estar instalada la máquina virtual de Java 1.5 o superior.
- Debe estar instalado RabbitMQ 3.0.2.
- Debe estar instalado Apache 2.2.20.

Trabajador:

- Disponer de Mozilla Firefox 12.0 o superior.

Requerimientos de *hardware*:

Servidor:

- Microprocesador Pentium IV o superior.
- Mínimo 1.0 Gigabytes (GB) de RAM.
- Mínimo 20 GB de disco duro disponible.
- Disponer de conexión a la red local o externa.

Trabajador:

- Microprocesador Pentium III o superior.
- Mínimo 256 Megabytes (MB) de RAM.
- Disponer de conexión a la red local o externa.

Requerimientos de apariencia o interfaz externa:

- El sistema debe contar con una interfaz amigable, donde el usuario pueda orientarse fácilmente.

Requerimientos de portabilidad:

- El sistema debe ser multiplataforma, razón por la cual podrá ser utilizado en cualquier sistema operativo y arquitectura de red.

Restricciones en el diseño e implementación:

- Se utilizará el lenguaje de programación Java para la implementación del servidor.
- Se utilizarán los lenguajes de programación HTML y JavaScript para la implementación del trabajador.
- Se utilizará XP como metodología de desarrollo de *software* para el análisis y diseño del sistema.
- Se utilizará UML como lenguaje de modelado y Visual Paradigm for UML como herramienta CASE para el modelado del sistema.

Requerimientos de transparencia:

- Los programadores o desarrolladores no tienen que encargarse de repartir el cálculo entre los trabajadores del sistema, solamente tienen que programar cómo dividir el problema en pequeñas

tareas, cómo resolverlas y cómo integrar las soluciones parciales a la solución final. La distribución de las tareas y el control de que estas se realicen es responsabilidad del sistema.

Requerimientos de eficiencia:

- La solución a los problemas debe obtenerse más rápido haciendo uso del sistema distribuido que haciendo uso de una sola computadora.

Requerimientos de fiabilidad:

- El sistema asegurará que el problema será resuelto, independientemente de los problemas ajenos que puedan surgir, ya sean fallos de red o de electricidad, apagado de máquinas, etc.

2.6 Metáfora

Una metáfora es una historia que describe cómo debería funcionar el sistema. La tarea de elegir una metáfora permite formar un conjunto de nombres que actúen como vocabulario para hablar sobre el dominio del problema. Este conjunto de nombres ayuda a la nomenclatura de clases y métodos del sistema [66].

La plataforma de procesamiento distribuido para la resolución de problemas complejos, empleando los navegadores como trabajadores, puede ser utilizada en cualquier institución donde existan máquinas conectadas en red que dispongan del navegador web Mozilla Firefox y que deseen resolver problemas de alto costo computacional utilizando los recursos existentes.

Para resolver un problema de este tipo el servidor lo divide en tareas más simples, a las que se le asigna el identificador del problema que las generó, y las envía a la cola de tareas del *middleware* de mensajería. El *plugin* instalado en las máquinas trabajadoras, apenas se abre el navegador, se suscribe a dicha cola, quedando a la espera de una tarea para procesar. El *middleware* asigna cada una de las tareas a un trabajador disponible. El trabajador procesa la tarea recibida a través del *plugin* y con el resultado de dicho procesamiento crea una respuesta que envía a la cola de respuestas del *middleware*, quedando nuevamente disponible para ejecutar otra tarea. A cada respuesta se le asigna el identificador de la tarea a la que corresponde. En caso de que la conexión entre el *plugin* y el *middleware* se pierda antes de ser entregada la respuesta, ya sea porque se cerró el navegador o se desconectó la máquina trabajadora de la red, etc., el *middleware* asume que el trabajador no pudo realizar la tarea y la coloca nuevamente en la cola de tareas. Una vez situada la respuesta en la cola de respuestas, el *middleware* se la envía al servidor, y este último asocia por el identificador cada respuesta que recibe con el problema correspondiente, el cual la incorpora a su solución. En caso de que la conexión entre el servidor y el *middleware* se pierda antes de ser entregada la respuesta, el middleware la coloca nuevamente en la cola de respuestas.

2.7 Arquitectura

El diseño de la arquitectura de un sistema es el proceso mediante el cual se define una solución para los requisitos técnicos y operacionales del mismo. Este proceso define qué componentes forman el sistema y cómo se relacionan entre ellos. El objetivo final de la arquitectura es identificar los requisitos que producen un impacto en la estructura del sistema y reducir los riesgos asociados con la construcción del mismo. La arquitectura debe soportar los cambios futuros del *software*, del *hardware* y de las funcionalidades demandadas por los clientes [68].

2.7.1 Estilos arquitectónicos

Los estilos arquitectónicos son la herramienta básica de un arquitecto a la hora de dar forma a la arquitectura de una aplicación. Un estilo arquitectónico se puede entender como un conjunto de principios que define a alto nivel un aspecto de la arquitectura. Los estilos arquitectónicos son indicaciones abstractas de cómo dividir en partes el sistema y de cómo estas partes deben interactuar. Lo normal en una arquitectura es que no se base en un solo estilo arquitectónico, sino que combine varios de dichos estilos para obtener las ventajas de cada uno [68].

2.7.1.1 Cliente/Servidor

EL estilo arquitectónico Cliente/Servidor define una relación entre dos aplicaciones, en las cuales una de ellas (cliente) envía peticiones a la otra (servidor). Es un estilo para sistemas distribuidos en el cual el cliente realiza una o más peticiones, espera por las respuestas y las procesa a su llegada, mientras que el servidor envía los datos en respuesta a las peticiones realizadas por los clientes conectados [68].

Algunas de las ventajas que brinda este estilo arquitectónico son [67]:

- Mayor seguridad, ya que los datos se almacenan en el servidor que generalmente ofrece más control sobre la seguridad.
- Acceso centralizado a los datos, lo que facilita su acceso y actualización.

Específicamente, la plataforma a desarrollar implementará una arquitectura Cliente/Servidor en la que la máquina servidor se comporta como cliente enviando solicitudes de resolución de tareas a los trabajadores, los cuales se comportan como servidores ya que son los encargados de atender las solicitudes, procesando las tareas y enviando las respuestas de dicho procesamiento. Para facilitar la comunicación se incorpora un middleware de mensajería que permite soportar llamadas asíncronas entre el cliente y el servidor. (Ver Figura 2.2)

Figura 2.2: Arquitectura Cliente/Servidor del sistema.

2.7.1.2 N-Capas

El estilo arquitectónico N-Capas se basa en una distribución jerárquica de los roles y las responsabilidades para proporcionar una división efectiva del problema a resolver. Los roles indican el tipo y la forma de interacción con otras capas, y las responsabilidades las funcionalidades que implementan. Este estilo descompone los servicios de forma que la mayoría de las interacciones ocurren solo entre capas vecinas. Las capas de una aplicación pueden residir en la misma máquina o pueden estar distribuidas entre varios equipos. Cada capa solo contiene la funcionalidad relacionada con las tareas de esa capa y cada nivel agrega las responsabilidades y abstracciones del nivel inferior. Además, las capas inferiores no tienen dependencias de las capas superiores [68].

Algunas de las ventajas que brinda este estilo arquitectónico son [68]:

- Abstracción, ya que los cambios se realizan a alto nivel y se puede incrementar o reducir el nivel de abstracción que se usa en cada capa del modelo.
- Aislamiento, ya que se pueden realizar actualizaciones en el interior de las capas sin que esto afecte al resto del sistema.
- Independencia, ya que elimina la necesidad de considerar el *hardware* y el despliegue así como las dependencias con interfaces externas.

Específicamente, el servidor de la plataforma a desarrollar quedará estructurado en 3 capas: Presentación, Negocio y Acceso a Datos. (Ver Figura 2.3)

Figura 2.3: Arquitectura en capas del servidor.

Capa de Presentación: en esta capa se encuentran todas las interfaces que interactúan con el usuario. Es la encargada de presentar el sistema al usuario, comunicándole y capturando información en un mínimo de proceso. Esta capa se comunica únicamente con la capa inmediatamente inferior Capa de Negocio.

Capa de Negocio: es en esta capa donde se definen y establecen todas las reglas del negocio que deben cumplirse. En ella se encuentran la clase controladora y el resto de las clases que modelan el negocio. Es la encargada de recibir las peticiones del usuario y enviar las respuestas tras el proceso. Esta capa se comunica con la Capa de Presentación, para recibir las solicitudes y presentar los resultados, y con la Capa de Acceso a Datos, para solicitar el almacenamiento o recuperación de datos.

Capa de Acceso a Datos: es la encargada de acceder a los datos, recibe las solicitudes de almacenamiento o recuperación de información desde la capa inmediatamente superior Capa de Negocio.

2.8 Plan de entregas

El plan de entregas se elabora con el objetivo de fijar qué período de tiempo puede tardar la implementación, definiéndose las fechas en que serán liberadas las versiones funcionales del producto [66]. (Ver Anexo III)

2.9 Plan de iteraciones

Como parte del ciclo de vida de un proyecto que utiliza la metodología XP se crea el plan de iteraciones, que tiene como objetivo mostrar la duración y el orden en que serán implementadas las historias de usuario dentro de cada iteración [66].

Para la construcción de la plataforma propuesta se han definido 13 historias de usuario que serán desarrolladas en 2 iteraciones. (Ver Anexo IV)

2.10 Tarjetas CRC

Las tarjetas CRC (Clase, Responsabilidad, Colaboración) son una técnica utilizada en XP para diseñar la solución informática según el paradigma orientado a objetos. Su principal ventaja es que todo el equipo contribuye a la elaboración del diseño de la solución.

Durante el diseño de la propuesta de solución se identificaron un conjunto de clases que aunque no tienen responsabilidades resultan claves para la implementación de la misma, estas son *IProblem* e *IData*, utilizadas para lograr la extensibilidad del sistema; *ExcuteTask*, *WaitTask* y *EndTask*, utilizadas para indicar el tipo de tarea que se genera en cada momento e implican cambios de estado en el problema; *Task*, *ParameterDescription*, *LibraryDescription* y *Response*, utilizadas para contener información imprescindible para el funcionamiento del sistema.

El resto de las clases identificadas contienen las responsabilidades necesarias para llevar a cabo las funcionalidades del sistema, estas son *Environment*, encargada de establecer la comunicación con el *middleware* de mensajería; *RunTime*, encargada de gestionar los problemas; *ProblemRunTimeInstance*, encargada de describir el problema, generar las tareas y unificar las respuestas; e *InformationManager*, encargada de gestionar los datos persistentes del sistema. A continuación se muestra la tarjeta CRC de la clase *ProblemRunTimeInstance*. (Ver Tabla 2.6) (El resto de las tarjetas CRC pueden ser consultadas en el Anexos V)

ProblemRunTimeInstance	
Responsabilidades	Colaboradores
Generar la próxima tarea (genera la próxima tarea que será enviada al *middleware* de mensajería).	IProblem. Task. ExecuteTask. EndTask.
Unificar la respuesta de una tarea (integra la respuesta de una tarea a su solución).	IProblem. Response.
Retornar su nombre.	IProblem.
Describir las librerías necesarias para la resolución de las tareas.	IProblem. LibraryDescription.
Describir los tipos de tareas que deben ser generadas.	IProblem. TaskDescription.
Describir los parámetros iniciales necesarios para su ejecución.	IProblem. ParameterDescription.
Retornar su descripción general.	IProblem.

Tabla 2.6: Tarjeta CRC de la clase *ProblemRunTimeInstance*.

2.11 Patrones de diseño

Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de *software*. Proporcionan un esquema para refinar los subsistemas o componentes de un sistema, o las relaciones entre ellos. Estos patrones solucionan problemas específicos del diseño y hacen los diseños orientados a objetos más flexibles, elegantes y reutilizables [69].

A continuación se muestran los patrones de diseño tenidos en cuenta para alcanzar una mayor calidad en el diseño.

2.11.1 Patrones GRASP

Los patrones GRASP (acrónico en inglés de *General Responsibility Assignment Software Patterns*) representan los principios básicos de la asignación de responsabilidades a objetos, expresados en forma de patrones.

Experto: este patrón plantea que se debe asignar una responsabilidad al experto en información, es decir, a la clase que cuenta con la información necesaria para cumplir la responsabilidad [70].
Una de las clases donde se evidencia la utilización del patrón Experto es la clase *InformationManager*, que contiene la información necesaria para realizar las responsabilidades de salvar y restaurar la información del servidor. (Ver Anexo VI)

Creador: este patrón es el encargado de guiar la asignación de responsabilidades relacionadas con la creación de objetos. Plantea que se debe asignar la responsabilidad de crear una instancia de una clase Y a una clase X solamente cuando [70]:

- X contiene a Y.
- X es una agregación (o composición) de Y.
- X almacena a Y.
- X tiene los datos de inicialización de Y (datos que requiere su constructor).
- X usa a Y.

En la clase *RunTime* se evidencia la utilización del patrón Creador. Esta clase es la encargada de crear instancias de la clase *ProblemRunTimeInstance*, pues contiene una colección de objetos de este tipo, así como toda la información necesaria para la inicialización de los mismos. (Ver Anexo VII)

Controlador: este patrón plantea que se debe asignar la responsabilidad del manejo de un mensaje de los eventos de un sistema a una clase que represente una de las siguientes opciones [70]:

- El sistema global (controlador de fachada).
- La empresa u organización global (controlador de fachada).
- Algo en el mundo real que es activo (por ejemplo, el papel de una persona) y que pueda participar en la tarea (controlador de tareas).

En la clase *Environment* se evidencia la utilización del patrón Controlador, pues esta es la encargada de manejar los diferentes hilos mediante los cuales se controla la lógica del negocio. (Ver Anexo VIII)

Alta cohesión: este patrón plantea que se debe asignar una responsabilidad de modo que la cohesión siga siendo alta. Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realizan un trabajo enorme. Este patrón propone el diseño de clases con

responsabilidades moderadas en su área funcional, donde cada una colabore con las demás para llevar a cabo una tarea [70].

La utilización del patrón Alta Cohesión se evidencia en cada una de las clases del sistema, ya que cada una de ellas realiza las responsabilidades correspondientes a su área funcional y colabora con otras clases para ejecutar las tareas. Un ejemplo de ello es la clase *Environment*. (Ver Anexo IX)

Bajo Acoplamiento: el acoplamiento es una medida de la fuerza con que una clase está conectada a otras. Este patrón propone el diseño de clases más independientes, lo que reduce el impacto del cambio y facilita la reutilización en otros sistemas [70].

La utilización del patrón Bajo Acoplamiento se evidencia mediante la implementación de la interfaz *IData* por la clase *InformationManager*, garantizando que en caso de ocurrir cambios en la forma de implementar las funcionalidades de *InformationManager* estos tengan el menor impacto posible en el resto de las clases. (Ver Anexo X)

2.11.2 Patrones GOF

Los patrones GOF (acrónico en inglés de Gang of Four) se clasifican en 3 categorías basadas en su propósito: creacionales, estructurales y de comportamiento.

Creacionales: los patrones creacionales se encargan de la creación de instancias de los objetos. Abstraen la forma en que se crean los objetos, permitiendo tratar las clases a crear de forma genérica, dejando para después la decisión de qué clase crear o cómo crearla.

Estructurales: los patrones estructurales se ocupan de cómo las clases y objetos se combinan para formar grandes estructuras y proporcionar nuevas funcionalidades. Tratan de conseguir que los cambios en los requisitos de la aplicación no ocasionen cambios en las relaciones entre los objetos.

Comportamiento: los patrones de comportamiento plantean la interacción y cooperación entre las clases. Son utilizados para organizar, manejar y combinar comportamientos.

A continuación se describe el patrón GOF aplicado en el diseño del sistema:

Singleton: este patrón se emplea cuando se quiere mantener un único punto de acceso a una única instancia de una clase en lugar de cualquier otra forma de visibilidad. La idea clave es que la clase X defina un método estático “getInstancia” el cual proporciona una única instancia de X, y su constructor se declara privado [70].

En la clase *RunTime* se evidencia la utilización del patrón *Singleton*. Esta clase implementa el método *getInstance*, el cual proporciona una única instancia de la misma, y su constructor es privado. (Ver Anexo XI)

2.11.3 Otros patrones de diseño utilizados

Con el objetivo de lograr un diseño claro, flexible y reutilizable, también se utilizaron los siguientes patrones de diseño.

Simple Factory: el patrón *Simple Factory* se utiliza para seleccionar y devolver una instancia de una clase a partir de varias clases similares, dependiendo de los datos suministrados a la fábrica. Por lo general, todas las clases similares tienen una clase padre común y métodos comunes, pero cada una de ellas realiza una tarea diferente y está optimizada para diferentes tipos de datos [71].
En la clase *ProblemRunTimeInstance* se evidencia el uso del patrón *Simple Factory*, pues esta es la encargada de crear y devolver instancias de las clases hijas de la clase *Task*, dependiendo de los datos que se le suministre. (Ver Anexo XII)

***Null Object*:** facilita una alternativa para utilizar *null* indicando la ausencia de un objeto al que delegue una operación. Utilizando *null* para indicar la ausencia de cada uno de los objetos requeridos se hace necesario realiza una pregunta para ver si es *null* antes de cada llamada a los métodos de otros objetos. En lugar de utilizar *null*, el patrón *Null Object* utiliza una referencia a un objeto que no hace nada [69].
En las clases *WaitTask* y *EndTask* se evidencia el uso del patrón *Null Object*, pues estas clases no contienen ninguna funcionalidad y son utilizadas para indicar la ausencia de tareas a procesar en ese momento. (Ver Anexo XIII)

2.12 Diagrama de clases del diseño

Luego de elaboradas las tarjetas CRC, donde se identificaron las clases del sistema y sus principales responsabilidades, se confecciona el diagrama de clases del diseño. Este diagrama aunque no es un artefacto propio de XP, permite representar gráficamente cada una de las clases y las relaciones que se establecen entre ellas.
En el diagrama de clases del diseño puede observarse la estructura del servidor, quedando reflejada la distribución de sus clases en capas. (Ver Anexo XIV)

2.13 Conclusiones parciales

En el presente capítulo se elaboró el modelo de dominio, que constituyó la primera visión del sistema y sirvió como punto de partida para el diseño del mismo. A partir de su realización se pudo concluir que los componentes imprescindibles en una plataforma de procesamiento distribuido son: el servidor y el trabajador.
También fueron definidas las historias de usuario, así como los requisitos no funcionales, lo que proporcionó un conocimiento detallado de los requerimientos del cliente, y se elaboraron el plan de entregas y el plan de iteraciones, que permitieron conocer la duración de la implementación, así como el

orden en que serán implementadas las historias de usuario. Toda esta información permitió llegar a la conclusión de que la construcción del sistema tardará un total de 6 meses y será desarrollado en 2 iteraciones, obteniéndose en cada una de ellas una versión ejecutable del mismo.

Además, se confeccionó la propuesta de solución y se describieron los estilos arquitectónicos empleados para su diseño, pudiendo concluirse que el sistema estará compuesto por tres componentes fundamentales: el servidor, el *middleware* de mensajería y el *plugin*, y que implementa una arquitectura cliente-servidor, donde el servidor estará estructurado en tres capas: la capa de presentación, la capa de negocio y la capa de acceso a datos.

Finalmente, se identificaron las clases pertenecientes al servidor y se crearon las tarjetas CRC referentes a cada una de ellas, lo que permitió concluir que el funcionamiento del servidor será guiado por las clases *Environment* y *RunTime*.

Capítulo III: Implementación y prueba

3.1 Introducción

En el presente capítulo se muestran los principales componentes del sistema, las relaciones que existen entre ellos y la distribución de los nodos necesarios para el despliegue de la aplicación. Además, se describe la implementación del software a través de tareas de ingeniería y estándares de codificación, y se muestran las interfaces gráficas, explicando cada una de ellas para un mayor entendimiento acerca del funcionamiento del sistema. Finalmente se muestran las pruebas realizadas al sistema, las cuales tienen el objetivo de asegurar la calidad y el correcto funcionamiento del mismo.

3.2 Tareas de ingeniería

En XP, todo el trabajo que se realiza en cada una de las iteraciones es expresado en tareas de programación, a las que se le asigna a un programador como responsable y son llevadas a cabo por parejas de programadores [66].

A continuación se muestran las tareas de ingeniería implicadas en la realización de cada una de las historias de usuario. (Ver Tabla 3.1) (Las especificaciones de dichas tareas pueden ser consultadas en el Anexo XV)

Iteración	Historia de usuario	Tareas
1	Crear tipo de problema.	- Cargar fichero. - Validar que existe toda la información necesaria. - Almacenar el tipo de problema.
	Resolver problema.	- Crear problema. - Generar tarea. - Asignar tarea a un trabajador. - Procesar tarea. - Retornar la respuesta. - Unificar la respuesta a la solución final.
	Mostrar listado de problemas resueltos.	- Listar problemas resueltos.
	Mostrar detalles de un problema resuelto.	- Detallar problema resuelto.
	Monitorear problemas en proceso de resolución.	- Listar problemas en proceso de resolución. - Actualizar información.
	Salvar servidor.	- Salvar servidor.
	Restaurar servidor.	- Restaurar servidor.

2	Mostrar listado de tipos de problemas.	- Listar tipos de problemas.
	Eliminar tipo de problema.	- Listar tipos de problemas. - Mostrar mensaje de confirmación. - Eliminar problemas en proceso de resolución. - Eliminar tipo de problema.
	Mostrar listado de problemas.	- Listar problemas.
	Eliminar problema.	- Listar problemas. - Mostrar mensaje de confirmación. - Eliminar problema.
	Detener ejecución de un problema.	- Detener ejecución del problema.
	Reanudar ejecución de un problema.	- Reanudar ejecución del problema.

Tabla 3.1: Tareas de ingeniería.

3.3 Estándares de codificación

Al iniciar un proyecto de *software* es muy importante definir un estándar de codificación, pues asegura que todos los programadores trabajen de forma coordinada. Esto hace posible que la mantenibilidad del código del sistema que se desea desarrollar se pueda llevar a cabo de una forma más fácil. A continuación se describen los estándares de codificación utilizados durante la implementación del sistema.

3.3.1 Nombres de estructuras

Nombres de clases

Los nombres de las clases adoptan la notación UpperCamelCase y no se utiliza el guión bajo "_" como delimitador entre palabras. La notación UpperCamelCase define que cada palabra que conforma el nombre de los diferentes elementos de código comienza con la primera letra en mayúsculas y el resto en minúsculas; además, todos los nombres de las diferentes estructuras de código deben ser descriptivos. (La aplicación de este estándar de codificación en el sistema se muestra en el Anexo XVI)

Nombres de interfaces

Los nombres de las interfaces se comportan de la misma manera que los nombres de las clases, solo que a estas se le antepone la letra "I" indicando que el nombre corresponde a una interfaz. (La aplicación de este estándar de codificación en el sistema se muestra en el Anexo XVII)

Nombres de atributos

Los atributos de las clases comienzan con guión bajo "_" seguido del nombre del atributo según la notación lowerCamelCase. Esta define que todas las palabras que conforman el nombre, excepto la primera, comienzan con la primera letra en mayúsculas y el resto en minúsculas. Además, todos los nombres de los atributos deben ser descriptivos. (La aplicación de este estándar de codificación en el sistema se muestra en el Anexo XVIII)

Nombres de parámetros y variables

Para los nombres de los parámetros de los métodos y las variables declaradas dentro de los mismos, se utiliza la notación lowerCamelCase. (La aplicación de este estándar de codificación en el sistema se muestra en el Anexo XIX)

3.3.2 Indentación

Longitud de la línea

Las líneas no cuentan con más de 80 caracteres ya que no son interpretadas correctamente por muchas terminales y herramientas.

Rotura de líneas

Cuando una expresión no entra en una línea es separada de acuerdo a los siguientes principios:

- Romper después de una coma.
- Romper antes de un operador.
- Realizar preferiblemente roturas de alto nivel (más a la derecha que el «padre») que de bajo nivel (más a la izquierda que el «padre»).
- Alinear la nueva línea con el comienzo de la expresión al mismo nivel de la línea anterior.
- Si las reglas anteriores provocan que el código parezca confuso o que este se acumule en el margen derecho, se utilizan 8 espacios.

(La aplicación de este estándar de codificación en el sistema se muestra en el Anexo XX)

3.3.3 Comentarios de implementación

Comentarios de bloque

Los comentarios de bloque (un bloque es cualquier código encerrado por llaves "{" y "}") se usan para dar descripciones de archivos, métodos, estructuras de datos y algoritmos. Estos se utilizan al comienzo de cada archivo, antes de cada método o en el interior de los métodos. Los comentarios de bloque en el interior de un método son indentados al mismo nivel que el código que describen. Un comentario de

bloque va precedido por una línea en blanco que lo separe del resto del código. (La aplicación de este estándar de codificación en el sistema se muestra en el Anexo XXI)

Comentarios de una línea

Pueden aparecer comentarios cortos de una única línea al nivel del código que sigue. Si un comentario no se puede escribir en una línea debe seguir el formato de los comentarios de bloque. Un comentario de una sola línea va precedido de una línea en blanco. (La aplicación de este estándar de codificación en el sistema se muestra en el Anexo XXII)

3.3.4 Declaraciones

Cantidad por línea

Se realiza solo una declaración por línea, ya que facilita los comentarios. (La aplicación de este estándar de codificación en el sistema se muestra en el Anexo XXIII)

Colocación

Las declaraciones se sitúan solo al principio de los bloques. No se debe esperar al primer uso para declararlas porque puede confundir al programador y limitar la portabilidad del código dentro de su ámbito. La excepción a la regla son los índices de los bucles *for*. (La aplicación de este estilo de codificación en el sistema se muestra en el Anexo XXIV)

Declaraciones de clases e interfaces

Al programar clases e interfaces, se siguen las siguientes reglas de formato:

- Ningún espacio en blanco entre el nombre de un método y el paréntesis "("que abre su lista de parámetros.
- La llave de apertura "{" aparece al final de la misma línea de la sentencia de declaración.
- La llave de cierre "}" empieza una nueva línea indentada ajustada a su sentencia de apertura correspondiente, excepto cuando no existen sentencias entre ambas, que debe aparecer inmediatamente después de la de apertura "{".
- Los métodos se separan con una línea en blanco.

(La aplicación de este estilo de codificación en el sistema se muestra en el Anexo XXV)

3.4 Diagrama de despliegue

Un diagrama de despliegue es un modelo de objetos que describe la distribución física del sistema en términos de cómo se distribuyen las funcionalidades entre los nodos. Cada nodo representa un recurso de cómputo, normalmente un procesador o un dispositivo similar.

A continuación se muestra el diagrama de despliegue del sistema desarrollado. (Ver Figura 3.1)

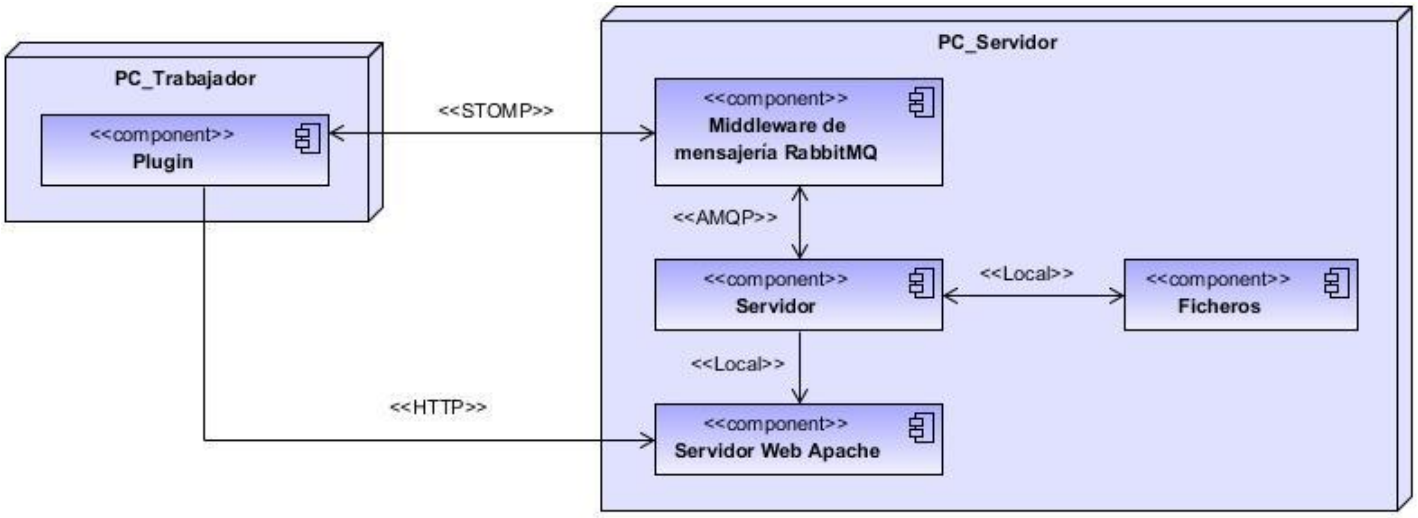


Figura 3.1: Diagrama de despliegue.

3.4.1 Descripción del diagrama de despliegue

La comunicación entre el servidor y el *plugin* se realiza teniendo como intermediario al *middleware* de mensajería RabbitMQ, utilizando el protocolo AMQP para comunicar el servidor con el *middleware* y el protocolo STOMP para comunicar el *middleware* con el *plugin*. Una de las funcionalidades más importantes de la plataforma desarrollada es el procesamiento de tareas en los trabajadores, lo que requiere en ocasiones de la utilización de librerías, razón por la cual se precisa que el servidor se comunique con el servidor web Apache, para publicar las librerías necesarias. Esta comunicación se realiza de manera local, ya que ambos componentes se encuentran instalados en la misma computadora. El *plugin* accede a las librerías publicadas en el servidor web Apache a través del protocolo HTTP. Todos los datos que deben ser guardados de forma persistente son almacenados por el servidor en ficheros de manera local.

3.5 Interfaces principales de la aplicación

La interfaz principal del sistema (Ver Figura 3.2) brinda la posibilidad de monitorear los problemas que se encuentran en proceso de resolución, y de detener o reanudar la ejecución de los mismos. De cada uno de estos problemas se muestra el tipo de problema al que pertenece, su identificador, el estado en que se encuentra, la cantidad de tareas enviadas a los trabajadores y la cantidad de respuestas recibidas; todos estos datos son actualizados periódicamente. Esta interfaz cuenta con una barra de menú que agrupa las funcionalidades proporcionadas por el sistema en dos menús desplegables. El menú “Tipos de problemas” ofrece las funcionalidades relacionadas con la gestión de los tipos de problemas (Ver Anexo XXVI), mientras que el menú “Problemas” ofrece las funcionalidades relacionadas con la gestión de los problemas (Ver Anexo XXVII).

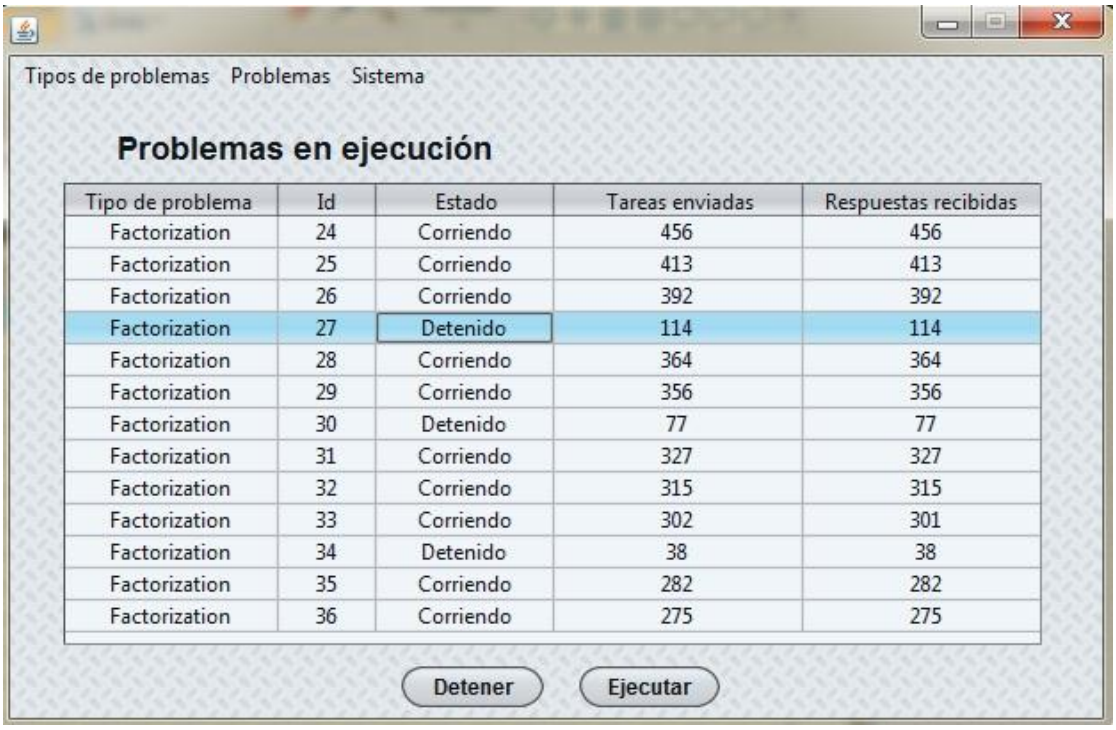


Figura 3.2: Interfaz principal del sistema.

La interfaz *ExecuteProblem* (Ver Figura 3.3) solicita al usuario que entre los valores iniciales de los parámetros de entrada del problema que desea ejecutar. Esta se crea de forma dinámica a partir de la información que brinda la clase Java perteneciente al tipo de problema previamente seleccionado.

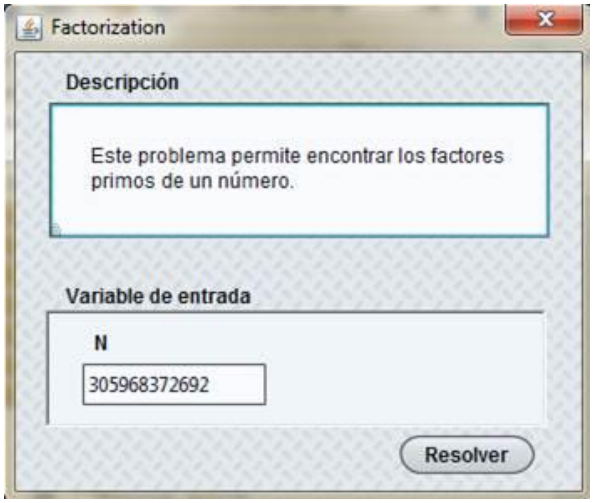


Figura 3.3: Interfaz *ExecuteProblem*.

La interfaz *ProblemDetails* (Ver Figura 3.4) muestra la información de un problema resuelto, especificando el tipo de problema al que pertenece, sus parámetros iniciales, la respuesta alcanzada y el tiempo que duró su ejecución.

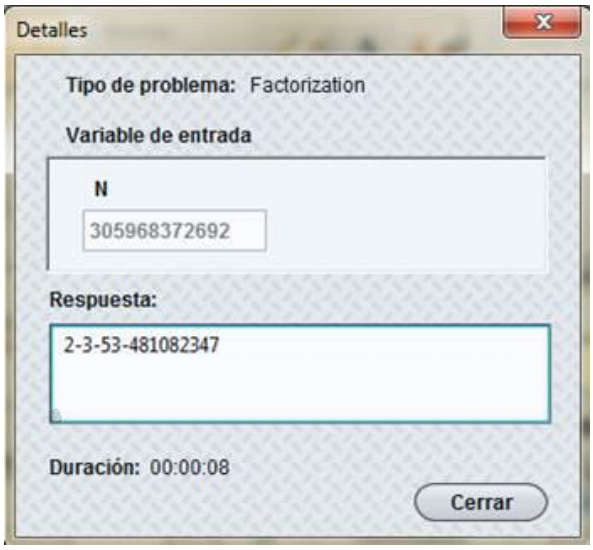


Figura 3.4: Interfaz *ProblemDetails*.

3.6 Cumplimiento de los principios de la computación distribuida

A continuación se describe como el sistema desarrollado cumple con cada uno de los principios de la computación distribuida.

3.6.1 Extensibilidad

Los sistemas de procesamiento distribuido para la inclusión de nuevas computadoras a la red como nodos de procesamiento (trabajadores), requieren de la instalación de un *software* en dichas computadoras que se encarga del procesamiento de los datos. A raíz de esto se han identificado un conjunto de restricciones que afectan directamente la extensibilidad a nivel de *hardware* de este tipo de sistema distribuido. Estas son la compatibilidad del sistema operativo de las computadoras que se desean incluir con dicho *software* y la disponibilidad de permisos por parte de los usuarios de estas computadoras para instalar el *software*.

La plataforma de procesamiento distribuido desarrollada garantiza la extensibilidad de *hardware* mediante la eliminación de estas restricciones, lo que amplía considerablemente la cantidad de computadoras que pueden colaborar con la resolución de problemas complejos. En la misma, el *software* de los trabajadores consiste en un *plugin* que se instala en el navegador web, por lo que no es necesario ningún permiso especial por parte de los usuarios para su instalación. El navegador utilizado por la plataforma es Mozilla Firefox, el cual es multiplataforma, lo que permite la adición de nuevas computadoras (trabajadores) independientemente del sistema operativo que posean.

Esta plataforma también garantiza la extensibilidad de *software* mediante la utilización de las interfaces *IProblem* e *IData*. *IProblem* estandariza la creación de nuevos tipos de problemas, así como la re-implementación de los existentes, mientras que *IData* estandariza la creación de nuevas formas de

almacenamiento y recuperación de los datos persistentes, así como la re-implementación de las existentes.

3.6.2 Escalabilidad

Una de las estrategias fundamentales llevadas a cabo con el fin de garantizar que el sistema desarrollado sea escalable fue la utilización del *middleware* de mensajería RabbitMQ, el cual soporta una gran cantidad de computadoras suscritas a sus colas. Además, el sistema ejecuta los problemas de manera concurrente, garantizando que aunque la cantidad de problemas para resolver aumente, todos puedan ser atendidos alternando su ejecución en el servidor.

Todo lo anteriormente planteado garantiza que el sistema pueda continuar funcionando sin necesidad de realizar cambios independientemente de la cantidad de trabajadores conectados y la cantidad de problemas ejecutándose.

3.6.3 Tratamiento a fallos

Por más confiable que pudiese parecer un sistema es necesario estar siempre preparado para cuando este eventualmente falle. Específicamente, en un sistema distribuido los fallos que se producen son parciales, pues algunos componentes fallan mientras otros siguen funcionando, dejando de brindarse solo los servicios ofrecidos por los componentes afectados.

En el sistema desarrollado los fallos que pueden ocurrir están determinados por la disponibilidad esporádica que presentan los trabajadores. En el peor de los casos la desconexión de un trabajador puede ocurrir durante el procesamiento de una tarea, lo que significaría que la tarea se pierda y por tanto, no se pueda obtener la solución final del problema al que dicha tarea pertenece.

3.6.3.1 Detección de fallos

Para dar tratamiento a un fallo es necesario primeramente detectar que este está ocurriendo. En el sistema desarrollado el *middleware* de mensajería es el encargado de llevar a cabo la detección de fallos, pues este establece un canal de comunicación con cada uno de los trabajadores que se suscriben a la cola de tareas, lo que permite detectar automáticamente cuándo un trabajador se desconecta.

3.6.3.2 Enmascaramiento de fallos

El sistema realiza el enmascaramiento de fallos mediante la corrección de los mismos apenas son detectados, de manera que el usuario no percibe en ningún momento que está ocurriendo un fallo.

3.6.3.3 Recuperación ante fallos

La recuperación ante fallos se lleva a cabo mediante la reasignación de tareas. Para ello el *middleware* de mensajería cuenta con una variable de confirmación, que es activada por el servidor indicándole que debe

esperar por su confirmación antes de liberar alguna tarea. En el momento en que una respuesta es recibida por el servidor, este envía el mensaje de confirmación esperado por el *middleware*. Esto garantiza que si un trabajador se desconecta antes de culminar el procesamiento de la tarea que le fue asignada, o la respuesta obtenida a partir de dicho procesamiento no es recibida por el servidor, el *middleware* al no haber desechado la tarea, puede colocarla nuevamente en la cola y reasignarla posteriormente a otro trabajador disponible.

3.6.4 Concurrencia

El sistema desarrollado está diseñado para atender una gran cantidad de problemas alternando la ejecución de los mismos, de manera que da la oportunidad a cada uno de generar una tarea cada determinado tiempo. Esto posibilita que un problema no tenga que esperar a que los demás culminen su ejecución para comenzar a ejecutarse.

El sistema cuenta con un grupo de trabajadores a los que se les asignan tareas para procesar. Esto posibilita que puedan existir varias tareas ejecutándose al mismo tiempo, es decir, si se tienen N trabajadores conectados al sistema, entonces se pueden estar ejecutando a la vez hasta N tareas.

De esta manera se garantiza que tanto la ejecución de los problemas como la ejecución de las tareas se realicen de manera concurrente.

3.6.5 Transparencia

El sistema ha sido desarrollado de manera que se muestra ante el usuario como un único sistema coherente, ocultando en todo momento su estructura distribuida. Sus trabajadores se encuentran suscritos a la cola de tareas del *middleware* de mensajería, siendo este último el único en conocer las direcciones IP de cada uno de ellos, direcciones a las que envía las tareas para ser procesadas. Esto garantiza que se mantenga una transparencia de acceso y de ubicación, pues el usuario en ningún momento necesita conocer la ubicación física de los trabajadores y accede a los recursos remotos como si se trataran de los recursos locales de un solo sistema.

Además de cumplir con los dos tipos de transparencia más importantes para un sistema distribuido, la transparencia de acceso y la transparencia de ubicación, el sistema mantiene una transparencia frente a fallos, pues los mismos son tratados sin notificar al usuario de su ocurrencia.

3.7 Validación del sistema

Las pruebas constituyen una de las actividades fundamentales en el proceso de desarrollo de un *software*. Estas tienen como objetivo verificar que se ha dado cumplimiento a los requerimientos y garantizan la calidad del producto final.

XP divide las pruebas del sistema en dos grupos: pruebas unitarias, encargadas de verificar el código y diseñadas por los programadores, y pruebas de aceptación o pruebas funcionales, destinadas a evaluar si al final de una iteración se consiguieron las funcionalidades requeridas y definidas por el cliente y el probador [66]. A la plataforma desarrollada se le realizaron dichas pruebas, además de pruebas de rendimiento para comprobar la eficiencia y fiabilidad del sistema.

3.7.1 Pruebas unitarias

Las pruebas unitarias se realizan a las funcionalidades de las clases del sistema con el objetivo de obtener los posibles errores que pudieran ocurrir durante la ejecución del mismo [72]. A través de ellas se verifica que cierto módulo o funcionalidad se ejecuta dentro de los parámetros y especificaciones concretadas en los requisitos.

Las pruebas de unidad se realizaron de manera automatizada haciendo uso de la librería JUnit para la herramienta de desarrollo NetBeans. JUnit permite realizar la ejecución de clases Java de manera controlada, para poder evaluar si el funcionamiento de cada uno de los métodos de una clase se comporta como se espera.

A continuación se muestran las pruebas unitarias realizadas a algunas de las funcionalidades más importantes. (Ver desde Tabla 3.2 hasta Tabla 3.4) (Los resultados de las pruebas unitarias realizadas a cada una de las clases, para validar cada uno de sus métodos, pueden ser consultados en el Anexo XXVIII)

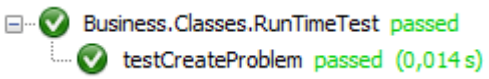
Prueba de Unidad		
Nombre: testCreateProblem.		
Estado: Satisfactoria.	**Tipo:** Caja blanca.	**Última ejecución:** 10/05/2013.
Ejecutado por: Heidy Luis Díaz.	**Verificado por:** Jorge Landrian García.	
Entrada: String javaClass.		
Criterio de aceptación: Crea un problema.		
Resultado: 		

Tabla 3.2: Prueba de unidad para el método *CreateProblem*.

Prueba de Unidad		
Nombre: testGetNextTask.		
Estado: Satisfactoria.	**Tipo:** Caja blanca.	**Última ejecución:** 10/05/2013.
Ejecutado por: Sael José Berrillo Borrero.	**Verificado por:** Jorge Landrian García.	
Entrada:		
Criterio de aceptación: Obtener la próxima tarea a ejecutar.		
Resultado:		

Business.Classes.RunTimeTest passed
testGetNextTask passed (0,006 s)

Tabla 3.3: Prueba de unidad para el método *GetNextTask*.

Prueba de Unidad		
Nombre: testDeliverResponseProblem.		
Estado: Satisfactoria.	**Tipo**: Caja blanca.	**Última ejecución:** 10/05/2013.
Ejecutado por: Sael José Berrillo Borrero.	**Verificado por:** Jorge Landrian García.	
Entrada:		
Criterio de aceptación: Unificar una respuesta a un problema.		
Resultado: The test passed.(0,062 s) Business.Classes.RunTimeTest passed testDeliverResponseProblem passed (0,0 s)		

Tabla 3.4: Prueba de unidad para el método *DeliverResponseProblem*.

3.7.1.1 Análisis de resultados

Las pruebas unitarias fueron aplicadas al sistema en 3 iteraciones. Luego de implementadas las historias de usuario planificadas para la primera iteración del sistema se realizó la primera iteración de pruebas, donde se detectaron 26 no conformidades que fueron corregidas durante la siguiente iteración del sistema. Una vez culminada la implementación de las historias de usuario planificadas para la segunda iteración del sistema, y corregidas las no conformidades de la iteración anterior, se realizó la segunda iteración de pruebas donde se detectaron 14 no conformidades, que fueron corregidas en la propia iteración del sistema en que fueron detectadas debido a que esta constituye la iteración final. Finalmente se realizó una tercera iteración de pruebas en la que no se detectaron no conformidades obteniéndose un resultado satisfactorio. (Ver Figura 3.5)

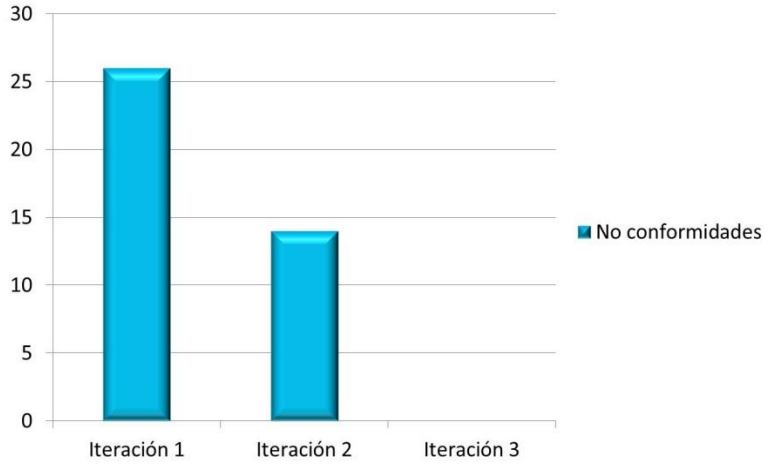


Figura 3.5: Resultados de las pruebas unitarias por iteraciones.

3.7.2 Pruebas de aceptación

Las pruebas de aceptación son muy importantes pues representan la satisfacción del cliente con el producto desarrollado. Son creadas en base a las historias de usuario y consideradas como "pruebas de caja negra". Los clientes son responsables de verificar que los resultados de estas pruebas sean correctos. Una historia de usuario no se puede considerar terminada hasta tanto pase correctamente todas las pruebas de aceptación [73].

A continuación se muestran las pruebas de aceptación aplicadas a las historias de usuario del sistema que tienen prioridad alta para el cliente. (Ver desde Tabla 3.5 hasta Tabla 3.9) (El resto de las pruebas de aceptación aplicadas a las historias de usuario pueden ser consultadas en el Anexo XXIX)

Caso de Prueba de Aceptación	
Código de caso de prueba: HU1_CP1	**Nombre de la historia de usuario:** Crear tipo de problema.
Responsable de la prueba: Heidy Luis Díaz.	
Descripción de la prueba: Prueba de funcionalidad para crear un tipo de problema.	
Condiciones de ejecución: Debe existir un fichero .zip que contenga la clase Java y los ficheros JavaScript.	
Entrada/Pasos de ejecución: ➢ El usuario carga el fichero .zip. ➢ Se verifica que el fichero .zip contenga toda la información necesaria para crear el tipo de problema. ➢ Se almacena la información del tipo de problema.	
Resultado esperado: Se muestra una notificación al usuario informándole que se ha creado correctamente el tipo de problema.	
Evaluación de la prueba: Prueba satisfactoria.	

Tabla 3.5: HU1_CP1 Crear tipo de problema.

Caso de Prueba de Aceptación	
Código de caso de prueba: HU4_CP4	**Nombre de la historia de usuario:** Resolver problema.
Responsable de la prueba: Sael José Berrillo Borrero.	
Descripción de la prueba: Prueba de funcionalidad para resolver un problema.	
Condiciones de ejecución: Debe existir algún tipo de problema. Debe estar iniciado el RabbitMQ. Debe existir algún cliente conectado.	
Entrada/Pasos de ejecución: ➢ Se muestra un listado con los nombres de los tipos de problemas existentes.	

➢ El usuario selecciona el tipo de problema que desea resolver. ➢ El usuario introduce los valores de los parámetros iniciales requeridos por el tipo de problema seleccionado. ➢ Se verifica que los valores estén en el formato correcto. ➢ Se resuelve el problema.
Resultado esperado: Se resuelve satisfactoriamente el problema.
Evaluación de la prueba: Prueba satisfactoria.

Tabla 3.6: HU4_CP4 Resolver problema.

Caso de Prueba de Aceptación	
Código de caso de prueba: HU7_CP7	**Nombre de la historia de usuario:** Mostrar listado de problemas resueltos.
Responsable de la prueba: Sael José Berrillo Borrero.	
Descripción de la prueba: Prueba de funcionalidad para listar los problemas resueltos.	
Condiciones de ejecución: Debe existir algún problema resuelto.	
Entrada/Pasos de ejecución: ➢ Se muestra un listado con el identificador y el tipo de problema de cada uno de los problemas resueltos.	
Resultado esperado: Se muestra satisfactoriamente el listado de los problemas resueltos.	
Evaluación de la prueba: Prueba satisfactoria.	

Tabla 3.7: HU7_CP7 Mostrar listado de problemas resueltos.

Caso de Prueba de Aceptación	
Código de caso de prueba: HU8_CP8	**Nombre de la historia de usuario:** Mostrar detalles de un problema resuelto.
Responsable de la prueba: Sael José Berrillo Borrero.	
Descripción de la prueba: Prueba de funcionalidad para mostrar los detalles de un problema resuelto.	
Condiciones de ejecución: Debe existir algún problema resuelto.	
Entrada/Pasos de ejecución: ➢ El usuario selecciona el problema resuelto que desea detallar. ➢ Se muestra una ventana con los detalles del problema resuelto seleccionado.	
Resultado esperado: Se muestran satisfactoriamente los detalles del problema resuelto.	
Evaluación de la prueba: Prueba satisfactoria.	

Tabla 3.8: HU8_CP8 Mostrar detalles de un problema resuelto.

Caso de Prueba de Aceptación

Código de caso de prueba: HU9_CP9	**Nombre de la historia de usuario:** Monitorear problemas en proceso de resolución.
Responsable de la prueba: Heidy Luis Díaz.	
Descripción de la prueba: Prueba de funcionalidad para monitorear los problemas que estén en proceso de resolución.	
Condiciones de ejecución: Debe existir algún problema en proceso de resolución.	
Entrada/Pasos de ejecución: ➢ Se muestra un listado con el identificador, el tipo de problema, el estado y las cantidades de tareas enviadas y respuestas recibidas, de cada uno de los problemas en proceso de resolución. ➢ Se actualiza periódicamente la información mostrada.	
Resultado esperado: Se muestra satisfactoriamente como trascurre el proceso de resolución de cada uno de los problemas que estén en dicho proceso.	
Evaluación de la prueba: Prueba satisfactoria.	

Tabla 3.9: HU9_CP9 Monitorear problemas en proceso de resolución.

3.7.2.1 Análisis de resultados

Las pruebas de aceptación fueron aplicadas al sistema en 3 iteraciones. La primera iteración de pruebas se realizó luego de implementadas las historias de usuario planificadas para la primera iteración del sistema, detectándose 8 no conformidades que fueron corregidas durante la siguiente iteración del sistema. La segunda iteración de pruebas se realizó una vez culminada la implementación de las historias de usuario, planificadas para la segunda iteración del sistema, y corregidas las no conformidades de la iteración anterior, detectándose 5 no conformidades que fueron corregidas en la propia iteración del sistema en que fueron detectadas debido a que esta constituye la iteración final. Finalmente se realizó una tercera iteración de pruebas en la que no se detectaron no conformidades obteniéndose un resultado satisfactorio. (Ver Figura 3.6)

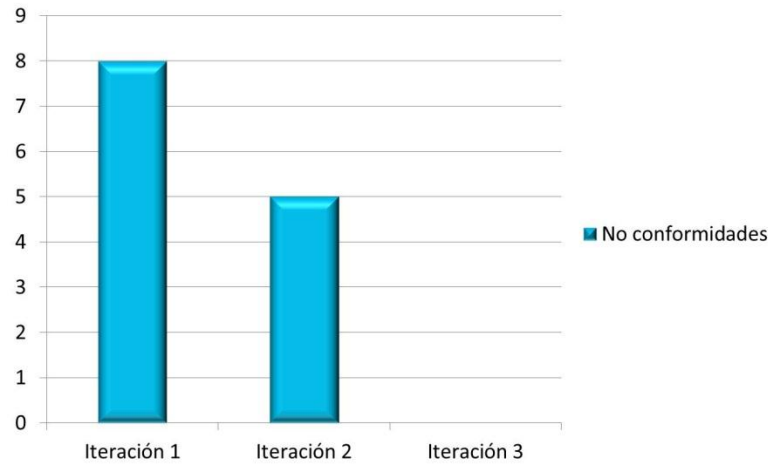


Figura 3.6: Resultados de las pruebas de aceptación por iteraciones.

3.7.3 Pruebas de rendimiento

Las pruebas de rendimiento realizadas a la plataforma permitieron realizar un análisis del comportamiento de los tiempos de respuesta en dependencia de la cantidad de trabajadores conectados. Estas pruebas consistieron en ejecutar un mismo problema mientras varía la cantidad de trabajadores. El problema seleccionado para la ejecución de las pruebas fue el cálculo de los factores primos del número 49604938271294.

3.7.3.1 Análisis de resultados

El problema seleccionado fue procesado de manera centralizada, es decir, utilizando una sola computadora, obteniendo un tiempo de respuesta de 3 segundos, así como de manera distribuida, haciendo uso de la plataforma desarrollada y empleando 1, 5, 10 y 15 trabajadores, obteniéndose un tiempo de respuesta de 40, 15, 7 y 5 segundos respectivamente (Ver Figura 3.7). Esto demuestra que a medida que la cantidad de trabajadores que colaboran con el sistema aumenta los tiempos de respuesta disminuyen, por lo que a pesar de que en este caso el procesamiento distribuido obtuvo un mayor tiempo de respuesta, principalmente debido al costo agregado a la distribución de tareas por la latencia de la red, si se continúa aumentando la cantidad de trabajadores los resultados serán considerablemente mejores a los obtenidos de manera centralizada.

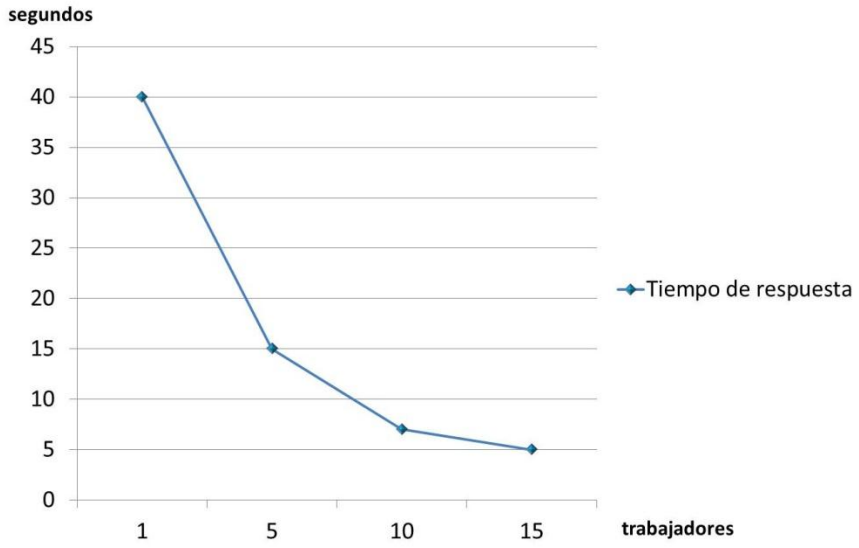


Figura 3.7: Resultados de las pruebas de rendimiento.

3.8 Conclusiones parciales

En el presente capítulo se establecieron los estándares de codificación, que al ser empleados garantizaron que la implementación del sistema se realizara de manera organizada, por lo que se concluye que el código de dicho sistema será fácil de mantener por los programadores.

Además, se describió cómo el sistema garantiza la extensibilidad, la escalabilidad, el tratamiento a fallos, la concurrencia y la transparencia, demostrando que el mismo ha sido desarrollado cumpliendo con los principios de la computación distribuida.

Mediante la realización de pruebas unitarias y de aceptación se comprobó el correcto funcionamiento de cada uno de los componentes del *software* y se verificó la satisfacción del cliente con la solución. A partir de estas pruebas se encontraron un conjunto de no conformidades que fueron la base fundamental para que el sistema mejorara su funcionamiento. También se realizaron pruebas de rendimiento que permitieron demostrar que a medida que la cantidad de trabajadores que colaboran con el sistema aumenta los tiempos de respuesta disminuyen, concluyendo que el sistema con la cantidad adecuada de trabajadores puede resolver el problema en un tiempo menor al que demoraría un solo ordenador. A partir de la realización de estas pruebas se puede concluir que el sistema desarrollado cumple con cada uno de los requerimiento especificados por el cliente.

Conclusiones

En la presente investigación se realizó un estudio de los principales conceptos relacionados con la computación distribuida, a partir del cual se concluye que la computación voluntaria constituye una fuerte alternativa para resolver problemas que requieren una eleva demanda de cómputo, ya que logra reunir gran capacidad de procesamiento mediante la utilización de ordenadores personales distribuidos por todo el mundo.

Se realizó un análisis de los principales sistemas de procesamiento distribuido enfocados a la computación voluntaria, tanto a nivel nacional como internacional, a partir del cual se pudo concluir que existen sistemas capaces de realizar las funcionalidades deseadas, pero requieren de la instalación previa de un *software* en las computadoras que donan sus recursos a la ejecución de problemas, por lo que su utilización no resulta factible.

Se desarrolló una plataforma de procesamiento distribuido capaz de resolver problemas que requieren de una elevada demanda de cómputo. La misma utiliza un *plugin* en el navegador como *software* para los trabajadores, dando la posibilidad a los usuarios que no cuentan con privilegios administrativos sobre las computadoras, de donar los recursos computacionales de las mismas a la ejecución de este tipo de problemas. Por lo que se puede llegar a la conclusión de que el sistema desarrollado es una plataforma no invasiva, que posibilita que una mayor cantidad de usuarios puedan colaborar con la resolución de problemas, lo que permite alcanzar una gran capacidad de procesamiento.

Mediante la realización de pruebas unitarias y de aceptación se comprobó el correcto funcionamiento de cada uno de los componentes del *software* y se verificó la satisfacción del cliente con la solución. También se realizaron pruebas de rendimiento que permitieron evaluar el comportamiento de los tiempos de respuesta en dependencia de la cantidad de trabajadores conectados al sistema. A partir de la realización de estas pruebas se puede concluir que el sistema desarrollado cumple con cada uno de los requerimiento especificados por el cliente.

Recomendaciones

Como parte del proceso de mejora continua del desarrollo de *software* se propone la siguiente recomendación que tributaría a un producto de mayor calidad.

- Desarrollar una consola de administración web que permita la gestión de los problemas mediante la web y atender a múltiples usuarios.

Referencias bibliográficas

1. Schickard, W. (2012). *Historia de la computación*. Obtenido el 20 de Octubre del 2012 desde http://www.colegiomedizbolio.com.mx/site_2012/wp-content/uploads/2012/08/Historia-de-la-computaci%C3%B3n.doc.

2. Solano, Y. Z. S.; Fraile, L. F. y Ocupacional, S. (1997). *Introducción a la informática*. Obtenido el 20 de Octubre del 2012 desde http://www.informatica.bligoo.com.co/media/users/8/418866/files/31285/INTRODUCCI_N_A_LA_INFORM_TIC123.docx.

3. Moreno, C. (2008). *Plataforma Computing@ home*. Tesis de Pregrado, Universidad Autónoma de Barcelona.

4. Duquesne, M. (2007). *Herramientas para la producción de materiales didácticos para las modalidades de enseñanza semipresencial y a distancia*. Obtenido el 10 de Noviembre del 2012 desde http://scielo.sld.cu/scielo.php?script=sci_arttext&pid=S1024-94352007000800008.

5. Mesa, A. y Branch, J. W. (2008). *Implementación de un cluster homogéneo para la resolución de problemas de alta complejidad computacional*. Obtenido el 28 de Octubre del 2012 desde http://www2.unalmed.edu.co/~pruebasminas/index.php?option=com_docman&task=doc_view&gid=435&tmpl=component&format=raw&Itemid=285.

6. Serrano, M. C. M. y Prada, S. A. O. (2001). *ANÁLISIS Y DISEÑO DE UNA ESTRATEGIA DE INTERACCIÓN ENTRE RECURSOS DISTRIBUIDOS DE UNA INFRAESTRUCTURA DE CÓMPUTO EN REDES HETEROGÉNEAS*. Tesis de Pregrado, Universidad Industrial de Santander.

7. Boldrin, F. (2007-2009). *Web Distributed Computing Systems*. Tesis de Doctorado, Universidad de Ferrara.

8. Tanenbaum, A. S. & Van Steen, M. (2001). *Distributed Systems: Principles and Paradigms*. Amsterdam: Pearson Prentice Hall.

9. Coulouris, G.; Dollimore, J. y Kindberg, T. (2001). *Sistemas Distribuidos, Conceptos y Diseño*. Madrid: Addison Wesley.

10. Rojo, J. O. (2003). *Introducción a los Sistemas Distribuidos*. Obtenido el 25 de Octubre del 2012 desde http://augcyl.org/?page_id=231.

11. Foster, I.; Kesselman, C. & Tuecke, S. (2001). *The anatomy of the grid*. Obtenido el 10 de Noviembre del 2012 desde http://books.google.es/books?hl=es&lr=&id=b4LWXLRBRLsC&oi=fnd&pg=PA171&dq=Foster+The+anatomy+of+the+grid+&ots=GSRoDdR8RX&sig=Rp-rbyAwA_asVDODJRbouLc5Uxo#v=onepage&q&f=false.

12. Mersenne Research, Inc. (1996-2013). *Great Internet Mersenne Prime Search GIMPS*. Obtenido el 28 de Octubre del 2012 desde http://www.mersenne.org/.

13. Alexandrov, A. D; Ibel, M.; Schauser, K. E. & Scheiman, C. J. (1997). *SuperWeb: research issues in Java-based global computing*. Obtenido el 20 de Noviembre del 2012 desde http://secs.ceas.uc.edu/~raosa/research/grid/javelin/96-superweb.pdf.

14. Nisan, N.; London, S.; Regev, O. & Camiel, N. (1998). *Globally distributed computation over the internet-the popcorn project*. Obtenido el 15 de Noviembre del 2012 desde www.cs.huji.ac.il/~popcorn/documentation/popcorn-submit.doc.

15. Kedem, Z.; Baratloo, A.; Karaul, M. & Wyckoff, P. (1997). *Charlotte: Metacomputing on the Web*. Obtenido el 15 de Noviembre del 2012 desde http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.39.5171&rep=rep1&type=pdf.

16. Sarmenta, L. (1998). *Bayanihan: Web-based volunteer computing using Java*. Obtenido el 24 de Noviembre del 2012 desde http://link.springer.com/chapter/10.1007/3-540-64216-1_67#.

17. California, U. (2012). *SETI@home*. Obtenido el 28 de Octubre del 2012 desde http://setiathome.berkeley.edu/.

18. Vijay Pande, U. & Stanford, U. (2000-2012). *Folding@home Distributed Computing*. Obtenido el 28 de Octubre del 2012 desde http://folding.stanford.edu/Spanish/HomePage.

19. California, U. (2013). *BOINC*. Obtenido el 28 de Octubre del 2012 desde http://boinc.berkeley.edu/.

20. INRIA/IN2P3. (2000-2008). *XtremWeb: the Open Source Platform for Desktop Grids*. Obtenido el 28 de Octubre del 2012 desde http://www.xtremweb.net/.

21. XtremWeb-CH. (2012). *XtremWeb-CH*. Obtenido el 2 de Noviembre del 2012 desde http://www.xtremwebch.net/index.html.

22. l'Accélérateur Linéaire, L. (2012). *XtremWeb-HEP*. Obtenido el 2 de Noviembre del 2012 desde http://www.xtremweb-hep.org/spip.php?rubrique13.

23. Michigan, U. (2010). *Predictor@home-BOINC*. Obtenido el 28 de Octubre del 2012 desde http://boinc.berkeley.edu/wiki/Predictor@home.

24. Washington, U. (2013). *Rosetta@home*. Obtenido el 28 de Octubre del 2012 desde http://boinc.bakerlab.org/rosetta/.

25. Allen, B. (2013). *Einstein@Home*. Obtenido el 28 de Octubre del 2012 desde http://einstein.phys.uwm.edu/.

26. Lodygensky, O.; Fedak, G.; Neri, V.; Cordier, A. & Cappello, F. (2003). *Augernome & XtremWeb: Monte Carlos computation on a global computing platform*. Obtenido el 22 de Noviembre del 2012 desde http://arxiv.org/pdf/cs/0307066v1.pdf.

27. Jorge, J. y Sánchez, E. (2011). *Análisis y Mejoras de Sistemas de Cómputo Voluntario*. Tesis de Maestría, Universidad Nacional de Córdoba.

28. García, C. R. & Miralles, D. M. (2009). *T-arenal v2.0: Desarrollo del back-end*. Tesis de Pregrado, Universidad de las Ciencias Informáticas.

29. Keane, T. (2004). *A general-purpose heterogeneous distributed computing system*. Obtenido el 24 de Noviembre del 2012 desde http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.101.8580&rep=rep1&type=pdf.

30. Aguilera, L.; García, C. R. & Resin, C. (2012). *A Multi-Server Approach for Distributing Tasks.*

31. Avison, D. & Fitzgerald, G. (2003). *Information systems development: methodologies, techniques and tools*. Maidenhead: McGraw Hill.

32. Figueroa, R. G.; Solís, C. J. y Cabrera, A. A. (2008). *Metodologías Tradicionales vs. Metodologías Ágiles*. Obtenido el 7 de Noviembre del 2012 desde http://tg-tatiana-oquendo.googlecode.com/svn/trunk/articulo-metodologia-de-sw-formato.doc.

33. Carrillo, I.; Pérez, R. y Rodríguez, A. (2008). *METODOLOGÍA DE DESARROLLO DEL SOFTWARE*. Obtenido el 7 de Noviembre del 2012 desde http://solusoft-g11.googlecode.com/files/Metodologias%20de%20desarrollo.pdf.

34. Cáceres, P.; Marcos, E. y Kybele, G. (2001). *Procesos ágiles para el desarrollo de aplicaciones Web*. Obtenido el 7 de Noviembre del 2012 desde http://dlsi.ua.es/webe01/articulos/s112.pdf.

35. Amaro, S. D. y Valverde, J. C. (2007). *Metodologías Ágiles*. Tesis de Pregrado, Universidad Nacional de Trujillo.

36. Canós, J.; Letelier, P. y Penadés, M. C. (2003). *Metodologías Ágiles en el desarrollo de Software*. Obtenido el 10 de Noviembre del 2012 desde http://noqualityinside.com.ar/nqi/nqifiles/XP_Agil.pdf.

37. Cockburn, A. (2005). *Crystal Clear: A Human-Powered Methodology for Small Teams*. Salt Lake: Addison-Wesley.

38. Fuentes, L. y Vallecillo, A. (2004). *Una introducción a los perfiles UML*. Obtenido el 24 de Noviembre del 2012 desde http://c3po.eui.upm.es/file.php/24/UMLProfiles-Novatica04.pdf.

39. Owen, M. & Raj, J. (2003). *BPMN and business process management*. Obtenido el 24 de Noviembre del 2012 desde http://www.bpmn.org/Documents/6AD5D16960.BPMN_and_BPM.pdf.

40. Booch, G.; Rumbaugh, J.; Jacobson, I.; Sáez, J. y García, J. (1999). *El lenguaje unificado de modelado*. Madrid: Addison Wesley.

41. González, P.; González, A. A. y Gallud, J. A. (1995). *Herramientas CASE:¿ Cómo incorporarlas con éxito en nuestra organización?*. Obtenido el 26 de Noviembre del 2012 desde http://www.uclm.es/ab/educacion/ensayos/pdf/revista10/10_17.pdf.

42. G. S. I. (2007). *Rational Rose Enterprise*. Obtenido el 26 de Noviembre del 2012 desde http://www.rational.com.ar/herramientas/roseenterprise.html.

43. Altova, Inc. (2010). *ALTOVA*. Obtenido el 26 de Noviembre del 2012 desde http://www.altova.com/umodel.html.

44. Visual Paradigm, Int. (2006). *Visual Paradigm*. Obtenido el 26 de Noviembre del 2012 desde http://www.visual-paradigm.com/product/vpuml/.

45. Madera, A. Y. y Torres, J. C. (2013). *Diseño de componente para el framework zathura code que permita la migración de código fuente de formularios con tecnología oracle forms 6i a componentes con tecnología java*. Tesis de Pregrado, Universidad de San Buenaventura.

46. Flanagan, D. (2011). *JavaScript: the definitive guide*. O'Reilly Media.

47. Zukowski, J. (2003). *Programación en Java 2 J2SE 1.4*. Madrid: Anaya Multimedia.

48. Oaks, S. (2001). *Java security*. Sebastopol: O'Reilly Media.

49. Arias, L. F. (2012). *Diseño, desarrollo y publicación de la página Web de "Electrogar"*. Tesis de Pregrado, Universidad Nacional Abierta y a Distancia.

50. Oracle, C. (2013). *NetBeans IDE Features*. Obtenido el 3 de Diciembre 2012 desde https://netbeans.org/features/index.html.

51. Caponi, M.; Rodríguez, P.; Rodríguez, L. y Rivero, D. (2008). *Mensajería en sistemas de información*. Informe de Proyecto de Grado, Universidad de la República.

52. Jankolovska, S. & Zherajikj, B. (2012). *Access Point Framework: - towards a more scalable ESB solution*. Tesis de Pregrado, KTM.

53. Piël, N. (2010). *ZeroMQ an introduction*. Obtenido el 26 de Noviembre del 2012 desde http://nichol.as/zeromq-an-introduction.

54. AMQP. (2013). *Products and Success Stories | AMQP*. Obtenido el 26 de Noviembre del 2012 desde http://www.amqp.org/about/examples.

55. StatCounter. (1999-2013). *StatCounter Global Stats*. Obtenido el 3 de Diciembre del 2012 http://gs.statcounter.com/.

56. Mozilla, F. (2012). *Mozilla Public License*. Obtenido el 12 de Diciembre del 2012 http://www.mozilla.org/MPL/.

57. Rivas, D. (2012). *Mozilla Firefox-Descargar*. Obtenido el 12 de Diciembre del 2012 http://mozilla-firefox.uptodown.com/.

58. Mozilla, F. (2012). *Builder y SDK :: Centro de Desarrolladores :: Complementos para Firefox*. Obtenido el 12 de Diciembre del 2012 desde https://addons.mozilla.org/es/developers/builder.

59. Mozilla, F. (2012). *Add-on SDK Documentation*. Obtenido el 12 de Diciembre del 2012 desde https://addons.mozilla.org/en-US/developers/docs/sdk/latest/.

60. Pérez, J. P. (2007). *Diseño de un sistema de supervisión remota para equipos de climatización en centrales telefónicas y nodos IP*. Tesis de Pregrado, Instituto Tecnológico de Costa Rica.

61. Apache Software, F. (2012). *The Apache HTTP Server Project*. Obtenido el 12 de Diciembre del 2012 desde http://httpd.apache.org/.

62. Apache Software, F. (2012). *Módulos de MultiProcesamiento (MPMs)*. Obtenido el 10 de Enero del 2013 desde http://httpd.apache.org/docs/2.2/es/mpm.html.

63. Betzler, B. (2010). *SmartCloud tip: Install IIS web server on Windows 2008 R2*. Obtenido el 23 de Febrero del 2013 desde http://www.ibm.com/developerworks/cloud/library/cl-iiswindows/.

64. Templin, R. (2007). *Introduction to IIS Architectures: The Official Microsoft IIS Site*. Obtenido el 2 de Marzo del 2013 desde http://www.iis.net/learn/get-started/introduction-to-iis/introduction-to-iis-architecture#Components.

65. Sánchez, S. (2012). *Análisis y UML "Modelo Conceptual"*. Obtenido el 28 de Noviembre del 2012 desde http://es.scribd.com/doc/86842976/unidad5madmodeladoanalisismodeloconceptual-13026968704227-phpapp01.

66. Letelier, P. (2006). *Metodologías ágiles para el desarrollo de software: eXtreme Programming (XP)*. Obtenido el 24 de Noviembre del 2012 desde http://www.cyta.com.ar/ta0502/b_v5n2a1.htm.

67. Hernández, A. (2010). *REQUISITOS A PARTIR DEL MODELO DEL NEGOCIO*. Obtenido el 16 de Enero del 2013 desde http://rii.cujae.edu.cu/index.php/revistaind/article/viewArticle/132.

68. de la Torre, C.; Castro, U. Z.; Calvarro, N. J.; Ramos, M. A.; Manteiga, C.; Cortés, F. y García, I. (2010). *Guía de arquitectura N-Capas orientada al dominio con .NET 4.0*. Krasis Press.

69. Martínez, J. F. J. (2000). *Guía de construcción de software en Java con patrones de diseño*. Tesis de Pregrado, Universidad de Oviedo.

70. Larman, C. (1999). *UML y patrones*. Pearson.

71. Cooper, J. W. (2002). *Introduction to design patterns in C#*. Addison-Wesley.

72. Mendoza, M. A. (2004). *Metodologías De Desarrollo De Software*. Obtenido el 4 de Diciembre del 2012 desde http://www.willydev.net/InsiteCreation/V1.0/descargas/cualmetodologia.pdf.

73. Joskowicz, J. (2008). *Reglas y prácticas en Extreme Programming*. Obtenido el 4 de Diciembre del 2012 desde http://iie.fing.edu.uy/~josej/docs/XP%20-%20Jose%20Joskowicz.pdf.

Bibliografía consultada

Lüer, A. y Bustos, J. (2009). *Laboratorios de computación como cluster HPC*. Obtenido el 23 de Noviembre del 2012 desde http://ceur-ws.org/Vol-558/Art_25.pdf.

Bianchi, O. M. y Repetto, A. J. (2012). *Computación Distribuida para Seguridad Informática*. Obtenido el 23 de Noviembre del 2012 desde http://sedici.unlp.edu.ar/bitstream/handle/10915/19212/Documento_completo.pdf?sequence=1.

Serrano, F. (2008). *El proyecto Ibercivis de computación voluntaria: un esbozo*. Obtenido el 25 de Noviembre del 2012 desde http://www.rediris.es/difusion/publicaciones/boletin/82-83/ponencia1.2A.pdf.

Chávez, F.; Guisado, J. L.; Lombraña, D. y Fernández, F. (2007). *Una Herramienta de Programación Genética Paralela que Aprovecha Recursos Públicos de Computación*. Obtenido el 25 de Noviembre del 2012 desde http://personal.us.es/jlguisado/publicaciones/MAEB2007_preprint.pdf.

Echeverry, L. M. y Delgado, L. E. (2007*). CASO PRÁCTICO DE LA METODOLOGÍA ÁGIL XP AL DESARROLLO DE SOFTWARE*. Tesis de Pregrado, UNIVERSIDAD TECNOLÓGICA DE PEREIRA.

Hernán, S. M. (2004). *Diseño de una Metodología Ágil de Desarrollo de Software*. Tesis de Pregrado, Universidad de Buenos Aires.

Schildt, H. (2001). *Java 2 Manual de referencia*. Madrid: McGraw-Hill.

Pérez, I. G.; Sánchez, A. J. y Hernández, D. V. (2002). *Java Threads (Hilos en Java)*. Departamento de Informática y Automática. Universidad de Salamanca.

Frentzen, J. y Sobotka, H. (1999). *Superutilidades para JavaScript*. Madrid: McGraw-Hill.

Alvarado, R. J. y Loría, J. A. (2008). *Secretos de Herencia y Polimorfismo junto con Reflexión*. Obtenido el 10 de Enero del 2013 desde http://www.di-mare.com/adolfo/cursos/2008-2/pp-OOPrflx.pdf.

Speegle, G. D. (2002). *Practical Guide for Java Programmers*. San Francisco: Morgan Kaufmann.

Oaks, S. & Wong, H. (2004). *Java Threads. Sebastopol*: O'Reilly Media.

Goodman, D.; Morrison, M. & Eich, B. (2007). JavaScript Bible. New York: John Wiley & Sons, Inc.

Glosario de términos

AMQP (acrónico en inglés de Advanced Message Queuing Protocol): es un protocolo de estándar abierto en la capa de aplicaciones de un sistema de comunicación orientado a mensajes.

CISED (Centro de Identificación y Seguridad Digital): centro de desarrollo de soluciones integrales, productos y servicios en el campo de la identificación y la seguridad digital. Se encuentra estrechamente vinculado a la empresa comercializadora Albet SA y a la Universidad de la Ciencias Informáticas, lo que ofrece una amplia posbilidad a su capital humano para la investigación, desarrollo e innovación. Cuenta además con un equipo de especialistas de alto nivel, con gran experiencia y reconocimiento a nivel nacional e internacional.

GNU LGPL (acrónico en ingles de GNU *Lesser General Public License*): es una licencia de *software* creada por la *Free Software Foundation* que pretende garantizar la libertad de compartir y modificar el *software* cubierto por ella, asegurando que el *software* es libre para todos sus usuarios.

***Middleware*:** capa de *software* mediante la cual el sistema puede abstraerse de las redes, *hardware*, lenguajes de programación y sistemas operativos, sobre los que se ejecuta.

Organización virtual (OV): conjunto de individuos y/o instituciones definidas por reglas que controlan el modo en que comparten sus recursos. Básicamente, son organizaciones unidas para lograr objetivos comunes.

PFLOPS: las Operaciones de Coma Flotante por Segundo (FLOPS, acrónico en inglés de *Floating Point Operations per Second*) son una medida del rendimiento de una computadora, especialmente en cálculos científicos que requieren un gran uso de operaciones de coma flotante. Las computadoras exhiben un amplio rango de rendimientos en coma flotante, por lo que a menudo se usan unidades mayores que el FLOPS como megaFLOPS (MFLOPS, 10^6 FLOPS), gigaFLOPS (GFLOPS, 10^9 FLOPS), teraFLOPS (TFLOPS, 10^{12} FLOPS), petaFLOPS (PFLOPS, 10^{15} FLOPS), exaFLOPS (EFLOPS, 10^{18} FLOPS). Por ejemplo: 1 petaflop equivale a 1 000 000 000 000 000 cálculos por segundo.

RabbitMQ: servidor de mensajería multiplataforma que implementa completamente el protocolo AMQP.

TIC (Tecnologías de la Información y las Comunicaciones): se refiere al conjunto de herramientas, aplicaciones y soportes informáticos que permiten procesar, almacenar y representar información en las más variadas formas.

Anexos

Anexo I: Comparación entre las metodologías ágiles y las metodologías tradicionales

Metodologías Ágiles	Metodologías Tradicionales
Basadas en heurísticas provenientes de prácticas de producción de código.	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo.
Especialmente preparadas para cambios durante el proyecto.	Cierta resistencia a los cambios.
Impuestas internamente (por el equipo).	Impuestas externamente.
Proceso menos controlado, con pocos principios.	Proceso mucho más controlado, con numerosas políticas y normas.
No existe contrato tradicional o al menos es bastante flexible.	Existe un contrato prefijado.
El cliente es parte del equipo de desarrollo.	El cliente interactúa con el equipo de desarrollo mediante reuniones.
Grupos pequeños (menos de 10 integrantes) y trabajando en el mismo sitio.	Grupos grandes y posiblemente distribuidos.
Pocos artefactos.	Más artefactos.
Pocos roles.	Más roles.
Menos énfasis en la arquitectura del *software*.	La arquitectura del *software* es esencial y se expresa mediante modelos.

Tabla I.1: Diferencia entre las metodologías ágiles y las metodologías tradicionales [35].

Anexo II: Historias de usuario

Historia de Usuario	
Código: HU_2	**Nombre historia de usuario:** Mostrar listado de tipos de problemas.
Modificación de historia de usuario número: ninguna.	
Programador: Heidy Luis Díaz.	**Iteración asignada:** 2.
Prioridad: Baja.	**Puntos estimados:** 1 (semanas).
Riesgo en desarrollo: Bajo.	**Puntos reales:** 1 (semanas).
Descripción: Se muestra un listado con todos los tipos de problemas existentes.	
Observaciones: En caso de no existir ningún tipo de problema, se le notifica al usuario.	

Tabla II.1: HU_2 Mostrar listado de tipos de problemas.

Historia de Usuario	
Código: HU_3	**Nombre historia de usuario:** Eliminar tipo de problema.
Modificación de historia de usuario número: ninguna.	
Programador: Heidy Luis Díaz.	**Iteración asignada:** 2.
Prioridad: Media.	**Puntos estimados:** 1 (semanas).
Riesgo en desarrollo: Medio.	**Puntos reales:** 1 (semanas).
Descripción: Se muestra un listado con todos los tipos de problemas existentes, del cual el usuario debe seleccionar el que desea eliminar. Una vez seleccionado, el sistema muestra un mensaje de confirmación.	
Observaciones: En caso de no existir ningún tipo de problema, se le notifica al usuario. En caso de no seleccionar algún tipo de problema, se le notifica al usuario. Una vez que el tipo de problema es eliminado, se le notifica al usuario.	

Tabla II.2: HU_3 Eliminar tipo de problema.

Historia de Usuario	
Código: HU_5	**Nombre historia de usuario:** Mostrar listado de problemas.
Modificación de historia de usuario número: ninguna.	
Programador: Sael José Berrillo Borrero.	**Iteración asignada:** 2.
Prioridad: Baja.	**Puntos estimados:** 1 (semanas).
Riesgo en desarrollo: Bajo.	**Puntos Reales:** 1 (semanas).
Descripción: Se muestra un listado con todos los problemas existentes.	
Observaciones: En caso de no existir ningún problema, se le notifica al usuario.	

Tabla II.3: HU_5 Mostrar listado de problemas.

Historia de Usuario	
Código: HU_6	**Nombre historia de usuario:** Eliminar problema.
Modificación de historia de usuario número: ninguna.	
Programador: Sael José Berrillo Borrero.	**Iteración asignada:** 2.
Prioridad: Media.	**Puntos estimados:** 1 (semanas).
Riesgo en desarrollo: Medio.	**Puntos reales:** 1 (semanas).
Descripción:	

Se muestra un listado con todos los problemas existentes, del cual el usuario debe seleccionar el que desea eliminar. Una vez seleccionado el sistema muestra un mensaje de confirmación.
Observaciones: En caso de no existir ningún problema, se le notifica al usuario. En caso de no seleccionar algún problema, se le notifica al usuario. Una vez que el problema es eliminado, se le notifica al usuario.

Tabla II.4: HU_6 Eliminar problema.

Historia de Usuario	
Código: HU_10	**Nombre historia de usuario:** Detener ejecución de un problema.
Modificación de historia de usuario número: ninguna.	
Programador: Heidy Luis Díaz.	**Iteración asignada:** 2.
Prioridad: Baja.	**Puntos estimados:** 1 (semanas).
Riesgo en desarrollo: Bajo.	**Puntos reales:** 1 (semanas).
Descripción: Una vez mostrado el listado de problemas en proceso de resolución, el usuario tiene la posibilidad de detener la ejecución de cada uno de ellos, seleccionando del listado el que desea detener. Después de seleccionado y marcada la opción de detener, el sistema detiene la ejecución de dicho problema.	
Observaciones: En caso de marcar la opción de detener sin seleccionar algún problema, se le notifica al usuario. Si el problema que se desea detener ya se encuentra detenido, se le notifica al usuario. Si el problema que se desea detener se encuentra en espera o concluyendo, no podrá ser detenido y se le notifica al usuario.	

Tabla II.5: HU_10 Detener ejecución de un problema.

Historia de Usuario	
Código: HU_11	**Nombre historia de usuario:** Reanudar ejecución de un problema.
Modificación de historia de usuario número: ninguna.	
Programador: Heidy Luis Díaz.	**Iteración asignada:** 2.
Prioridad: Baja.	**Puntos estimados:** 1 (semanas).
Riesgo en desarrollo: Bajo.	**Puntos reales:** 1 (semanas).
Descripción: Una vez mostrado el listado de problemas en proceso de resolución, el usuario tiene la posibilidad de reanudar la ejecución de cada uno de ellos, seleccionando del listado el que desea reanudar. Después de seleccionado y marcada la opción de reanudar, el sistema	

reanuda la ejecución de dicho problema.
Observaciones: En caso de marcar la opción de reanudar sin seleccionar algún problema, se le notifica al usuario. Si el problema que se desea reanudar ya se encuentra ejecutándose, se le notifica al usuario. Si el problema que se desea reanudar se encuentra en espera o concluyendo, no podrá ser reanudado y se le notifica usuario.

Tabla II.6: HU_11 Reanudar ejecución de un problema.

Historia de Usuario	
Código: HU_12	**Nombre historia de usuario:** Salvar servidor.
Modificación de historia de usuario número: ninguna.	
Programador: Sael José Berrillo Borrero.	**Iteración asignada:** 1.
Prioridad: Media.	**Puntos estimados:** 1 (semanas).
Riesgo en desarrollo: Bajo.	**Puntos reales:** 1 (semanas).
Descripción: Se almacena en la máquina servidor toda la información gestionada por el sistema.	
Observaciones:	

Tabla II.7: HU_12 Salvar servidor.

Historia de Usuario	
Código: HU_13	**Nombre historia de usuario:** Restaurar servidor.
Modificación de historia de usuario número: ninguna.	
Programador: Sael José Berrillo Borrero.	**Iteración asignada:** 1.
Prioridad: Media.	**Puntos estimados:** 2 (semanas).
Riesgo en desarrollo: Medio.	**Puntos reales:** 2 (semanas).
Descripción: Al iniciar el sistema se obtiene toda la información guardada en el disco duro de la máquina servidor. Toda esta información es cargada nuevamente en el sistema, el cual queda en el mismo estado en que se encontraba antes de ser apagado.	
Observaciones: En caso de no existir la información requerida, se le notifica al usuario. En caso de que la información no esté en el formato requerido, se le notifica al usuario.	

Tabla II.8: HU_13 Restaurar servidor.

Anexo III: Plan de entregas

Entregable	Iteración 1	Iteración 2

Plataforma de procesamiento distribuido.	Febrero del 2013.	Mayo del 2013.

Tabla III.1: Plan de entregas.

Anexo IV: Plan de iteraciones

Iteración	No. HU	Historia de usuario	Duración estimada (semanas)
1	HU_1	Crear tipo de problema.	11
	HU_4	Resolver problema.	
	HU_7	Mostrar listado de problemas resueltos.	
	HU_8	Mostrar detalles de un problema resuelto.	
	HU_9	Monitorear problemas en proceso de resolución.	
	HU_12	Salvar servidor.	
	HU_13	Restaurar servidor.	
2	HU_2	Mostrar listado de tipos de problemas.	9
	HU_3	Eliminar tipo de problema.	
	HU_5	Mostrar listado de problemas.	
	HU_6	Eliminar problema.	
	HU_10	Detener ejecución de un problema.	
	HU_11	Reanudar ejecución de un problema.	

Tabla IV.1: Plan de iteraciones.

Anexo V: Tarjetas CRC

Environment	
Responsabilidades	Colaboradores
Abrir conexión (establece un canal de comunicación con el *middleware* de mensajería).	
Cerrar conexión (cierra el canal de comunicación con el *middleware* de mensajería).	
Enviar tarea (envía la tarea a la cola de tareas del *middleware* de mensajería).	RunTime. Task.
Recibir respuesta (recibe la respuesta enviada por el *middleware* de mensajería).	RunTime. Response.
Obtener cantidad de trabajadores disponibles	

(obtiene la cantidad de trabajadores disponible suscritos a la cola de tareas del *middleware* de mensajería).	

Tabla V.1: Tarjeta CRC de la clase Environment.

RunTime	
Responsabilidades	Colaboradores
Crear un tipo de problema.	ProblemRunTimeInstance. InformationManager.
Eliminar un tipo de problema.	InformationManager.
Crear un problema (crea una instancia de un tipo de problema).	ProblemRunTimeInstance.
Eliminar un problema (elimina una instancia de un tipo de problema).	
Obtener el problema a ejecutar (determina a qué problema le corresponde ser procesado en cada momento).	ProblemRunTimeInstance. State.
Obtener la próxima tarea (obtiene la próxima tarea que se enviará al *middleware* de mensajería).	ProblemRunTimeInstance. Task. ExecuteTask. EndTask. ProblemTask.
Asignar una respuesta a un problema (asigna la respuesta de una tarea al problema correspondiente).	ProblemRunTimeInstance. Response. State.
Arrancar la ejecución de un problema.	ProblemRunTimeInstance. State.
Detener la ejecución de un problema.	ProblemRunTimeInstance. State.
Poner en espera la ejecución de un problema.	ProblemRunTimeInstance. State.
Concluir la ejecución de un problema.	ProblemRunTimeInstance. State.
Finalizar la ejecución de un problema.	ProblemRunTimeInstance. State.
Obtener el listado de los problemas pendientes.	
Obtener el listado de los problemas resueltos.	

Obtener el listado de problemas detenidos.	
Obtener un problema por el identificador.	ProblemRunTimeInstance.
Salvar tarea.	InformationManager.
Salvar respuesta.	InformationManager.
Restaurar el servidor.	InformationManager.

Tabla V.2: Tarjeta CRC de la clase RunTime.

InformationManager	
Responsabilidades	Colaboradores
Salvar un tipo de problema (salva la información que debe ser persistente del tipo de problema).	
Cargar un tipo de problema (obtiene toda la información, previamente salvada, perteneciente a un tipo de problema).	
Actualizar un tipo de problema (actualiza la información almacenada perteneciente a un tipo de problema).	
Eliminar un tipo de problema (elimina toda la información gualdada perteneciente a un tipo de problema).	
Obtener el listado de los tipos de problemas (obtiene el nombre de los tipos de problemas almacenados).	
Salvar una tarea (salva la información que debe ser persistente de una tarea).	
Cargar una tarea (obtiene la información, previamente salvada, perteneciente a una tarea).	
Cargar el código de una tarea (obtiene el código contenido en el fichero JavaScript perteneciente a una tarea).	
Eliminar una tarea (elimina toda la información guardada perteneciente a una tarea).	
Salvar una respuesta (salva la información que debe ser persistente de una respuesta).	
Cargar una respuesta (obtiene la información,	

previamente salvada, perteneciente a una respuesta).	
Eliminar una respuesta (elimina toda la información guardada perteneciente a una respuesta).	
Salvar el servidor (salva toda la información gestionada por el servidor).	
Restaurar el servidor (obtiene toda la información previamente salvada del servidor).	

Tabla V.3: Tarjeta CRC de la clase InformationManager.

Anexo VI: Aplicación del patrón Experto en el sistema

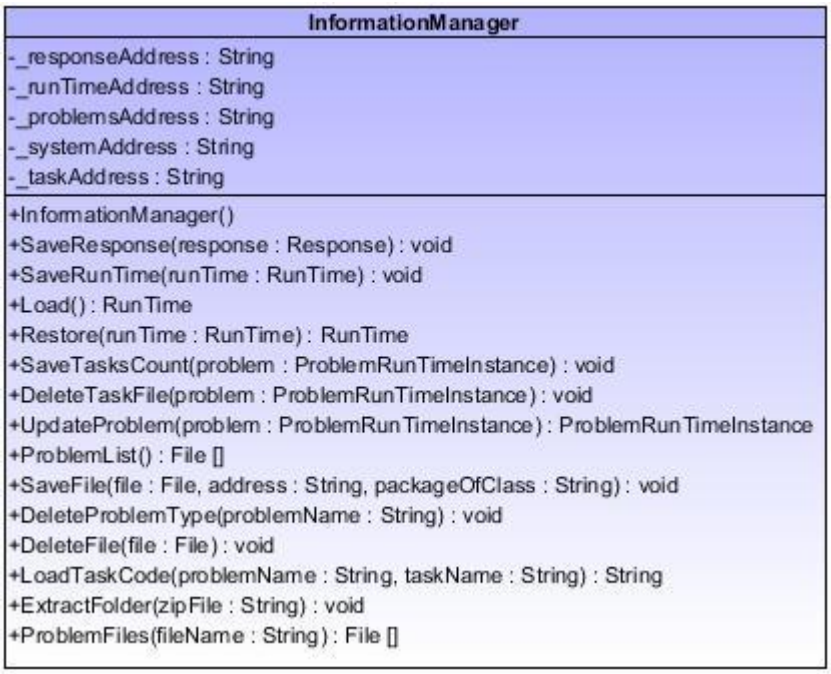


Figura VI.1: Ejemplo de la aplicación del patrón Experto en el sistema.

Anexo VII: Aplicación del patrón Creador en el sistema

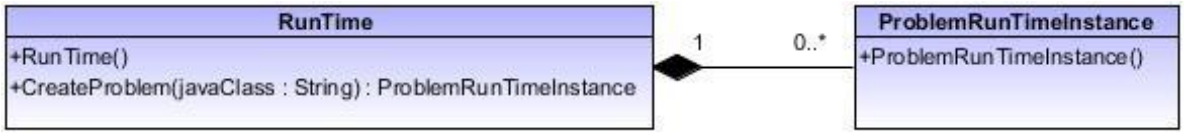


Figura VII.1: Ejemplo de la aplicación del patrón Creador en el sistema.

Anexo VIII: Aplicación del patrón Controlador en el sistema

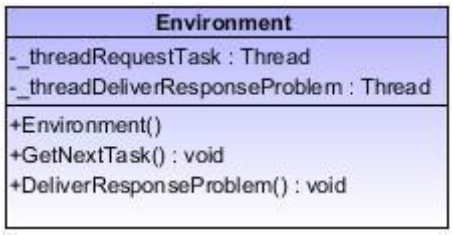


Figura VIII.1: Ejemplo de la aplicación del patrón Controlador en el sistema.

Anexo IX: Aplicación del patrón Alta Cohesión en el sistema

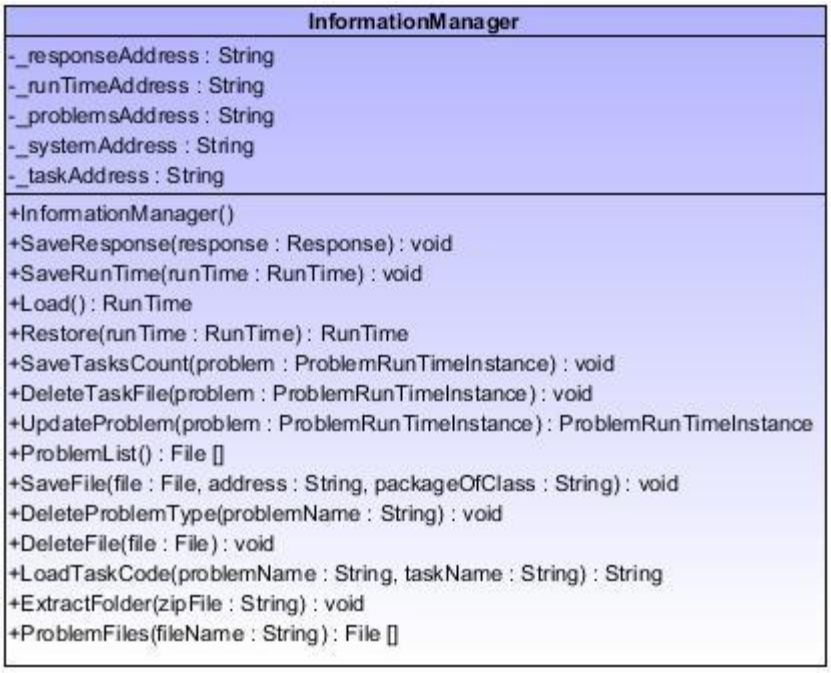


Figura IX.1: Ejemplo de la aplicación del patrón Alta Cohesión en el sistema.

Anexo X: Aplicación del patrón Bajo Acoplamiento en el sistema

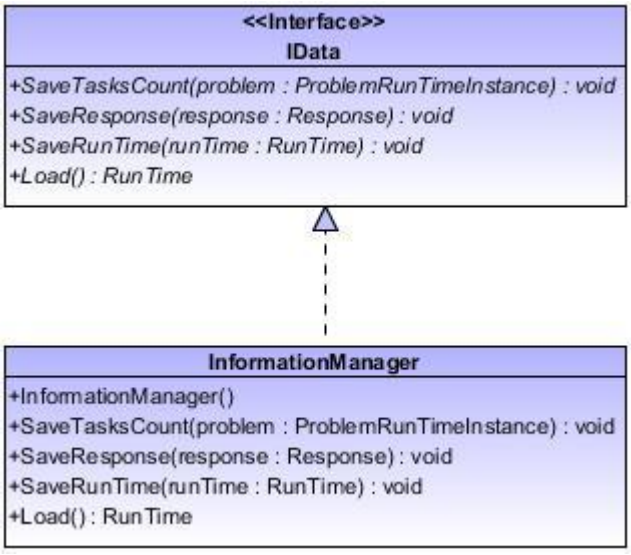


Figura X.1: Ejemplo de la aplicación del patrón Bajo Acoplamiento en el sistema.

Anexo XI: Aplicación del patrón *Singleton* en el sistema

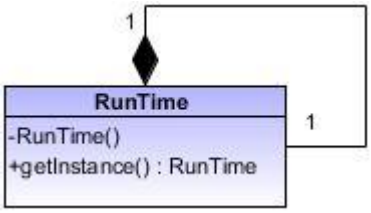


Figura XI.1: Ejemplo de la aplicación del patrón *Singleton* en el sistema.

Anexo XII: Aplicación del patrón *Simple Factory* en el sistema

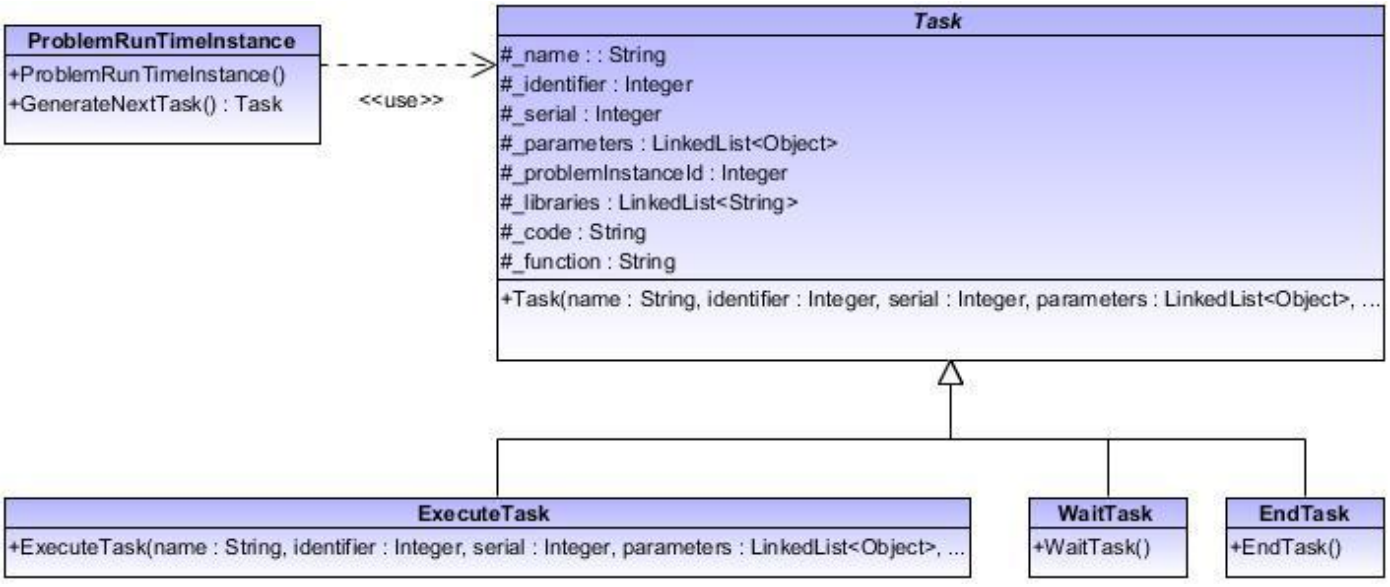


Figura XII.1: Ejemplo de la aplicación del patrón *Simple Factory* en el sistema.

Anexo XIII: Aplicación del patrón *Null Object* en el sistema

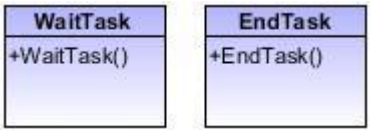
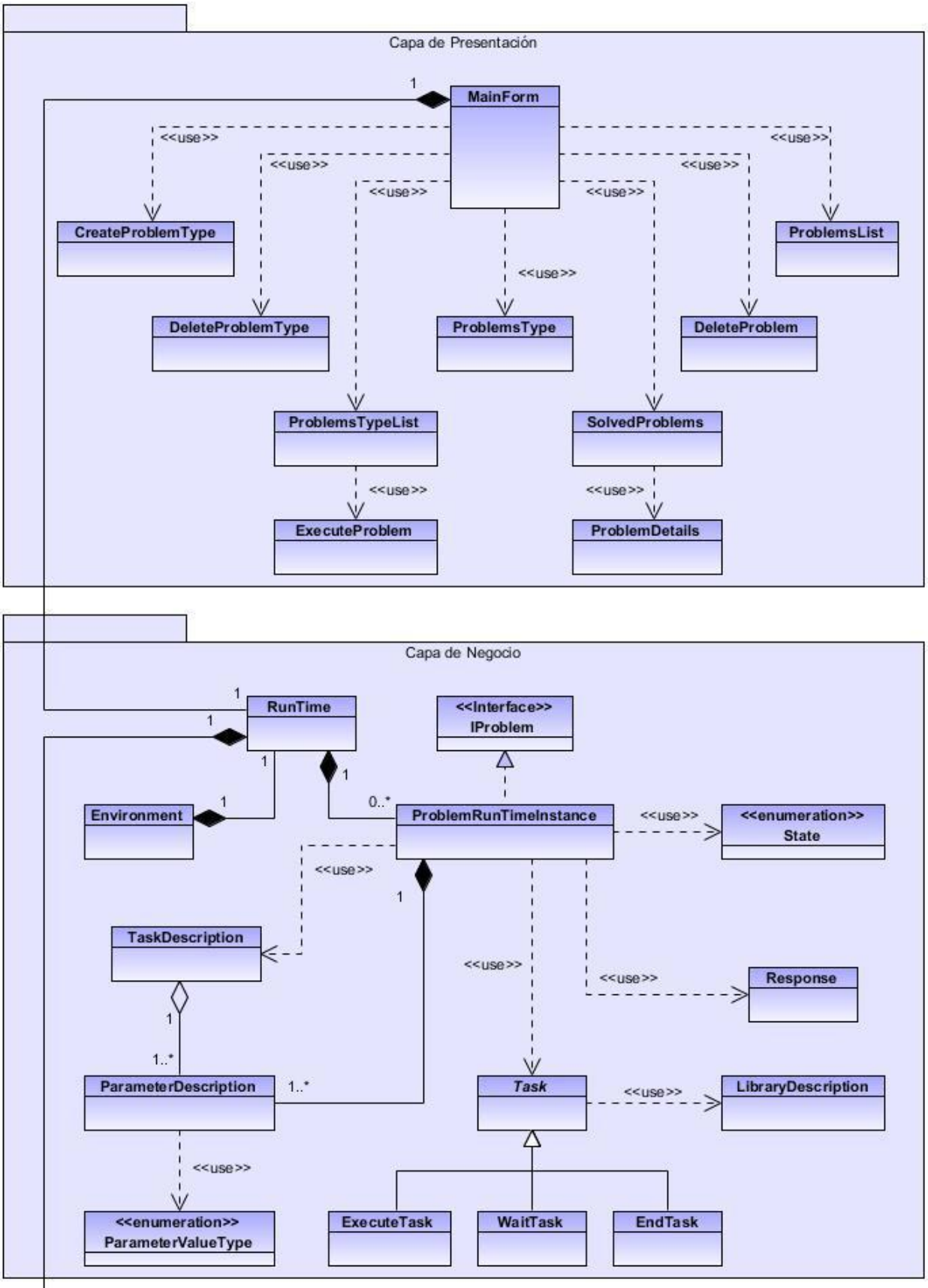


Figura XIII.1: Ejemplo de la aplicación del patrón *Null Object* en el sistema.

Anexo XIV: Diagrama de clases del diseño del servidor


Capa de Presentación
MainForm
CreateProblemType
ProblemsList
DeleteProblemType
ProblemsType
DeleteProblem
ProblemsTypeList
SolvedProblems
ExecuteProblem
ProblemDetails
<<use>>
Capa de Negocio
RunTime
<<Interface>>
IProblem
Environment
ProblemRunTimeInstance
<<enumeration>>
State
TaskDescription
Response
ParameterDescription
Task
LibraryDescription
<<enumeration>>
ParameterValueType
ExecuteTask
WaitTask
EndTask

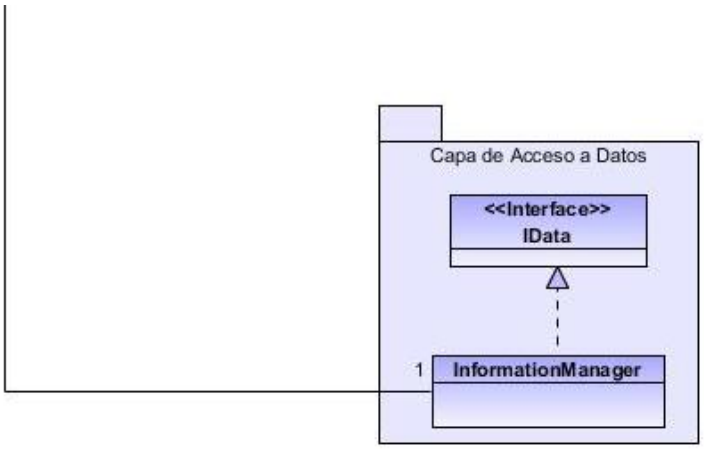
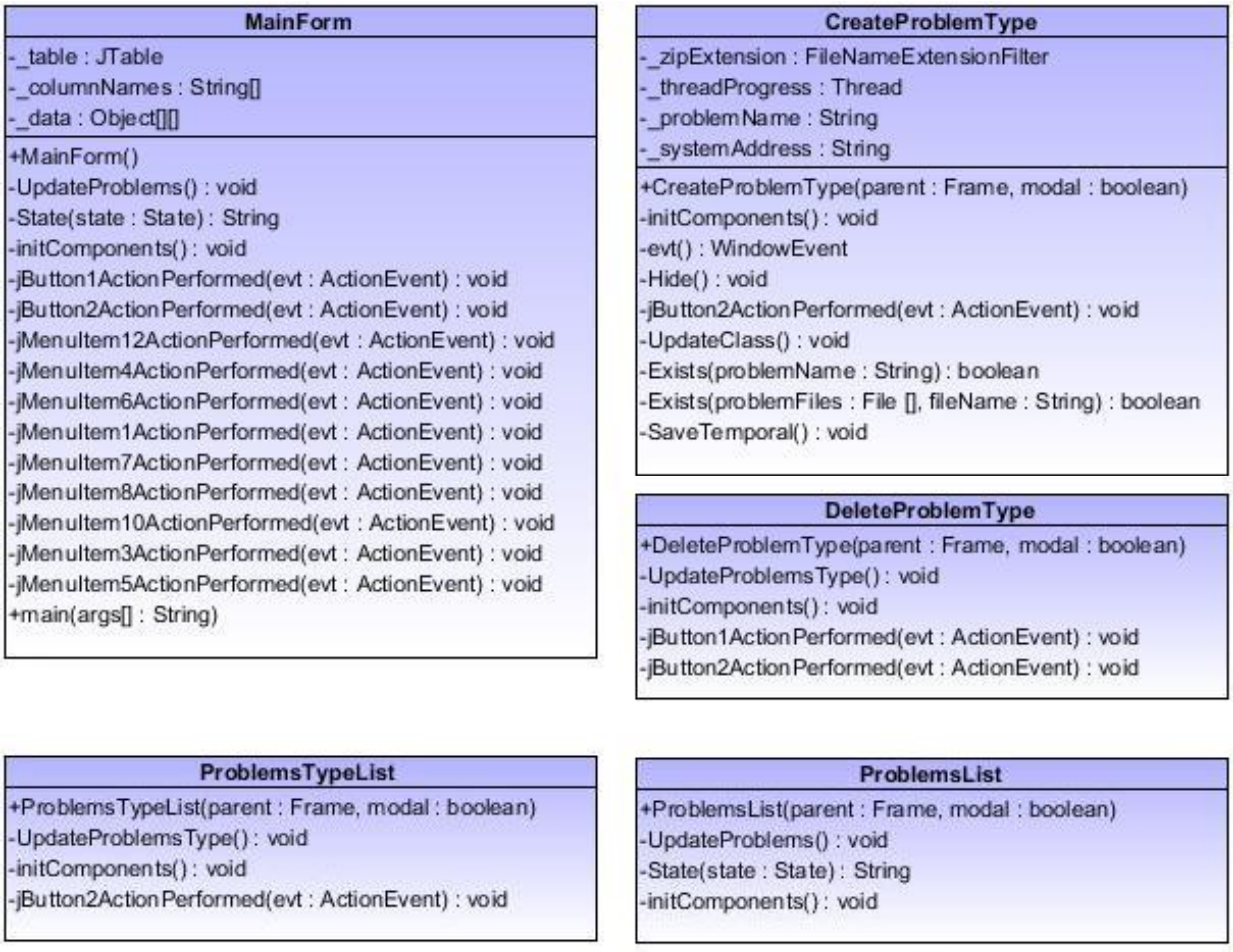


Figura XIV.1: Diagrama de clases del diseño del servidor.

Especificación de las clases de la Capa de Presentación



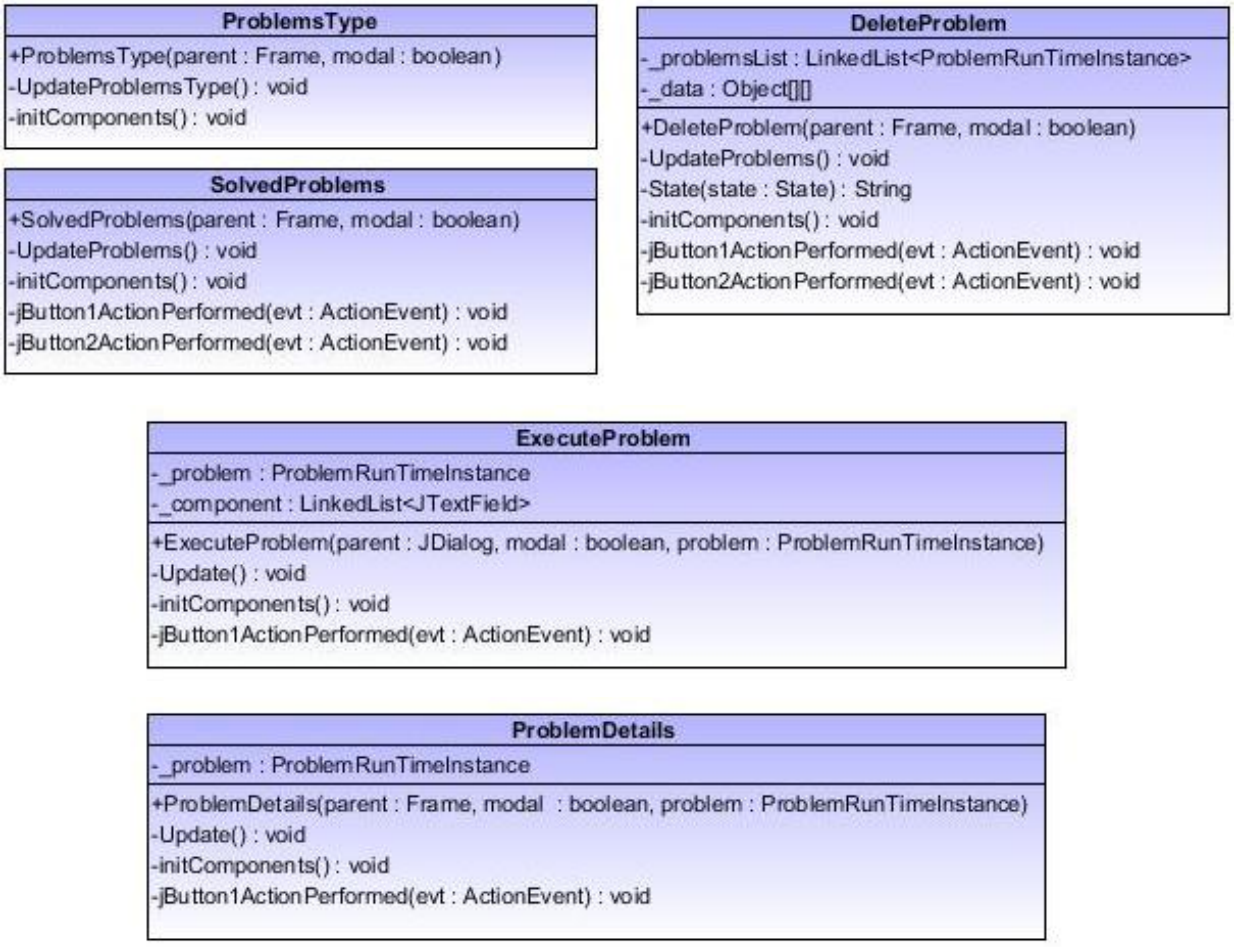


Figura XIV.2: Especificación de la las clases de la Capa de Presentación.

Especificación de las clases de la Capa de Negocio


RunTime
-_runTime : RunTime
-_listPendingProblems : LinkedList<ProblemRunTimeInstance>
-_listResolvedProblems : LinkedList<ProblemRunTimeInstance>
-_countCreatedProblems : int
-_positionProblemToExecute : int
-_countStoppedProblems : int
-RunTime()
+getInstance() : RunTime
+AddProblem(problemRunTimeInstance : ProblemRunTimeInstance) : void
-NextProblemToExecute() : void
+GetNextTask() : Task
+DeliverResponseProblem(response : Response) : boolean
+getInformationManager() : InformationManager
+getListPendingProblems() : LinkedList<ProblemRunTimeInstance>
+getListResolvedProblems() : LinkedList<ProblemRunTimeInstance>
+getCountStoppedProblems() : int
+setCountStoppedProblems(countStoppedProblems : Integer) : void
+getProblemById(problemId : Integer) : ProblemRunTimeInstance
+getCountCreatedProblems() : int
+getPositionProblemToExecute() : int
+CreateProblem(javaClass : String) : ProblemRunTimeInstance
+DeleteProblemById(problemId : Integer) : void
+StoppedProblems() : LinkedList<ProblemRunTimeInstance>
+DeleteProblemsByName(problemName : String) : void
+Play(problem : ProblemRunTimeInstance) : void
+Stop(problem : ProblemRunTimeInstance) : void
+Wait(problem : ProblemRunTimeInstance) : void
+Terminate(problem : ProblemRunTimeInstance) : void
+Conclude(problem : ProblemRunTimeInstance) : void
+SaveResponse(response : Response) : void
+RestartSystem() : void
<<Interface>>
IProblem
+getName() : String
+GetDescription() : String
+GetParametersDescription() : LinkedList<ParameterDescription>
+GetLibrariesDescription() : LinkedList<LibraryDescription>
+setProblem(parameters : LinkedList<Object>) : void
+GetTasksDescription() : LinkedList<TaskDescription>
+GenerateNextTask() : Task
+UnifyResponseProblem(response : Response) : void
+getSolution() : String
ProblemRunTimeInstance
-_name : String
-_problemId : Integer
-_state : State
-_numberCreatedTasks : Integer
-_numberResolvedTasks : Integer
-_problemIsntance : IProblem
-_executionStart : Calendar
-_executionTime : String
-_responsesId : LinkedList<Integer>
-_solution : String
+ProblemRunTimeInstance(urlJava : String)
+getName() : String
+GetLibrariesDescription() : LinkedList<LibraryDescription>
+GetTasksDescription() : LinkedList<TaskDescription>
+GenerateNextTask() : Task
+GetDescription() : String
+GetParametersDescription() : LinkedList<ParameterDescription>
+UnifyResponseProblem(response : Response) : void
+setProblem(parameters : LinkedList<Object>) : void
+getSolution() : String
+UpdateExecutionStart() : void
+getProblemId() : Integer
+setProblemId(problemId : Integer) : void
+getState() : State
+setState(state : State) : void
+getNumberCreatedTasks() : Integer
+getNumberResolvedTasks() : Integer
+getExecutionTime() : String
+setExecutionTime(executionTime : String) : void
+getParameters() : LinkedList<ParameterDescription>
+setProblemInstance(problemInstance : IProblem) : void
+getExecutionStart() : Calendar
+setSolution(solution : String) : void
+UpdateSolution() : void
Environment
-_threadRequestTask : Thread
-_threadDeliverResponseProblem : Thread
-_channel : Channel
-_consumer : QueueingConsumer
-_queueOfTasks : String
-_queueOfResponses : String
-_countAllowTasks : Integer
-_gson : Gson
+Environment()
+GetNextTask() : void
+DeliverResponseProblem() : void
+ConsumersCount() : int
ParameterDescription
-_parameterName : String
-_valueType : ParameterValueType
-_defaultValue : Object
+ParameterDescription(parameterName : String, parameterValueType : ParameterValueType, defaultValue : Object)
+getParameterName() : String
+setParameterName(parameterName : String) : void
+getValueType() : ParameterValueType
+setValueType(valueType : ParameterValueType) : void
+getDefaultValue() : Object
+setDefaultValue(defaultValue : Object) : void
<<enumeration>>
ParameterValueType
Cadena
Entero
Entero_Grande
Lógico
Real
Lista_Cadena
Lista_Entero
Lista_Lógico
lista_Real

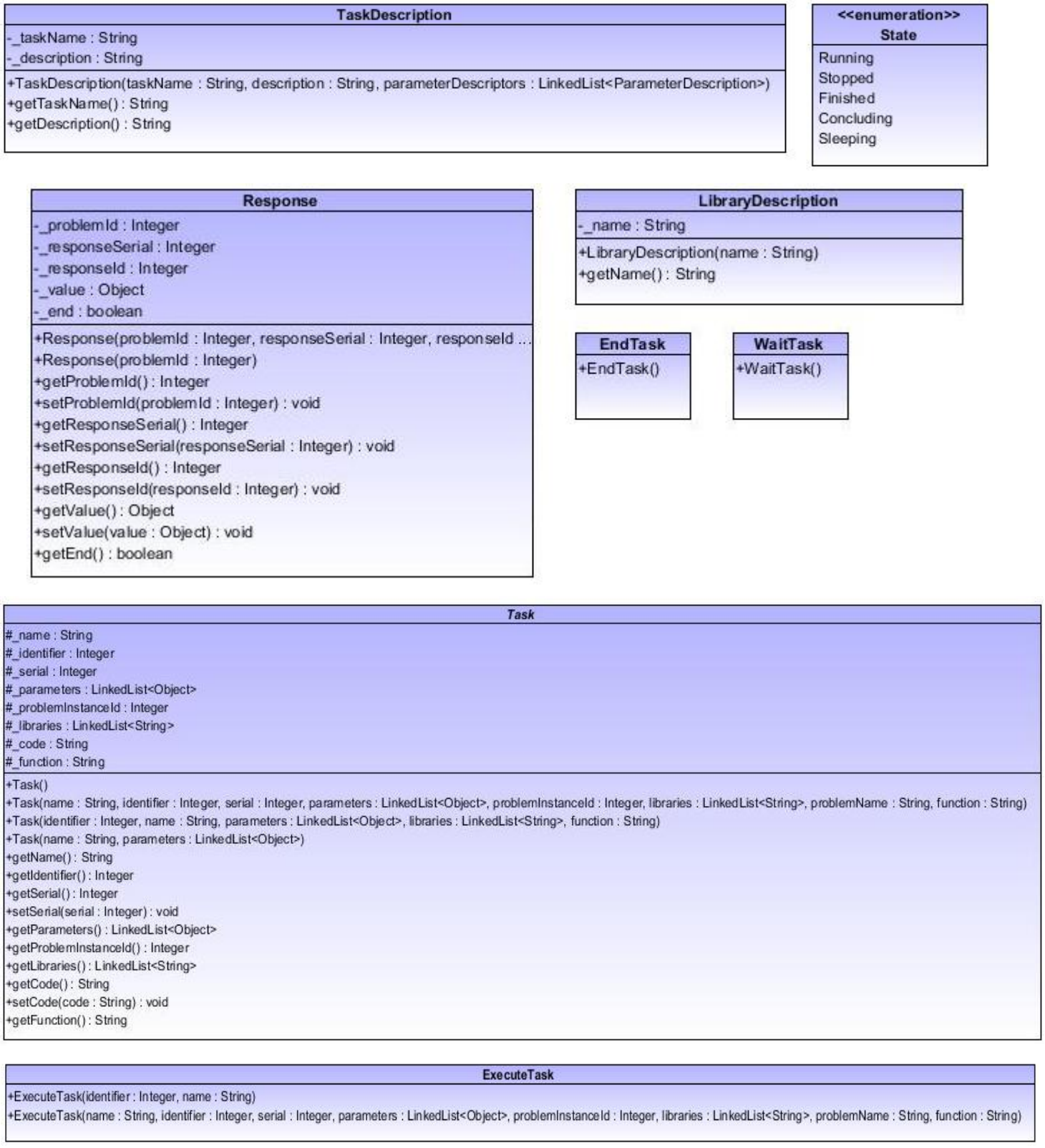


Figura XIV.3: Especificación de la las clases de la Capa de Negocio.

Especificación de las clases de la Capa de Acceso a Datos

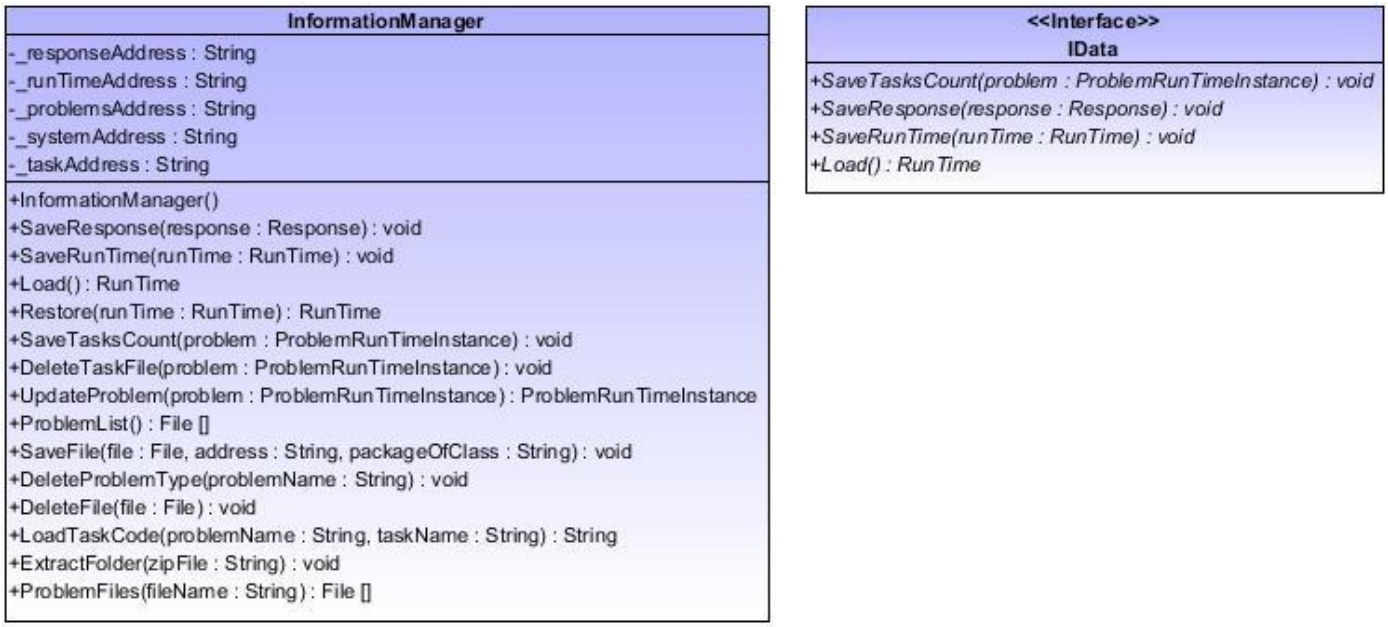


Figura XIV.4: Especificación de la las clases de la Capa de Acceso a Datos.

Anexo XV: Especificación de las tareas de ingeniería

	Tarea de Ingeniería
Número de tarea: HU1_T1	**Historia de usuario:** Crear tipo de problema.
Nombre: Cargar fichero.	
Tipo: Desarrollo.	**Puntos estimados:** 3 (días).
Fecha inicio: 19/11/2012.	**Fecha fin:** 21/11/2012.
Responsable: Heidy Luis Díaz.	
Descripción: El sistema solicita al usuario que cargue el fichero comprimido con extensión .zip que debe contener la clase de Java, que representa el tipo problema, y los ficheros JavaScript, que representan los tipos de tareas en que puede ser divido un problema de este tipo y las librerías necesarias para el procesamiento de dichas tareas.	

Tabla XV.1: HU1_T1 Cargar fichero.

	Tarea de Ingeniería
Número de tarea: HU1_T2	**Historia de usuario:** Crear tipo de problema.
Nombre: Validar que exista toda la información necesaria.	
Tipo: Desarrollo.	**Puntos estimados:** 2 (días).
Fecha inicio: 22/11/2012.	**Fecha fin:** 23/11/2012.
Responsable: Heidy Luis Díaz.	
Descripción: Mediante el uso de la clase Java, se obtiene la auto-descripción del tipo de problema, información referente a qué tipos de tareas pueden ser generadas y qué librerías son	

necesarias para la ejecución de las mismas. Con esta información el sistema verifica que existan todos los ficheros correspondientes a las tareas y las librerías.

Tabla XV.2: HU1_T2 Validar que exista toda la información necesaria.

	Tarea de Ingeniería
Número de tarea: HU1_T3	**Historia de usuario:** Crear tipo de problema.
Nombre: Almacenar el tipo de problema.	
Tipo: Desarrollo.	**Puntos estimados:** 2 (días).
Fecha inicio: 28/11/2012.	**Fecha fin:** 29/11/2012.
Responsable: Heidy Luis Díaz.	
Descripción: Toda la información del tipo de problema es almacenada en el disco duro de la máquina servidor, la clase Java y los ficheros JavaScript referentes a los tipos de tareas se almacenan en un directorio, y los ficheros JavaScript referentes a las librerías se publican en el servidor web.	

Tabla XV.3 HU1_T3 Almacenar el tipo de problema.

	Tarea de Ingeniería
Número de tarea: HU2_T1	**Historia de usuario:** Mostrar listado de tipos de problemas.
Nombre: Listar tipos de problemas.	
Tipo: Desarrollo.	**Puntos estimados:** 1 (días).
Fecha inicio: 08/03/2013.	**Fecha fin:** 08/03/2013.
Responsable: Heidy Luis Díaz.	
Descripción: Se busca en el directorio donde se encuentran almacenados los tipos de problemas el nombre de cada uno de ellos y se muestran en un lista.	

Tabla XV.4: HU2_T1 Listar tipos de problemas.

	Tarea de Ingeniería
Número de tarea: HU3_T1	**Historia de usuario:** Eliminar tipo de problema.
Nombre: Listar tipos de problemas.	
Tipo: Desarrollo.	**Puntos estimados:** 1 (días).
Fecha inicio: 13/03/2013.	**Fecha fin:** 13/03/2013.
Responsable: Heidy Luis Díaz.	
Descripción: Se busca en el directorio donde se encuentran almacenados los tipos de problemas el nombre de cada uno de ellos, se muestran en un lista, dándole la posibilidad al usuario de seleccionar el que desea eliminar.	

Tabla XV.5: HU3_T1 Listar tipos de problemas.

Tarea de Ingeniería	
Número de tarea: HU3_T2	**Historia de usuario:** Eliminar tipo de problema.
Nombre: Mostrar mensaje de confirmación.	
Tipo: Desarrollo.	**Puntos estimados:** 1 (días).
Fecha inicio: 15/03/2013.	**Fecha fin:** 15/03/2013.
Responsable: Heidy Luis Díaz.	
Descripción: Se muestra un mensaje de confirmación alertando al usuario de que en caso de existir problemas en proceso de resolución, del tipo que se desea eliminar, serán eliminados del sistema, y se le da la posibilidad al usuario de reafirmar o cancelar la operación de eliminación.	

Tabla XV.6: HU3_T2 Mostrar mensaje de confirmación.

Tarea de Ingeniería	
Número de tarea: HU3_T3	**Historia de usuario:** Eliminar tipo de problema.
Nombre: Eliminar problemas que se encuentren en proceso de resolución.	
Tipo: Desarrollo.	**Puntos estimados:** 2 (días).
Fecha inicio: 18/03/2013.	**Fecha fin:** 19/03/2013.
Responsable: Heidy Luis Díaz.	
Descripción: Se buscan todos los problemas del tipo de problema que se desea eliminar y se eliminan del sistema.	

Tabla XV.7: HU3_T3 Eliminar problemas que se encuentren en proceso de resolución.

Tarea de Ingeniería	
Número de tarea: HU3_T4	**Historia de usuario:** Eliminar tipo de problema.
Nombre: Eliminar tipo de problema.	
Tipo: Desarrollo.	**Puntos estimados:** 1 (días).
Fecha inicio: 21/03/2013.	**Fecha fin:** 21/03/2013.
Responsable: Heidy Luis Díaz.	
Descripción: Toda la información del tipo de problema almacenada en el disco duro de la PC servidor se elimina, al igual que las librerías publicadas en el servidor web.	

Tabla XV.8: HU3_T4 Eliminar tipo de problema.

Tarea de Ingeniería	
Número de tarea: HU4_T1	**Historia de usuario:** Resolver problema.

Nombre: Crear problema.	
Tipo: Desarrollo.	**Puntos estimados:** 4 (días).
Fecha inicio: 03/12/2012.	**Fecha fin:** 07/12/2012.
Responsable: Sael José Berrillo Borrero.	
Descripción: Se muestra un listado con todos los tipos de problemas existentes, dándole la posibilidad al usuario de seleccionar de qué tipo es el problema que desea resolver. Una vez seleccionado, el sistema obtiene, mediante el uso de la clase Java perteneciente al tipo de problema seleccionado, los parámetros iniciales requeridos para su resolución. Posteriomente se le solicita al usuario que introduzca los valores deseados para cada uno de ellos y se valida que estos valores estén en el formato correcto. Finalmente, con toda esta información se crea el problema.	

Tabla XV.9: HU4_T1 Crear problema.

	Tarea de Ingeniería
Número de tarea: HU4_T2	**Historia de usuario:** Resolver problema.
Nombre: Generar tarea.	
Tipo: Desarrollo.	**Puntos estimados:** 2 (días).
Fecha inicio: 11/12/2012.	**Fecha fin:** 12/12/2012.
Responsable: Sael José Berrillo Borrero.	
Descripción: Se crea una tarea utilizando la clase Java perteneciente al tipo de problema que se desea resolver, la cual especifica cómo debe ser dividido el problema en múltiples tareas más pequeñas y qué librerías son necesarias para su ejecución. A esta tarea se le incorpora el código JavaScript, que se obtiene del fichero JavaScript correspondiente almacenado en el disco duro de la máquina servidor.	

Tabla XV.10: HU4_T2 Generar tarea.

	Tarea de Ingeniería
Número de tarea: HU4_T3	**Historia de usuario:** Resolver problema.
Nombre: Asignar tarea a un trabajador.	
Tipo: Desarrollo.	**Puntos estimados:** 11 (días).
Fecha inicio: 14/12/2012.	**Fecha fin:** 25/12/2012.
Responsable: Sael José Berrillo Borrero.	
Descripción: Se adiciona la tarea a la cola de tareas del RabbitMQ, quien es el encargado de asignarla a un trabajador que se encuentre disponible.	

Tabla XV.11: HU4_T3 Asignar tarea a un trabajador.

	Tarea de Ingeniería

Número de tarea: HU4_T4	**Historia de usuario:** Resolver problema.
Nombre: Procesar tarea.	
Tipo: Desarrollo.	**Puntos estimados:** 8 (días).
Fecha inicio: 07/01/2013.	**Fecha fin:** 15/01/2013.
Responsable: Sael José Berrillo Borrero.	
Descripción: Se incluyen las librerías necesarias y se interpreta el código JavaScript. Con el resultado del procesamiento realizado se crea una respuesta.	

Tabla XV.12: HU4_T4 Procesar tarea.

	Tarea de Ingeniería
Número de tarea: HU4_T5	**Historia de usuario:** Resolver problema.
Nombre: Retornar la respuesta.	
Tipo: Desarrollo.	**Puntos estimados:** 2 (días).
Fecha inicio: 17/01/2013.	**Fecha fin:** 18/01/2013.
Responsable: Sael José Berrillo Borrero.	
Descripción: Se adiciona la respuesta a la cola de respuestas del RabbitMQ, quien es el encargado de entregarla al servidor.	

Tabla XV.13: HU4_T5 Retornar la respuesta.

	Tarea de Ingeniería
Número de tarea: HU4_T6	**Historia de usuario:** Resolver problema.
Nombre: Unificar la respuesta a la solución final del problema.	
Tipo: Desarrollo.	**Puntos estimados:** 3 (días).
Fecha inicio: 21/01/2013.	**Fecha fin:** 23/01/2013.
Responsable: Sael José Berrillo Borrero.	
Descripción: La respuesta se integra con la solución final del problema correspondiente mediante el uso de la clase Java perteneciente a dicho problema. Esta clase especifica cómo debe ser integrada la respuesta. Si la respuesta es la última esperada por el problema, se da por terminado su proceso de resolución.	

Tabla XV.14: HU4_T6 Unificar la respuesta a la solución final del problema.

	Tarea de Ingeniería
Número de tarea: HU5_T1	**Historia de usuario:** Mostrar listado de problemas.
Nombre: Listar problemas existentes.	
Tipo: Desarrollo.	**Puntos estimados:** 1 (días).
Fecha inicio: 03/04/2013	**Fecha fin:** 03/04/2013

Responsable: Sael José Berrillo Borrero.
Descripción: Se muestra la información necesaria de todos los problemas existentes.

Tabla XV.15: HU5_T1 Listar problemas existentes.

	Tarea de Ingeniería
Número de tarea: HU6_T1	**Historia de usuario:** Eliminar problema.
Nombre: Listar problemas.	
Tipo: Desarrollo.	**Puntos estimados:** 1 (días).
Fecha inicio: 05/04/2013.	**Fecha fin:** 05/04/2013.
Responsable: Sael José Berrillo Borrero.	
Descripción: Se muestra la información necesaria de todos los problemas existentes, dándole la posibilidad al usuario de seleccionar el que desea eliminar.	

Tabla XV.16: HU6_T1 Listar problemas.

	Tarea de Ingeniería
Número de tarea: HU6_T2	**Historia de usuario:** Eliminar problema.
Nombre: Mostrar mensaje de confirmación.	
Tipo: Desarrollo.	**Puntos estimados:** 1 (días).
Fecha inicio: 08/04/2013.	**Fecha fin:** 08/04/2013.
Responsable: Sael José Berrillo Borrero.	
Descripción: Se muestra un mensaje de confirmación, dándole al usuario la posibilidad de reafirmar o cancelar la operación de eliminación.	

Tabla XV.17: HU6_T2 Mostrar mensaje de confirmación.

	Tarea de Ingeniería
Número de tarea: HU6_T3	**Historia de usuario:** Eliminar problema.
Nombre: Eliminar problema.	
Tipo: Desarrollo.	**Puntos estimados:** 1 (días).
Fecha inicio: 08/04/2013.	**Fecha fin:** 08/04/2013.
Responsable: Sael José Berrillo Borrero.	
Descripción: Se elimina el problema de la lista de problemas existentes en el sistema.	

Tabla XV.18: HU6_T3 Eliminar problema.

	Tarea de Ingeniería
Número de tarea: HU7_T1	**Historia de usuario:** Mostrar listado de problemas resueltos.
Nombre: Listar problemas resueltos.	

Tipo: Desarrollo.	**Puntos estimados:** 1 (días).
Fecha inicio: 24/01/2013.	**Fecha fin:** 24/01/2013.
Responsable: Sael José Berrillo Borrero.	
Descripción: Se muestra parte de la información de todos los problemas resueltos.	

Tabla XV.19: HU7_T1 Listar problemas resueltos.

	Tarea de Ingeniería
Número de tarea: HU8_T1	**Historia de usuario:** Mostrar detalles de un problema resuelto.
Nombre: Detallar problema resuelto.	
Tipo: Desarrollo.	**Puntos estimados:** 3 (días).
Fecha inicio: 22/01/2013.	**Fecha fin:** 25/01/2013.
Responsable: Sael José Berrillo Borrero.	
Descripción: Se muestra toda la información referente al problema resuelto seleccionado.	

Tabla XV.20: HU8_T1 Detallar problema resuelto.

	Tarea de Ingeniería
Número de tarea: HU9_T1	**Historia de usuario:** Monitorear problemas en proceso de resolución.
Nombre: Listar problemas en proceso de resolución.	
Tipo: Desarrollo.	**Puntos estimados:** 1 (días).
Fecha inicio: 27/01/2013.	**Fecha fin:** 27/01/2013.
Responsable: Heidy Luis Díaz.	
Descripción: Se muestra la información necesaria de todos los problemas en proceso de resolución.	

Tabla XV.21: HU9_T1 Listar problemas en proceso de resolución.

	Tarea de Ingeniería
Número de tarea: HU9_T2	**Historia de usuario:** Monitorear problemas en proceso de resolución.
Nombre: Actualizar información.	
Tipo: Desarrollo.	**Puntos estimados:** 2 (días).
Fecha inicio: 27/01/2013.	**Fecha fin:** 29/01/2013.
Responsable: Heidy Luis Díaz.	
Descripción: Se actualiza periódicamente la información de los problemas en proceso de resolución, observándose cómo evoluciona este proceso.	

Tabla XV.22: HU9_T2 Actualizar información.

Tarea de Ingeniería	
Número de tarea: HU10_T1	**Historia de usuario:** Detener ejecución de un problema.
Nombre: Detener ejecución del problema.	
Tipo: Desarrollo.	**Puntos estimados:** 1 (días).
Fecha inicio: 15/04/2013.	**Fecha fin:** 15/04/2013.
Responsable: Heidy Luis Díaz.	
Descripción: Se detiene la ejecución de un problema seleccionado, cambiando su estado a detenido.	

Tabla XV.23: HU10_T1 Detener ejecución del problema.

Tarea de Ingeniería	
Número de tarea: HU11_T1	**Historia de usuario:** Reanudar ejecución de un problema.
Nombre: Reanudar ejecución del problema.	
Tipo: Desarrollo.	**Puntos estimados:** 1 (días).
Fecha inicio: 26/04/2013.	**Fecha fin:** 26/04/2013.
Responsable: Heidy Luis Díaz.	
Descripción: Se reanuda la ejecución de un problema seleccionado, cambiando su estado a corriendo.	

Tabla XV.24: HU11_T1 Reanudar ejecución del problema.

Tarea de Ingeniería	
Número de tarea: HU12_T1	**Historia de usuario:** Salvar servidor.
Nombre: Salvar servidor.	
Tipo: Desarrollo.	**Puntos estimados:** 3 (días).
Fecha inicio: 30/01/2013.	**Fecha fin:** 01/02/2013.
Responsable: Sael José Berrillo Borrero.	
Descripción: Se serializa toda la información gestionada por el sistema y se almacena en ficheros en el disco duro de la máquina servidor.	

Tabla XV.25: HU12_T1 Salvar servidor.

Tarea de Ingeniería	
Número de tarea: HU13_T1	**Historia de usuario:** Restaurar servidor.
Nombre: Restaurar servidor.	
Tipo: Desarrollo.	**Puntos estimados:** 8 (días).
Fecha inicio: 04/02/2013.	**Fecha fin:** 12/02/2013.
Responsable: Sael José Berrillo Borrero.	
Descripción:	

Se obtiene toda la información guardada en el disco duro de la máquina servidor, se de-serializa y se carga nuevamente en el sistema.

Tabla XV.26: HU13_T1 Restaurar servidor.

Anexo XVI: Aplicación del estándar de codificación definido para los nombres de clases

```
public class ExecuteTask extends Task {
    //...
```

Figura XVI.1: Ejemplo del estándar de codificación para los nombres de clases aplicado en el sistema.

Anexo XVII: Aplicación del estándar de codificación definido para los nombres de interfaces

```
public interface IProblem {
    //...
```

Figura XVII.1: Ejemplo del estándar de codificación para los nombres de interfaces aplicado en el sistema.

Anexo XVIII: Aplicación del estándar de codificación definido para nombres de atributos

```
private Integer _problemId;
private Integer _numberCreatedTasks;
private Integer _numberResolvedTasks;
```

Figura XVIII.1: Ejemplo del estándar de codificación para los nombres de atributos aplicado en el sistema.

Anexo XIX: Aplicación del estándar de codificación definido para nombres de parámetros y variables

```
public ProblemRunTimeInstance(String urlJava) {
    //...
```

Figura XIX.1: Ejemplo del estándar de codificación para los nombres de parámetros y variables aplicado en el sistema.

Anexo XX: Aplicación del estándar de codificación definido para la rotura de líneas

```
public ExecuteTask(String name, Integer identifier, Integer serial,
                   LinkedList<Object> parameters, Integer problemInstanceId,
                   LinkedList<String> libraries, String problemName,
                   String function) {
    //...
```

Figura XX.1: Ejemplo del estándar de codificación para la rotura de líneas aplicado en el sistema.

Anexo XXI: Aplicación del estándar de codificación definido para los comentarios de bloque

```
/*
 * @return the positionProblemToExecute
 */
```

Figura XXI.1: Ejemplo del estándar de codificación para los comentarios de bloque aplicado en el sistema.

Anexo XXII: Aplicación del estándar de codificación definido para los comentarios de línea

```
public Environment() throws Exception {
    Connection connection;
    /* To establish the connection with the RabbitMQ */

    ConnectionFactory factory = new ConnectionFactory();
    factory.setHost("localhost");
    connection = factory.newConnection();
     _channel = connection.createChannel();
    //...
    }
```

Figura XXII.1: Ejemplo del estándar de codificación para los comentarios de línea aplicado en el sistema.

Anexo XXIII: Aplicación del estándar de codificación definido para la cantidad de declaraciones por línea

```
Task task;
Task t;
```

Figura XXIII.1: Ejemplo del estándar de codificación para la cantidad de declaraciones por línea aplicado en el sistema.

Anexo XXIV: Aplicación del estándar de codificación definido para la colocación de las declaraciones

```
public Task GetNextTask() {
    ProblemRunTimeIntance problemToExecute;
    Task task;

    if (_positionProblemToExecute == -1) {
        return null;
    }
    else {
        problemToExecute = _listPendingProblems.get(_positionProblemToExecute);
        task = problemToExecute.GetNextTask();
        //...
```

Figura XXIV.1: Ejemplo del estándar de codificación para la colocación de las declaraciones aplicado en el sistema.

Anexo XXV: Aplicación del estándar de codificación definido para la declaración de clases e interfaces

```
public class TaskDescription {
    private String _taskName;
    private String _description;
    private LinkedList<ParameterDescription> _parameterDescriptors;

    public TaskDescription(String taskName, String description,
            LinkedList<ParameterDescription> parameterDescriptors) {
        _taskName = taskName;
        _description = description;
        _parameterDescriptors = parameterDescriptors;
    }

    public String getTaskName() {
        return _taskName;
    }
```

Figura XXV.1: Ejemplo del estándar de codificación para la declaración de clases e interfaces aplicado en el sistema.

Anexo XXVI: Interfaz principal con el menú "Tipos de problemas" desplegado

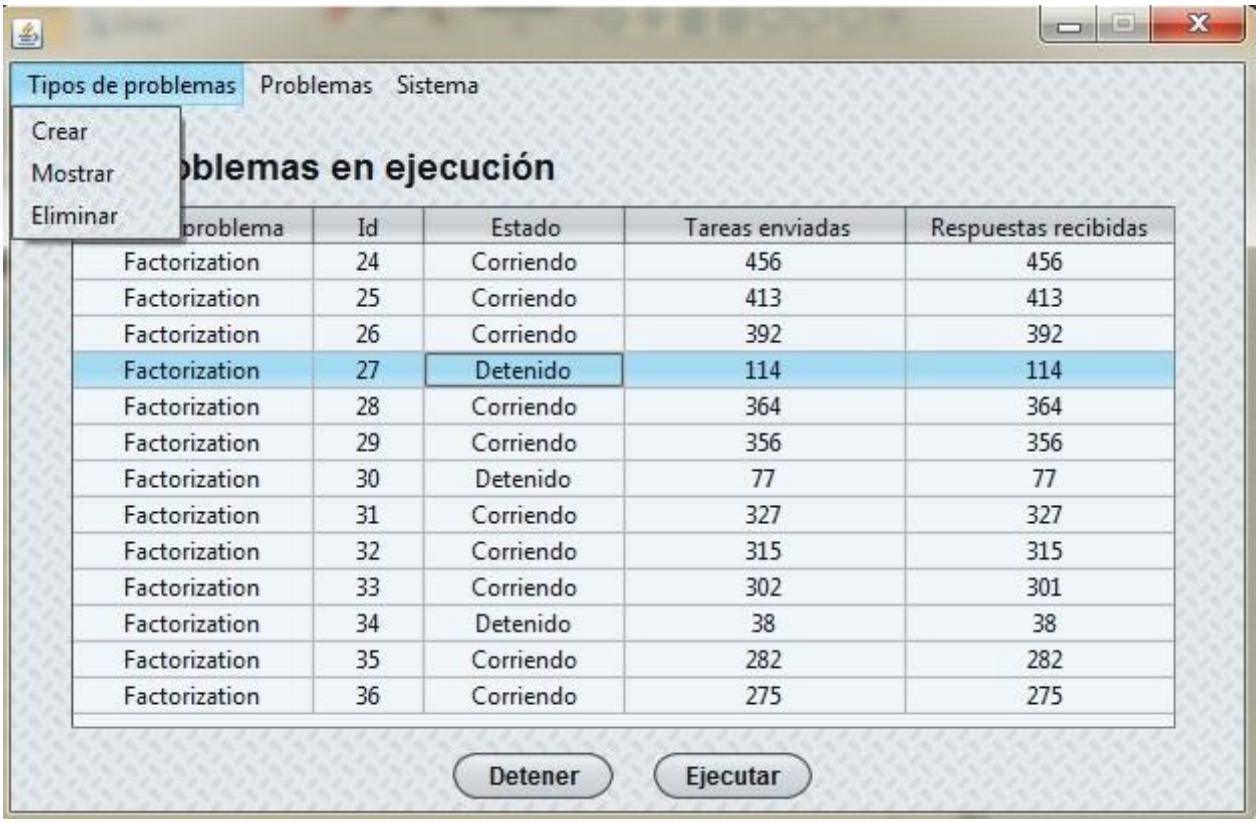


Figura XXVI.1: Interfaz principal con el menú "Tipos de problemas" desplegado.

Anexo XXVII: Interfaz principal con el menú "Problemas" desplegado

Figura XXVII.1: Interfaz principal con el menú "Problemas" desplegado.

Anexos XXVIII: Resultado de las pruebas unitarias

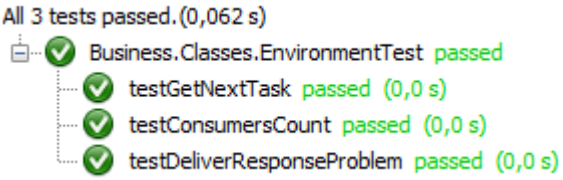


Figura XXVIII.1: Resultados de las pruebas unitarias realizadas a la clase *Environment*.

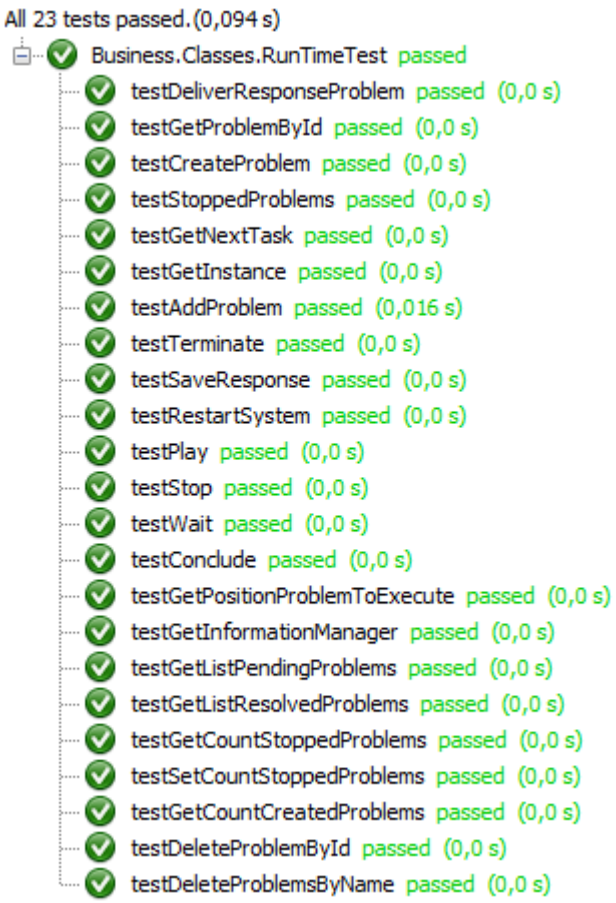


Figura XXVIII.2: Resultados de las pruebas unitarias realizadas a la clase *RunTime*.

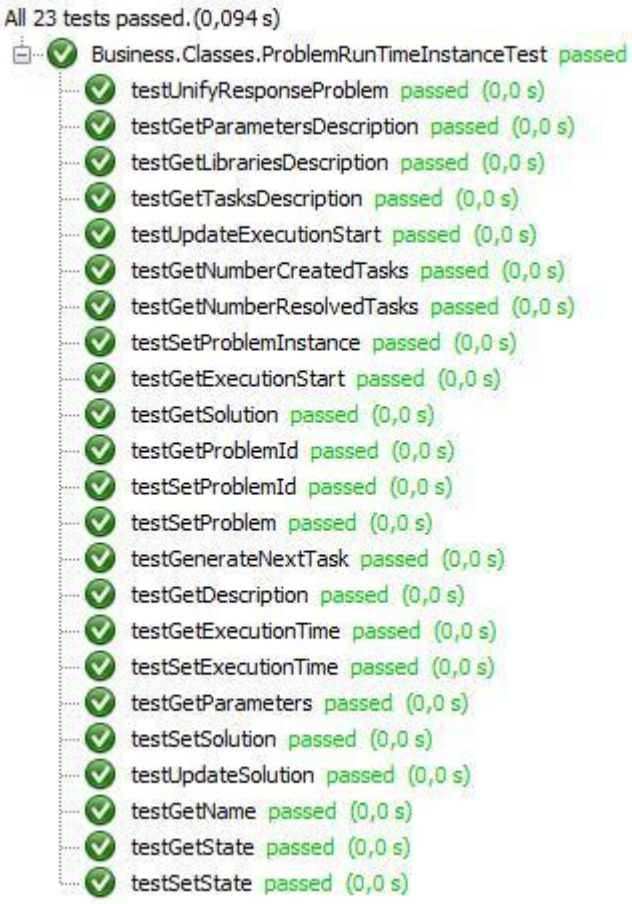


Figura XXVIII.3: Resultados de las pruebas unitarias realizadas a la clase ProblemRunTimeInstance.

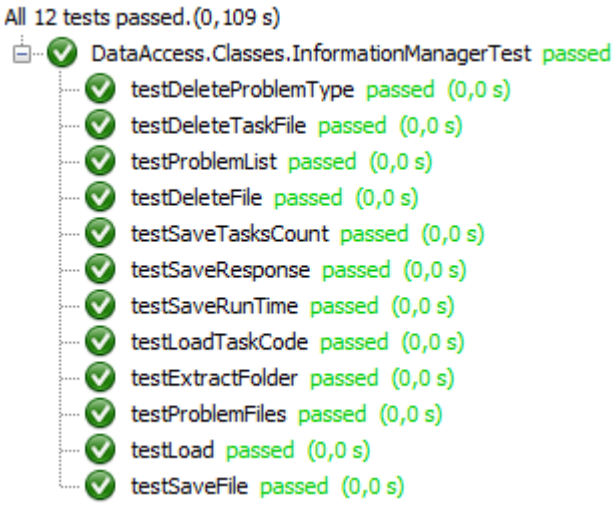


Figura XXVIII.4: Resultados de las pruebas unitarias realizadas a la clase InformationManager.

Anexos XXIX: Pruebas de aceptación

	Caso de Prueba de Aceptación
Código de caso de	**Nombre de la historia de usuario:** Mostrar listado de tipos de

prueba: HU2_CP2	problemas.
Responsable de la prueba: Heidy Luis Díaz.	
Descripción de la prueba: Prueba de funcionalidad para listar los tipos de problemas.	
Condiciones de ejecución: Debe existir algún tipo de problema.	
Entrada/Pasos de ejecución: ➢ Se muestra un listado con los nombres de los tipos de problemas existentes.	
Resultado esperado: Se muestra satisfactoriamente el listado de los tipos de problemas existentes.	
Evaluación de la prueba: Prueba satisfactoria.	

Tabla XXIX.1: HU2_CP2 Mostrar listado de tipos de problemas.

Caso de Prueba de Aceptación	
Código de caso de prueba: HU3_CP3	**Nombre de la historia de usuario:** Eliminar tipo de problema.
Responsable de la prueba: Heidy Luis Díaz.	
Descripción de la prueba: Prueba de funcionalidad para eliminar un tipo de problema.	
Condiciones de ejecución: Debe existir algún tipo de problema.	
Entrada/Pasos de ejecución: ➢ Se muestra un listado con los nombres de los tipos de problemas existentes. ➢ El usuario selecciona el tipo de problema que desea eliminar. ➢ Se elimina el tipo de problema y los problemas de ese tipo que se encuentran en proceso de resolución.	
Resultado esperado: Se muestra una notificación al usuario informándole que se ha creado correctamente el tipo de problema.	
Evaluación de la prueba: Prueba satisfactoria.	

Tabla XXIX.2: HU3_CP3 Eliminar tipo de problema.

Caso de Prueba de Aceptación	
Código de caso de prueba: HU5_CP5	**Nombre de la historia de usuario:** Mostrar listado de problemas.
Responsable de la prueba: Sael José Berrillo Borrero.	
Descripción de la prueba: Prueba de funcionalidad para listar los problemas.	
Condiciones de ejecución: Debe existir algún problema.	
Entrada/Pasos de ejecución: ➢ Se muestra un listado con el identificador, el tipo de problema y el estado de cada uno de los problemas existentes.	
Resultado esperado: Se muestra satisfactoriamente el listado de los problemas existentes.	

Evaluación de la prueba: Prueba satisfactoria.

Tabla XXIX.3: HU5_CP5 Mostrar listado de problemas.

Caso de Prueba de Aceptación	
Código de caso de prueba: HU6_CP6	**Nombre de la historia de usuario:** Eliminar problema.
Responsable de la prueba: Sael José Berrillo Borrero.	
Descripción de la prueba: Prueba de funcionalidad para eliminar un problema.	
Condiciones de ejecución: Debe existir algún problema.	
Entrada/Pasos de ejecución: ➢ Se muestra un listado con el identificador, el tipo de problema y el estado de cada uno de los problemas existentes. ➢ El usuario selecciona el problema que desea eliminar. ➢ Se elimina el problema.	
Resultado esperado: Se muestra una notificación al usuario informándole que se ha eliminado correctamente el problema.	
Evaluación de la prueba: Prueba satisfactoria.	

Tabla XXIX.4: HU6_CP6 Eliminar problema.

Caso de Prueba de Aceptación	
Código de caso de prueba: HU10_CP10	**Nombre de la historia de usuario:** Detener ejecución de un problema.
Responsable de la prueba: Heidy Luis Díaz.	
Descripción de la prueba: Prueba de funcionalidad para detener la ejecución de un problema.	
Condiciones de ejecución: Debe existir algún problema ejecutándose.	
Entrada/Pasos de ejecución: ➢ El usuario selecciona el problema que desea detener. ➢ Se detiene la ejecución del problema seleccionado y se cambia su estado a detenido.	
Resultado esperado: Se detiene satisfactoriamente la ejecución del problema.	
Evaluación de la prueba: Prueba satisfactoria.	

Tabla XXIX.5: HU10_CP10 Detener ejecución de un problema.

Caso de Prueba de Aceptación	
Código de caso de prueba: HU11_CP11	**Nombre de la historia de usuario:** Reanudar ejecución de un problema.
Responsable de la prueba: Heidy Luis Díaz.	
Descripción de la prueba: Prueba de funcionalidad para reanudar la ejecución de un	

problema.
Condiciones de ejecución: Debe existir algún problema detenido.
Entrada/Pasos de ejecución: ➢ El usuario selecciona el problema que desea reanudar. ➢ Se reanuda la ejecución del problema seleccionado y se cambia su estado a corriendo.
Resultado esperado: Se reanuda satisfactoriamente la ejecución del problema.
Evaluación de la prueba: Prueba satisfactoria.

Tabla XXIX.6: HU11_CP11 Reanudar ejecución de un problema.

Caso de Prueba de Aceptación	
Código de caso de prueba: HU12_CP12	**Nombre de la historia de usuario:** Salvar servidor.
Responsable de la prueba: Sael José Berrillo Borrero.	
Descripción de la prueba: Prueba de funcionalidad para salvar el servidor.	
Condiciones de ejecución: Ninguna.	
Entrada/Pasos de ejecución: ➢ Se crean los directorios necesarios para salvar la información. ➢ Se almacena toda la información gestionada por el sistema.	
Resultado esperado: Se salva satisfactoriamente la información gestionada por el sistema.	
Evaluación de la prueba: Prueba satisfactoria.	

Tabla XXIX.7: HU12_CP12 Salvar servidor.

Caso de Prueba de Aceptación	
Código de caso de prueba: HU13_CP13	**Nombre de la historia de usuario:** Restaurar servidor.
Responsable de la prueba: Sael José Berrillo Borrero.	
Descripción de la prueba: Prueba de funcionalidad para restaurar el servidor.	
Condiciones de ejecución: Debe existir una salva del servidor.	
Entrada/Pasos de ejecución: ➢ Se obtiene la información almacenada en la salva previamente realizada. ➢ Se carga la información obtenida.	
Resultado esperado: Se restaura satisfactoriamente el servidor.	
Evaluación de la prueba: Prueba satisfactoria.	

Tabla XXIX.8: HU13_CP13 Restaurar servidor.