Universidad de las Ciencias Informáticas

Facultad 6

Título: Herramienta informática para realizar pruebas de rendimiento a aplicaciones cliente/servidor

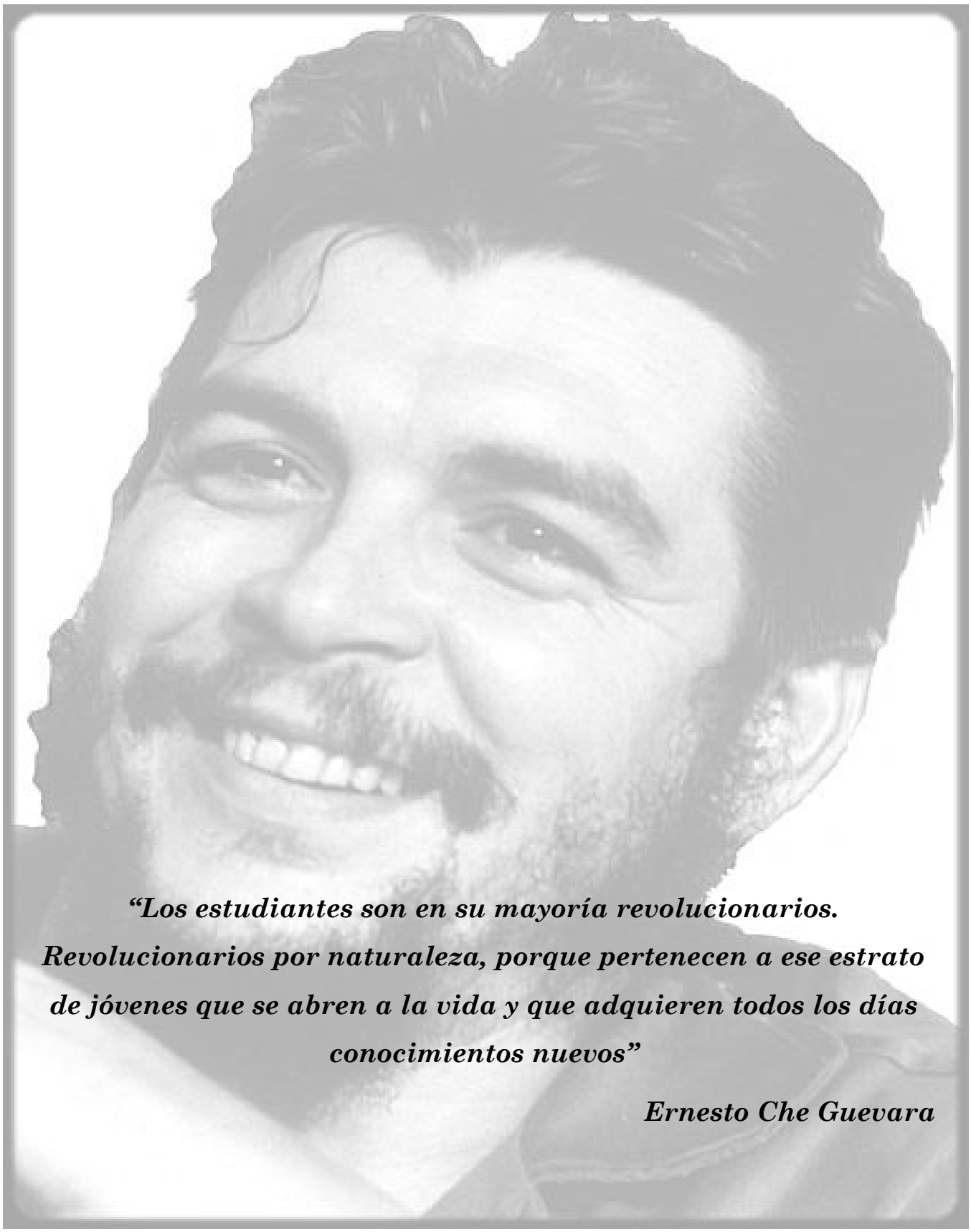
Trabajo de Diploma para optar por el título de
Ingeniero en Ciencias Informáticas

Autor: Eugenio Noda Castillo

Tutora: Ing. Adisley Reyes Crespo

La Habana, Cuba, Junio 2011.

"Año 53 de la Revolución"



"Los estudiantes son en su mayoría revolucionarios. Revolucionarios por naturaleza, porque pertenecen a ese estrato de jóvenes que se abren a la vida y que adquieren todos los días conocimientos nuevos"

Ernesto Che Guevara

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor de este trabajo y autorizo a la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de junio del año 2011.

Eugenio Noda Castillo

_____

Ing. Adisley Reyes Crespo

_____

DATOS DE CONTACTO

Tutora

Ing. Adisley Reyes Crespo.

Especialidad de Graduación: Ingeniería en Ciencias informáticas

Categoría Docente: Instructora

Años de Experiencia en el tema: 4

Año de Graduada: 2007

Correo Electrónico: areyesc@uci.cu

Agradecimientos

Quiero agradecer primeramente a mi madre por ser la razón de yo existir y estar en el lugar donde estoy, por darme aliento en toda la trayectoria estudiantil, por ser ejemplo siempre. A mi familia en general pues siempre me han dado su apoyo incondicional, mis hermanos, mi padre. Y más que todo mi abuela que siempre está y estará conmigo porque es el ejemplo que siempre seguí y siempre seguiré.

A mi tutora por ser una persona excepcional, por estar a mi lado siempre en momentos fáciles, difíciles, por darme siempre apoyo, por regañarme siempre cuando le discutía, por todo en general. Ojalá algún día podamos trabajar juntos, hacemos buen equipo.

Al profesor Lafaurie, y Edel por ser los que me ayudaron por parte de la aplicación, Al tribunal por ser siempre exigente, de ahí el resultado de mi trabajo.

A mis amigos que siempre estuvieron ahí dándome su apoyo incondicional, son muchos, pero todos me ayudaron mucho y les estoy muy agradecido por ello: Juanca, Migue, Osiris, los mamaos (Leo, Boris, Herryman, Manre, Mainoldis, Carlos), Yulio, Katiuska, Camilo, Alber, Rolando, Raidel, Mahela, Yenis, Abelito, Ana li, Felo,

el Rigo, Naivis, Capote, Osiel, Isael, Betty, Mario,Rubier, en fin; son muchos y nada, gracias a todos en general.

A mis amigos de Ciego que siempre me apoyaron y ayudaron incluso desde tan lejos: Binla (Yoa), Iselda, Eddy, Agnel, Yoan, el chino,Mico,Carlitos, Leidy, Jose. Gracias piquete.

Agradezco de manera general a todos los que me ayudaron a mi formación y que han compartido conmigo en algún momento de sus vidas,

Quisiera decir que sin el apoyo de todas estas personas nunca hubiera estado donde estoy.

Gracias una vez más a todos.

Dedicatoria

A quienes son y serán siempre ejemplos para mí:

Mi madre por ser la que siempre estuvo ahí dándome el ejemplo a seguir y el aliento a caminar por los senderos de la vida con buenos pasos.

A los que no están conmigo físicamente pero han sido parte de la realización de este sueño. (Mi abuela, Mi tía, y mi prima de la vida Danays).

A todos los que han compartido conmigo momentos importantes desde mis inicios hasta el momento, a todos mis amigos.

En especial a todos los que de una forma u otra me han ayudado en mi vida.

RESUMEN

El Laboratorio Industrial de Pruebas de Software (LIPS), perteneciente al Centro de Atención a la Calidad de Software (CALISOFT) es el encargado de realizar pruebas de rendimiento a aplicaciones cliente/servidor. Para la realización de las mismas utilizan una herramienta muy eficiente; “el JMeter”. Estas pruebas, con esta herramienta, dependiendo de la cantidad de usuarios pueden hacer que el ordenador donde se estén realizando las mismas se torne lento e incluso llegue a colapsar. El presente trabajo forma parte de la Plataforma de Tareas Distribuidas (T-arenal). Se presenta una herramienta que, con la utilización de esta plataforma, permite realizar pruebas de rendimientos (carga y estrés) a aplicaciones cliente/servidor. La misma, siguiendo la filosofía de trabajo de T-arenal logra disminuir la carga de trabajo en los clientes de prueba y aumenta la concurrencia en el servidor de prueba, ya que en lugar de simular usuarios mediante hilos de conexión desde una misma estación de trabajo, distribuye las peticiones de conexión entre los diferentes clientes de la plataforma, siendo estos los que se conectan al Servidor a probar, lográndose un mejor aprovechamiento de los recursos de cómputo disponibles en la Universidad.

PALABRAS CLAVE

Plataforma de Tareas Distribuida, Pruebas de calidad, Pruebas de rendimiento, T-arenal.

Tabla de Contenidos

INTRODUCCIÓN

Asegurar la calidad de los programas (Software), es una actividad que ha surgido como consecuencia de las fuertes demandas de los mismos en todos los procesos que se desarrollan en la actualidad; desde programas elementales de contabilidad hasta programas tan complejos como los espaciales. Por lo que las empresas que desarrollan este tipo de programa, dedican en su mayoría grandes esfuerzo para obtener soluciones de software con una alta calidad.

Según González J. “el aseguramiento de la calidad toma en cuenta todas aquellas acciones planificadas y sistemáticas necesarias para proporcionar la confianza de que un producto o servicio satisface los requisitos de calidad establecidos. Para que sea efectivo, se requiere una evaluación permanente de aquellos factores que influyen en la adecuación del diseño y de las especificaciones según las aplicaciones previstas” *(1)*.

Pressman, por su parte, asegura que “la garantía de calidad del software es una **actividad de protección** que se aplica a lo largo de todo el proceso de Ingeniería del Software, la cual engloba: métodos y herramientas de análisis, diseño, codificación y prueba; revisión de técnicas formales que se aplican durante cada paso; estrategia de prueba multiescalada; control de la documentación del software y de los cambios realizados; procedimientos que aseguren un ajuste a los estándares de desarrollo del software; y mecanismos de medida y de información” *(2)*.

Las pruebas de software, se consideran uno de los procesos más importantes en el diseño y construcción de aplicaciones, ya que constituyen las herramientas que la ingeniería de software emplea para asegurar la calidad de los productos. Es muy común que en las empresas de desarrollo de software existan equipos de trabajo dedicados a probar las funcionalidades de sus nuevos desarrollos mientras que se dedique poco o ningún tiempo a comprobar si estos cumplen con los requisitos mínimos de rendimiento. Estas pruebas son, hoy en día, cada vez más necesarias pues: los tiempos de respuesta por encima de lo aceptable, la excesiva variabilidad de los mismos en función de la carga del sistema y los problemas de fiabilidad o disponibilidad deben de considerarse errores tan graves como los de funcionalidad.

Cuba no está exenta de este desarrollo, logrando insertarse en el mercado informático mundial. Esto implica que cada día las empresas cubanas destinadas al desarrollo de software, tengan como reto brindar soluciones eficiente y con una alta calidad a sus clientes.

Entre las entidades productoras de software en Cuba se encuentran: La Empresa Cubana Nacional de Software (DeSoft), La Empresa Cubana Productora de Software para la Técnica Electrónica (Softel) y La Universidad de las Ciencias Informáticas (UCI). Esta última cuenta con el Centro para la Excelencia en el Desarrollo de Proyectos Tecnológicos (Calisoft) perteneciente al Ministerio de la Informática y las Comunicaciones, que se encarga de certificar la calidad de los productos que se desarrollan, además de facilitar la implementación de buenas prácticas en el proceso de desarrollo y mantenimiento de un software. Para esto, Calisoft cuenta con el Laboratorio Industrial de Pruebas de Software (LIPS), el cual asegura que cada artefacto tenga la calidad requerida para ser entregado al cliente.

En la Universidad se realizan diversas aplicaciones informáticas para empresas tanto nacionales como internacionales. Gran parte de estas aplicaciones se basan en arquitectura cliente-servidor, donde el cliente es una máquina que solicita una determinada petición al servidor y este le proporciona respuesta a dicha petición.

La realización de pruebas de rendimiento a este tipo de aplicación es fundamental, ya que permiten verificar si la capacidad del sistema es adecuada para la demanda de trabajo que soportará y detectar posibles puntos de ruptura (cuellos de botella) e ineficiencias proporcionando la información necesaria para un correcto dimensionado.

En el mundo existen diferentes aplicaciones destinadas a la realización de pruebas de rendimiento las cuales permiten simular los entornos en los que posteriormente serán implantados los sistemas. En la universidad, el LIPS utiliza la herramienta JMeter para realizar dichas pruebas, el mismo permite obtener resultados estadísticos sobre el desempeño del sistema, los tiempos de respuesta, la carga de usuario, en general el comportamiento del software bajo las determinadas condiciones de trabajo a las que se someta.

Su funcionamiento se basa en simular usuarios mediante la utilización de hilos concurrentemente sobre la aplicación a probar desde un mismo ordenador. Esto trae consigo que al realizar pruebas que superen los 500 usuarios a simular el ordenador se torne lento e incluso llegue a bloquearse,

imposibilitando la conclusión de la prueba planificada, esto puede pasar en computadoras de hasta 2 GB de memoria RAM.

Por lo que se define como **problema científico** ¿Cómo potenciar las pruebas de rendimiento que se realizan en la Universidad por el LIPS?

Como **objeto de estudio** se define el "Proceso de Pruebas de Calidad de Software", enmarcado en el **campo de acción** "Proceso de Pruebas de Rendimiento a Aplicaciones Cliente-Servidor".

Para dar solución al problema planteado, se define como **objetivo general:** Desarrollar una aplicación informática que, utilizando múltiples estaciones de trabajo, permita potenciar las pruebas de rendimiento a aplicaciones cliente-servidor.

Objetivos Específicos:

- ✗ Definir las funcionalidades que tendrá la herramienta para realizar pruebas de rendimiento a aplicaciones cliente-servidor.
- ✗ Realizar el diseño del sistema.
- ✗ Implementar el sistema.
- ✗ Realizar pruebas al sistema.

Para dar cumplimiento a los objetivos específicos se proponen las siguientes **tareas de la investigación:**

- Revisión del estado del arte acerca de los sistemas existentes en el mundo que permiten realizar pruebas de rendimiento a aplicaciones cliente-servidor.
- Definición de los requisitos funcionales y no funcionales de la herramienta.
- Desarrollo del Diagrama de Casos de Uso del sistema.
- Descripción de los Casos de Uso del sistema.
- Descripción de la vista de caso de uso.
- Descripción de la vista lógica y vista de despliegue.

- Selección de los patrones arquitectónicos.
- Descripción de los patrones de diseño propuestos.
- Desarrollo del diagrama de clases del diseño.
- Desarrollo del diagrama de componentes.
- Realización de las pruebas de unidad e integración del sistema.

La tesis está estructurada en tres capítulos, los cuales se describen a continuación:

Capítulo 1. Fundamentación Teórica, en este capítulo se realiza un estudio sobre la calidad del software y sus conceptos más significativos, los tipos de pruebas que se le realizan a los sistemas informáticos y las herramientas que se utilizan para medir el rendimiento en un software y se especifican las tecnologías y herramientas utilizadas para dar solución al problema planteado.

Capítulo 2. Descripción de la solución propuesta: En este capítulo se definen los requisitos funcionales y no funcionales de la herramienta, se presenta el diagrama de caso de uso del sistema, así como las descripciones textuales de los casos de uso para comprender mejor el funcionamiento del sistema. Se expone la descripción de los patrones usados, tanto de diseño como arquitectónicos, así como los diagramas de clases del diseño; vista de casos de uso, vista lógica y el diagrama de despliegue.

Capítulo 3: Implementación y prueba del sistema: En este capítulo se describen los elementos del diseño en términos de componentes reflejados en el diagrama de componentes. Se describe el código fuente fundamental de la herramienta y posteriormente se realizan pruebas de validación para verificar el correcto funcionamiento de las funcionalidades de la misma.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA.

En el presente capítulo se realiza un estudio sobre la calidad del software y sus conceptos más significativos, los tipos de pruebas que se le realizan a los sistemas informáticos y las herramientas que se utilizan para medir el rendimiento en un software. También se define la arquitectura del sistema y se especifican las tecnologías y herramientas utilizadas para dar solución al problema planteado.

1.1 Pruebas de calidad de software

En la actualidad, uno de los factores más importantes en un software es la calidad del mismo. Para obtener un producto con eficacia; se utilizan metodologías o procedimientos a lo largo de todo el desarrollo del mismo en vista de lograr un producto con calidad.

Según Pressman, la calidad de software no es más que la concordancia con los requisitos funcionales y de rendimiento explícitamente establecidos con los estándares de desarrollo explícitamente documentados y con las características implícitas que se espera de todo producto desarrollado profesionalmente .*(2)*

Según el IEEE; prueba se define como una actividad en la cual un sistema o componente es ejecutado bajo condiciones específicas, se observan o almacenan los resultados y se realiza una evaluación de algún aspecto del sistema o componente.*(3)* Las “pruebas de calidad del software” son un eslabón fundamental en el proceso de desarrollo del software. Permiten evaluar el producto para conocer en qué condiciones se encuentra el mismo y de esta forma detectar errores que pudieran generar comportamientos erróneos durante su ejecución.

Partiendo de lo antes dicho, se deben tener en cuenta varios objetivos a la hora de realizar pruebas de calidad a un software. Los principales objetivos son:

- Encontrar defectos en el software
- Diseñar pruebas a lo largo de todo el ciclo de desarrollo del software para así corregir la mayor cantidad de errores posibles y ganar en tiempo y esfuerzo.
- Ofrecer un producto altamente seguro y confiable.

- Verificar que el software satisface las necesidades del cliente.

El proceso de pruebas en un software tiene éxito si descubre defectos en el mismo, de lo contrario fracasa. Es notable decir que las pruebas no pueden decir si un software es correcto sino la cantidad de defectos que presenta el mismo. En el proceso de pruebas al software existen niveles en los que se ejecutan diferentes tipos de prueba con objetivos específicos. Teniendo en cuenta esto, las pruebas se agrupan por niveles de acuerdo a las diferentes etapas del proceso de desarrollo. Entre estos niveles se encuentra el nivel de pruebas unitarias, el cual se encarga de realizar pruebas de procedimientos o subrutinas (caja blanca). También se encuentran las pruebas de integración que es donde se verifica que las unidades trabajen de forma conjunta. Las pruebas de aceptación son otras pruebas importantes pues son las que hacen posible que el cliente esté de acuerdo con lo que se ha probado y así aceptar el software. Existen las pruebas de validación que se realizan con el objetivo de comprobar que el software, una vez ensamblado, funciona de acuerdo a las expectativas del cliente. Pero cuando de rendimiento se trata, se toman las pruebas de sistema.

1.1.1 Pruebas de Sistema

Las pruebas de sistema coinciden con las fases finales de las pruebas de integración. Tiene como objetivo verificar el sistema para comprobar si este cumple sus requisitos. A medida que aumenta la complejidad de los sistemas y aumenta la demanda de calidad, se hacen necesarios procesos y métodos que permitan obtener buenos conjuntos de pruebas del sistema. Este trabajo describe los modelos necesarios para generar de manera sistemática un conjunto de pruebas que permitan verificar la implementación de los requisitos funcionales de un software. *(4)*

Entre los tipos de pruebas más importantes en este nivel se encuentran:

- **Funcionales**: Las pruebas funcionales están desarrolladas bajo la perspectiva del usuario, confirmando que el sistema hace lo que los usuarios esperan que haga, es decir, que se cumplan satisfactoriamente los requisitos funcionales.
- **De Usabilidad**: Se intenta determinar si el sistema será fácil de utilizar para sus usuarios.

- **Pruebas de carga**: Una prueba de carga se ejecuta para comprender el comportamiento de una aplicación ante una carga determinada. Esta carga puede ser el número de usuarios esperado ejecutando un número de transacciones durante un tiempo determinado. El resultado de esta prueba dará el tiempo de respuesta de todas las transacciones críticas hasta llegar al punto de ruptura del sistema.
- **Pruebas de estrés (*stress*)**: Estas pruebas son utilizadas normalmente para someter la aplicación al límite de su funcionamiento, mediante la ejecución de un número de usuarios superior al esperado. La concurrencia es uno de los factores que más afecta el desempeño, calidad y operabilidad de productos de software que funcionen en ambientes cliente-servidor, debido a que, comúnmente, la carga transaccional del servidor (donde se ejecuta el producto de software) es muy alta a consecuencia de las múltiples conexiones o peticiones de usuarios para realizar procesos en él.

  El objetivo de las pruebas de stress es intentar romper el sistema. La idea es encontrar las circunstancias bajo las cuales se colapsará. Es importante porque puede revelar defectos en tiempo real, así como las áreas débiles donde el diseño defectuoso podría causar la indisponibilidad del servicio. Esto es particularmente importante para los sistemas en tiempo real, donde los acontecimientos imprevisibles pueden ocurrir, dando por resultado las cargas de la entrada que exceden lo descrito en los documentos de los requisitos funcionales *(13)*.
- **De rendimiento**: Estas pruebas buscan medir la eficiencia de un producto de software en términos de su rendimiento computacional. En este tipo de prueba se verifica que los requisitos de eficiencia y tiempo de respuesta concuerden con los que arroja el programa.

  Estas pruebas permiten que los probadores optimicen o ajusten el sistema, es decir, que optimicen la asignación de los recursos de sistema. Por ejemplo, los probadores pueden encontrar que necesitan reasignar posiciones de la memoria o modificar el nivel de la prioridad de ciertas operaciones de sistema. Los usuarios o los clientes deben

articular los objetivos del rendimiento en los documentos de los requisitos, además de indicarlos claramente en el plan de pruebas del sistema *(13)*.

Esta prueba no está aislada de las pruebas de carga y *stress*, ya que busca, en cada uno de los casos, medir el rendimiento del producto de software bajo ciertos criterios especificados por alguna de las mismas. Es así como la prueba de rendimiento es transversal a las pruebas de carga y stress respectivamente.

- **De volumen**: Se realizan para asegurar que el producto de software esté en capacidad de manejar los volúmenes de datos que los requisitos funcionales establecen.
- **De seguridad**: Se realizan para asegurar que el producto de software cumple con los estándares exigidos en los requisitos no funcionales y, además, que contemple los esquemas de seguridad mínimos reconocidos en la industria de software. La prueba de la seguridad evalúa las características del sistema que se relacionan con la disponibilidad, integridad y confidencialidad de los datos y de los servicios del sistema *(13)*.
- **De entorno**: Orientadas a probar las interacciones entre el sistema y otros sistemas en su entorno.

En la actualidad existen varias herramientas que automatizan estas pruebas de rendimiento entre otras funcionalidades.

1.2 Herramientas que se utilizan para pruebas de rendimiento

Existen herramientas que permiten medir el rendimiento de un sistema. A continuación se describen algunas de estas herramientas, hay que destacar que la mayoría de las mismas son propietarias, pero no todas, ejemplo el JMeter.

1.2.1 JMeter

JMeter es una herramienta Open Source (Código abierto) realizada en java (es un proyecto de Apache) que se utiliza para realizar pruebas de rendimiento, normalmente contra aplicaciones web. JMeter nos permite realizar simulaciones de gran carga en el servidor, red o aplicación para comprobar

su “fuerza” y para analizar el rendimiento ante diferentes tipos de sobrecarga. Las características principales de Apache JMeter son las siguientes:

- Permite cargar y realizar pruebas sobre distintos tipos de servidores: Web (HTTP, HTTPS), SOAP, Bases de datos (JDBC), LDAP, JMS, Email (POP3 e IMAP).
- Multiplataforma: Unix (Solaris, Linux, entre otros), Windows (98, NT, XP, entre otros), OpenVMS Alpha 7.3+.
- Multihilo: permite realizar testeos de forma concurrente.
- Permite comprobar cómo se comportará una aplicación web ante multitud de usuarios y la velocidad de carga que tendrá. *(5)*

1.2.2 Rational Performance Tester

IBM Rational Performance Tester es la herramienta que permite la creación, ejecución y análisis de pruebas de rendimiento que ayuda a los equipos a validar la escalabilidad y confiabilidad de las aplicaciones basadas en Web antes del despliegue en producción. También disponible con las extensiones IBM Rational Performance Tester para SAP y Siebel (SAP 4.6, 4.7 y Siebel v7.7 y v7.8), Citrix y SOA.

Entre las características más resaltantes están:

- Simplifica la creación de pruebas, la generación de cargas y la recopilación de datos para garantizar que las aplicaciones se amplíen hasta a miles de usuarios concurrentes.
- Interfaz de usuario basada en Windows y Linux.
- Permite grandes pruebas de múltiples usuarios con un mínimo de recursos de hardware.
- Permite un reconocimiento inmediato de los problemas de rendimiento con informes en tiempo real. *(6)*

1.2.3 JCrawler

Aplicación Opensource para realizar test de estrés a aplicaciones web. Se introduce la URL y se puede realizar la navegación web. Admite redirecciones HTTP y cookies. Es independiente de la plataforma, posee un modo consola y es sencillo de configurar. Es apropiado para portales complejos en los que hay que probar todas las páginas del portal y no solo algunas URLs. JCrawler está basado en ataques/segundo y no en "X" hilos atacando una web (ya que en estos últimos casos puede que realmente se estén dando por ejemplo solo 2 ataques por segundo aún teniendo 200 hilos).

Entre sus características más significativa se tienen:

- Fácil de configurar. Toda la configuración se reduce a un fichero de configuración XML.
- Independiente del Sistema Operativo.
- Código Abierto: ofrece confianza, ya que en cuanto se detecte un error (bug), será solucionado en un muy breve período de tiempo. (7)

1.2.4 SOLEX

Es una herramienta Open Source para testeo creado para ser implantado como un plugin en Eclipse. Solex permite grabar la sesión de un usuario, para poder ser utilizado posteriormente y ser repetida, de manera que aseguremos la no regresión de la aplicación. También nos permite realizar pruebas de estrés y rendimiento contra la aplicación web. (8)

1.3 Sistema Distribuido

Se define como un sistema informático compuesto por un conjunto de nodos de procesamiento comunicados y coordinados mediante una red que permite el intercambio de mensajes entre los mismos. *(9)*

1.3.1 Plataforma de Cálculo Distribuido T-arenal.

La Plataforma de Tareas Distribuidas (T-arenal) es un producto informático que ofrece una alternativa de cómputo y que aglutina en un solo conjunto varias estaciones de trabajo sin intentar eliminar o

pretender aminorar las amplias posibilidades de aplicación de los modelos paralelos, sino de complementar todos los medios disponibles en una gran "supercomputadora virtual".

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

Servidor Central: Su principal función, es la de chequear y planificar la asignación de las diferentes tareas a los servidores de peticiones. Además 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 a servir. Realiza también el seguimiento de todos los eventos que ocurren durante su funcionamiento.

Servidor de Peticiones: Es el responsable de solicitar una tarea al servidor central, así como atender y controlar la realización de la misma. La tarea a ser atendida, será dividida en pequeñas subtareas 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 subtareas generadas, para finalmente construir el resultado de la tarea original.

Cliente: El 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. Múltiples clientes pueden realizar peticiones de trabajo al servidor de peticiones.

Además, cumple con los aspectos esenciales de un sistema de cómputo distribuido: transparencia, eficiencia, flexibilidad, escalabilidad y fiabilidad.

Transparencia: Oculta la naturaleza distribuida permitiendo que los usuarios interactúen con una aplicación de escritorio y trabajen como si se tratara solamente de una supercomputadora.

Eficiencia: La solución a los problemas se obtiene mucho más rápido haciendo uso del sistema distribuido, que la respuesta que daría una simple computadora.

Flexibilidad: Es lo suficientemente flexible, para que cualquier cambio a realizar no requiera la parada de todo el sistema y la recopilación de todo el código.

Escalabilidad: Se comporta estable, tanto en redes pequeñas que grandes. También garantiza un adecuado mantenimiento y actualización, empleando el mínimo de personal.

Fiabilidad: La protección que brinda al usuario es extrema, asegura al 100% que el problema del usuario será resuelto, independientemente de los problemas ajenos que existan, ya sean fallos de red, electricidad o el apagado de las máquinas.

Multiplataforma: Al ser un sistema desarrollado completamente en Java, está listo para ser instalado sobre cualquier tipo de arquitectura de hardware y sistemas operativos.

El sistema distribuido t-arenal se encuentra desplegado actualmente en la UCI. Se decidió usar dicha plataforma puesto que permite que el sistema funcione más cómodamente al realizar las pruebas de rendimiento; ya que este proceso se hace en paralelo en varias computadoras que realizan la misma acción (prueba) simultáneamente, permitiendo una mayor concurrencia en el servidor, dando así un tiempo de respuesta más exacto y rápido. T-arenal presenta una arquitectura Cliente-Servidor, de aquí que la herramienta posee dicha arquitectura. A continuación se describe la misma.

1.4 Arquitectura de software

La Arquitectura de Software es, a grandes rasgos, una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según se le percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema. *(10).*

La definición oficial que sigue la IEEE Std1471-2000 plantea: “La arquitectura del software es la organización fundamental de un sistema formada por sus componentes[1], las relaciones entre ellos y el contexto en el que se implantarán, y los principios que orientan su diseño y evolución”**.**

1.4.1 Arquitectura Cliente/Servidor

Con respecto a la definición de arquitectura cliente/servidor se encuentran las siguientes definiciones:

1 Bloques de construcción que conforman las partes de un sistema de software. A nivel de lenguajes de programación, pueden ser representados como módulos, clases, objetos o un conjunto de funciones relacionadas.

- Cualquier combinación de sistemas que pueden colaborar entre sí para dar a los usuarios toda la información que ellos necesiten sin que tengan que saber donde está ubicada.
- Es una arquitectura de procesamientos cooperativo donde uno de los componentes pide servicios a otro.
- Es un procesamiento de datos de índole colaborativo entre dos o más computadoras conectadas a una red.
- El término cliente/servidor es originalmente aplicado a la arquitectura de software que describe el procesamiento entre dos o más programas: una aplicación y un servicio soportante. (*15*)

Para el desarrollo del sistema; por las características que lo definen, se especifican las herramientas, tecnologías y metodologías a usar en el perfeccionamiento del mismo. En el capítulo 2 se presenta el patrón arquitectónico definido para la herramienta. A continuación se verán las herramientas y metodologías usadas para el desarrollo del sistema.

1.5 Metodología, tecnologías y herramientas.

Para la realización de cualquier proyecto de software, se debe definir qué metodologías y herramientas de desarrollo se ajustan al proyecto. No existe una metodología de software universal, esta debe adecuarse a las características específicas de cada proyecto (recursos y equipo de desarrollo) que exigen que el proceso sea configurable. La propuesta de la metodología, tecnología y herramienta que se muestra a continuación, surgen a partir de la relación que existirá entre la herramienta y la plataforma de tareas distribuidas (t-arenal) para su funcionamiento. Todas son tecnologías y herramientas libres, basado en lo anterior se definen a continuación las herramientas y metodologías para el desarrollo de la investigación.

1.5.1 Metodología de desarrollo de software (OpenUp)

OpenUP/Basic es un FrameWork de procesos de desarrollo de software de código abierto. Es un proceso modelo y extensible, dirigido a gestión y desarrollo de proyectos de software basados en desarrollo iterativo, ágil e incremental; y es aplicable a un conjunto amplio de plataformas y aplicaciones de desarrollo.

Este proceso de desarrollo unificado está basado en Rational Unified Process (RUP), desarrollado por IBM y reconocido mundialmente como uno de los procesos de desarrollo de software de mayor calidad, basándose en los principios de Adaptación, Importancia a los involucrados e interesados en los resultados del proyecto; Colaboración, Valor a la iteración; y Calidad Continua. *(11)*

1.5.2 Herramienta para la modelación visual del sistema (Visual Paradigm 6.4)

Para la modelación visual del sistema se emplea Visual Paradigm para UML en su versión 6.4. Es una herramienta Case profesional que utiliza "UML", ayudando a construir aplicaciones rápidamente, mejor y económicamente, soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Es una herramienta fácil de usar e instalar. Permite modelar todos los tipos de diagramas de clases, código inverso, generar código fuente desde diagramas y documentación en HTML/PDF. Además en el mismo se pueden modelar los prototipos de interfaz de usuarios, dando al desarrollador una vista de lo que será el sistema.*(16)*

1.5.3 Lenguaje de Modelado (UML)

Para dar solución al problema planteado se utiliza como lenguaje de modelado UML (Lenguaje Unificado de Modelado) como se plantea en el epígrafe anterior. Es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software. Además, es una notación unificada con la que se permite lograr un entendimiento entre los usuarios y los desarrolladores. *(17)*

1.5.4 Entorno de Desarrollo Integrado (Eclipse 3.3.0)

Eclipse es una plataforma de programación, usada para crear entornos integrados de desarrollo (del Inglés IDE). Eclipse fue desarrollado originalmente por IBM como el sucesor de su familia de herramientas para VisualAge. Eclipse es ahora desarrollado por la Fundación "Eclipse", una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto. Se decide usar dicha herramienta puesto que la plataforma t-arenal está implementada en la misma, lo cual hace posible que el trabajo sea más eficiente, ya que todas las actualizaciones que se hagan en cualquier instante se podrán ver desde la plataforma o desde la PC del desarrollador.*(12)*

1.5.5 Tecnología utilizada para la programación (Java 1.6)

La compañía Sun describe el lenguaje Java como “simple, orientado a objetos, distribuido, interpretado, robusto, seguro, de arquitectura neutra, portable, de altas prestaciones, multitarea y dinámico”.*(14)* Además, Java se caracteriza por ser simple, ya que elimina al 50% los errores más comunes de programación en los lenguajes C y C++ al eliminar muchas de las características de éstos, entre las que destacan: necesidad de liberar memoria, definición de tipos, aritmética de punteros, entre otros. Además de las características mencionadas con anterioridad, constituye un lenguaje “simple, orientado a objetos, distribuido, interpretado, robusto, seguro, de arquitectura neutra, portable, de altas prestaciones, multitarea y dinámico”. Además es el lenguaje que se usó para implementar la plataforma t-arenal, donde dicha plataforma brinda una librería que facilita el trabajo con la misma.

1.6 Conclusiones del capítulo

En este capítulo se realizó el estudio de los principales factores que hacen posible el desarrollo del sistema. Se fundamentaron los conceptos básicos que se necesitan conocer para el desarrollo del software, tales como: **calidad de software, pruebas, arquitectura**. Se argumentó además sobre las fases o niveles de pruebas por los que transita un sistema informático a lo largo de su desarrollo. Además de las herramientas usadas en el mundo automatizar la realización de pruebas de rendimiento a un software. Conjuntamente se seleccionó la metodología de desarrollo a utilizar **(OpenUP), Visual Paradigm** como herramienta CASE, **Eclipse** como IDE de desarrollo y **Java** como Lenguaje de Programación. Se utiliza la plataforma de tareas distribuidas **t-arenal** puesto que agiliza la funcionalidad del software (Realizar Pruebas de rendimiento) al usar cómputo distribuido que permite dividir la prueba en forma de tarea en varias subtareas en diferentes estaciones de trabajo, haciendo posible que la herramienta sea más factible y que no se produzcan cargas en los ordenadores donde se realizan los testeos.

CAPÍTULO 2: CARACTERÍSTICAS Y DISEÑO DEL SISTEMA.

En el presente capítulo se expone una descripción detallada de la solución propuesta. Se plantea cada uno de los requisitos funcionales y no funcionales que debe tener la herramienta, se presenta el diagrama de CU del sistema, así como la descripción teórica de los mismos. Además se expone la descripción de los patrones usados, tanto de diseño como arquitectónicos, así como los diagramas de clases del diseño; vista de casos de uso, vista lógica y el diagrama de despliegue.

2.1 Propuesta del sistema

Actualmente en la UCI, en el LIPS, se realizan pruebas de rendimiento al software enfocadas en carga y estrés. Para la realización de estas pruebas, se utiliza la herramienta JMeter. La aplicación de la herramienta JMeter en el LIPS, se hace desde un único ordenador generando conexiones en hilos. La creación de hilos de ejecución consume recursos de la computadora, como la memoria RAM, de ahí que físicamente exista un límite máximo de hilos y conexiones posibles a crear en un determinado intervalo de tiempo.

Es por esto, que se define la propuesta de una herramienta que se enfoque en medir el rendimiento de un sistema en cuanto a carga y estrés inicialmente, que logre aumentar el nivel de concurrencia en el servidor para llegar más rápido al punto de ruptura de la aplicación a probar, aprovechando los recursos de cómputo de la Universidad. En esta propuesta se utilizarán varias estaciones de trabajo generando hilos de ejecución sobre un servidor. En lugar de ser 100 hilos de ejecución en un mismo ordenador cada cierto tiempo, contar con más estaciones de trabajo propiciaría que se pueda distribuir esta carga de peticiones entre ellas. De esta forma se lograría aumentar el nivel de concurrencia en el servidor de la aplicación que se desee probar, la prueba podría realizarse en menos tiempo ya que la carga sería distribuida y se llegaría más rápido al punto de ruptura de la aplicación a probar.

¿De qué forma sería posible esto?

Se utilizaría la plataforma de tareas distribuidas t-arenal, aprovechando su ambiente distribuido explicado anteriormente.

Ejemplo de la solución propuesta

Se cuenta con el Servidor Central de t-arenal, un servidor de peticiones, y tres clientes pidiendo tareas, además del servidor de la aplicación a probar. Este servidor central de t-arenal almacena la tarea que pudiera ser, simular 150 hilos de ejecución sobre la aplicación a probar. Esta tarea es enviada al servidor de peticiones, quién se encarga de dividirla en pequeños subproblemas, (cada cliente realizará 50 hilos de ejecución) y las distribuye entre los mismos. Cada cliente realiza el procesamiento de su tarea y envía la respuesta al servidor de peticiones y éste se encarga de coleccionar el resultado de cada uno de los subproblemas, para construir el resultado final de la tarea original, y se la envía al servidor central de t-arenal. En total si cada PC es capaz de soportar por sus características físicas 50 hilos de ejecución, si hay tres PCs, entonces sería 3 * 50 hilos de ejecución que serían las 150 ejecuciones que no se realizarían desde un mismo ordenador, por lo tanto, se ganará en tiempo de ejecución, aumentará el nivel de concurrencia en el servidor de la aplicación a probar y las máquinas clientes no sufrirán degradación en su sistema.

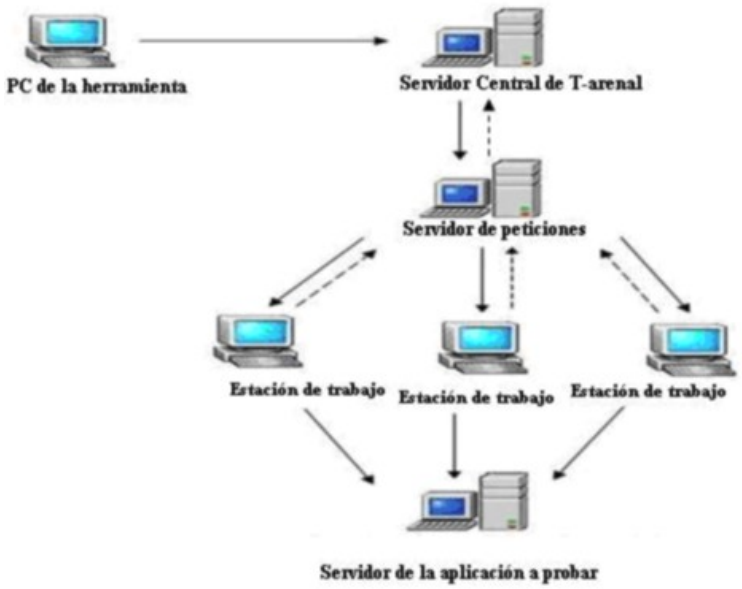


Figura 1: Herramienta integrada con T-Arenal.

2.2 Especificación de los Requisitos del Software.

El flujo de trabajo de requerimientos es uno de los más importantes, porque en él se establece qué es lo que tiene que hacer exactamente el sistema que se construya, de modo que los usuarios finales tienen que comprender y aceptar los requisitos que se especifiquen. Se dividen en dos grupos: los requisitos funcionales y los requisitos no funcionales.

2.2.1 Requisitos funcionales.

Los requisitos funcionales deben comprenderlo tanto los desarrolladores como los usuarios, ya que son los requerimientos que el sistema debe cumplir para su funcionamiento.

- **RF1.** Autenticar usuario a la plataforma t-arenal.

  El sistema debe permitir al usuario autenticarse en la plataforma t-arenal.

- **RF2.** Definir prueba para una aplicación web.

  El sistema debe permitirle al probador definir la prueba para una aplicación web en la herramienta.

- **RF3.** Definir prueba para una Base de datos (BD).

  El sistema debe permitirle al probador definir la prueba a una base de datos en la herramienta.

- **RF4**. Eliminar pruebas.

  El sistema debe permitir eliminar pruebas que se hayan definido.

- **RF5.** Ejecutar Pruebas.

  El sistema debe permitirle al probador ejecutar las pruebas definidas para obtener los resultados de las mismas.

- **RF6.** Obtener resultados de las pruebas ejecutadas.

  El sistema debe permitirle al probador ver los resultados de las pruebas que hayan terminado de ejecutarse.

2.2.2 Requisitos no funcionales.

Una vez analizados los requisitos funcionales, se hace necesario analizar los requisitos no funcionales, que no son más que las propiedades o cualidades que el producto debe tener. Entre los requisitos no funcionales del sistema se encuentran:

Apariencia o interfaz externa: El sistema debe contar con una interfaz amigable, donde el usuario pueda orientarse fácilmente.

Usabilidad: La herramienta estará dirigida a usuarios con un nivel medio o superior de informática. La herramienta tendrá un ambiente sencillo y será fácil de manejar para el usuario.

Fiabilidad: La interacción con el sistema, estará sometida a un proceso de autenticación del usuario. La plataforma t-arenal guarda constantemente los cambios, garantizando que si hay fallos ya sean fallos de red, electricidad, apagado de máquinas, entre otros que puedan surgir; no ocurra nada con el seguimiento de la tarea.

Seguridad

- Confidencialidad: Se requiere de usuario y contraseña para poder acceder a la información de las pruebas.
- Disponibilidad: En caso de tener el usuario y la contraseña, se le garantiza poder acceder a la herramienta en todo momento dependiendo del permiso que este tenga en la plataforma.

Portabilidad: El sistema es multiplataforma, razón por la cual podrá ser utilizado tanto en Windows como en Linux.

Software. Debe estar instalada la máquina virtual de Java SDK 1.5.x o superior. El servidor central de t-arenal, debe estar disponible, así como los clientes de prueba. Por consiguiente la plataforma t-arenal debe estar disponible conjuntamente con el servidor central.

Hardware: El ordenador donde se utilizará la herramienta debe tener como mínimo la cantidad de memoria RAM equivalente a los requerimientos de memoria de la máquina Virtual de Java.

Restricciones en el diseño e implementación

- Lenguaje de programación Java.
- Eclipse como herramienta de desarrollo.
- Se trabajará con la biblioteca (librería) t-arenal.
- Visual Paradigm como herramienta CASE.

2.3 Modelo de Casos de Usos.

2.3.1 Actores del sistema

Los actores representan a terceros fuera del sistema que interactúan con él. En el sistema que se describe se identificó un solo actor:

Tabla 1: Descripción de los actores del sistema.

Actores	Descripción
Probador	Representa al usuario que va a interactuar con las pruebas.

2.3.2 Casos de uso del sistema

Los casos de uso del sistema que aparecen a continuación, tienen como objetivo satisfacer los requisitos funcionales descritos con anterioridad.

- **CU Autenticar usuario**
- **CU Definir prueba.**
- **CU Definir prueba para la web.**
- **CU Definir prueba para la BD.**
- **CU Eliminar pruebas.**
- **CU Ejecutar pruebas.**
- **CU Obtener resultados.**

2.3.3 Diagrama de Casos de Uso del Sistema.

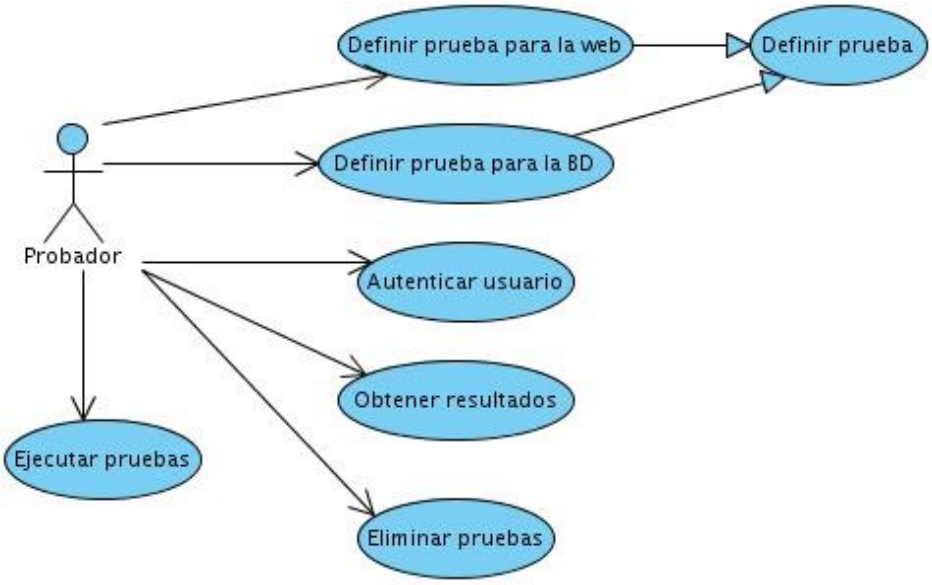


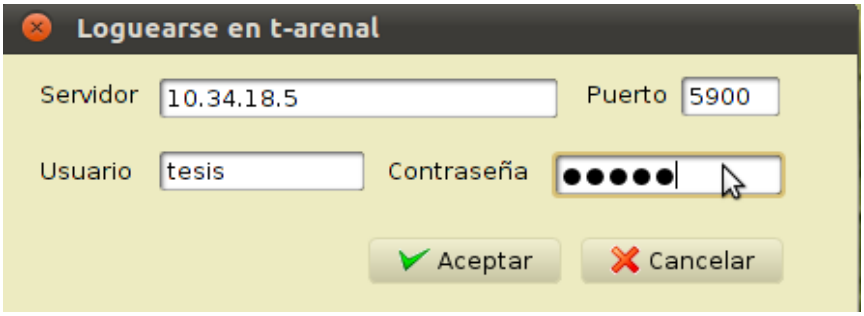
Figura 2: Diagrama de Casos de Uso del Sistema.

2.3.4 Descripción de los casos de uso del sistema.

A continuación se realiza la descripción textual de los casos de uso del sistema.

Tabla 2. Descripción del caso de uso Autenticar usuario a la plataforma t-arenal.

Caso de Uso:	Autenticar usuario a la plataforma t-arenal
Actores:	Probador
Resumen:	El caso de uso inicia cuando el probador inicia la aplicación. El probador introduce el usuario, la contraseña y la dirección de la plataforma y el puerto y se autentica en el sistema, terminando así el caso de uso.
Precondiciones:	El usuario debe existir en la base de datos de la plataforma t-arenal.
Referencias	RF1
Prioridad	Crítico
Flujo Normal de Eventos	

Acción del Actor	**Respuesta del Sistema**
1. El probador introduce los datos (dirección, puerto usuario, contraseña) y selecciona la opción aceptar.	**2.** Verifica que los datos sean correctos.
3. El probador accede a la aplicación satisfactoriamente. Termina el Caso de Uso.	
Prototipo de Interfaz de Usuario	
	
Flujos Alternos	
Acción del Actor	Respuesta del Sistema
	2.1 En caso de que la dirección o el puerto del servidor de t-arenal no sea correcto, se muestra un mensaje de error: “Verifique la dirección y el puerto”. Ir a la acción 1 del actor. **2.2** En caso de que el usuario o la contraseña estén incorrectos, el sistema muestra un mensaje de error: “Usuario o Contraseña incorrectos”. Ir a la acción 1 del actor.

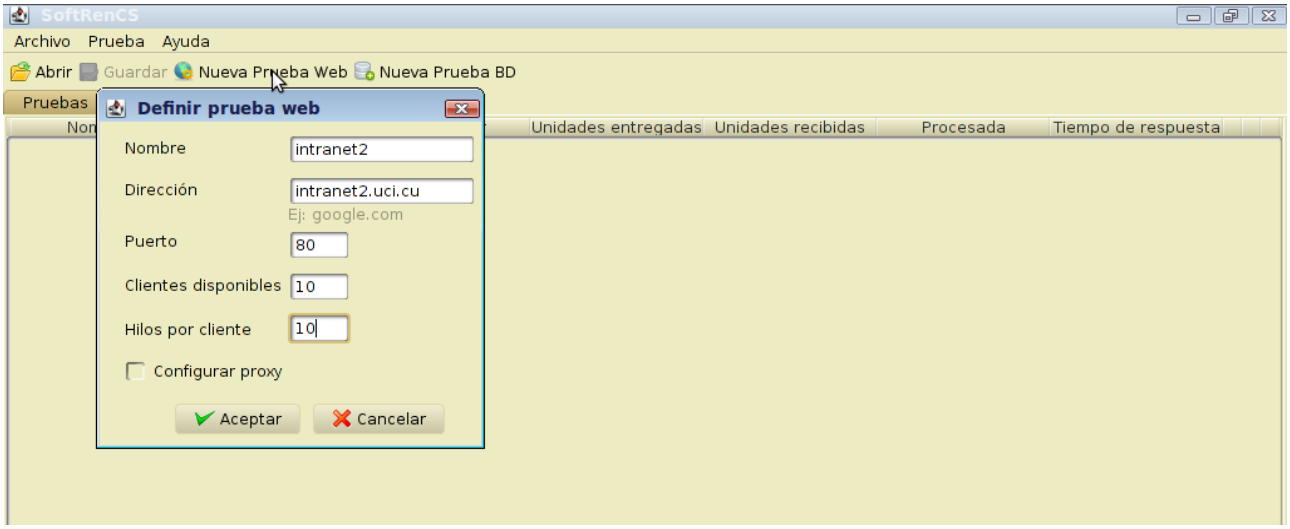
Prototipo de interfaz de usuario	
	
Poscondiciones	Queda autenticado el usuario en el sistema.

Tabla 3. Descripción del caso de uso Definir prueba.

Caso de Uso:	Definir prueba (Caso de uso base generalizado)	
Actores:	Probador	
Resumen:	El caso de uno inicia cuando el probador selecciona “Definir prueba” en el menú de Pruebas. Escoge el tipo de prueba a definir. Se realiza la sesión del caso de uso referente a la prueba seleccionada. Termina el caso de uso.	
Precondiciones:	El probador debe estar autenticado	
Referencias	RF2,RF3	
Prioridad	Crítico	
Flujo Normal de Eventos		
Acción del Actor		**Respuesta del Sistema**
1. El probador selecciona la acción “Nueva”.		**2.** El sistema muestra los tipos de pruebas a definir.
3. El probador escoge la prueba a definir. • Prueba Web • Prueba a una BD		**4.** Si escoge Prueba Web, ver caso de uso “Definir prueba Web”. **5.** Si escoge Prueba a una BD, ver caso de uso “Definir prueba BD”. **6.** Termina el caso de uso.

Tabla 4. Descripción del caso de uso Definir prueba Web.

Caso de Uso:	Definir prueba Web (Especializado)
Actores:	Probador
Resumen:	El caso de Uso inicia cuando el probador selecciona definir prueba web. Se introducen los datos de la página web a conectarse. Se guardan los datos de la página. Termina el caso de uso.
Precondiciones:	El probador debe estar autenticado

Referencias	RF2
Prioridad	Crítico
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. Entra los datos referentes a la prueba web (Nombre, Dirección de la página, puerto de entrada a la página, proxy, puerto del proxy, usuario, contraseña y dominio, clientes disponibles y cantidad de hilos por cliente).	**2.** Verifica que los datos sean correctos (nombre, puertos, cantidad de hilos por cliente, clientes disponibles).
3. El usuario selecciona la acción "Aceptar".	**4.** El sistema guarda los datos referentes a la prueba en un fichero. Termina el Caso de uso.
Prototipo de interfaz de usuario.	
	
Flujos Alternos	
Acción del Actor	Respuesta del Sistema
	2.1 Si el nombre comienza con número muestra un mensaje de error "El nombre no debe

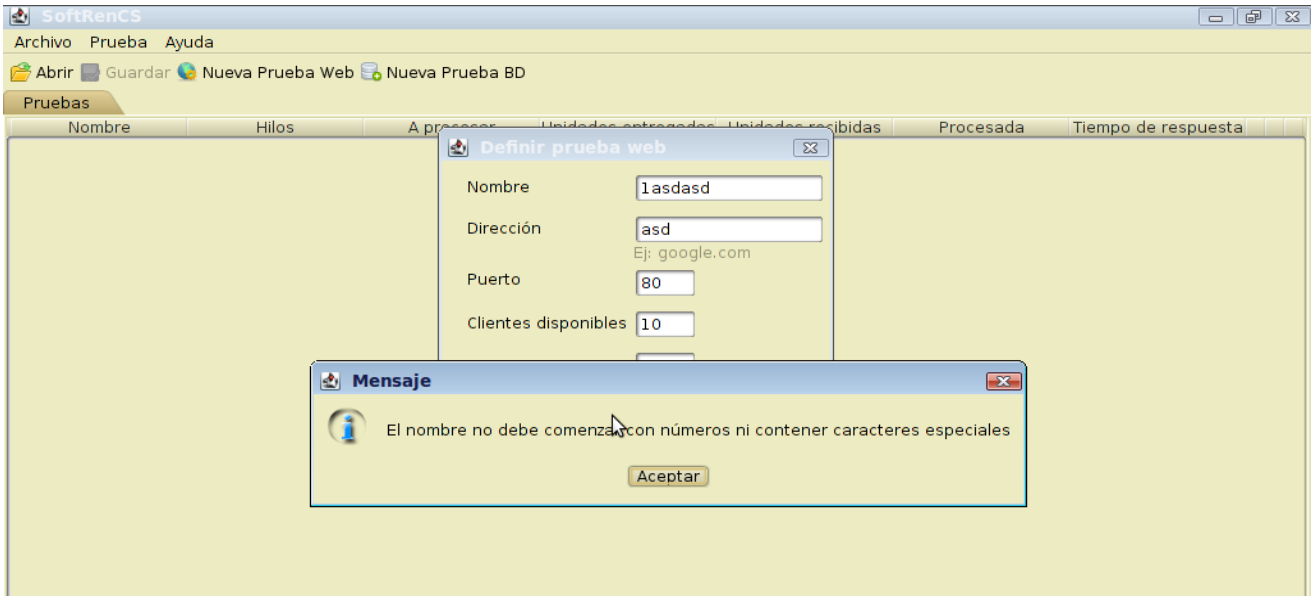
	comenzar con números ni tener caracteres especiales". Ir a la acción 1 del actor. **2.2** Si el usuario intenta escribir caracteres en los campos mencionados en la acción 2 del actor excluyendo el nombre. El sistema no permite escribir letras ni caracteres a no ser números.
Prototipo de interfaz de usuario	
	
Poscondiciones	Los datos de la prueba web quedan guardados en un fichero.

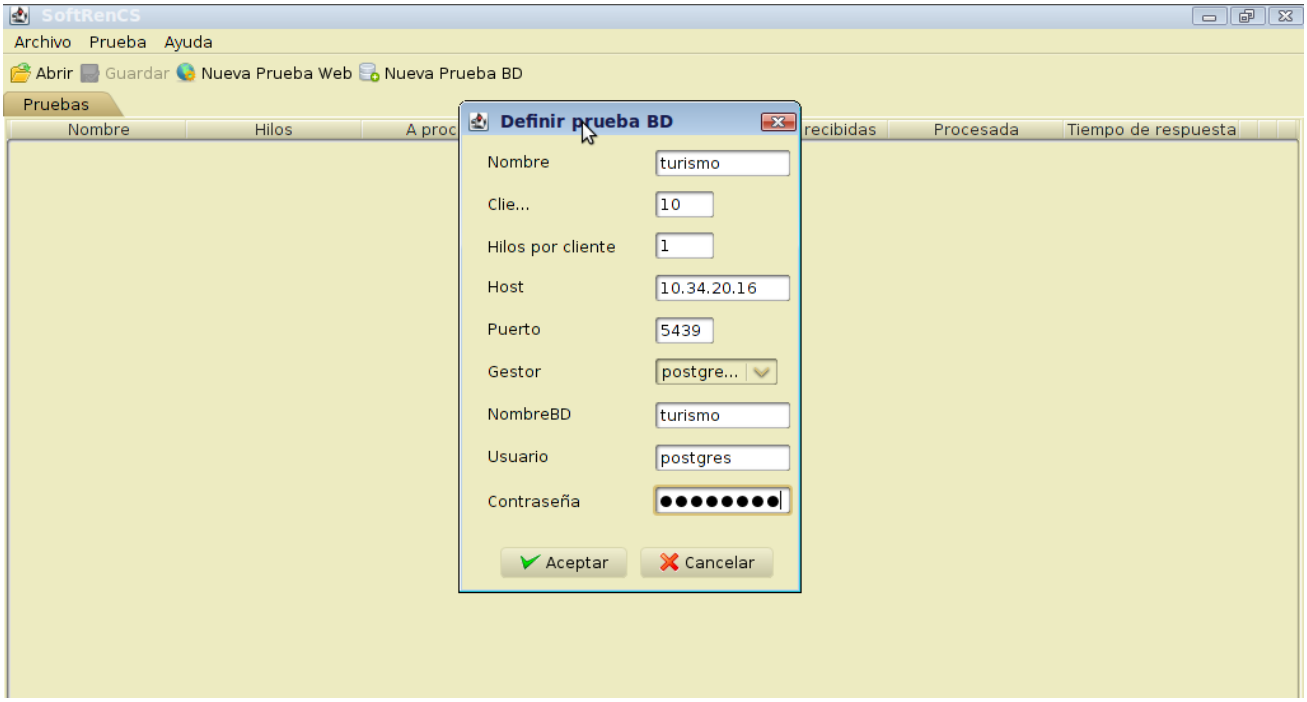
Tabla 5. Descripción del caso de uso Definir prueba BD.

Caso de Uso:	Definir prueba BD (Especializado)
Actores:	Probador
Resumen:	El caso de Uso inicia cuando el probador selecciona definir prueba a la BD. Se introducen los datos de la base de datos a probar. Se guardan los datos

	de la prueba. Termina el caso de uso.
Precondiciones:	El probador debe estar autenticado
Referencias	RF3
Prioridad	Crítico

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. Entra los datos referentes a la prueba BD (host, puerto, usuario, contraseña, nombre de base de datos y gestor de base de datos).	**2.** Verifica que los datos sean correctos (nombre, puerto, cantidad de hilos por cliente, clientes disponibles)
3. El usuario selecciona la acción "Aceptar".	**4.** El sistema guarda los datos referentes a la prueba en un fichero. Termina el Caso de uso.

Prototipo de interfaz de usuario



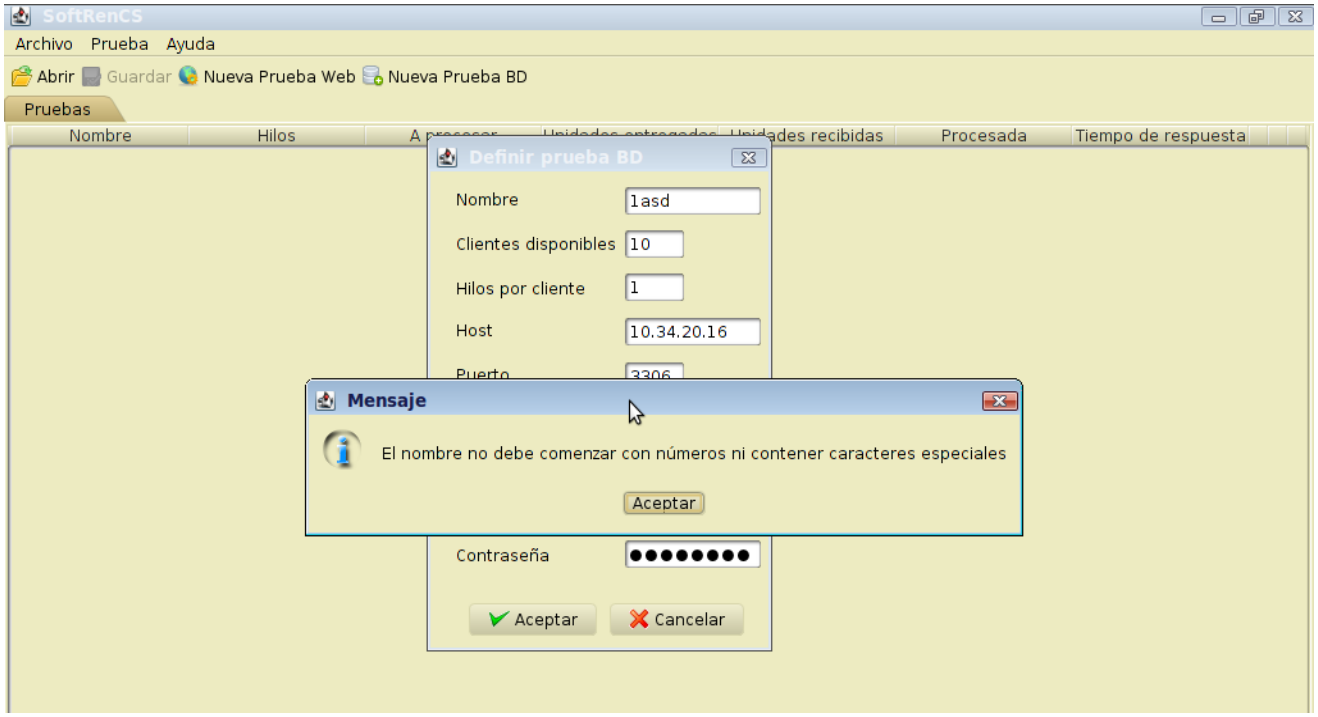
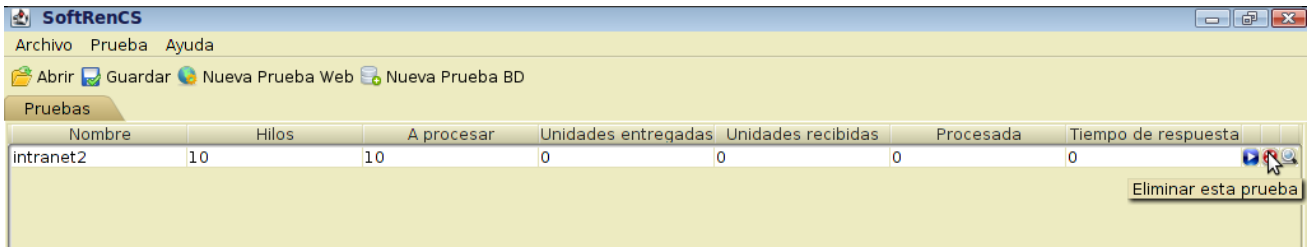
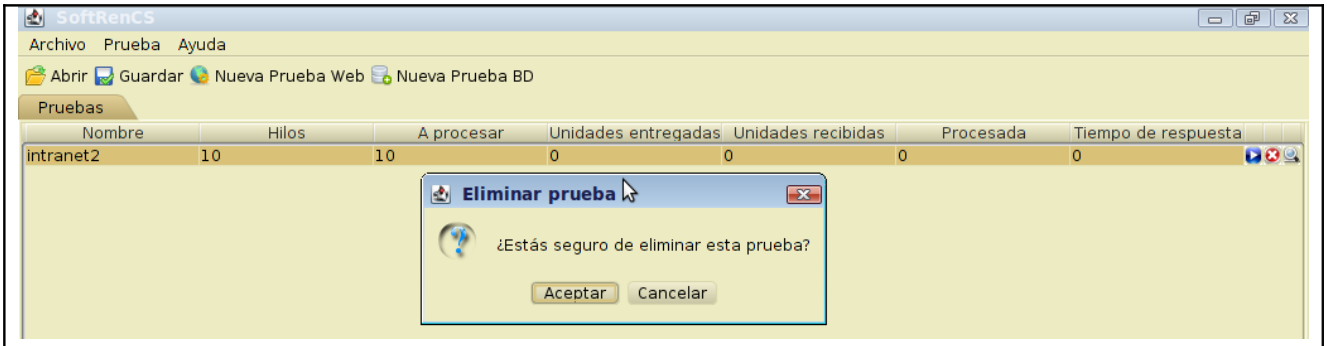
Flujos Alternos	
Acción del Actor	Respuesta del Sistema
	2.1 Si el nombre comienza con número muestra un mensaje de error "El nombre no debe comenzar con números ni tener caracteres especiales". Ir a la acción 1 del actor. **2.2** Si el usuario intenta escribir caracteres en los campos mencionados en la acción 2 del actor excluyendo el nombre. El sistema no permite escribir letras ni caracteres a no ser números.
Prototipo de interfaz de usuario	
	
Poscondiciones	Los datos de la base de datos quedan almacenados en un fichero.

Tabla 6. Descripción del caso de uso Eliminar pruebas.

Caso de Uso:	Eliminar pruebas
Actores:	Probador
Resumen:	El caso de Uso inicia cuando el probador selecciona elimina prueba en el botón "Eliminar" correspondiente a la prueba seleccionada. El sistema muestra un mensaje de confirmación. El probador decide eliminar la prueba. El sistema elimina la prueba seleccionada. Termina el caso de uso
Precondiciones:	Debe haber al menos una prueba definida.
Referencias	RF4
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El probador selecciona la opción eliminar prueba.	**2.** El sistema muestra un mensaje de confirmación "¿Está seguro de eliminar esta prueba?".
3. El probador selecciona la opción "Aceptar"	**4.** El sistema elimina la prueba satisfactoriamente. Termina el caso de uso.
Prototipo de interfaz de usuario	





Poscondiciones	Queda eliminada la prueba escogida por el probador.
Flujos Alternos	
Acción del Actor	Respuesta del Sistema
	2.1 Si el probador escoge la opción "Cancelar" se cancela la opción de eliminar prueba (CU). Termina el caso de uso.

Tabla 7. Descripción del caso de uso Ejecutar pruebas.

Caso de Uso:	Ejecutar pruebas
Actores:	Probador
Resumen:	El caso de Uso inicia cuando el probador selecciona la opción ejecutar prueba en el botón "Ejecutar" correspondiente a la prueba seleccionada. El sistema carga el fichero de información de la prueba ya almacenada y envia dicho fichero al servidor de t-arenal y ejecuta dicha prueba. Termina el caso de uso.
Precondiciones:	Debe estar definida la prueba.
Referencias	RF5
Prioridad	Crítico

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El probador selecciona la opción "Ejecutar prueba".	**2.** El sistema carga el fichero almacenado con los datos de la prueba seleccionada. **3.** El sistema sube el fichero de configuración al servidor de t-arenal y lo ejecuta. **4.** Termina el caso de uso.

Prototipo de interfaz de usuario

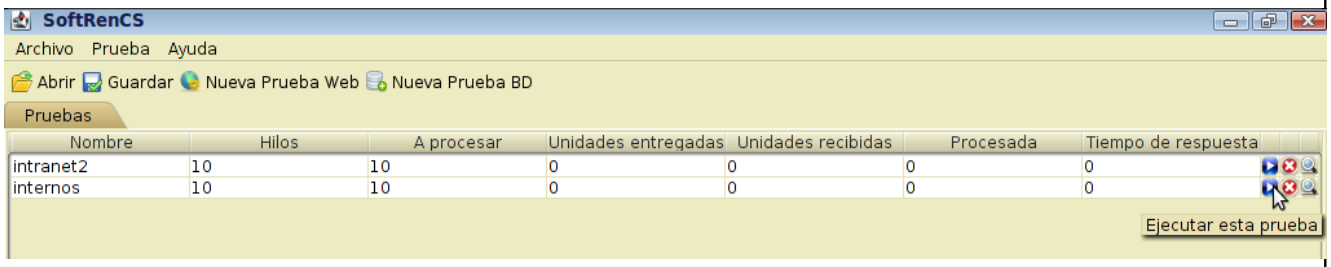
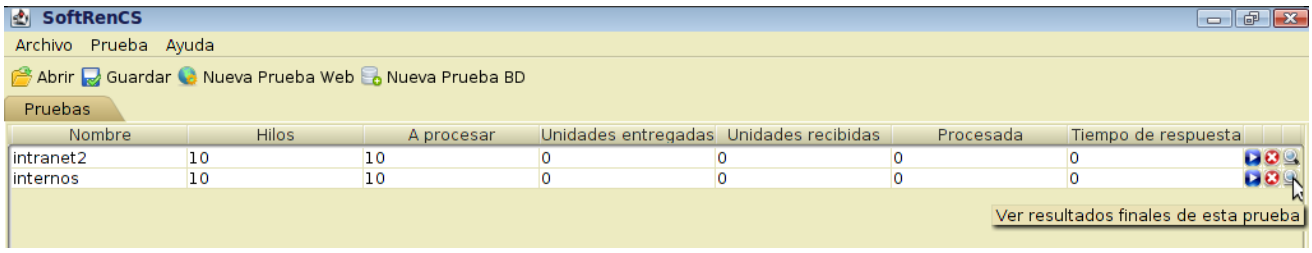
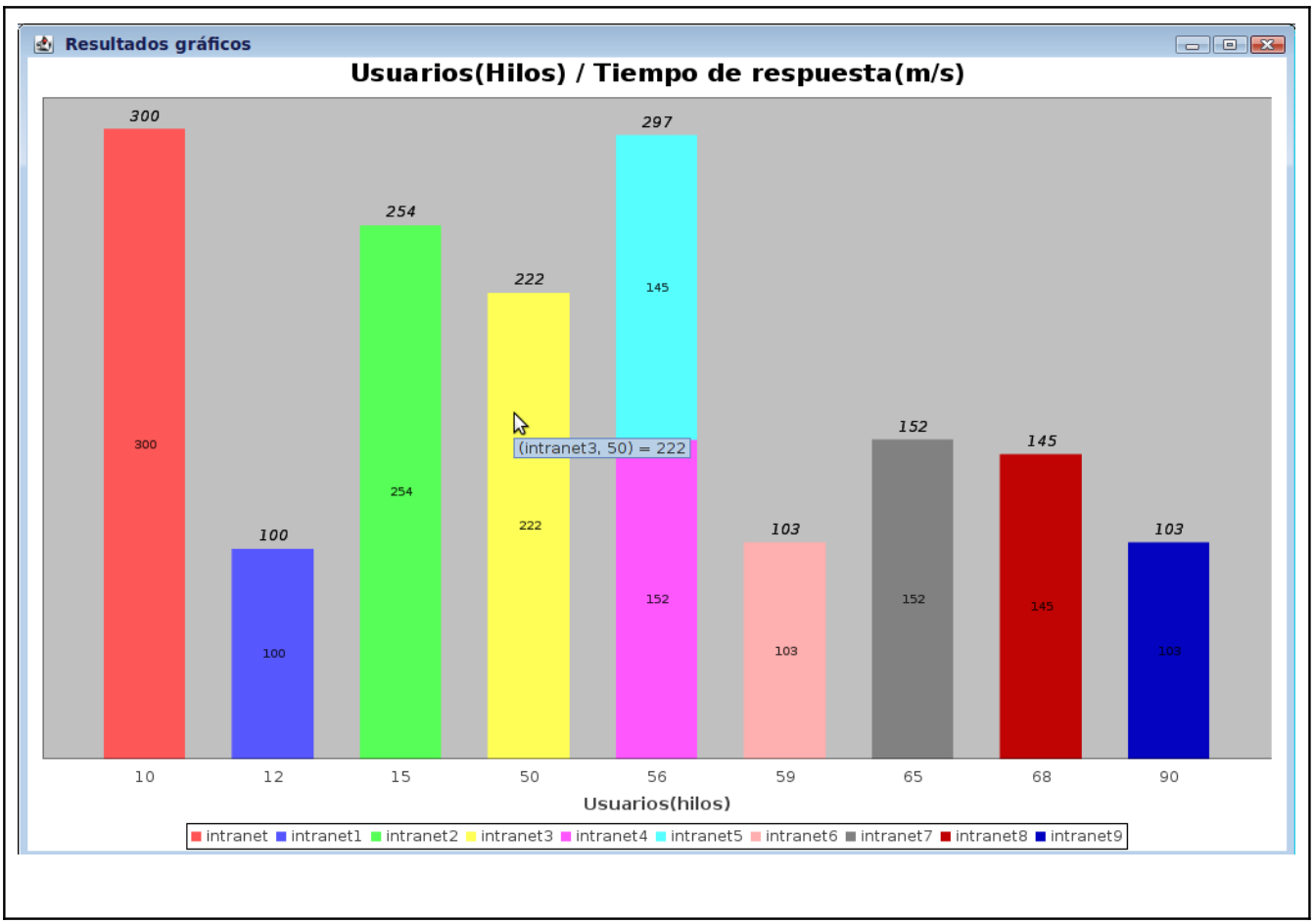


Tabla 8. Descripción del caso de uso Obtener Resultados.

Caso de Uso:	Obtener resultados
Actores:	Probador
Resumen:	El caso de uso inicia cuando el actor selecciona el botón “Ver resultados” correspondiente a la prueba seleccionada. El Probador escoge la prueba. El sistema muestra los resultados de la prueba escogida. Termina el caso de uso
Precondiciones:	Deben existir pruebas realizadas.
Referencias	RF6
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El probador selecciona la opción “Ver resultados”.	**2.** El sistema muestra un gráfico con los resultados de la prueba. Termina el caso de uso.
Prototipo de interfaz de usuario	





2.4 Descripción de la vista de caso de uso.

La vista de casos de uso contiene los casos de usos más significativos para la arquitectura, ya que describen las funcionalidades más importantes y críticas que deben realizarse con mayor prioridad, la cual se representa a continuación:

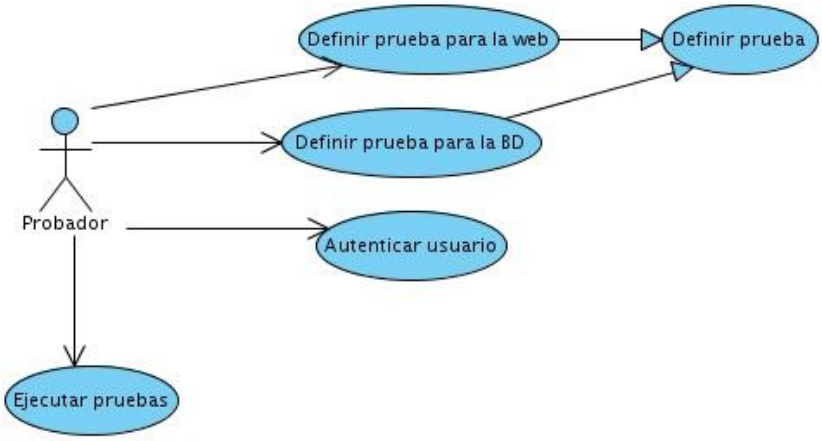


Figura 3: Diagrama de casos de usos significativos.

2.5 Patrones de diseño.

A continuación, se dan a conocer los patrones de diseño que se tienen en cuenta para la arquitectura del sistema.

- Controlador: Representa el sistema global, dispositivo o subsistema (controlador de fachada). Este patrón se encuentra presente en la herramienta, puesto que existe una clase que es capaz de realizar las acciones más importantes del sistema y que ésta implica otras clases.
- Alta cohesión: La cohesión es una medida de cuán relacionadas y enfocadas están las responsabilidades de una clase. Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme. Una clase con alta cohesión tiene la responsabilidad de realizar una única labor identificable en el sistema. Este patrón está enmarcado en la herramienta debido a que cada clase tendrá sus responsabilidades en la misma.
- Bajo acoplamiento: Dentro del modelo del diseño es necesario que colaboren algunas clases con las otras. Sin embargo, la colaboración debe mantenerse en un mínimo

aceptable. Si un modelo de diseño tiene un acoplamiento alto, las clases de diseño colaboran con todas las otras clases de diseño, el sistema es difícil de implementar, probar y mantener a través del tiempo.

2.6 Patrón arquitectónico Cliente-Servidor.

El patrón es una descripción del problema y la esencia de su solución, de forma que la solución pueda reutilizarse en diferentes situaciones, no constituye una especificación detallada, es una solución adecuada a un problema común. En la figura 4, se muestra el patrón utilizado cliente-servidor entre la herramienta y el sistema distribuido. *La PC_herramienta y los Clientes_t-arenal* actúan como clientes. El *Servidor t-arenal* siempre se comporta como servidor, en el momento de enviarle la respuesta de la tarea a la PC_herramienta y para darle una tarea al *Servidor de Peticiones*. El Servidor de peticiones se comporta como cliente o servidor. Como cliente al solicitar una tarea al *Servidor_t-arenal (Servidor Central)* y como servidor cuando distribuye la tarea a los *clientes_t-arenal* que éstos solicitan.

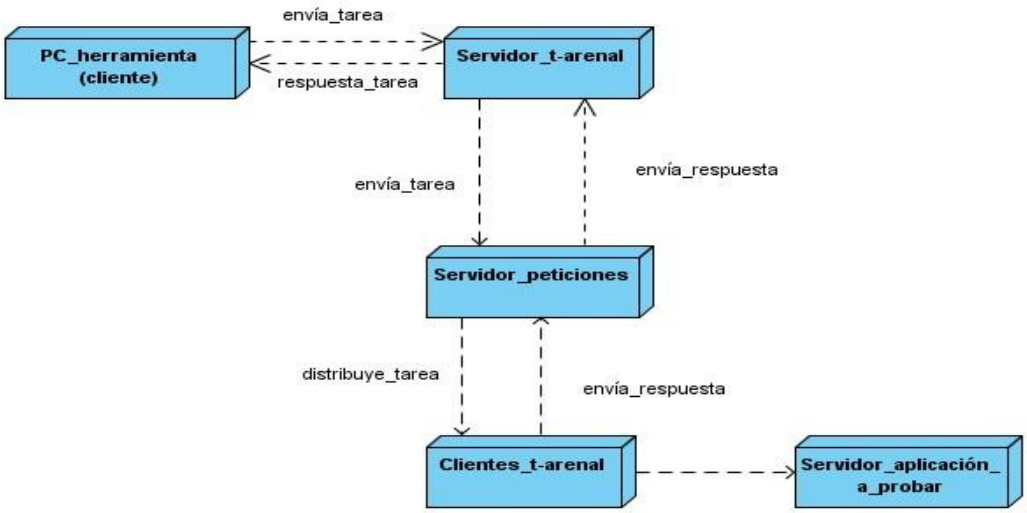


Figura 4: Patrón arquitectónico cliente – servidor aplicado al sistema.

2.7 Descripción de la vista lógica.

La vista lógica, contiene las clases del diseño más importantes, organizadas por paquetes y subsistemas en capas de trabajo. Esta vista describe los paquetes en los que se dividirá la herramienta. En la figura siguiente, se muestra la vista lógica de la herramienta.

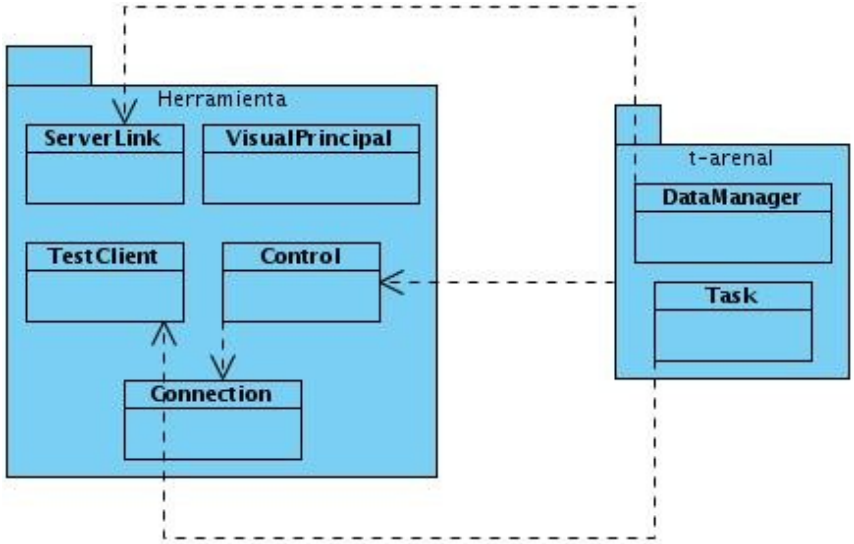


Figura 5: Vista lógica.

- **VisualPrincipal**: Representa la clase visual que interactúa con la plataforma t-arenal.
- **DataManager**: Clase de lado del servidor que se encarga de distribuir las tareas por los clientes de forma uniforme.
- **Task:** Clase que se ejecuta en los clientes una vez que se les asigna una tarea.
- **ServerLink:** Clase de la herramienta, que hereda de la clase **Datamanager**. En esta clase se implementan los métodos abstractos de la clase padre a conveniencia de las acciones de la herramienta, es la representación del **Datamanager** en la herramienta.

- **TestClient:** Clase que hereda de **Task.** Esta clase redefine el método abstracto de la clase padre, para ejecutar las tareas en los clientes de acuerdo a lo implementado, es la representación del **Task** en la herramienta.
- **Control**: Clase controladora, capaz de conectar la herramienta con t-arenal y hace el resto de las funcionalidades generales con las clases entidades.

2.8 Diagrama de clases del diseño

A continuación se muestra el diagrama de clases del diseño, el cual modela la relación entre las clases del sistema.

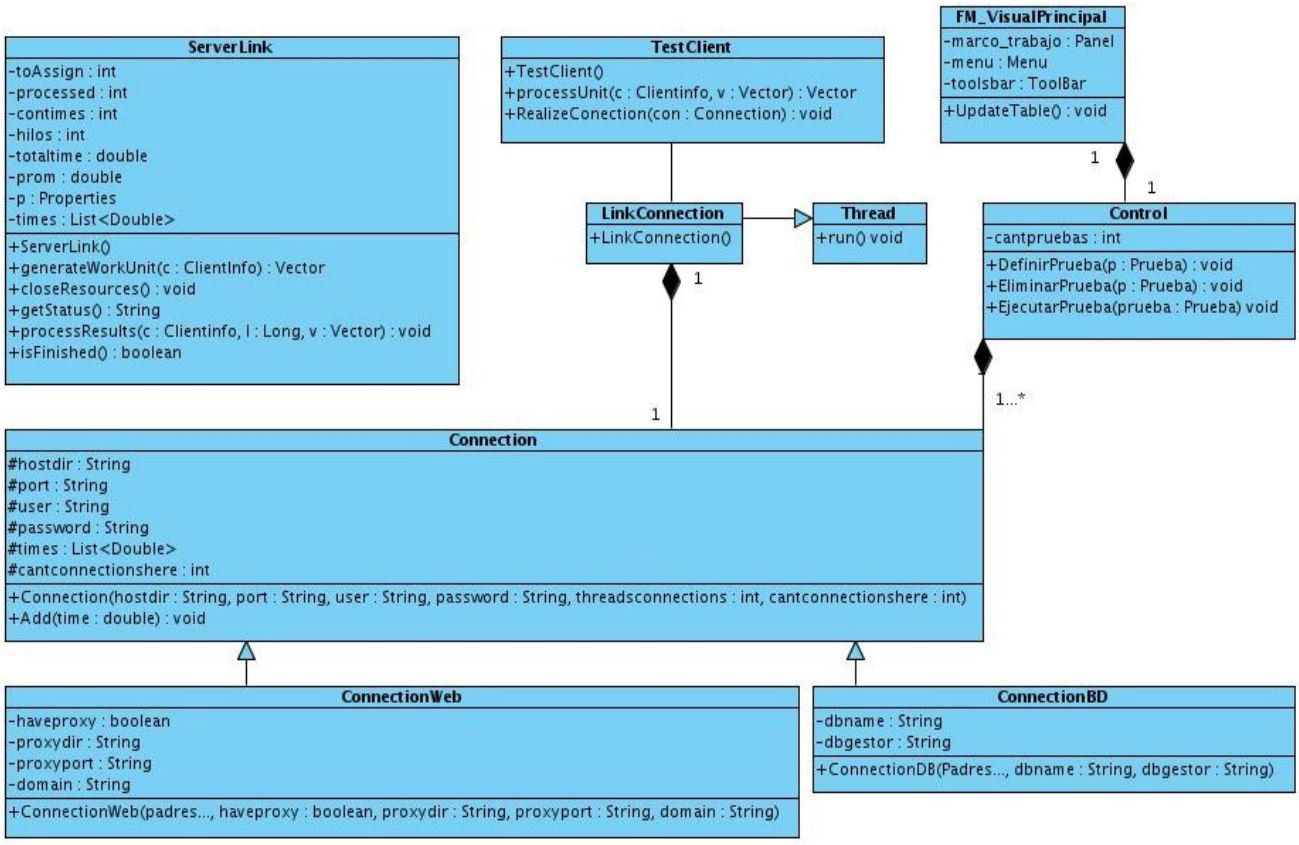


Figura 6: Diagrama de clases del diseño.

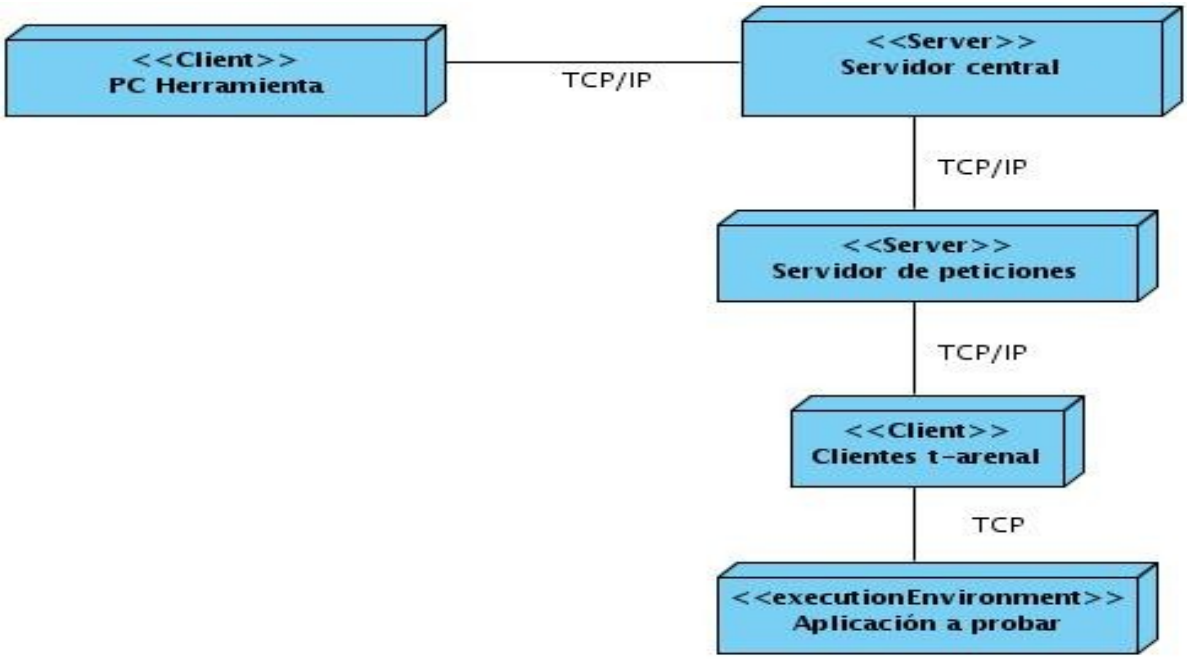


Figura 7: Vista de despliegue

2.9 Descripción de la vista de despliegue.

La vista de despliegue describe los principales nodos físicos que se necesitan para desplegar el sistema y la relación entre ellos mediante los protocolos de comunicación. En la figura 7, se muestra una propuesta de la vista de despliegue de la herramienta.

PC Herramienta: Se refiere a la computadora donde se encuentra la herramienta.

Servidor central: Se refiere al servidor central de t-arenal, el cual realizaría las acciones para distribuir las tareas a los servidores de peticiones.

Servidor de peticiones: Servidores encargados de distribuir las tareas a los clientes de prueba.

Cliente prueba: Clientes en los cuales se realizan las tareas enviadas desde el servidor (en este caso las pruebas).

Aplicación a probar: Aplicación Cliente-Servidor (Web o BD), a la cual se le hacen las pruebas.

2.10 Conclusiones del capítulo

En este capítulo se definieron los requisitos funcionales y no funcionales del sistema. Se definieron los actores que interactúan con el software, así como los casos de usos referentes a las funcionalidades del mismo. Se realizó el diagrama de casos de uso del sistema, así como la descripción de los mismos. Además se definió el modelo de diseño, donde se muestran los diagramas del mismo; tales como diagrama de clases del diseño, vista de despliegue, vista lógica, entre otros. Se definieron los patrones de diseños que identifican el funcionamiento del sistema en términos de relaciones de clases. Se definió el patrón arquitectónico Cliente-Servidor dado que la plataforma trabaja con el mismo.

CAPÍTULO 3: IMPLEMENTACIÓN Y VALIDACIÓN DEL SISTEMA.

En este capítulo se describe cómo los elementos del modelo de diseño se implementan en términos de componentes. Para esto se muestra el diagrama de componentes. Además se muestra el código fuente de los principales métodos de la aplicación. Finalmente se realizarán pruebas de validación al sistema para verificar su correcto funcionamiento.

3.1 Diagrama de componentes.

Un diagrama de componentes muestra las organizaciones y dependencias lógicas entre componentes de software, sean estos componentes de código fuente, binarios o ejecutables. Desde el punto de vista del diagrama de componentes se tienen en consideración los requisitos relacionados con la facilidad de desarrollo, la gestión del software, la reutilización y las restricciones impuestas por los lenguajes de programación y las herramientas utilizadas en el desarrollo.

Estereotipos utilizados.

- Archivo (File)
- Biblioteca (Library)

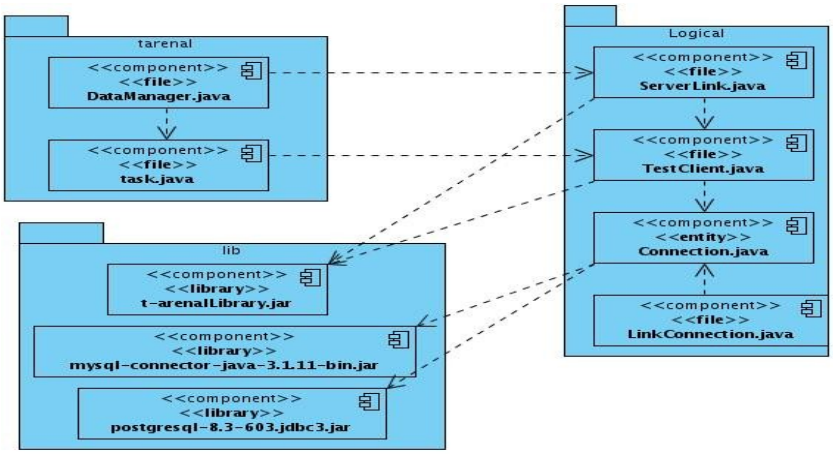


Figura 8: Diagrama de componentes.

3.2 Código fuente.

A continuación se describen las principales funcionalidades que se implementaron en la herramienta.

Existen seis clases fundamentales en la implementación de la aplicación.

3.2.1 ServerLink

Clase que hereda de la clase DataManager perteneciente a la librería (biblioteca) t-arenal, la cual se encarga de trabajar en la parte del servidor. Esta cuenta con cinco funcionalidades(métodos) fundamentales.

```
public Vector generateWorkUnit(ClientInfo arg0) throws Throwable {
    Vector vector = new Vector();
    if (cantUnidEntregadas == toAssign) {
       return null;
    }
    cantUnidEntregadas++;
    ipssinceserver.println(arg0.getIP());
    vector.add(p);
    vector.add(canthilosporclientes);
    return vector;
  }
```

Este método se encarga de enviar las unidades al servidor de peticiones y el mismo posteriormente a los clientes de t-arenal. Esta funcionalidad envía un vector con los datos referentes a la prueba almacenada en un fichero llamado datos.txt y la cantidad de hilos (usuarios a simular) que se ejecutarán en cada cliente. Este método se invocará mientras que el servidor no haya finalizado o la cantidad de unidades entregadas sea igual a la cantidad de unidades asignadas.

```
public void processResults(ClientInfo arg0, Long arg1, Vector arg2)
        throws Throwable {
    Vector response = arg2;
    Vector times = (Vector) response.get(0);
    Vector errors = (Vector) response.get(1);
    conttimes=0;
    //String PageContent = (String) arg2.get(2);
    for (int i = 0; i < errors.size(); i++) {
       errores.println(errors.get(i));
    }
    for (int j = 0; j < times.size(); j++) {
       totaltime += Double.parseDouble(times.get(j).toString());
       conttimes++;
       canttimes++;
    }
    processed += conttimes;
    prueba.println(times.size());
    cantUnidRecibidas++;
    ipssinceclients.println(arg0.getIP() + " Veces conectadas " + (times.size()));
    if (toAssign == cantUnidRecibidas) {
       prom = totaltime / canttimes;
       PrintStream p1 = new PrintStream(new File(PROBLEMDIRECTORY, "result.txt"));
```

```
        p1.println(getStatus());
        p1.close();
    }
}
```

Este método se encarga de recibir las unidades y procesar los resultados obtenidos de las mismas. Recibe un vector el cual contiene los resultados de cada unidad recibida desde los clientes. Estos resultados están dados por los tiempos de respuesta de cada hilo ejecutado en los clientes; así como los errores reportados de los mismos. Una vez obtenidos los resultados se actualizan las variables de respuesta final (processed, cantUnidades recibidas, totaltime, canttimes). Cuando se finalicen las peticiones, se guarda en un fichero el contenido del método getStatus().

```
public String getStatus() throws Throwable {
    double s = 0;
    double min = 0;
    if(prom<1000){
      s = Math.rint(prom*100)/100;
       timetext = "" + s + " ms";
    }
    if (prom >= 1000) {
       s = Math.rint((prom/1000)*100)/100;
       timetext = "" + s + " seg";
    }
    if (prom >= 60000) {
       min = Math.rint((prom/60000)*100)/100;
       timetext = "" + min + " min";
```

```
    }
    return "Tareas a Procesar: " + toAssign + "\n" + "Unidades entregadas: "
        + cantUnidEntregadas + "\n" + "Unidades Recibidas: " + cantUnidRecibidas + "\n"
        + "Tareas procesadas: " + processed + "\n" + "Promedio: " + timetext;
  }
```

Una vez finalizada la ejecución(*véase método isFinished()*), se invoca el método CloseResources(), el cual se encarga de cerrar los flujos de ficheros que se han quedado abiertos u otras funcionalidades.

```
public void closeResources() throws Throwable {
    ipssinceserver.close();
    ipssinceclients.close();
    prueba.close();
    errores.close();
  }
public boolean isFinished() throws Throwable {
    return toAssign == cantUnidRecibidas;
  }
```

3.2.2 TestClient

Clase que hereda de la clase Task de la librería t-arenal, la cual se ejecuta en los clientes de t-arenal. Cuenta con un método fundamental.

```
  public Vector processUnit(ClientInfo arg0, Vector arg1) throws Throwable {
    Properties p = (Properties) arg1.get(0);
    hilos = Integer.parseInt(p.getProperty("Conexionesporcliente"));
    Vector result = new Vector();
```

```
Connection con=null;
if (p.getProperty("TipoDeConexion").equalsIgnoreCase("Web")) {
   String nombre = p.getProperty("Nombre");
   String direccion = p.getProperty("Direccion");
   String puerto = p.getProperty("Puerto");
   boolean haveproxy = Boolean.parseBoolean(p.getProperty("TieneProxy"));
   String dirproxy = p.getProperty("DireccionProxy");
   String usuario = p.getProperty("Usuario");
   int cantconex = Integer.parseInt(p.getProperty("Conexiones"));
   int coninclient = Integer.parseInt(p.getProperty("Conexionesporcliente"));
   String dominio = p.getProperty("Dominio");
   String password = p.getProperty("Password");
   String proxyport = p.getProperty("PuertoProxy");
      con = new ConnectionWeb(nombre, direccion, puerto, usuario, password, cantconex,
coninclient, haveproxy, dirproxy, proxyport, dominio);
} else {
   if (p.getProperty("TipoDeConexion").equalsIgnoreCase("BD")) {
      String nombre = p.getProperty("Nombre");
      String direccionbd = p.getProperty("DireccionBD");
      String puertobd = p.getProperty("PuertoBD");
      String nombrebd = p.getProperty("NombreBD");
      String usuariobd = p.getProperty("UsuarioBD");
      int cantconex = Integer.parseInt(p.getProperty("Conexiones"));
```

```
            int coninclient = Integer.parseInt(p.getProperty("Conexionesporcliente"));
            String gestorbd = p.getProperty("GestorBD");
            String passwordbd = p.getProperty("PasswordBD");
                con = new ConnectionDB(nombre, direccionbd, puertobd, usuariobd, passwordbd,
cantconex, coninclient, nombrebd, gestorbd);
          }
        }
        RealizeLinkConnection(con, hilos);
        result.add(timesvector);
        result.add(errorsvector);
        return result;
    }
```

Este método es el que realiza la acción de ejecutar las pruebas, el mismo crea un objeto (*Connection*) a partir de los valores del fichero recibido desde el servidor de t-arenal. Una vez creado el objeto, se invoca al método *RealizeLinkConnection(con,hilos),* el cual se encarga de conectarse al servidor de aplicaciones recibido una n-hilos cantidad de veces.

```
 private void RealizeLinkConnection(Connection con, int hilos) throws InterruptedException {
        LinkConnection[] links = new LinkConnection[hilos];
        for (int i = 0; i < hilos; i++) {
           links[i] = new LinkConnection(con, hilos);
        }
        for (int i = 0; i < hilos; i++) {
           links[i].start();
```

```
        links[i].join();
    }
    for (int i = 0; i < hilos; i++) {
        timesvector.add(links[i].getTimerequest());
        if(!links[i].getError().equals(""))
        errorsvector.add(links[i].getError());
      }
    }
```

Una vez ejecutado el método anterior, se almacena en los vectores timesvector y errorsvector los tiempos de respuesta y los errores reportados de la conexión de cada hilo. (Véase clase LinkConnection). Luego envía la respuesta en un vector con estos dos vectores como parámetro al servidor de t-arenal.

3.2.3 LinkConnection

Clase que hereda de Thread, es la encargada de realizar las conexiones a las páginas o base de datos de pruebas a partir de su método principal.

```
public void run() {
    DoConnections();
  }
private void DoConnections() {
    String direccion = "";
    String dirfile = "/";
    // si es a una URL
    if (con instanceof ConnectionWeb) {
```

```
final String user;
ConnectionWeb topc = (ConnectionWeb) con;
String port = topc.getPort();
direccion = topc.getHostdir();
if (topc.Haveproxy()) {
   System.setProperty("http.proxyHost", topc.getProxydir());
   System.setProperty("http.proxyPort", topc.getProxyport());
   System.setProperty("http.proxyUser", topc.getUser());
   System.setProperty("http.proxyPassword", topc.getPassword());
   if (!topc.getDomain().equalsIgnoreCase("")) {
      user = topc.getDomain() + "\\" + topc.getUser();
   } else {
      user = topc.getUser();
   }
   final String pass = topc.getPassword();
   Authenticator.setDefault(new Authenticator() {
      @Override
      public PasswordAuthentication getPasswordAuthentication() {
         return new PasswordAuthentication(user, pass.toCharArray());
      }
   });
}
```

```
    String[] dirfiledir = direccion.split("/");
    direccion = dirfiledir[0];
    for (int j = 1; j < dirfiledir.length; j++) {
        dirfile += dirfiledir[j] + "/";
    }
    try {
        URL url = new URL("Http", direccion, Integer.parseInt(port), dirfile);
        java.net.URLConnection conurl = url.openConnection();
        inicialtime = System.currentTimeMillis();
        BufferedReader in = new BufferedReader(
                new InputStreamReader(conurl.getInputStream(), "UTF-8"));
        finaltime = System.currentTimeMillis();
        timerequest = finaltime - inicialtime;
    } catch (Exception e) {
        error =e.getMessage();
        return;
    }
} else {
    ConnectionDB topc = (ConnectionDB) con;
    String gestor = topc.getDbgestor();
    if (gestor.equalsIgnoreCase("postgresql")) {
        try {
```

```
            Class.forName("org.postgresql.Driver");
        } catch (ClassNotFoundException e) {
            error = "Verifique el gestor";
            con.AddError(error);
        }
    } else {
        try {
            Class.forName("com.mysql.jdbc.Driver");
        } catch (ClassNotFoundException e) {
            error = "Verifique el gestor";
            con.AddError(error);
        }          }
    String host = topc.getHostdir();
    String dabasename = topc.getDbname();
    String port = topc.getPort();
    String user = topc.getUser();
    String pass = topc.getPassword();
    String concat = "jdbc:" + gestor + "://" + host + ":" + port
            + "/" + dabasename;
    inicialtime = System.currentTimeMillis();
    java.sql.Connection condb = null;
    try {
```

```
            condb = DriverManager.getConnection(concat, user, pass);
            finaltime = System.currentTimeMillis();
            timerequest = finaltime - inicialtime;
            //condb.close();
        } catch (SQLException e) {
            error = e.getMessage();
            return;
        }
    } System.setProperties(null);
}
```

Este método es el encargado de realizar la conexión tanto a la página Web como a la base de datos y calcular el tiempo de respuesta de dicha conexión, así como los errores que esta pueda tener.

3.2.4 Clases entidad

Clase Connection: Clase entidad padre con los datos comunes a las conexiones tanto web como base de datos.

Clase ConnectionWeb: Clase entidad hija de la clase *Connection* la cual tiene los atributos de la web.

Clase ConnectionBD: Clase entidad hija de la clase *Connection* la cual tiene los atributos de la Base de datos.

3.3 Pruebas de Validación.

Una vez terminada la implementación, se realizaron pruebas de validación para verificar que las funcionalidades del sistema satisfacen las necesidades del cliente y que el mismo realiza las pruebas de forma satisfactoria.

Se tomó como muestra el servidor web[2] al cual se le realizaron las siguientes pruebas.

2 http://10.34.18.5:8888/osiatis

Primeramente se realizaron 10 pruebas incrementando la cantidad de usuarios en una escala de 100.

Clientes a usar: Es la cantidad definida por el probador al realizar las pruebas, ya que conoce la cantidad de clientes t-arenal listos para trabajar.

Clientes usados: Es la cantidad real de clientes que estaban disponibles y activos al realizar las pruebas.

Hilos por cliente: Es la cantidad de hilos definidos por el probador que se ejecutarán en cada cliente usado.

Tiempo de respuesta (ms): Es el tiempo de respuesta promedio que demoraron las conexiones.

Tiempo real de prueba (seg): Es el tiempo real que demora en realizarse las pruebas.

Usuarios simulados = Clientes a usar * hilos por clientes

Después de realizar las pruebas se obtuvieron los siguientes resultados.

Tabla 9: Resultados obtenidos para escala de 100 usuarios (osiatis).

Clientes a usar	Clientes usados	Hilos por cliente	Tiempo de respuesta (ms)	Tiempo real de prueba (seg)
100	11	1	296,6	85
100	3	2	328,22	110
100	15	3	450,3	116
100	16	4	378,09	80
100	8	5	510,1	135
100	8	6	515,9	90
100	10	7	635,71	115
100	14	8	651,8	100
100	16	9	590,75	88
100	13	10	697,15	121

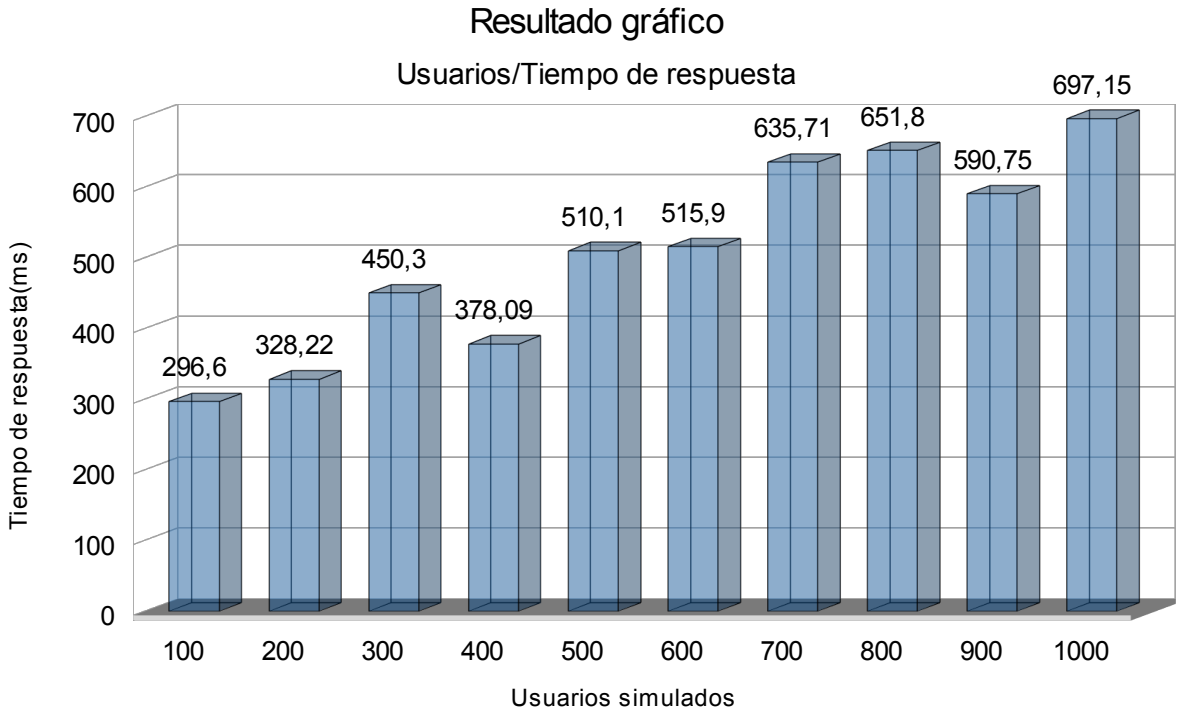


Figura 9: Resultados gráficos (Escala de 100 usuarios)

Posteriormente se realizaron pruebas al mismo sitio pero en este caso con una escala de 1000 usuarios para aumentar la carga y observar los resultados comenzando simulando 2000 usuarios ya que en la prueba anterior terminó con 1000.

Tabla 10: Resultados obtenidos para escala de 1000 usuarios(osiatis).

Clientes a usar	Clientes usados	Hilos por cliente	Tiempo de respuesta (ms)	Tiempo real de prueba (seg)
100	5	20	640,2	280
100	10	30	1488,19	291
100	5	40	670,13	340
100	5	50	681,9	550
100	4	60	672,7	498
100	7	70	712,6	580
100	5	80	681,1	571
100	7	90	701,71	602
100	10	10	2105,8	709
100	5	110	710,15	615

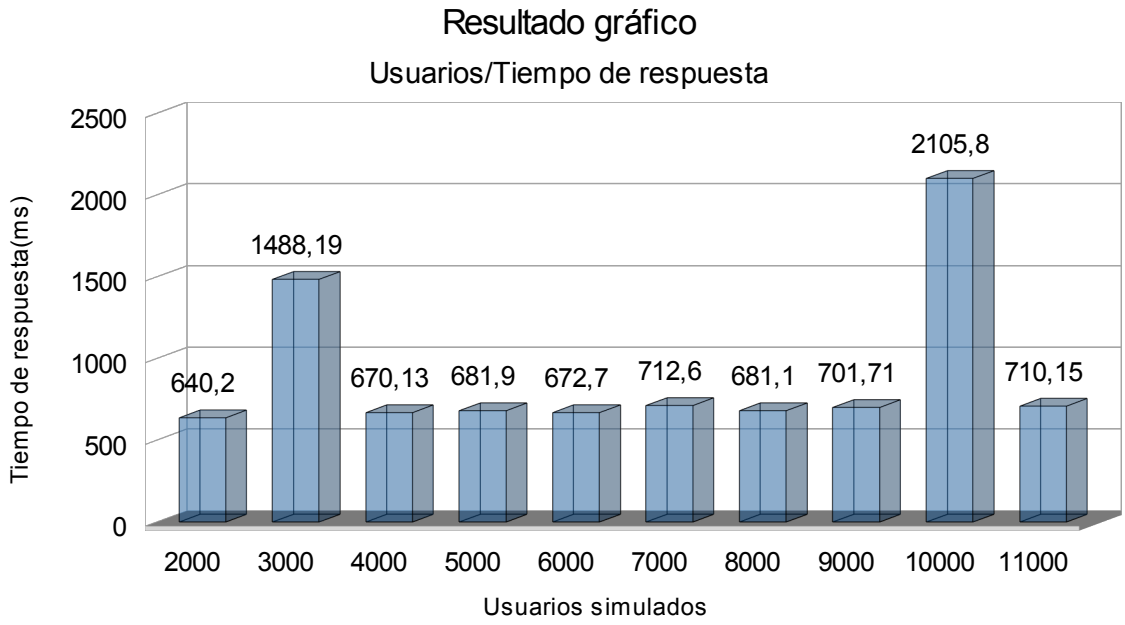


Figura 10: Resultados gráficos (Escala de 1000 usuarios)

Como se pudo observar, este servidor no llegó a su punto de ruptura, puesto que puede soportar una carga de 11000 usuarios. Este resultado depende de varios factores, fundamentalmente de la cantidad de clientes de t-arenal que estén disponibles y activos para realizar las pruebas. De esta forma mientras más clientes activos haya, mayor será la concurrencia en el servidor. Además depende también de la velocidad de la red así como otros factores de rendimiento que hacen posible los resultados obtenidos.

Luego se tomó como muestra el servidor de base de datos (turismo). Perteneciente al grupo de almacenes del centro DATEC.

Tabla 11: Resultados obtenidos para escala de 10 usuarios(turismo).

Clientes a usar	Clientes usados	Hilos por cliente	Tiempo de respuesta (ms)	Tiempo real de prueba (seg)
10	5	1	103,2	43
10	5	2	99,8	59
10	4	3	75,3	33
10	5	4	87,5	37
10	7	5	116,98	62
10	5	6	121,4	79
10	9	7	135,56	82
10	5	8	37,94	61
10	7	9	47,85	58
10	5	10	110,73	45
10	5	11	101,52	67
10	9	12	95,9	72
10	7	13	Error	82

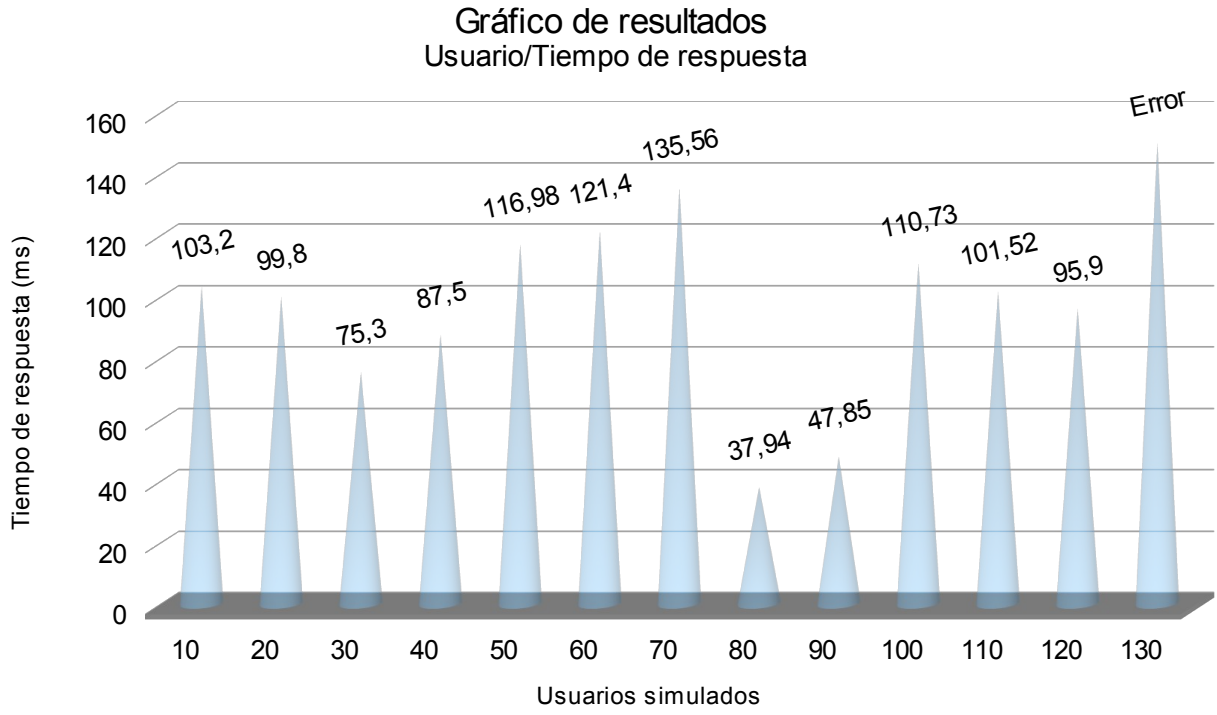


Figura 11: Resultados gráficos (Escala de 10 usuarios)

Luego de haber realizado las pruebas a esta base de datos podemos llegar a la conclusión que la misma soporta como máximo 120 usuarios ya que cuando se trató de simular 130 usuarios, dio error, por lo que encontramos el punto de ruptura en el valor 120, siendo este el valor máximo de conexiones que dicha base de datos puede soportar.

3.4 Conclusiones del capítulo

En este capítulo se definió el diagrama de componentes en vista de paquetes. Además se expuso el código fuente de las funcionalidades principales de la herramienta. Finalmente se realizaron pruebas al servidor web osiatis y al servidor de bases de datos turismo para validar que los requisitos funcionales funcionan correctamente y ver los resultados de las pruebas realizadas.

CONCLUSIONES

1. Se definieron las funcionalidades de la herramienta para realizar pruebas de rendimiento de forma satisfactoria.
2. Se realizó el diseño del sistema usando tres patrones de diseño fundamentales, alta cohesión, bajo acoplamiento y el patrón controlador. Además se usó como patrón arquitectónico el patrón cliente-servidor.
3. Se realizó la implementación del sistema haciendo uso de la biblioteca(librería) t-arenal.
4. Se realizaron pruebas de validación al sistema realizando pruebas a dos servidores (osiatis y turismo) obteniendo el resultado de los testeos de forma satisfactoria.

RECOMENDACIONES

1. Implementar las funcionalidades que permitan realizar pruebas a consultas a las bases de datos.
2. Implementar las funcionalidades que permitan realizar pruebas a otros servidores de aplicaciones, como subversion, de datos, de correo, entre otros.

REFERENCIAS BIBLIOGRÁFICAS

1. Gonzales, J. Las normas de la calidad del software. Addison-Wesley. Iberoamericana. España. 2002. [Citado 25/11/2010].
2. Pressman, R. Ingeniería del Software: Un enfoque Práctico. McGraw Hill. 2002. [Citado 25/11/2010]
3. **IEEE.** Computer Dictionary. Computer Society 1990. [Citado 27/11/2010].
4. Javier J. Gutiérrez, María J. Escalona, Manuel Mejías y Antonia M. Reina . MODELOS DE PRUEBAS PARA PRUEBAS DEL SISTEMA . Universidad de Sevilla . 2006. [Citado 3/12/2010]. Disponible en: (http://users.dsic.upv.es/workshops/dsdm06/files/dsdm06-07-Gutierrez.pdf)
5. Manual de usuario JMeter (ingles): [Citado 5/12/2010.]

   Disponible en:(http://jakarta.apache.org/jmeter/usermanual/index.html.)
6. Corporación Sybven. Portal de Sybven. 2011. [Citado 9/12/2010]. Disponible en: (http://www.corporacionsybven.com/portal/index.php?option=com_content&view=article&id=91:rational-performance-tester&catid=77:calidad-de-software-&Itemid=104).
7. JCrawler. A perfect Load. Sitio Oficial de Jcrawler. [Citado 15/12/2010]. Disponible en : (http://jcrawler.sourceforge.net/).2004.
8. Solex. Web application testing with Eclipse. Sitio Oficial de Solex. [Citado 21/12/2010]. Disponible en: (http://solex.sourceforge.net/).
9. FJRP, FMBR 2008/09 ccia SCS-Sistemas Cliente/Servidor . Marzo 2008. [Citado 7/01/2011]
10. Reynoso . Carlos Billy. Introducción a la Arquitectura de Software . Marzo de 2004 . [Citado 15/01/2011].
11. CBASQA. Proceso de Desarrollo Open UP. Septiembre 2008. [Citado 21/01/2011]. Disponible en (http://cbasqa.wordpress.com/2008/09/02/proceso-de-desarrollo-openup/).

12. Sitio Oficial de Eclipse. Disponible en: (http://www.eclipse.org). [Citado 22/01/2011].

13. **J, CARLOS M. ZAPATA.** *CICLO-P: UN MÉTODO PARA EL ACOPLAMIENTO DE PRUEBAS AL CICLO DE VIDA DEL SOFTWARE.* 2010.[citado 21/2/2011]

14. Javier García de Jalón . TECNUM. Aprenda Java como si estuviera en primero. 2000 [citado 18/2/2011].

15. JOSE GUILLERMO VALLE. UNIMINUTO . TECNOLOGÍA EN INFORMÁTICA . 2005. [citado 20/3/2011].

16. Pablo Nicolás Baeza Baeza. Visual Paradigm DB Visual Architect SQL. D y T Systems. [citado 23/3/2011].

17. Pressman, R. Ingeniería del Software: Un enfoque Práctico. McGraw Hill. 2002.[Citado 2/4/2011].

BIBLIOGRAFÍA

1. Gonzales, J. Las normas de la calidad del software. Addison-Wesley. Iberoamericana. España. 2002.
2. Pressman, R. Ingeniería del Software: Un enfoque Práctico. McGraw Hill. 2002.
3. **IEEE.** Computer Dictionary. Computer Society 1990.
4. Javier J. Gutiérrez, María J. Escalona, Manuel Mejías y Antonia M. Reina . MODELOS DE PRUEBAS PARA PRUEBAS DEL SISTEMA . Universidad de Sevilla . 2006.
5. Manual de usuario (ingles): (http://jakarta.apache.org/jmeter/usermanual/index.html.)
6. Corporación Sybven. Portal de Sybven. 2011. Disponible en: (http://www.corporacionsybven.com/portal/index.php?option=com_content&view=article&id=91:rational-performance-tester&catid=77:calidad-de-software-&Itemid=104).
7. JCrawler. A perfect Load. Sitio Oficial de JCrawler. Disponible en : (http://jcrawler.sourceforge.net/).
8. Solex. Web application testing with Eclipse. Sitio Oficial de Solex. Disponible en: (http://solex.sourceforge.net/).
9. FJRP, FMBR 2008/09 ccia SCS-Sistemas Cliente/Servidor . Marzo 2008.
10. Reynoso . Carlos Billy. Introducción a la Arquitectura de Software . Marzo de 2004 .
11. CBASQA. Proceso de Desarrollo Open UP. Septiembre 2008.
12. Sitio Oficial de Eclipse. Disponible en: (http://www.eclipse.org).
13. **J, CARLOS M. ZAPATA.** *CICLO-P: UN MÉTODO PARA EL ACOPLAMIENTO DE PRUEBAS AL CICLO DE VIDA DEL SOFTWARE.* 2010.
14. Carnegie Mellon University, "Capability Maturity Model® Integration (CMMI) Versión 1.1", U.S. Patent No.15213-3890, 2006.

15. JOSE GUILLERMO VALLE. UNIMINUTO . TECNOLOGÍA EN INFORMÁTICA . 2005.

16. Pablo Nicolás Baeza Baeza. Visual Paradigm DB Visual Architect SQL. D y T Systems.

17. Ignacio Esmite, Mauricio Farías,et al. *Automatización y Gestión de las Pruebas Funcionales usando Herramientas Open Source.* Centro de Ensayos de Software (CES), Universidad de la República Montevideo. Uruguay .

18. Ruth Alarcón, et al. *Infraestructura de pruebas para una plataforma de infraestructura de negocio: lecciones aprendidas de una experiencia académica.España,* Septiembre de 2008 Revista Española de Innovación, Calidad e Ingeniería de Software, Vol. 4. ISSN: 1885-4486.

19. Delmys Pozo, et al . *Procedimiento para pruebas de intrusión en aplicaciones Web.*España: Septiembre de 2009, Revista Española de Innovación, Calidad e Ingeniería de Software, Vol. 5. ISSN: 1885-4486.

20. Manual de Usuario de T-Arenal.

21. LARMAN, C. UML y Patrones. Introducción al análisis y diseño orientado a objetos. Mexico: 1999. ISBN 970-17-0261-1.

22. Nevedrov, Dmitri. Using JMeter to Performance Test Web Services. 2006. Disponible en: http://dev2dev.bea.com/pub/a/2006/08/jmeter-performance-testing.html

23. Roger.Pressman. Ingenieria.del.Software.6th.Ed.McGraw-Hill

24. Pedro Pablo Gómez Martín . Diseño de Sistemas Operativos Avanzados . Concurrencia en Java. 2010

25. Javier Luna Velásquez . Jfreechart . 5 de junio del 2007.

26. Javier García de Jalón. TECNUM. Aprenda Java como si estuviera en primero. 2000.

ANEXOS

Imágenes de la aplicación final.

Figura 12: Logueándose al servidor central t-arenal UCI

Figura 13: Adicionando prueba a turismo

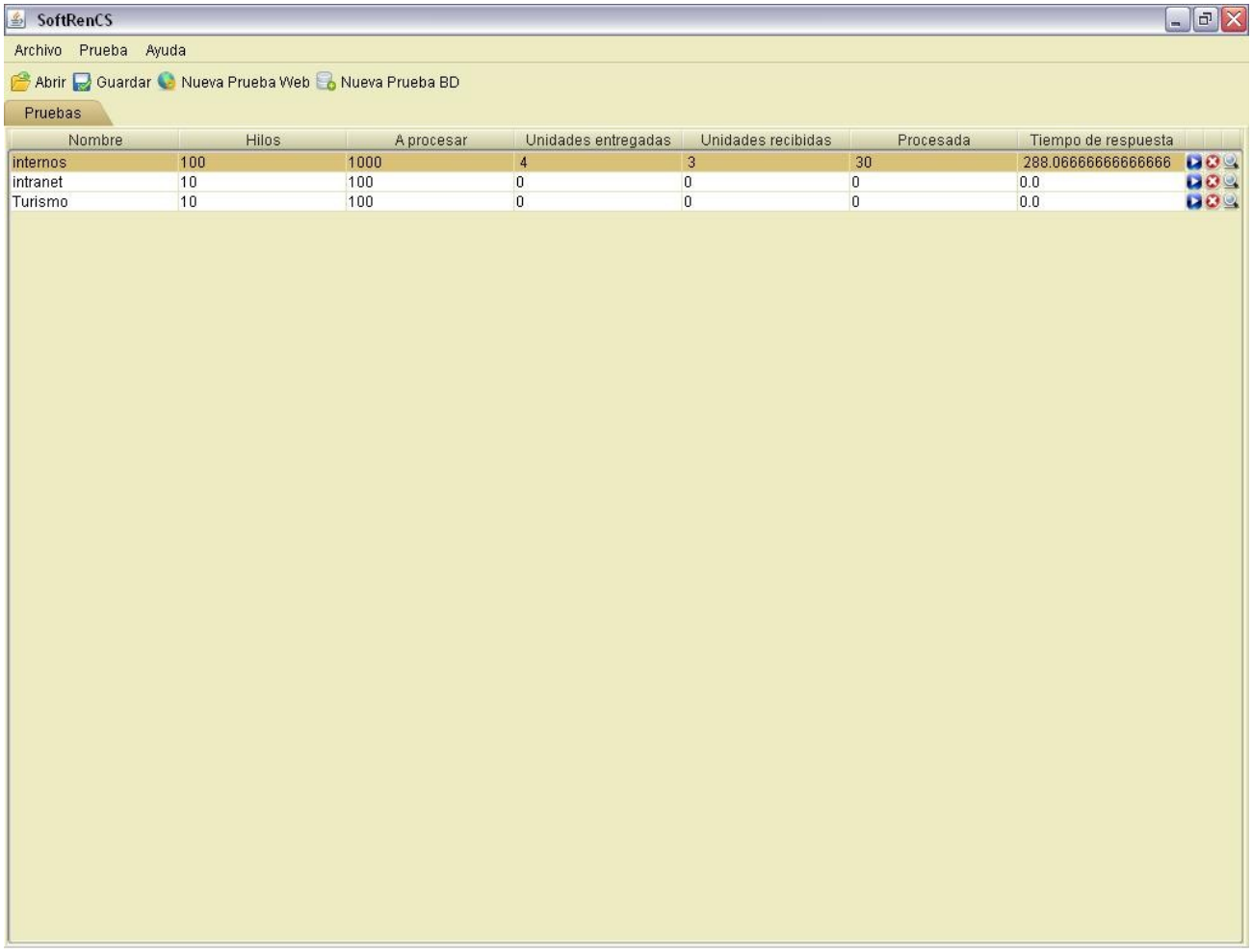


Figura 14: Pruebas ejecutándose