ANÁLISIS DE VIABILIDAD PARA LA CENTRALIZACIÓN DE ENTORNOS DISTRIBUIDOS

Memòria del Treball Final de Carrera
d'Enginyeria de Telecomunicació
realitzat per
Robert Requena Rubio
i dirigit per
Jordi Verdú Tirado
Bellaterra, 17 de Setembre de 2008

Universitat Autònoma de Barcelona

etse
Escola Tècnica Superior d'Enginyeria

El sotasignat, Jordi Verdú Tirado

Professor/a de l'Escola Tècnica Superior d'Enginyeria de la UAB,

CERTIFICA:

Que el treball a què correspon aquesta memòria ha estat realitzat sota la seva direcció per en Robert Requena Rubio

I per tal que consti firma la present.

Signat:

Bellaterra, 17 de Setembre de 2008

Índice de contenidos

Índice de figuras

Índice de tablas

1. INTRODUCCIÓN

La mayoría de las empresas tienen gran cantidad de aplicaciones que deben ser distribuidas entre una amplia gama de usuarios, desde empleados en funciones técnicas con dispositivos de acceso simples hasta los que hacen el trabajo intelectual, como los gerentes y ejecutivos con ordenadores muy potentes.

Existen tres tipos principales de aplicaciones a nivel de empresa, las cliente-servidor, las aplicaciones Web y las de escritorio. Los usuarios necesitan acceder continuamente a estas aplicaciones y a mucha información (correo electrónico, páginas Web, documentos...) para hacer su trabajo en una empresa.

Hay dos formas de afrontar este aumento de necesidades de información: la primera es sobre la marcha, usuario por usuario y aplicación por aplicación y la segunda es de modo sistemático. La primera manera ha terminado siempre con una compleja red de acceso a los datos de forma que es poco eficiente y difícil de manejar mientras que de la segunda manera todo el sistema queda definido de forma más clara y permite una escalabilidad y flexibilidad mucho mayores. Una plataforma de acceso para toda la organización que conecte todos los puntos donde se reclama información con todos los puntos donde se encuentra dicha información (centralización), que ofrezca la información bajo demanda mediante un enfoque orientado al servicio ha demostrado ser la mejor manera de distribuir la creciente información necesaria en una empresa.

Es importante, por lo tanto, comprobar que todo funcione correctamente, que el sistema permite y soporta el acceso a todos los usuarios y que las aplicaciones que hay no saturan la red ni el servidor de forma que reduzca la eficiencia del sistema.

Los tests, principalmente los de carga o performance, son los encargados de comprobar in situ que el sistema centralizado no fallará. El hecho de que el test tenga que hacerse en un servidor remoto permite una gran variedad de opciones y de maneras de afrontarlo dependiendo además del tipo de servidor y del tipo de aplicaciones a testear.

En este proyecto profundizaremos en cada uno de los diferentes conceptos mostrados en esta introducción trayéndolos al mundo real de una empresa mostrando que detrás de cada una de las definiciones teóricas podemos y debemos encontrar una funcionalidad real. Después, comparando las diferentes opciones que se nos presentan, valorándolas y eligiendo la mejor entre ellas llegaremos a una opción útil y eficaz que nos permita realizar un test de carga real a una aplicación real, pues este es el objetivo final de este proyecto.

La prueba se realizará sobre una aplicación de gestión bancaria que denominaremos Titicaca y hemos de comprobar cual es el máximo número de usuarios que pueden acceder simultáneamente a ella en cada uno de los servidores en que está instalada, teniendo como número ideal el de 30 usuarios concurrentes en cada servidor para una estimación de entre 5000 y 10000 usuarios finales repartidos por todos los servidores de que dispone el banco.

Hasta llegar a este test final el proceso que sufre el código es muy largo con muchas pruebas intermedias que permiten al desarrollador corregir errores y presentar un producto final que debe superar las exigencias de un test como el que se plantea en este proyecto. Veremos aquí, en la primera parte del mismo, todo este proceso de diferentes pruebas internas que sigue la aplicación antes de someterse a esta última prueba que dictaminará en qué medida está lista para ser utilizada por los usuarios finales.

Todas estas pruebas internas son siempre realizadas por parte de los desarrolladores y es por eso que las situaremos en el lado "cliente" mientras que explicaremos también todo el proceso desde el lado del "proveedor de servicios" que se encargará del test final y cuyo trabajo comienza una vez ha finalizado el del cliente. Así pues veremos y evaluaremos las distintas opciones que se le presentan al proveedor una vez le entregan la aplicación y cuál es el proceso desde que recibe la petición de realización del test hasta que lo culmina con la entrega de sus informes de evaluación.

2. TESTING

El testing es el proceso que permite verificar y determinar la calidad de un programa informático.

En este capítulo veremos qué es el testing en profundidad además de explicar cuales son los principales tipos de pruebas de software que se realizan en dicho proceso.

2.1 El ciclo de software

El desarrollo de cualquier programa informático o software requiere una serie de procesos o etapas. Este conjunto de etapas se denomina Ciclo de Software.

El testing es la quinta de estas fases del Ciclo de software. El proceso de testeo consiste en ejecutar el programa a comprobar y mediante diferentes técnicas descubrir los errores que pueda tener dicho programa.

El primer puesto de este ciclo corresponde al análisis de requisitos en que el ingeniero establece los requerimientos que el programa ha de tener. En nuestro caso se parte de un programa ya antiguo de gestión bancaria que, pese a que ha sufrido gran cantidad de actualizaciones de forma constante desde su creación se está quedando obsoleto, ya no sólo funcionalmente si no también estéticamente y en facilidad de manejo y comprensión. La nueva aplicación se encargará de realizar las mismas funciones que la antigua pero de una forma más eficiente a la vez que sencilla y clara para el usuario. Uno de los principales objetivos es el reducir de forma importante el tiempo de aprendizaje de la herramienta haciéndola más intuitiva y visual.

Después de este análisis preliminar la siguiente fase es la especificación donde el ingeniero se encarga de describir detalladamente el software a crear. La importancia de esta fase es relativa si hablamos del núcleo interno del programa y cobra la máxima relevancia a la hora de diseñar interfaces externos, pues estos han de ser, pase lo que pase por debajo, lo más estables posible. Estamos, con Titicaca, en una de esas ocasiones en que la importancia de esta fase para el núcleo interno es muy baja ya que el programa está perfectamente definido desde hace mucho tiempo. En cambio, como ya hemos comentado, será muy importante el interfaz de usuario pues es una de las partes que más cambios va a sufrir durante el proceso de creación del software.

Antes de empezar con la programación propiamente dicha se tiene que determinar el diseño y la arquitectura del software. En esta parte del ciclo se han de tener en cuenta elementos como el hardware o la red a la que estarán conectados los ordenadores donde vaya a correr el programa. Nuestra herramienta de gestión bancaria ha de estar disponible para todos los empleados del banco por lo que es importantísima esta fase del ciclo de

software. Titicaca será utilizado por entre 5000 y 10000 usuarios que se conectaran desde distintos sitios del planeta a un servidor que ha de ofrecerles el programa de forma rápida y eficiente. El cálculo del número de servidores dependerá pues del número de usuarios capaces de utilizar cada uno de ellos simultáneamente. En este caso, si cogemos el número ideal de 30 usuarios/servidor, tendríamos entre 166 y 334 servidores.

Después de estos pasos el ingeniero está listo para la programación en sí misma, cuya complejidad y duración depende en gran medida del lenguaje de programación escogido para realizar el código. En nuestro caso el lenguaje elegido es Java a través de la plataforma Eclipse de IBM.

Cuando el proceso de programación ha terminado es cuando llega el proceso de testing. El testing se encarga de que el proceso de especificación no haya sido en balde y permite comprobar que todas las tareas que indicaba dicha especificación se cumplen correctamente. Esta es la última fase que evaluaremos en este proyecto y la podemos dividir en dos fases: la de pruebas internas, que realizará el cliente de forma privada mientras desarrolla el código y que le permitirá corregir los errores más importantes. La segunda fase es la de pruebas externas en la que el proveedor se encarga de realizar una prueba final de forma ajena al cliente que permite evaluar su trabajo final y establecer, en este caso, el número de usuarios finales que podrán ser utilizados en cada servidor. Además es habitual descubrir algún error que haya pasado desapercibido al desarrollador debido a la naturaleza de las pruebas que realiza durante la programación.

Completan el ciclo de software la documentación del proyecto (manuales técnicos, diagramas, pruebas, manuales de usuario...) y el mantenimiento en que el ingeniero ha de corregir posibles errores y mejorar el software con sucesivas actualizaciones que recojan nuevos requisitos. Esta fase es la que mayor tiempo dura en el Ciclo de software siendo habitualmente 2/3 del tiempo total. En nuestro caso esta fase escapa de nuestros objetivos por lo que no entraremos a evaluarla y será, en todo caso, responsabilidad del cliente.

Es habitual encontrar empresas encargadas exclusivamente a esta parte del proceso y casi cualquier consultoría informática o empresa de ingeniería informática tienen un departamento de testing. Este último caso es el de nuestra empresa, de cuyo departamento de testing formé parte durante la realización de este proyecto.

2.2 Entorno de pruebas

Las pruebas de software necesitan un entorno particular aislado diferente al entorno de producción (donde el usuario final utilizará la aplicación). Este entorno de pruebas, aún siendo diferente al de producción ha de tener las mismas condiciones que éste para así recrear perfectamente la situación con la que ha trabajado el ingeniero y con la que se encontrará el usuario final. Los propios fabricantes de hardware se encargan habitualmente de proporcionar las herramientas necesarias para reproducir los entornos finales.

Es importante esta separación de entornos gemelos para que los errores que se detecten se deban exclusivamente al diseño del producto y no vengan causados por diferencias entre dichos entornos que escapen al control del ingeniero.

Para el test sobre Titicaca el banco nos facilita el acceso a una granja (conjunto de servidores) dedicada exclusivamente a la realización de tests. En uno de los servidores de la granja se ha instalado el software y reproducido el entorno final que encontrará el usuario. Para el acceso nos facilitan 30 usuarios virtuales con los que realizaremos el test y que se encargarán de simular el trabajo de 30 personas simultáneas sobre Titicaca.

2.3 Errores

Podemos distinguir dos tipos de errores en una aplicación, los de programación (*bugs*) y los defectos de forma. Los defectos de forma se dan cuando el programa no realiza una acción prevista por el usuario, por ejemplo cuando al pulsar un botón nos aparece una ventana que no es la que esperábamos. Los errores de programación, en cambio, tienen que ver con un fallo en la semántica del código.

Un bug, denominado así porque en 1944 una polilla fue la causante de un importante error en el ordenador Mark II al haber abierto uno de los muchos relés que formaban parte de él, muy típico es la aparición de *Memory Leaks* o pérdidas de memoria. Estas pérdidas de memoria se producen cuando una porción de memoria reservada para un proceso no queda liberada cuando dicho proceso termina de ejecutarse.

Los errores de programación no tienen porqué provocar un defecto de forma. Siguiendo con el ejemplo de las Memory Leaks, es posible que un programa funcione correctamente durante un tiempo aún con ese bug aunque si el tiempo de ejecución de dicho software es lo suficientemente alto el programa acaba con toda la memoria disponible y deja de funcionar. No es nada extraño que este tipo de errores se pasen por alto inicialmente (un test corto en tiempo probablemente no daría con el bug) por lo que es importante realizar tests como los que vamos a realizar para poder localizar este tipo de fallos.

No es habitual, en cambio, encontrar con un test como el que llevaremos a cabo, defectos de forma pues estos errores son fácilmente detectables por el propio desarrollador. Aún así, en caso de encontrarlos, suelen surgir durante la preparación de la prueba. Un desarrollador que conoce el programa que está realizando, sabe lo que ha de pasar y lo que ha de hacer para que pase, es por eso que las pruebas que puede realizar, aunque él no lo crea, están destinadas siempre a resultar satisfactorias. Por ejemplo, en un campo en que te piden el número de una tarjeta de crédito (o en la que siempre hubiera que poner un número de tarjeta de crédito), es posible que al programador nunca se le ocurra que alguien que desconozca la aplicación pueda introducir letras en ese campo. En este caso es muy importante que el probador externo realice el flujo normal que va a realizar el test de forma manual y así favorecer el descubrimiento de estos errores que para un programador que se conoce su software de arriba a abajo resultan, por obvios, ilocalizables.

2.4 Alpha y beta

A menudo las empresas, además de realizar un proceso de testeo interno, ya sea en un departamento de la propia empresa o contratando a una externa, optan por publicar una versión no definitiva que los usuarios finales se encargaran de probar durante un tiempo. Estas versiones de las aplicaciones reciben el nombre de beta.

Durante la fase de desarrollo es posible también realizar una versión básica que permite ir haciendo algunas pruebas iniciales. Esta versión básica es la versión denominada alpha.

Indudablemente, esta opción de las versiones beta, aunque es útil para detectar cierto tipo de errores, no puede ser, y de hecho no es, una fase clave en el proceso de testeo pues la importancia de dicha fase no permite dejar la responsabilidad de las pruebas en manos del usuario final.

Para el proyecto Titicaca el cliente se encarga de hacer llegar al proveedor una versión que cree definitiva, habiendo superado las fases alpha y beta (si es que se realizaron) antes de empezar nuestro test.

2.5 Detección

Hoy en día la detección de errores a tiempo a la hora de desarrollar un software es clave para el éxito de dicha aplicación. El aumento de la complejidad y, por lo tanto, de los costes que conlleva hoy en día el desarrollo de una aplicación obliga a que la detección de errores tenga que ser una tarea eficaz y rápida. Esto es debido principalmente a que el tiempo que se tarde en encontrar un fallo es directamente proporcional al dinero necesario para corregirlo. Conforme vayamos tardando más tiempo en encontrar el error más se irá encareciendo el proceso de corrección de dicho error.

Para detectar los errores lo antes posible se realizan diferentes pruebas durante todo el proceso que, aunque irremediablemente va después de la programación del código probado, se pueden intercalar en mitad del proceso mediante la creación de versiones intermedias del software final o el estudio de módulos de código individuales.

2.6 Tipos de pruebas

Existen muchos y diferentes tipos de pruebas de software de las cuales en este punto veremos una pequeña muestra en las que destacamos probablemente las más extendidas en el mundo del testing de empresa.

2.6.1 Prueba unitaria

Una prueba unitaria ha de formar parte de un proceso de prueba que incluya también al menos una prueba de integración pues es un paso previo para esta última.

La prueba unitaria permite analizar un módulo aislado del código implementado de forma que comprobará que dicho módulo funciona correctamente por separado. El hecho de que todos los módulos funcionen perfectamente de forma aislada no significa necesariamente que el software en conjunto también lo haga. De ahí la necesidad de realizar, una vez pasadas las pruebas unitarias, una prueba de integración que certifique el correcto funcionamiento de la aplicación.

Las pruebas unitarias suelen ser realizadas por los propios programadores encargados de desarrollar el código. Sirven como filtro inicial para detectar los primeros errores rápidamente así como para comprobar que los pequeños cambios que se puedan introducir en la implementación no afectan al módulo completo.

Para Titicaca todas estas pruebas están del lado cliente y para nosotros son transparentes. Hemos de suponer que se han pasado todos estos filtros durante el desarrollo del código antes de que nos llegue a nosotros el resultado final. En caso de que esto no hubiera sido así, aunque nuestro trabajo resultaría más importante si cabe, debería enfocarse de otra manera para poder abarcar todas las funcionalidades del software y no sólo una parte, que será lo que nosotros haremos puesto que nuestro objetivo no es comprobar que el software funcione sino comprobar cuantos usuarios concurrentes es capaz de soportar sin dejar de funcionar.

2.6.2 Pruebas de integración

Una prueba de integración consiste en verificar el funcionamiento de una aplicación una vez se han completado las diferentes pruebas unitarias. Para poder completar la prueba de integración los diferentes módulos testados se combinan y testan como un sólo módulo.

Estas pruebas, aunque es habitual que se encarguen también a consultores externos o, al menos, a testers profesionales, en nuestro caso correspondieron también al cliente de modo que no fue necesario realizar un esfuerzo mayor y más completo para el diseño del test.

2.6.3 Pruebas de sistema

Las pruebas de sistema nos permiten verificar que el software testado cumple con los requerimientos iniciales.

Para ello se suele utilizar el método de testeo conocido como de caja negra. El que realiza la prueba no conoce el contenido del sistema que debe testear, sólo conoce los requerimientos de dicho sistema. Así pues, con una determinada entrada la salida debe ser una concreta y solamente una. Con esta prueba, pues, no es posible detectar errores concretos por parte del tester y es en este caso el programador el que, interpretando el resultado de la prueba que le ofrece el tester, debe descubrir donde se produce el error.

Hemos de suponer que esta prueba la realizó el cliente puesto que tampoco entraba dentro de nuestros objetivos del proyecto Titicaca.

2.6.4 Pruebas funcionales

Una prueba funcional se encarga de verificar y comprobar las funcionalidades que se esperan del software. En una prueba funcional se esperan detectar principalmente defectos de forma aunque también es posible, por supuesto, detectar bugs.

Para estas pruebas es habitual que se requieran los servicios de alguien ajeno al programador puesto que el conocer de primera mano la aplicación perjudica a la hora de realizar las pruebas. Una persona que desconozca la aplicación puede llegar, por error o desconocimiento, a probar funcionalidades o casos de error que al programador por absurdos o ilógicos desde su punto de vista, nunca comprobaría.

Los departamentos de testing de las empresas suelen estar divididos en dos principales grupos: el de test funcional y el de performance. El departamento de test funcional se encarga, como ya hemos dicho, de comprobar que el software funcione perfectamente y responda de forma correcta a los eventos que se reproduzcan. Este test sí fue realizado por nuestra misma empresa aunque no es el objetivo de este proyecto entrar en el proceso ni analizar los resultados que este obtuvo. Cuando la petición de test llegó a nuestras manos el test funcional ya había sido superado de forma satisfactoria.

2.6.5 Pruebas de carga

Estas pruebas se encargan de comprobar la estabilidad de una aplicación o un sistema. Consiste en bombardear la aplicación (o el sistema) con un número determinado o no de peticiones simultáneas con la intención de sobrecargarla. El objetivo es comprobar:

Si el número de peticiones está predeterminado: que la aplicación o el sistema soportan la carga de peticiones (o usuarios) que los requerimientos exigen sin sobrepasar los límites de consumo de recursos (CPU, Memoria, tiempo de respuesta, eficiencia...) máximos.

Si el número de peticiones es indeterminado: cuál es el número máximo de peticiones que la aplicación o el sistema soportan antes de alcanzar los límites de consumo de recursos.

Este tipo de test es muy importante en los entornos distribuidos ya que permite conocer las limitaciones del sistema en cuanto a carga de usuarios se refiere y nos permite así mejorar el balanceo de carga. Es decir, permite decidir cuantos usuarios y cuando serán dirigidos a un servidor o a otro.

Titicaca es un proyecto en el que principalmente se realiza un test de carga sobre los servidores y sobre la aplicación. En un primer momento puede parecer que el número de peticiones está predeterminado puesto que se espera que cada servidor permita trabajar a 30 usuarios simultáneos. Aún así, como este número es más un objetivo "utópico" que una realidad, las pruebas que realizaremos serán con un número de peticiones indeterminadas. Es decir, empezaremos con un número bajo de usuarios simultáneos e iremos aumentándolo hasta averiguar cual es el tope del servidor y de la aplicación. Este tope lo marcará el consumo de recursos de Titicaca puesto que los recursos de que dispone el servidor son fijos son los de la aplicación los que hay que optimizar para conseguir el resultado esperado.

2.6.6 Pruebas de prestaciones

También llamado test de performance, combina el test de carga con el test funcional.

Así pues permite, a la vez, verificar el correcto funcionamiento de la aplicación a nivel de concepto (funcionalidades) así como a nivel técnico (consumo de recursos). Como el test de carga, es especialmente útil en entornos distribuidos. Es fácil confundirlo con una prueba de carga puesto que una parte importante del test de performance es estresar la aplicación aunque, como decimos, es igualmente importante comprobar que la aplicación funciona tal y como se esperaba.

En las pruebas de carga es normal que, para sobrecargar el sistema o la aplicación, se ataque determinada funcionalidad de dicho sistema. En este caso no queremos comprobar la funcionalidad si no sobrecargar el sistema con peticiones a través de dicha función. Así pues, aunque a priori, un test de carga pueda parecerse a un test de prestaciones, debemos tener clara la diferencia.

En el caso que estamos estudiando sólo probaremos un par de funcionalidades de la aplicación para comprobar como responde ante un ataque grande. Así pues, la funcionalidad que se prueba no es tan importante como el hecho de bombardear de peticiones al programa. Es por eso que el test a Titicaca es un test de carga y no uno de prestaciones.

3. CENTRALIZACIÓN

La centralización es una forma de organización en el entorno de la informática y las telecomunicaciones. Con la centralización de todos los servicios que requiere una red (software, impresoras...) se consigue minimizar gastos y el tiempo de reacción ante cualquier problema o cambio (p.e. actualización del software). Con esta centralización tenemos una plataforma de acceso que abarca toda la organización y conecta todos los puntos desde donde se reclama información con todos los puntos donde se encuentra dicha información ofreciéndola bajo demanda. Esto es, sólo dando la información a quien la necesite cuando la necesite. Con esto evitamos tanto la sobrecarga de la red al minimizar el flujo de datos que circulan por ella como que los clientes tengan demasiada información en su PC.

Existen dos modelos de centralización: en el primero, el servidor remoto provee un escritorio de Windows o de Linux a varias terminales de usuario. A estos usuarios se les llama *thin client*. En el otro modelo un ordenador normal actúa temporalmente como servidor remoto ofreciendo su escritorio al resto de ordenadores conectados a una WAN (Wide Area Network) que normalmente es Internet para permitir el teletrabajo. El software utilizado tanto en el teletrabajo como por los thin clients es conocido como Aplicaciones de Escritorio Remoto (*Remote Desktop Applications*).

En el banco nos encontramos básicamente con usuarios thin client. Y es que todo el trabajo se realiza en los servidores dejando a los usuarios con lo justo. En nuestro caso particular, como empresa externa que trabaja para el banco, disponemos de acceso al servidor vía Escritorio Remoto pues necesitamos, como testers en mi caso particular o desarrolladores en otras secciones de la empresa, mucha más potencia y prestaciones de las que puede ofrecer un thin client.

En este capítulo explicaremos porqué la centralización se ha impuesto en el modelo actual de red en que los Escritorios Remotos son la base así como la organización de las redes centralizadas.

3.1 Sistemas distribuidos

Los sistemas distribuidos son sistemas cuyos componentes hardware y software, que están en ordenadores conectados en red, se comunican entre ellos y son capaces de coordinar sus acciones, para el logro de un objetivo común. La comunicación entre los componentes se establece mediante un protocolo prefijado por un esquema cliente-servidor.

Los sistemas distribuidos han de tener tres características imprescindibles: concurrencia, carencia de reloj global e independencia entre los componentes.

La concurrencia consiste en que todos los recursos disponibles de la red puedan utilizarse simultáneamente por los usuarios sin que esto suponga un conflicto. El hecho de que yo esté utilizando cierto componente software (p.e. el Word) no ha de impedir que otro usuario que lo solicite después de mí pueda utilizarlo. En los componentes software (p.e. una impresora) lógicamente se generarán colas pero éstas no impedirán que el usuario trabaje normalmente (las colas son transparentes para el usuario).

La siguiente característica, la carencia de un reloj global, obliga a que los diferentes componentes funcionen asíncronamente. Es decir, se comunican entre ellos cuando es necesario e independientemente de cuando lo hagan el resto de componentes. Con esto distribuimos la carga de la red, el intercambio de datos, de forma constante a lo largo del día y evitamos sobrecargas del sistema. De todas maneras, es inevitable que a ciertas horas del día (a primera hora de la mañana y a primera hora de la tarde) la cantidad de información que circula por la red del sistema es mucho mayor que durante el resto de la jornada. Es por esto que los momentos iniciales del test de carga que realizaremos, cuando todos los usuarios se conectan en una franja de tiempo muy pequeña, son muy importantes y darán una buena medida de lo que es capaz el servidor y de lo bien o mal implementado que está Titicaca.

Por último, como ya hemos dicho, también es necesario que dichos componentes sean independientes entre ellos. El hecho de que un elemento de la red falle no puede afectar al resto, que han de poder seguir trabajando con los elementos a los que no haya afectado dicho fallo.

3.2 Esquema cliente-servidor

Para este esquema necesitamos al menos dos máquinas distintas. Una que hará las tareas de servidor y la otra de cliente. Habitualmente de un servidor dependen más de un cliente como podemos ver en la figura 3.1.

Un servidor se encarga de proporcionar servicios al cliente que se lo demande. Estos servicios pueden ser o la ejecución de un programa, el acceso a una base de datos o el acceso a un dispositivo hardware, como puede ser una impresora. El cliente no tiene por sí mismo la capacidad de utilizar estos servicios y requiere la presencia del servidor para poder obtenerlos.

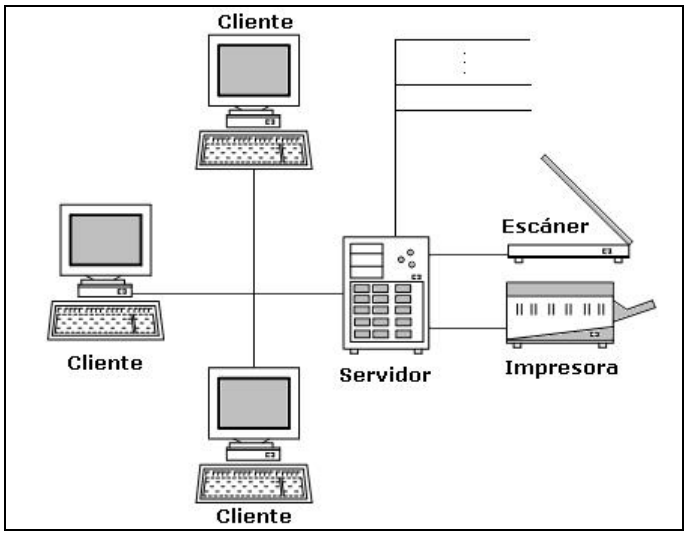


Fig. 3.1. Esquema habitual cliente-servidor

Un cliente, como vemos en la figura, puede ser un PC que tiene funcionalidad independiente del servidor, con sus propios recursos (disco duro, memoria...) y que, en un momento dado requiere los servicios del servidor, ya sea para imprimir un documento o para acceder a un software determinado. Otra opción para trabajar con un servidor remoto son los thin clients, mucho menos independientes que los PCs.

El test lo realizaremos desde un PC que gestionará la conexión de los usuarios virtuales (teóricamente hasta 30) que realizarán el test sobre el servidor simulando cada uno de ellos la actuación de un thin client trabajando sobre el servidor durante un periodo de tiempo prolongado. Todos ellos demandarán al servidor el acceso a la aplicación Titicaca y trabajarán simultáneamente sobre ella.

3.3 Thin Client

Un thin client o "cliente liviano" es una máquina que funciona como cliente en un esquema cliente-servidor de forma totalmente dependiente. El servidor es el encargado de toda la lógica de programación y, por lo tanto, de todas las tareas de procesamiento. Un thin client sólo necesita poder conectarse a la red para poder trabajar remotamente en el servidor.

Aparecen como contraprestación a los ordenadores habituales (ya sean PC's u ordenadores portátiles) ya que permiten ahorrar tanto en hardware como en software.

Los thin clients permiten mantener toda la información centralizada, reducen el coste de hardware (no tienen disco duro ni un procesador demasiado potente) y consumen muy poca energía. Su tamaño también suele ser reducido (no mucho mayor que un módem o un *router*) permitiendo así ahorrar todo el espacio que necesita una torre de PC y no son capaces de almacenar datos mejorando así la seguridad del sistema.

Para un banco parece pues, la mejor forma de trabajar. Todos los empleados de las sucursales disponen de un thin client desde el que realizar sus tareas diarias sin problemas y no pueden almacenar y/o robar información puesto que no disponen de memoria propia. Esta forma de trabajar es mucho más segura pero exige un control muy severo para que el sistema funcione perfectamente y es que es impensable que la red de todo un banco se caiga de forma accidental sin tener controlada una alternativa. Volvemos a ver, con nuestro ejemplo, lo importante que es realizar pruebas como las que se nos propuso y que estamos tratando aquí.

3.4 Escritorios Remotos

Los escritorios remotos permiten la centralización de las aplicaciones habituales para los usuarios (navegadores, procesadores de texto...). Así, no es necesario que el usuario disponga en su máquina local del software instalado.

Los thin clients son así simples terminales de entrada/salida mientras que los PC suelen tener instaladas las aplicaciones más comunes y utilizan los escritorios remotos para aplicaciones que, ya sea por relevancia, temas de seguridad o por coste de licencias, la empresa prefiere mantener en un servidor aparte.

Para el acceso a los escritorios remotos existen varios programas diferentes que utilizan, cada uno de ellos, un protocolo de comunicaciones propio permitiendo así la elección entre diversas opciones que, como veremos más adelante, serán un elemento importante en la elección de alternativas a la hora de realizar las pruebas de *performance* sobre un servidor.

3.5 Ventajas e inconvenientes de la centralización de recursos

Podemos decir sin riesgo a equivocarnos que las ventajas de la centralización superan ampliamente a las desventajas. A continuación vamos a ver cuales son estas ventajas que nos ofrece y cuales son los inconvenientes que, por supuesto, se nos presentan también.

3.5.1 Ventajas

Uno de los motivos más importantes por los que se impone el modelo de centralización de los recursos es el ahorro económico que supone. Es mucho más barato añadir un nuevo cliente que conectar al servidor (sobretodo si se trata de un thin client) que no añadir un nuevo puesto de trabajo independiente de la red montada. Asimismo, a la hora de actualizar software o hardware es mucho más barato hacerlo en un sólo servidor que no tener que ir puesto por puesto, PC por PC, actualizándolo, tanto en dinero, en recursos humanos como en tiempo.

La escalabilidad de la red también es muy grande, es fácil aumentarla añadiendo nuevos clientes o, en caso necesario, aumentar el número de servidores centrales. Tener la información en uno (o en unos pocos servidores) permite que la duplicación o el aumento del hardware sean mucho más rápidos, fiables y baratos.

La capacidad de compartir recursos entre diversos usuarios (impresoras, escáneres, software, dispositivos de almacenamiento...) es otra de las grandes ventajas de estos sistemas así como el hecho de poder distribuir la carga de trabajo entre diferentes ordenadores. Puesto que todos tienen acceso al mismo servidor, la carga para el hardware de cada uno de ellos es menor mejorando así la eficiencia de trabajo y el tiempo empleado en completar una misma tarea en el servidor entre varias personas a la vez es también mucho menor que si el mismo trabajo lo realiza una sola persona en local, además del riesgo de perder los datos que esto supone.

3.5.2 Inconvenientes

Un problema que se presenta habitualmente en estos entornos de trabajo es el de decidir cuantas funcionalidades, cuantos servicios o cuanta información hemos de dejar en el servidor y cuanta ha de estar en el lado del usuario. Es responsabilidad de los administradores de la red decidirlo antes de ponerla en marcha y es por tanto un problema de diseño más que un problema del sistema en sí, una vez solucionado este apartado antes de construir la red, este inconveniente no se nos volverá a presentar salvo que se lleve a cabo una reestructuración de los permisos y funcionalidades para los usuarios.

Es posible que, si no se ha dimensionado bien la red, se puedan producir problemas con la comunicación cliente-servidor, llegando a sufrir pérdidas de mensajes, saturación del tráfico... Aunque también es un problema de diseño y arquitectura previo a la red en sí es importante la realización de tests de carga o de performance antes de dar por buena una nueva actualización de un servidor o de una red puesto que, debido a su alta escalabilidad, el redimensionamiento de la red se realiza relativamente a menudo.

El principal problema que se presenta es el de la seguridad. El hecho de compartir datos entre los usuarios la condiciona mucho. Aún así, la multitud de sistemas de seguridad, encriptaciones y accesos restringidos permiten, a estas alturas, tener unas redes bastante seguras.

4. PLATAFORMAS DE ACCESO A SERVIDORES REMOTOS

Existen diversas aplicaciones que permiten el acceso desde la máquina local, ya sea un PC o un thin client, a los servidores remotos y el control de este acceso desde el servidor. Estas aplicaciones permiten el acceso a diferentes aplicaciones y al escritorio de una máquina remota (como cliente) así como el manejo de las conexiones remotas (como servidor). Prácticamente cada una de estas aplicaciones utiliza su propio protocolo de conexión.

En este capítulo veremos las opciones más comunes que podemos encontrar en cuanto a plataformas de acceso a servidores remotos se refiere. No entraremos en demasiadas características técnicas y será más bien una descripción superficial de las herramientas. También veremos una evaluación de las diferentes alternativas presentadas.

4.1 Terminal Service. Microsoft

Terminal Services es la plataforma de acceso a escritorios remotos que *Microsoft* integra en *Windows*. *Terminal Services* permite tanto la opción cliente como la opción servidor.

Terminal Services ofrece, en su opción servidor, el componente *Terminal Server*. Este componente se encarga de manejar la autenticación de clientes, de poner a disposición de éstos las aplicaciones de forma remota así como de aplicar las restricciones necesarias dependiendo de los privilegios del grupo de usuarios al que pertenezca el cliente.

En su opción cliente, *Terminal Services* incluye dos aplicaciones para Windows: *Remote Desktop* y *Remote Assistance*. Como viene integrada en Windows prácticamente todo el mundo dispone de ella.

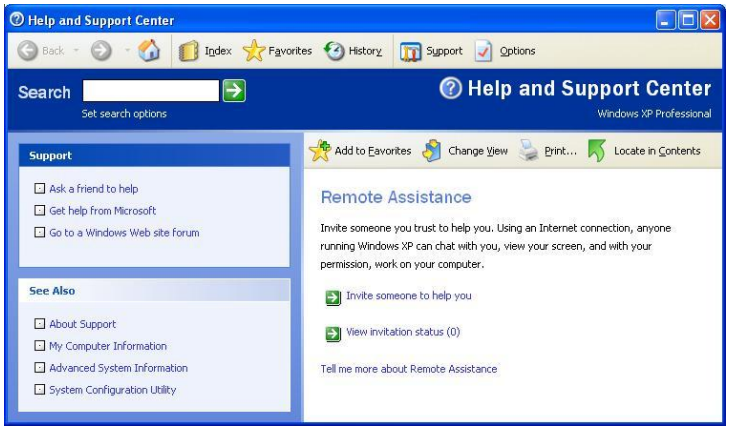


Fig. 4.1. Remote Assistance

Remote Assistance permite recibir asistencia técnica a través de la conexión remota sin necesidad de que el técnico se desplace físicamente adonde esté el ordenador. A través de esta aplicación el servicio técnico adquiere temporalmente el control sobre el escritorio del cliente e intenta resolver los problemas surgidos.

Fig. 4.2. Remote Desktop

Remote Desktop permite al cliente utilizar una máquina en red de forma remota. Para acceder a la máquina remota son necesarios su usuario y su contraseña. La máquina a la que accedemos queda completamente redirigida al cliente: sonido, recursos hardware y software, dispositivos externos como cámaras Web pueden ser utilizados por el cliente como si estuvieran en su ordenador. La apariencia de dicha máquina se muestra al cliente con la misma imagen que aparece en local por lo que el cliente observa e interactúa de forma remota de la misma manera que lo haría si lo hiciese en local.

4.2 Virtual Network Computing.

Virtual Network Computing (VNC) es una plataforma de acceso a escritorios remotos realizada bajo el concepto de software.

VNC funciona gráficamente, trabajando sobre la pantalla del servidor en el cliente de forma que el cliente se mueve sobre píxeles y no sobre botones o ventanas. Todos los recursos del servidor son redirigidos al cliente que puede hacer uso de ellos remotamente. El servidor tiene la opción de funcionar como servidor HTTP permitiendo así que el cliente, sin necesidad de haber descargado previamente el VNC, pueda ver la pantalla compartida. En este caso, cuando el navegador intenta acceder al servidor descarga automáticamente el cliente VNC. El sistema operativo en el que trabajen cliente y servidor no tiene por qué ser el mismo.

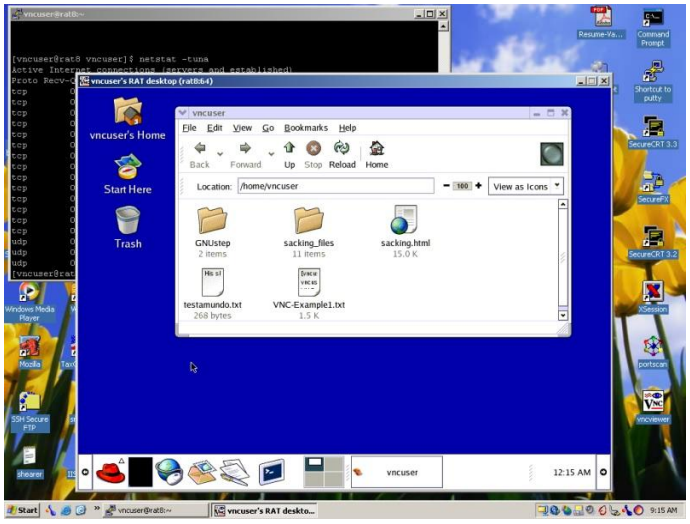


Fig. 4.3. Vista de una conexión VNC desde Windows XP.

4.3 MetaFrame. Citrix

MetaFrame es la plataforma de acceso a escritorios remotos que ofrece la compañía *Citrix*. Parte del código fuente de esta empresa lo utiliza Microsoft para su Terminal Server mientras que otra parte ha permitido lanzar la plataforma *MetaFrame* (en otras versiones de *Windows* se ha llamado *Presentation Server* y *XenApp*). *MetaFrame* utiliza el protocolo ICA (*Independent Computing Architecture*) para establecer las conexiones remotas y funciona siempre sobre una máquina con *Windows* como sistema operativo.

MetaFrame realiza una conexión de alto nivel con la que el cliente trabaja exactamente igual que en local. La conexión se puede realizar desde cualquier máquina al servidor *Citrix* y requiere autenticación propia. Para la conexión el usuario necesita un cliente ICA que contiene la información sobre la conexión (derechos del usuario, aplicaciones que tiene disponibles...)

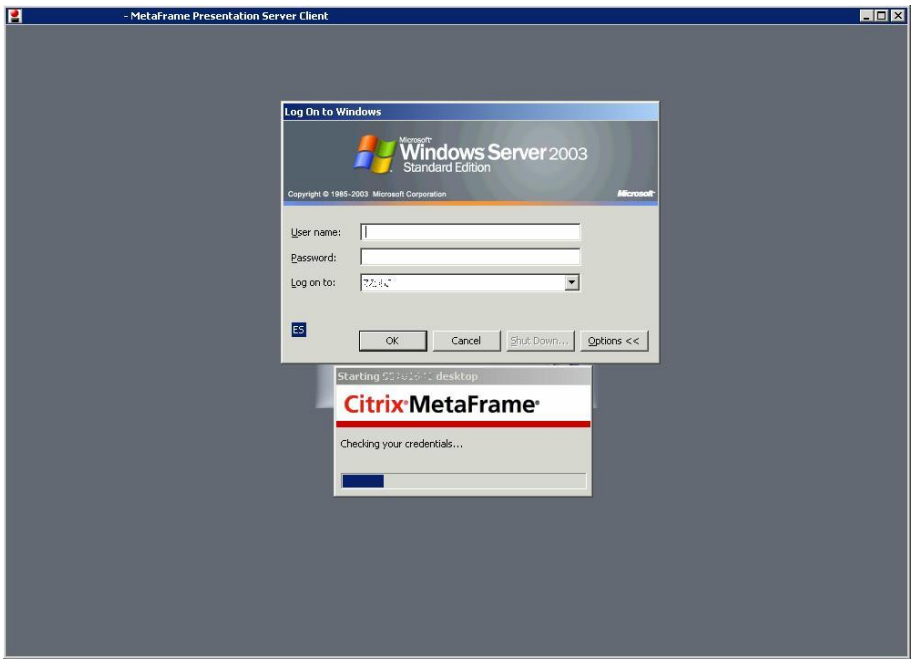


Fig. 4.4. Pantalla de autenticación a un servidor a través de una conexión MetaFrame

4.4 X-Window. Unix

X-Window es una plataforma de acceso a escritorios remotos para los sistemas *Unix*. Para ello utiliza el protocolo *XProtocol*.

En un principio, *X-Window* era sólo un sistema de ventanas que funcionaba como interfaz gráfica para *Unix*. A partir de ahí, aparecen diferentes componentes que se encargan de que esta interfaz funcione en red permitiendo el acceso remoto desde diferentes clientes al escritorio de un servidor. Actualmente se utiliza la versión X11 de este componente.

Como el resto de aplicaciones de software libre, *X-Window* es independiente del sistema operativo. En este caso, funciona también de forma gráfica aunque, a diferencia de VNC, requiere de un gestor de ventanas independiente para ser operativo. Sin este componente la conexión remota no funciona. Es decir, el conjunto X11 + gestor de ventanas realiza una conexión de alto nivel.

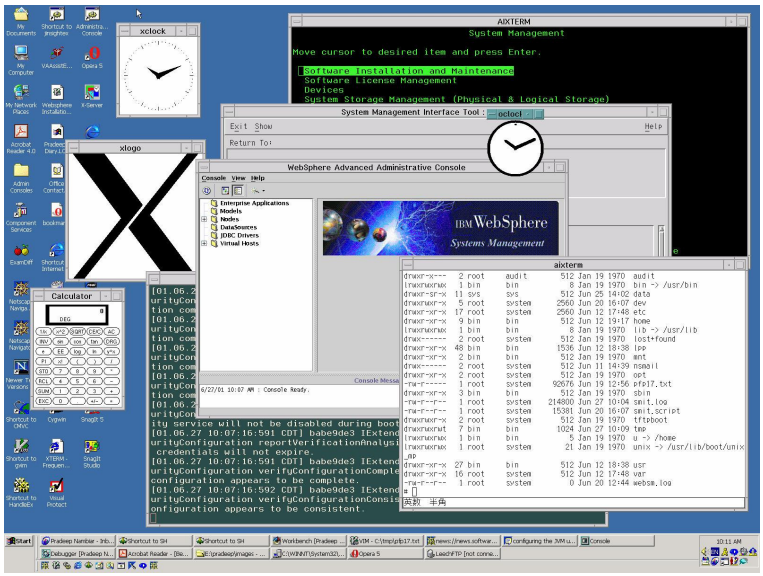


Fig. 4.5. Sesión X-Window sobre plataforma Windows

4.5 Secure Global Desktop. Sun Microsystems

Secure Global Desktop, aunque fue diseñado y creado por la compañía *Tarantella*, forma parte actualmente de *Sun Microsystems*.

La mayoría de sistemas operativos permiten utilizar SGD con la única condición de disponer un navegador (*Internet Explorer*, *Mozilla Firefox* o *Safari*) con el componente *Java Runtime Environment* instalado. La primera vez que el cliente se conecta al servidor a través del navegador, el componente *SGD client* se descarga y se instala. Utiliza el protocolo AIP (*Adaptative Internet Protocol*).

El hecho de que la conexión sea vía navegador permite que las sesiones se mantengan y puedan cerrarse en un ordenador (o en un thin client) y abrirse en otra manteniendo el último estado de la conexión.

Para el control de la conexión desde el servidor presenta el *Object Manager* que puede actualizar las aplicaciones que el usuario puede ver sin necesidad de que dicho cliente cierre su sesión.

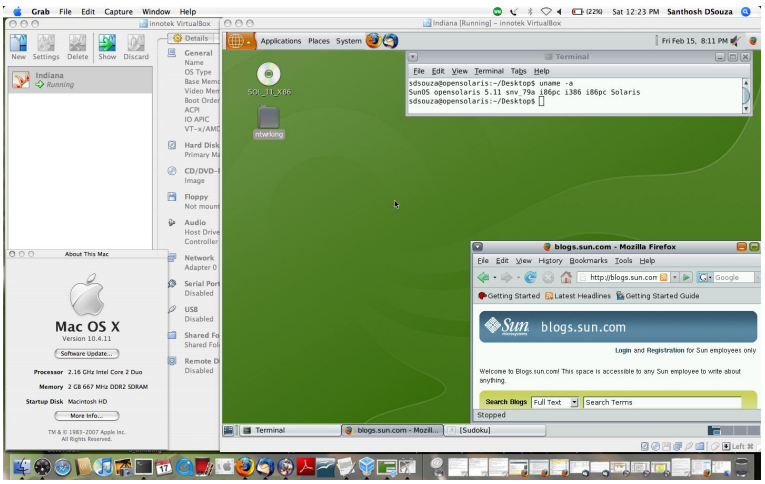


Fig. 4.6. Vista de una sesión SGD sobre Apple

4.6 Evaluación de alternativas

Dependiendo de las características de los equipos de que dispongamos y de la capacidad económica que tengamos deberemos elegir una opción u otra.

De las opciones que hemos visto VNC es la única que, siendo de libre acceso, nos permite trabajar en *Windows*. Sus limitaciones a la hora de trabajar debido a que sus conexiones son gráficas impiden un gran desarrollo de las conexiones y un aprovechamiento total de todas las opciones que una conexión de este tipo nos permite teóricamente. Es por eso que para cuestiones docentes puede ser una buena herramienta. Permite las conexiones remotas entre los alumnos y el profesor y sirve para unos propósitos no demasiados ambiciosos como pueden ser los de una clase en la universidad o en el instituto.

En caso de disponer de tecnología *Unix* podría también utilizarse la plataforma *X-Window*. En todo caso parece más apropiada para instituciones públicas más que para empresas privadas ya que el simple hecho de su filosofía de software libre no permite llegar, por falta de medios, a los niveles que un desarrollo de una empresa privada y por lo tanto, al nivel de prestaciones que puede llegar a exigirse. En todo caso es una buena alternativa, la mejor, si el sistema operativo utilizado es *Unix* y no se tiene un presupuesto suficiente. Es por eso que parece más recomendable para instituciones públicas como ayuntamientos o gobiernos autonómicos en que los requerimientos del software, en teoría, no han de ser demasiados.

Los servicios integrados de *Microsoft*, precisamente por eso, por ser integrados no presentan muchas de las prestaciones que otros programas. De hecho, es a partir de estas limitaciones desde donde se desarrollan las otras plataformas, aprovechando la base que ofrece Windows para ofrecer un mejor servicio. Aunque es cierto que las opciones que presenta, tanto desde el lado servidor como el lado cliente son bastante buenas y permiten realizar conexiones a escritorios remotos y gestionarlas de forma bastante eficiente, las limitaciones técnicas que presenta y las desventajas que conlleva la convierten en una herramienta circunstancial y como complemento de alguna más compleja.

Así pues nos quedan las dos propuestas comerciales que hemos presentado: *Citrix* y *SGD*.

SGD es el producto que ofrece la gigantesca *Sun*, la gran competidora de *Microsoft*. Aunque su plataforma es absolutamente compatible con *Windows* utiliza otro de sus propios productos, *Java Runtime Environment*, para hacerla funcionar. Esto presenta algunas ventajas, como el hecho de que sea independiente del sistema operativo puesto que, como ya hemos dicho, funciona con casi cualquier navegador mientras tengamos instalado el JRE manteniendo además la sesión de usuario.

MetaFrame, en cambio, basa toda su tecnología en *Windows* y, aunque es de una empresa diferente, el acuerdo que tienen *Citrix* y *Microsoft* les permite intentar conseguir el mayor rendimiento que se puede ofrecer de las bases que ofrece *Windows* (*Terminal Services*).

La gran diferencia entre SGD y *MetaFrame* es la necesidad del primero de un navegador para funcionar mientras que la plataforma de *Citrix* nos permite la conexión a un escritorio remoto sin necesidad de navegadores. Esta ventaja de *MetaFrame* respecto a SGD parece más importante que la ventaja del producto *Sun* de mantener la sesión y poder retomarla en otro punto de la red puesto que en las empresas habitualmente las conexiones de un usuario se realizan siempre desde el mismo PC o thin client.

Es por todas estas razones expuestas que la mejor elección para la mayoría de las empresas (Sistema Operativo *Windows* y puestos de trabajo fijos) considero que es *MetaFrame* de *Citrix* o cualquiera de sus versiones que varían con la versión de Windows instalada.

En nuestro caso, utilizaremos esta conexión de *Citrix* para acceder al servidor donde realizaremos el test. Disponemos como sistema operativo *Windows XP* mientras que en los servidores encontramos *Windows Server 2003*. Partiendo de esta base, la opción *MetaFrame* es la más natural y también la mejor para nuestros intereses. En todo caso, es el banco el que gestiona los servidores así pues, ellos son los que eligen la plataforma de acceso.

5. EJEMPLO DE UNA PRUEBA DE CARGA. “TITICACA ”

Un test de prestaciones permite verificar el correcto funcionamiento de una aplicación o un conjunto de ellas tanto a nivel de concepto (funcionalidades) como técnico (recursos).

Este tipo de pruebas son especialmente útiles en entornos distribuidos puesto que los servidores remotos van a sufrir un estrés (mayor o menor dependiendo de su objetivo) debido a los diferentes usuarios que pueden estar funcionando a la vez y ha de responder a todos por igual.

En este capítulo veremos todo el proceso necesario para la realización de un test de prestaciones de una forma práctica, con un caso real como ejemplo.

5.1 Preparación

La preparación de un test se divide en tres fases, una primera fase en que el cliente pide que se comprueben las prestaciones de algún producto suyo, una segunda fase en que el encargado de realizar del test analiza la petición y pone las condiciones que considera necesarias y una última parte en que, una vez de acuerdo, se plasma todo lo acordado en las fases anteriores en una propuesta que el cliente debe evaluar y aceptar si lo cree conveniente.

5.1.1 Requerimientos

Esta es la primera fase de un test, en ella el cliente contacta con el proveedor de servicios para comprobar la viabilidad de realizar las pruebas que han pensado para su producto.

A la hora de definir los requerimientos es importante también la opinión del tester puesto que la mayoría de las veces, el cliente tiene una idea vaga de lo que quiere hacer y es el encargado de realizar las pruebas el que tiene que dar forma a lo que se busca para hacerlo realizable según los medios que posea.

En el ejemplo que nos ocupa, el del software que llamaremos “Titicaca ”, no fue necesario ese intercambio de impresiones entre cliente y proveedor puesto que ambos llevan trabajando mucho tiempo juntos y tienen bastante claro, unos lo que pueden exigir y los otros lo que pueden ofrecer.

A continuación podemos ver una tabla con los requerimientos que pone la empresa cliente para realizar el test:

ID	DESCRIPTION
R01	Based on an GUI and Browser based GUI, perform a load test only the following FrontEnd: • Titicaca GUI (rich client GUI)
R02	Test scenarios are : *Scenario I*: Concurrent users performing isolated usecase • 20 concurrent users doing the same usecase, closing the frontend after each usecase • Monitoring of o CPU load o Memory consumption o Response time *Scenario II*: Concurrent users using the frontend over a period of time • 20 concurrent users performing a set of defined usecases over a defined amount of time without closing the frontend application after each usecase • Monitoring of o CPU load over time o Memory consumption over time o Response time development
R03	Usecase Description for Scenario I • Start GUI • Log in • Open work queue • Open work item and associated documents (TIFFS) • Open document functionality only available in the eclipse FrontEnd • Close work item • Log off • Close GUI Main objective of this usecase is to check possible application memory leaks
R04	Usecase Description for Scenario II • Start GUI • Log in • Open work queue • Open and close multiple windows o Functionality available by links (IProcessClient Browser) or Buttons, double-clicks and menu items (eclipse based FrontEnd) Main objective of this usecase is to check the stability of the application.
R05	Generation of a document reporting the test results (CPU load, memory consumption, response time, etc.).

Tabla 5.1. Requerimientos para el test de la aplicación Titicaca

5.1.2 Asunciones

Como respuesta a los requerimientos del cliente, el proveedor del servicio debe establecer una serie de puntos que se consideran necesarios e imprescindibles para la realización en perfectas condiciones del test.

Estos puntos son asumidos por el cliente y se compromete a cumplirlos como parte del contrato que se firmará. En caso de que no sea así el proveedor no garantiza que el test pueda realizarse correctamente. Es habitual que estas asunciones por parte del cliente no se cumplan al completo durante todo el tiempo que dura el test pero en ese caso es imprescindible que tenga las medios necesarios para remediar el problema en el menor tiempo posible sin perjuicio para la empresa que se encarga del test.

Podemos ver debajo de estas líneas la tabla en que quedaron recogidas las asunciones para el proyecto de este ejemplo:

ID	DESCRIPTION
AS01	Test environment infrastructure is available (servers, etc.).
AS02	20 test users with access to the test environment.
AS03	Test users available and with proper rights to access the FrontEnd of Titicaca according to described test scenarios.
AS04	Test environment based on a W2K3 Server platform.
AS05	Detailed testcases will be available at the beginning of the implementation.
AS06	No changes on the FrontEnd of Titicaca during the load test cycle (neither versions updates nor configuration changes).
AS07	Chosen testing tool is available and able to be installed on a local development computer.
AS08	The sub-application MDS has no special technical feature that leads to a higher effort to automate the corresponding testcases.
AS09	The Company is the requester for installations and assumes product license costs, but since there is not a special owner for the proposed testing tool, it seems that the license costs will be charged to the project or department who use it.

Tabla 5.2. Asunciones para el test de la aplicación Titicaca

5.1.3 Propuesta

Teniendo en cuenta los requerimientos y las asunciones el proveedor debe realizar una propuesta que satisfaga al cliente presentándole, si es posible, más de una opción entre las que el cliente pueda elegir la que le mejor le convenga a sus intereses, indicando en cada caso las ventajas y desventajas de cada una de las propuestas así como el impacto en el mantenimiento en caso de que la opción ofertada lo requiera.

Una vez presentada la propuesta el trabajo previo por parte del proveedor termina y sólo le queda esperar la decisión final del cliente, tanto elegir si el proveedor es el elegido de entre todos los que se hayan presentado a concurso como elegir cual es la opción de las presentadas por dicho proveedor, si es que se han presentado varias, que más se ajusta a sus intereses (económicos y técnicos).

En el ejemplo que estamos siguiendo encontramos dos diferentes opciones:

Opción 1:

- Ventajas:

ID	DESCRIPTION
AV01	Cost reduction by reusing license of testing tool what is now in place.
AV02	Low effort for developing, since the current scripts can be used without less additional effort. Only a new scenario to include the new scripts for Titicaca must be created.
AV03	Short timeline to begin development and the load test execution.

Tabla 5.3. Ventajas de la Opción 1 de la propuesta

- Desventajas:

ID	DESCRIPTION
DAV01	Manual start of test user sessions and also their scripts makes not possible to measure detailed resource consumption, only a general average.
DAV02	It is not possible to get the real responses times for any application or test user, it is only possible to get measures of general CPU load and memory consumption.
DAV03	No information available about causes of test execution errors (environment/server, test tool, scripts, performance, etc.). Wrong values on results cannot be isolated.
DAV04	No information available about specific test users activity and in consequence about a specific test scenario.
DAV05	This is only a temporary solution; it is a workaround while a corporate testing tool is defined. This means, efforts invested in this solution will give a unique test result and cannot be amortized.
DAV06	It is not possible to generate automatic project documentation with a complete load test analysis.

Tabla 5.4. Desventajas de la Opción 1 de la propuesta

- Propuesta:

The overall effort estimation for all scenarios (I+II+III) is **27 MDs**, including analysis, set-up, development, test and delivery
Following a detailed description of the tasks covered:

	effort
TOTAL	***27,00***
Proof of Concept	*1,00*
Application Analysis	*3,00*
In this phase, all applications involved in the Load Test will be analyzed from a performance point of view. All information sent by each step performed by the user will be examined in order to detect which fields need to be parameterized or correlated	
Scripts Development	*5,00*
During the development phase, all business flows will be recorded using Visual Test. Scripts will be parameterized and correlated.	
Scenarios Creation	*3,00*
Once the business flows are developed, scenarios need to be built with scripts. Within the scenarios, duration, number of users and server monitoring will be configured.	
Project documentation	0,00
No automatic project documentation can be generated.	
Execution	9,00
All scenarios previously created will be performed. During the execution phase, a daily execution report will be delivered with a summary of all executions performed and general results	
Results Analysis	2,00
After the execution, scripts must be developed to gather values from the perfmon logs. An analysis is also necessary to extract the meaningful information.	
Report	1,00
Current management summary will be deliver adding information about Titicaca	
Project Management	3,00
project tracking and reporting tasks	

Tabla 5.5. Descripción de la propuesta de la Opción 1.

Opción 2:

- Ventajas:

ID	DESCRIPTION
AV01	Highly scalable scenarios. If a load increase is required, from simulating 20 to 50 users, no additional effort is required.
AV02	All performance results at one side. LoadRunner lets collect all information from (application response times and server monitoring) and create graphs automatically. Furthermore, following information can be gathered: • Running Virtual Users • Error statistics (providing the type of error and the line number in the script) • Transactions per second • Transactions Response Time (Average, Min, Max) • Windows Resources o CPU o Memory o Paging • Network Traffic
AV03	Scripts consistency. As LoadRunner is not a functional test tool, any change in the presentation layer doesn’t affect to the script.
AV04	Error management. Information regarding the execution is displayed live, seeing all users simulated and if there is any error.
AV05	Test Reports. LoadRunner lets you create HTML reports for any execution. It also includes a Word Report generation tool to automatically summarize and display the test's significant data

Tabla 5.6. Ventajas de la Opción 2 de la propuesta.

- Desventajas:

ID	DESCRIPTION
DAV01	All scripts included in the Crosstest must be created for LoadRunner. Existing scripts can not be reused.
DAV02	Additional LoadRunner license to use Citrix ICA protocol is required (see general chapter assumptions)
DAV03	LoadRunner environment is shared by other projects. This environment must be booked for all Crosstest executions.

Tabla 5.7. Desventajas de la Opción 2 de la propuesta

- Propuesta:

The overall effort estimation for the solution for all scenarios (I+II+III) is **69,5 MDs**, including analysis, set-up, development, test and delivery
Following a detailed description of the tasks covered:

	effort
TOTAL	***69,50***
Proof of Concept	*2,00*
Application Analysis	*16,00*
In this phase, all applications involved in the Load Test will be analyzed from a performance point of view. All information sent by each step performed by the user will be examined in order to detect which fields need to be parameterized or correlated	
Scripts Development	*33,00*
During the development phase, all business flows will be recorded using Mercury Virtual User Generation through the Cirtrix ICA protocol. Scripts will be parameterized and correlated.	
Scenarios Creation	*1,00*
Once the business flows are developed, scenarios need to be built with LoadRunner Controller. Within the scenarios, duration, number of users and server monitoring will be configured.	
Project documentation	5,00
Execution	3,00
All scenarios previously created will be performed. During the execution phase, a daily execution report will be delivered with a summary of all executions performed and general results	
Results Analysis	2,00
From the execution, several graphs and tables are created from LoadRunner. An analysis is necessary to extract the meaningful information.	
Report	0,50
From the execution, several graphs and tables are created from LoadRunner. An analysis is necessary to extract the meaningful information.	
Project Management	7,00
project tracking and reporting tasks	

Tabla 5.8. Descripción de la propuesta de la Opción 2

5.2 Desarrollo del código

En esta parte comienza el verdadero trabajo del probador ya que, aunque en la fase anterior tiene mucho que decir y su opinión tiene que ser muy tenida en cuenta es ahora donde realmente empieza su trabajo de probador.

El desarrollo del código también se divide en tres fases, dos previas en que el probador se encarga de conocer la herramienta a testear y de preparar teóricamente el test y una última fase en que se desarrolla el código propiamente dicho.

En nuestro ejemplo el cliente optó por la primera opción por lo que el trabajo se completó con la herramienta Rational Visual Test de IBM.

5.2.1 Prueba de concepto

La prueba de concepto consiste en una prueba para que el tester se familiarice con la aplicación para la cual ha de crear el test. Normalmente consiste en una reunión, ya sea en persona o a través de teleconferencia, en la que algún desarrollador de la herramienta en cuestión o alguien que la conozca suficientemente bien muestra al probador como funciona ésta. Es habitual que se aproveche esta misma reunión para definir los testcases que quiere que se realicen.

Cada testcase prueba una pequeña parte de la aplicación o un determinado flujo de acciones dentro de ella. Por ejemplo, abrir una aplicación y a continuación cerrarla. En este caso tenemos un testcase sencillo que se encarga de comprobar que la aplicación se abre correctamente y se cierra sin problemas. Aunque pueda parecer absurdo, este pequeño testcase puede servir para detectar Memory Leaks (memoria que no se libera adecuadamente) ocurridos al cerrar.

Una vez realizada esta reunión el probador comprueba por su cuenta que los testcases realizados de forma manual funcionan y plasma la definición del testcase en un documento que el responsable del test del lado del cliente tiene que confirmar. Es esta una forma de evitar problemas de malentendidos entre lo que quería el cliente y lo que entendió el tester.

Con el test de Titicaca este documento quedó de esta manera:

Test Scenario

A unique test scenario is defined for this test with following requirements:

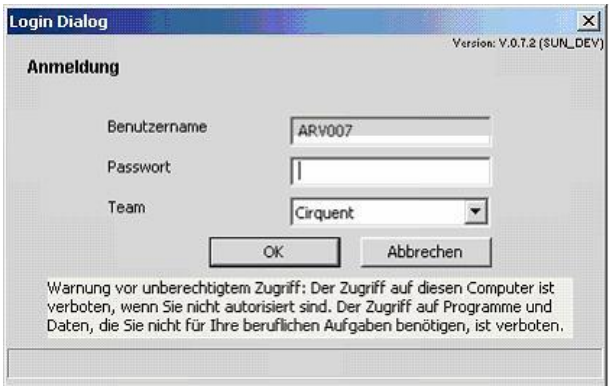
- 20 test users are connecting and logging into the Citrix Farm
- All these test users star the Titicaca application
- Each test user selects the same working list
- Each of these test users selects a different work item from the list.
- With the selected work item each test user executes the test case
- The test case execution will be repeated several times but without closing the Titicaca application
- The test execution will be performed during a whole working day.

Test Cases

There are many functional test cases but they will be executed as a unique one with following steps:

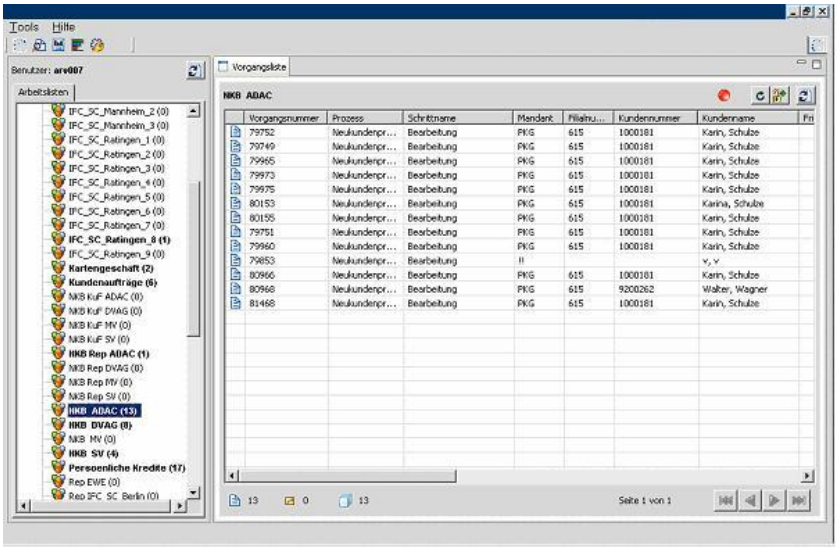
1. Start the Titicaca application by clicking on the desktop on the icon [Titicaca]

2. Perform the logon



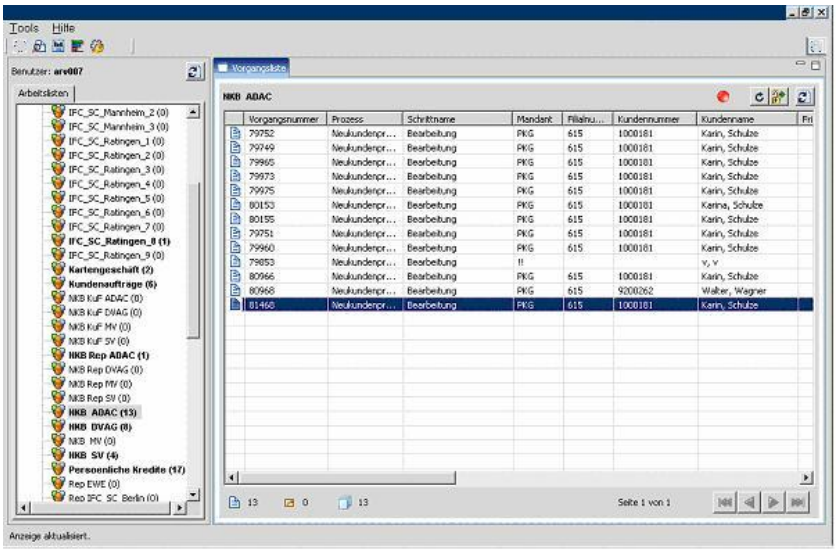
After the application is started and the test user is logged on, on the left side a list must be shown.

3. On the Tab list,



click a item from the list.

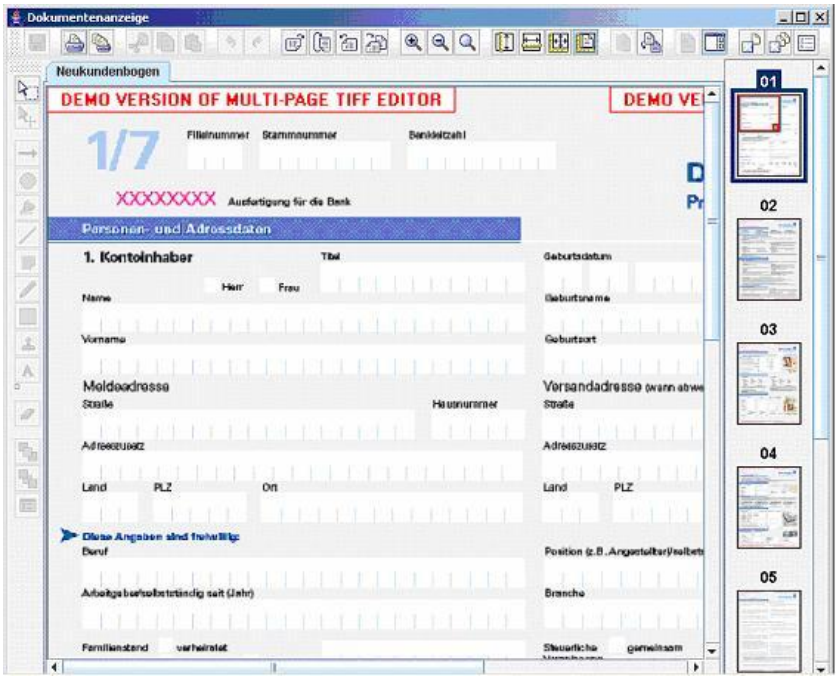
4. On the right window,



select the work item corresponding to this test user
Remarks: each test user must have an own or different defined test working item.

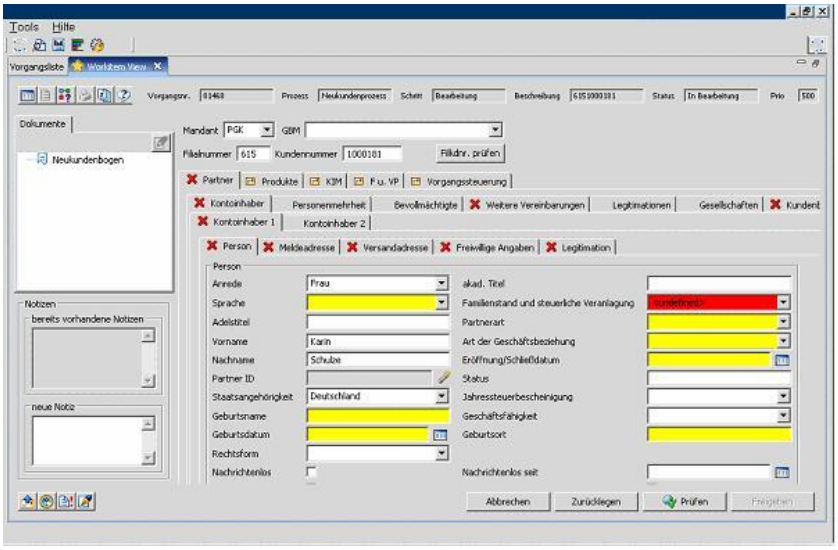
5. Double click on the selected item.

   The following document will be opened:



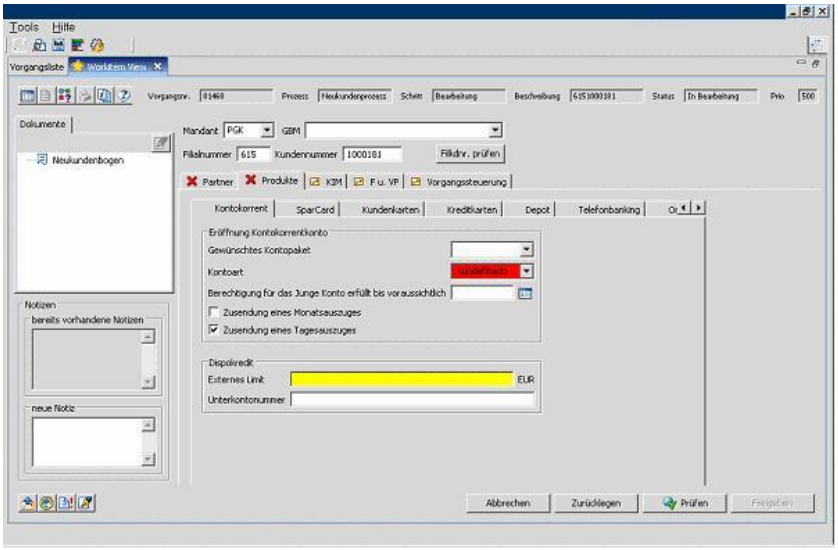
6. Minimize this document (keep it opened).

7. On the Tab <Workitem view>,



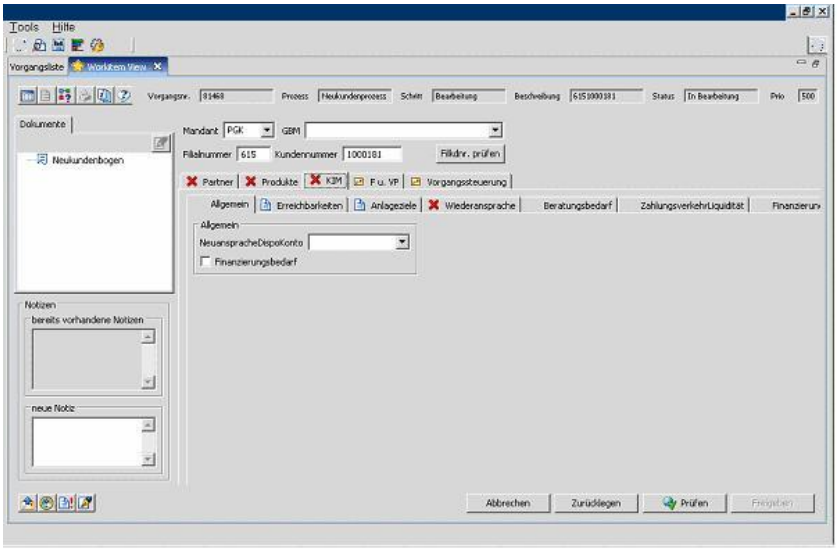
click on Tab [Partner], and
wait 30 seconds.

8. On the same dialog,



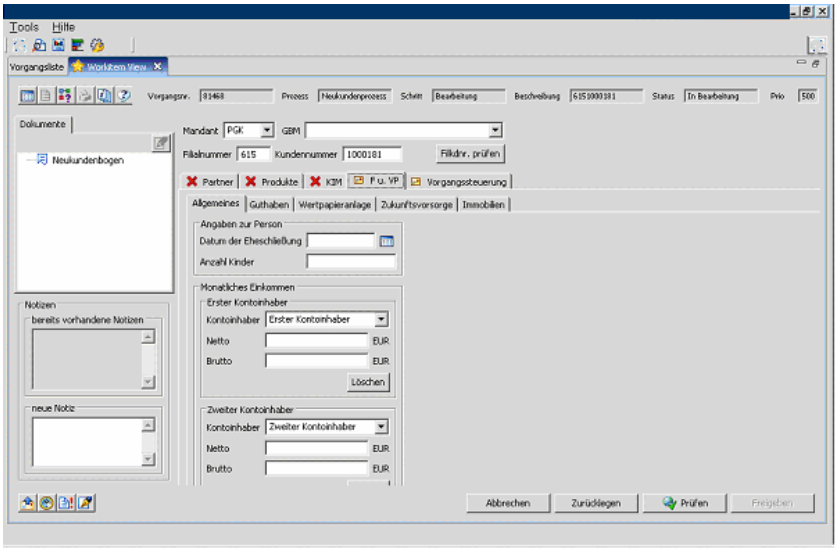
click on Tab [Produkte], and
wait 30 seconds.

9. On the same dialog,



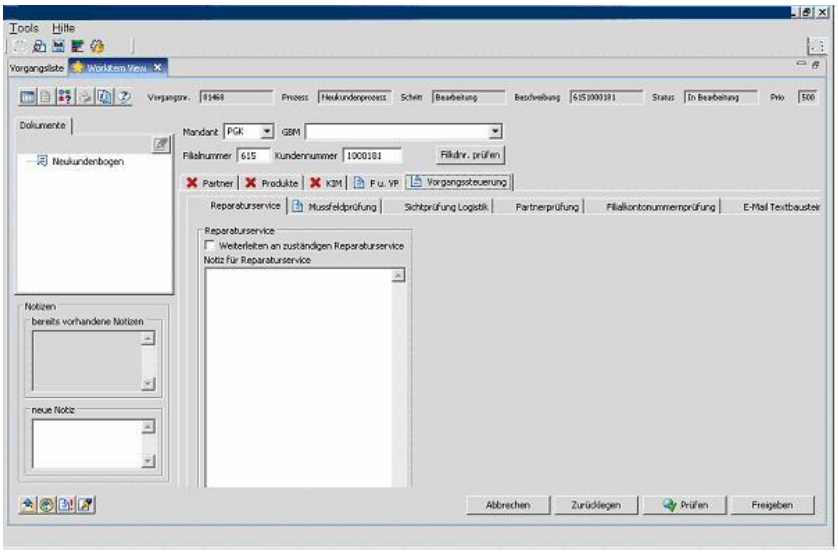
click on Tab [KIM], and
wait 30 seconds.

10. On the same dialog,



click on Tab [F u. VP], and
wait 30 seconds.

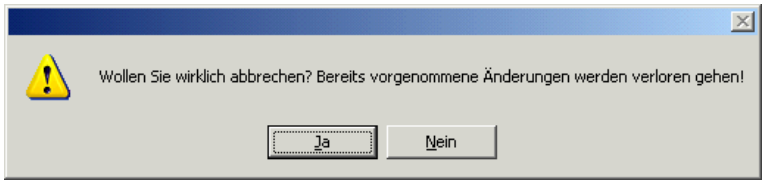
11. On the same dialog,



click on Tab [Vorganssteuerung], and
wait 30 seconds.

12. On the same dialog click on button [Abbrechen]

13. On the confirmation dialog click on button [Ja]



The [Workitem View] will be closed.

At this point the application must be kept open and the test case must be repeated.

5.2.2 Diseño

Una vez superada la prueba de concepto el tester tiene que decidir como afrontar la prueba. Dependiendo de la herramienta se elegirá una forma de actuar u otra, una forma de crear los testcases u otra.

En nuestro caso tenemos que la herramienta es Rational Visual Test. Ya la hemos utilizado antes así que estamos familiarizados con ella y sabemos cual es la mejor forma de afrontar el test por experiencias pasadas.

Crearemos pues los escenarios necesarios, en teoría 30 diferentes, que serán los pasos que seguirá cada usuario virtual mientras dure la prueba. Además tenemos que separar todo el testcase principal (que hemos visto en la prueba de concepto) en diferentes testcases más pequeños: login, selección de elemento de la lista... El objetivo es dividir el test en partes lo más pequeñas posibles que, sin alterar en exceso el desarrollo normal, permitan detectar y corregir errores de forma rápida.

En concreto el objetivo diseñado era el siguiente:

- 30 Escenarios diferentes con dos diferencias básicas:
    1. usuario de acceso al servidor
    2. elemento de la lista a seleccionar

- 3 diferentes testcases:
    1. login
    2. selección de elemento
    3. selección de pestañas

5.2.3 Scripting

Esta parte es propiamente el desarrollo del código y consiste en plasmar en el lenguaje de la herramienta de test todo lo que se ha estado planificando hasta ahora.

El *Rational Visual Test* dispone de la posibilidad de grabar la actuación del usuario y transformarla automáticamente en código. Esta opción es una buena primera aproximación y permite establecer el esqueleto de lo que será el test.

Una vez se han grabado las acciones el código que nos presenta la herramienta ha de ser pulido y completado para tener en cuenta los posibles errores, retrasos cuando el servidor esté más o menos saturado, etc.

Este parte del trabajo, aunque requiere de cierta habilidad y experiencia para minimizarla, se basa bastante en el método ensayo y error. Con el esqueleto conseguido con la grabación directa, si se ejecuta, no se suele conseguir más que en el primer momento en que pueda fallar, el test falle.

Es por eso que una vez detectado el posible fallo del test, que no de la herramienta, hay que corregirlo, más rápidamente si se tiene experiencia y menos si apenas se conoce la herramienta. Como ya hemos dicho, ya hemos trabajado anteriormente con *Rational Visual Test* por lo que este proceso, aunque es el más largo de toda la fase de desarrollo se simplificó bastante gracias a los anteriores tests realizados.

5.3 Ejecución

En nuestro caso se realizaron tres ejecuciones diferentes. El escenario para todas las ejecuciones fue el mismo mientras que lo que variaba era el número de usuarios virtuales conectados al servidor. En el primer test se utilizaron 20 usuarios, en el segundo 10 y en el tercero 15.

El número de ejecuciones teórico es de una por test pero en nuestro caso, debido a los resultados obtenidos, se optó por realizar la prueba hasta 3 veces diferentes. Después de cada ejecución se realizó el correspondiente análisis y se entregó un informe al cliente. Por comodidad hemos decidido agrupar la explicación de cada prueba por fases (ejecución, análisis e informe) en vez de por ejecución que obligaría a volver adelante y atrás cada vez.

5.3.1 Primer test

Como teóricamente se esperaba que los servidores aguantaran hasta 30 usuarios se decidió realizar un test que a priori era conservador, considerando sólo 20 usuarios simultáneos trabajando con Titicaca en un mismo servidor.

Al realizar el test resultó que las estimaciones eran demasiado optimistas y aún con "sólo" 20 usuarios la ejecución no fue tan bien como se esperaba. Vemos a continuación como, al haber transcurrido tres horas de trabajo con Titicaca, el servidor quedó saturado, algo inaceptable teniendo en cuenta que una jornada laboral es de, al menos, 8 horas.

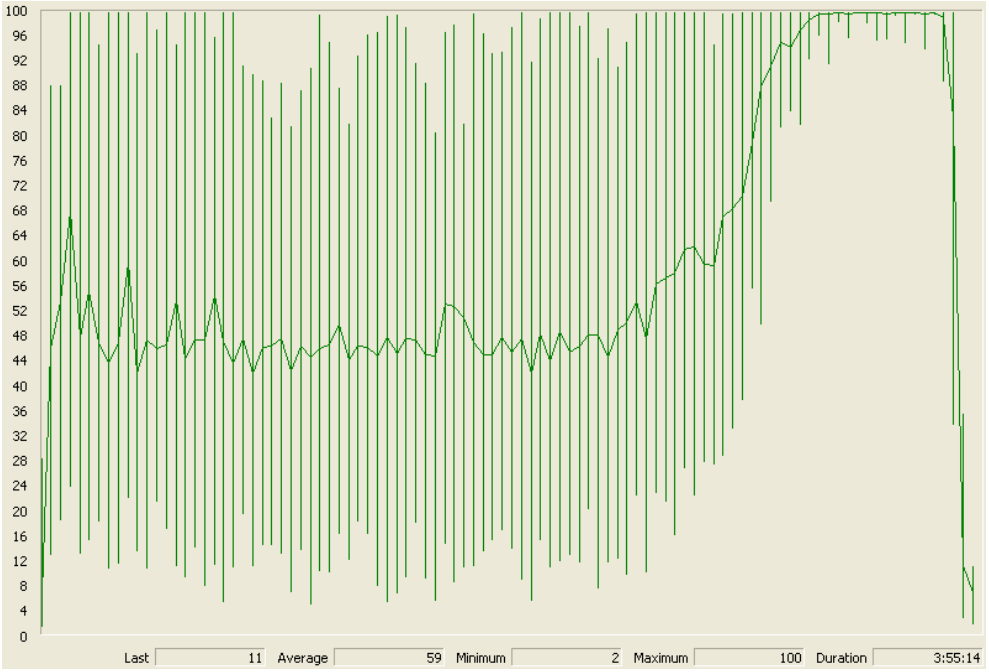


Fig 5.1. Comportamiento de la CPU del servidor durante la ejecucion del primer test

Podemos ver en la figura 5.1 como el consumo se dispara al cabo de unas tres horas aproximadamente y alcanza el 100% durante la última media hora en que la aplicación dejó de funcionar correctamente. Además, es fácilmente apreciable que el consumo medio mientras el programa trabajaba de forma correcta es de casi un 50%, un porcentaje de CPU muy elevado si tenemos en cuenta que sólo estamos haciendo funcionar una sola herramienta.

5.3.2 Segundo test

En vista de los resultados obtenidos con el primer test el cliente se planteó realizar otra prueba después de hacer algunos cambios en el código de Titicaca. Así pues, un par de semanas después de realizar la primera ejecución se preparó una segunda.

El escenario era el mismo (mismo testcase definido sobre una versión diferente del mismo programa) por lo que lo único que variaba era el número de usuarios. Esta vez, teniendo en cuenta los resultados anteriores, se optó por una posición más conservadora aún y se decidió realizar una prueba con 10 usuarios concurrentes. Así pues diez sesiones diferentes se abrieron en el servidor para realizar el test.

Los resultadotes fueron esta vez mucho mejores, el consumo medio había bajado sensiblemente y el servidor no se había saturado en ningún momento dando esperanzas de que los cambios realizados funcionaban y habían mejorado la eficiencia de Titicaca.

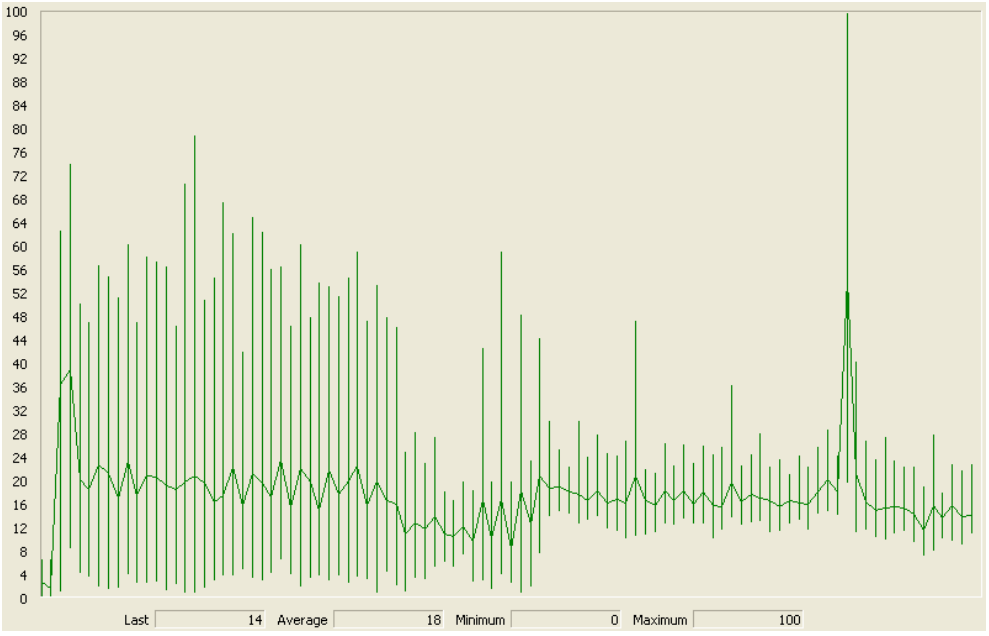


Fig. 5.2. Comportamiento de la CPU del servidor durante la ejecución del segundo test

5.3.3 Tercer test

El resultado del segundo test hizo ser optimista al cliente y éste nos pidió un tercer test para confirmar la tendencia de mejora. En este tercer test serían 15 los usuarios virtuales y si los resultados eran tan buenos como en el anterior se lanzaría un cuarto test que definiría el número de usuarios por servidor que podrían utilizar.

Pero este tercer test estuvo en la línea del primero. Los cambios no habían surgido el efecto vislumbrado por el segundo test y, aunque el servidor no se saturaba con los 15 usuarios simultáneos, el programa dejaba de funcionar correctamente al cabo de unas tres horas de ejecución.

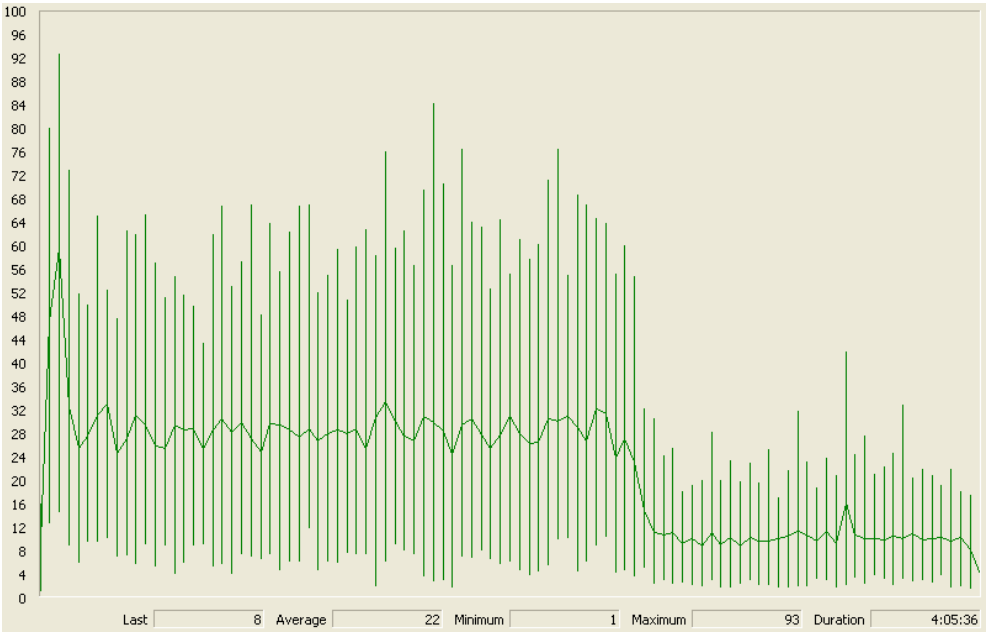


Fig. 5.3. Comportamiento de la CPU del servidor durante la ejecución del tercer test

Podemos ver en la figura 5.3 como el consumo de CPU cae al cabo de esas tres horas debido a que el programa deja de funcionar correctamente y, por lo tanto, de consumir recursos.

5.4 Análisis de resultados

El análisis y tratamiento de resultados nos permitirá siempre elaborar un informe útil y con información que aporte algo al cliente. Para realizarlo debemos primero organizar los datos recogidos durante el test.

Como vimos en los requerimientos los datos principales que se piden son el consumo de CPU, del cual hemos visto los gráficos obtenidos en el apartado anterior, y la memoria consumida por el programa del total de la memoria disponible del servidor. Estos datos se obtienen, en nuestro caso, a través de una herramienta de *Windows* llamada *Performance Monitor*. Este servicio que ofrece Windows permite almacenar los datos de la máquina en un archivo .csv que es compatible con Excel y con algún otro procesador de texto simple como *Notepad* o *UltraEdit*.

El *Performance Monitor* se lanza en el servidor para que registre los datos de consumo de CPU y Memoria durante toda la ejecución. Después estos datos se guardan, como ya hemos dicho, en un archivo .csv y con él trabajaremos para el análisis de los resultados que nos ofrezca el test.

A partir de los datos obtenidos, como podemos ver en la figura 5.4, se generan, con un programa diseñado por nosotros para trabajar con ese formato de datos, una serie de gráficos y de archivos de texto (Resultados parciales en la figura 5.4) que nos los presentan de una forma mucho más legible y visual. A partir de estos archivos y gráficos, trabajando con *Excel* y *Access* generamos nuevos gráficos y documentos que serán muy útiles a la hora de realizar el informe (Resultados finales en la figura 5.4).

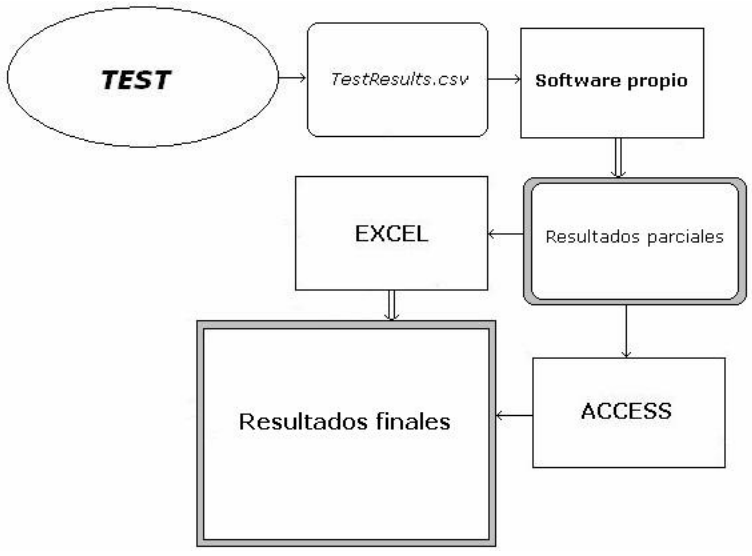


Fig. 5.4. Esquema del proceso de generación de gráficos y documentos útiles a partir de los obtenidos del test

5.5 Informe

El objetivo del informe es ofrecer al cliente la suficiente información de forma clara y concisa. No hay que olvidar que el cliente no ha de saber de testing e incluso no tiene porqué conocer los detalles de la aplicación sobre la que se realiza la prueba. Es por eso que el informe ha de mostrar los resultados de forma visual y rápida, con gráficos y tablas, a poder ser coloreadas de forma instintiva: verde si el test ha ido bien y rojo se ha ido mal.

Es importante también, en caso de que el resultado de la prueba no sea el esperado, justificarlo, indicando cuales pueden ser los errores que han causado el mal resultado y sugiriendo, siempre dejando al cliente la decisión final, las posibles soluciones para los problemas detectados.

En nuestro caso presentamos un informe después de la primera ejecución y, en vista de los resultados, el cliente decidió realizar una segunda y una tercera pruebas por lo que tuvimos que presentar tres informes diferentes.

Resource usage	green	amber	red
RAM [MB]	<= 3400	> 3400 <= 3680	> 3680
CPU [%]	<= 55	> 55 <= 65	> 65

Tabla 5.9. Criterio de evaluación para los resultados obtenidos en el test

5.5.1 Primer test

Para el primer test se presentó un informe que incluyó el gráfico expuesto en la figura 5.1 para mostrar de forma visual el resultado de la prueba, en este caso sólo el comportamiento de la CPU. Se incluyó también la tabla 5.10 con los valores significativos de la memoria consumida y la CPU durante la ejecución y la tabla 5.11 donde siguiendo los criterios expuestos en la tabla 5.9 se evalúan los resultados obtenidos.

Resource	Min.	Average	Max.
CPU	2 %	59 %	100 %
RAM	1916,01 MB	3470,08 MB	3674,12 MB

Tabla 5.10. Valores mínimo, medio y máximo de los consumos de CPU y Memoria.

		Measure	Status
Resource usage	**RAM**	3731,53 MB	**red**
	CPU	59 %	**amber**
General performance		**Not acceptable**	

Tabla 5.11. Resultados del primer test.

Como los resultados no ofrecen dudas se incluyó también en el informe un apartado de conclusiones en el que se recomendaba revisar principalmente el consumo de memoria y se apuntaba la posibilidad de la existencia de Memory Leaks puesto que aún cuando el programa quedaba colgado y no hacía nada el consumo de memoria seguía creciendo indefinidamente.

5.5.2 Segundo test

El segundo test tuvo lugar unas semanas después del primero. Los desarrolladores del software habían hecho algunos cambios que creían que podían haber solucionado si no de forma total, al menos en parte, el problema de consumo de memoria que hacía que el servidor se saturase. El informe presentado siguió la misma plantilla que el anterior: incluyó el gráfico expuesto en la figura 5.10 para mostrar de forma visual el comportamiento de la CPU, la tabla 5.12 de memoria y CPU consumidas y la tabla 5.13 donde siguiendo los criterios expuestos en la tabla 5.9 se evalúan los resultados obtenidos.

Resource	Min.	Average	Max.
CPU	0 %	18 %	100 %
RAM	1494,71 MB	2504,18 MB	2646,12 MB

Tabla 5.12. Valores mínimo, medio y máximo de los consumos de CPU y Memoria.

		Measure	Status
Resource usage	**RAM**	2504,18 MB	**green**
	CPU	18 %	**green**
General performance		**Acceptable**	

Tabla 5.13. Resultados del segundo test.

En este caso los resultados fueron muy buenos y animaron al cliente a realizar una tercera prueba para comprobar si el aumento de usuarios hasta 15 mejoraba el resultado obtenido en el primer test.

5.5.3 Tercer test

En este tercer test el informe varió ligeramente ya que decidimos, puesto que la versión de la herramienta era la misma que en el segundo test, realizar una comparativa entre los resultados obtenidos. Así pues, el informe estaba basado en los anteriores pero sustituyendo las tablas presentadas por unas en que se podían ver los resultados del segundo test también. Es por eso que, además de figura 5.10 para mostrar de forma visual el comportamiento de la CPU, presentábamos la tabla 5.14 con los valores significativos de CPU y memoria, la tabla 5.15 donde se muestra la variación en % de los valores obtenidos tanto de memoria como de CPU, Por último incluimos la tabla 5.16 donde siguiendo los criterios expuestos en la tabla 5.9 se muestran los resultados obtenidos de las dos pruebas.

Resource	# users	Execution	Min.	Average	Max.
CPU	10	*Test-2*	0 %	18 %	100 %
CPU	15	*Test-3*	1 %	22 %	93 %
RAM	10	*Test-2*	1494,71 MB	2504,18 MB	2646,12 MB
RAM	15	*Test-3*	1916,01 MB	3470,08 MB	3674,12 MB

Tabla 5.14. Valores mínimo, medio y máximo de los consumos de CPU y Memoria del segundo y tercer test.

		Variation 2-3	Measure Test-2	Measure Test-3
Resource usage	**RAM**	**+ 38,57 %**	3470,08 MB	2504,18 MB
	CPU	**+ 4 %**	22 %	18 %

Tabla 5.15. Comparativa entre los resultados medios de la segunda y la tercera prueba.

		Measure 3	Status 3	Measure 2	Status 2
Resource usage	**RAM**	3470,08 MB	amber	2504,18 MB	green
	CPU	22 %	green	18 %	green
General performance		Acceptable		Acceptable	

Tabla 5.16. Resultados de ambos tests.

Teniendo en cuenta que el test tuvo que pararse al cabo de tres horas de ejecución debido a que el programa dejó de funcionar y viendo la progresión del consumo de memoria (casi un 40% más con sólo 5 usuarios más) se optó por dar por finalizados los tests y dejar en 10 usuarios en vez de los 30 previstos al principio el número máximo de usuarios por servidor.

6. CONCLUSIONES

Con este proyecto hemos pretendido descubrir, explicar y evaluar las diferentes opciones que se presentan en un entorno de trabajo real a la hora de realizar tests, en nuestro caso, de carga.

En este proyecto hemos visto el desarrollo de un test desde el momento en que surge la necesidad de realizar un test hasta el momento en que se entrega el informe con los resultados de dicha prueba. Hemos comprobado como el proceso es mucho más grande que solamente realizar las pruebas que se piden y que la ejecución de las mismas es sólo una pequeña parte del global, igual de necesaria e imprescindible que el resto.

También hemos visto en qué consiste el testing desgranando los más importantes tipos de prueba que podemos realizar y que, de hecho, se realizan durante la parte de testeo del ciclo de software. Hemos podido describir qué son y cómo funcionan los sistemas distribuidos y el porqué de su gran supremacía actual en el mundo de la empresa. Además hemos evaluado algunas de las diferentes alternativas que se presentan en el mercado de cara a trabajar en estos entornos centralizados.

Nos hemos decidido, a la hora de elegir una opción para trabar por, en un entorno distribuido Windows, utilizar la herramienta *MetaFrame* de *Citrix* porque sus características encajan perfectamente con nuestros objetivos. Estos objetivos, realizar un test en el que 30 usuarios trabajaran simultáneamente con un programa de gestión bancaria que hemos llamado Titicaca en un servidor Windows, eran fácilmente accesibles con esta herramienta que nos ofrece la compañía *Citrix* y llegamos a la misma conclusión a la que había llegado el banco y nos decidimos también por *MetaFrame*.

Para terminar, hemos entrado en detalle en un test real realizado por una empresa para comprobar que de la teoría a la realidad, de lo que se plasma en una propuesta a lo que al final se acaba realizando normalmente hay una gran diferencia. La realización de tres diferentes ejecuciones con tres diferentes configuraciones tanto de test como de software a probar nos ha permitido obtener diferentes conclusiones después de cada una de las ejecuciones que han obligado al cliente a tomar decisiones inesperadas muy diferentes a las que teóricamente tenían que tomar.

El proyecto nos ha traído una aproximación al mundo del testing, un mundo cada vez más extendido hoy en día, tanto con la creación de empresas dedicadas exclusivamente a esto como en departamentos dentro de las grandes consultoras informáticas.

El futuro de las grandes empresas pasa por un incremento de su presupuesto para testing puesto que es este departamento el que permitirá reducir costes de mantenimiento al minimizar los errores antes de la salida del producto al mercado. Así mismo el número de empresas dedicadas exclusivamente a las pruebas de software será cada vez más importante ya que la realización de los tests se está convirtiendo en un elemento imprescindible para que cualquier proyecto tenga éxito.

Hemos podido ver, con este proyecto una perspectiva global de un mundo que se vuelve imprescindible en una sociedad cada vez más informatizada en el que el I+D y las nuevas tecnologías se están convirtiendo en la base de una sociedad avanzada como la nuestra y que pretende estar a la altura del norte de Europa o de Estados Unidos.

7. BIBLIOGRAFÍA

1. http://citrix.com/
2. http://www.sun.com/
3. http://www.microsoft.com/
4. http://www.monografias.com/
5. http://www.csi.map.es/
6. http://www.x.org/
7. http://www.realvnc.com/

ABSTRACT

EVERY DAY IS MORE USUAL TO FIND ALL THE COMMON SOFTWARE AND HARDWARE OF THE COMPANYS DISTRIBUTED AND MANAGED BY DIFFERENT SERVERS WHAT SERVE TO THE USER ALL THEY NEED ONLY WHEN THEY ASK FOR IT. THIS INFORMATION DISTRIBUTION SYSTEM IS CALLED CENTRALIZATION.

THIS DISTRIBUTION SYSTME REQUIRES A CONTINUOUS MAINTENANCE IN ORDER TO ATTEND ALL THE USERS DEMANDS. DUE TO THE CENTRALIZATION THIS MAINTENANCE BECAMES AN EASY PROCESS THAT INVOLVES ONLY THE SERVER UPDATE. THE SERVER CHECKING IS NOW VERY IMPORTANT BECAUSE IT'S NECESSARY TO CHECK IF THE UPDATES ANSWER PROPERLY TO THE USERS REMOTE DEMANDS.

IN THIS PROJECT WE HAVE ANALIZED HOW ALL THIS TESTS TO ASSURE THE RIGHT PERFORMANCE OF THE REMOTE SERVERS ARE DONE. WE HAVE CONSIDERED THE ENVIRONMENT WHERE THE TESTS ARE PERFORMED AND THE NECESSARY TOOLS TO DO THEM. TO COMPLETE THIS INFORMATION WE HAVE SEEN A PARTICULAR EXAMPLE OF A LOAD TEST.

RESUMEN

CADA VEZ ES MAS HABITUAL ENCONTRAR TANTO EL SOFTWARE COMO EL HARDWARE COMÚN DE LAS EMPRESAS DISTRIBUIDO Y GESTIONADO EN DIFERENTES SERVIDORES QUE SE ENCARGAN DE SERVIR AL USUARIO AQUELLO QUE NECESITA SOLO CUANDO ESTE LO PIDE. ESTE SISTEMA DE DISTRIBUCIÓN DE LA INFORMACION SE LLAMA CENTRALIZACION.

ESTE SISTEMA DE DISTRIBUCION REQUIERE UN MANTENIMIENTO CONSTANTE PARA ASÍ PODER ATENDER TODAS LAS DEMANDAS DE LOS USUARIOS. EL MANTENIMIENTO SE CONVIERTE, GRACIAS A LA CENTRALIZACIÓN EN ALGO RELATIVAMENTE SENCILLO PUESTO QUE SOLO ES EN EL SERVIDOR DONDE SE TIENEN QUE REALIZAR LOS CAMBIOS, ACTUALIZACIONES O INSTALACION DE NUEVO SOFTWARE. ES IMPORTANTE ENTONCES COMPROBAR QUE ESTAS NUEVAS ACTUALIZACIONES DEL SERVIDOR RESPONDERAN CORRECTAMENTE CUANDO LOS USUARIOS LAS REQUIERAN REMOTAMENTE.

EN ESTE PROYECTO NOS HEMOS ENCARGADO DE ANALIZAR COMO SE REALIZAN LAS COMPROBACIONES NECESARIAS PARA ASEGURAR EL CORRECTO FUNCIONAMIENTO DE LOS SERVIDORES REMOTOS CONSIDERANDO TANTO EL ENTORNO EN EL QUE SE REALIZAN COMO LAS HERRAMIENTAS NECESARIAS PARA LLEVARLO A CABO. PARA COMPLETAR LA INFORMACIÓN NOS HEMOS CENTRADO EN UN EJEMPLO PARTICULAR DE TEST DE CARGA.

RESUM

CADA COP ES MES HABITUAL TROBAR-NOS TANT EL SOFTWARE COM EL HARDWARE COMU D'UNA EMPRESA DISTRIBUIT I GESTIONAT A DIFERENTS SERVIDORS QUE S'ENCARREGUEN DE SERVIR L'USUARI TOT ALLÒ QUE NECESSITI NOMES QUAN HO DEMANI. AQUEST SISTEMA DE DISTRIBUCIÓ DE L' INFORMACIO S'ANOMENA CENTRALITZACIO.

AQUEST SISTEMA DE DISTRIBUCIO FA NECESSARI UN MANTENIMENT CONSTANT PER PODER ATENDRE TOTES LES DEMANDES DELS USUARIS. EL MANTENIMENT ARRIBA A SER, GRACIES A LA CENTRALITZACIO, EN UNA COSA RELATIVAMENT SENZILLA DEGUT A QUE NOMES ES AL SERVIDOR ON S'HAN DE REALITZAR ELS CANVIS, ACTUALITZACIONS O INSTAL·LACIONS DE NOU SOFTWARE. LLAVORS ES IMPORTANT COMPROVAR QUE AQUESTAS NOVES ACTUALITZACIONS DEL SERVIDOR RESPONDRAN CORRECTAMENT QUAN ELS USUARIS LES DEMANDIN REMOTAMENT.

EN AQUEST PROJECTE ENS HEM ENCARREGAT D'ANALITZAR COM ES REALITZEN LES COMPROBACIONS NECESSARIES PER ASEGURAR EL CORRECTE FUNCIONAMENT DELS SERVIDORS REMOTS TOT CONSIDERANT L'ENTORN EN EL QUE ES REALITZEN I LES EINES QUE NECESSITAREM PER DUR-LES A TERME. PER COMPLETAR L' INFORMACIÓ ENS HEM CENTRAT EN UN EXEMPLE PARTICULAR D'UN TEST DE CÀRREGA.