CAPÍTULO 3: ARQUITECTURA PROPUESTA

3.1. JAVA

3.1.1. INTRODUCCIÓN

Como se mencionó en el primer capítulo de esta memoria, este proyecto nació de la idea de incorporar a la aplicación de la que procede un interfaz de usuario que permitiera manejarla de un modo sencillo e intuitivo. Unido a esto, el concepto de interfaz *web* dejó claro el que iba a ser el lenguaje de programación a emplear: Java [Java'01]. De hecho, Java no sólo es un lenguaje de programación, sino también una plataforma sobre la que hacer funcionar los programas.

3.1.1.1. JAVA COMO LENGUAJE DE PROGRAMACIÓN

Java es un lenguaje de programación de alto nivel entre cuyas características están las que vienen definidas por esta serie de palabras: sencillo, orientado a objeto, independiente del sistema operativo empleado, reutilizable, distribuido, de alto rendimiento, interpretado, concurrente, robusto, dinámico y seguro.

En la mayoría de los lenguajes de programación, o bien se compila, o bien se interpreta un programa para poder ejecutarlo en un ordenador. La peculiaridad de Java está en que ambas acciones se llevan a cabo, es decir, primero se compila el código a un lenguaje intermedio llamado *bytecode* (código de byte), que es único para todos los sistemas operativos. Cada tipo de máquina tiene un intérprete que se dedica a ejecutar las instrucciones que contiene el código compilado, es decir: se compila una única vez y se interpreta cada vez que se quiera ejecutar el programa.

El proceso viene ilustrado en la siguiente figura:

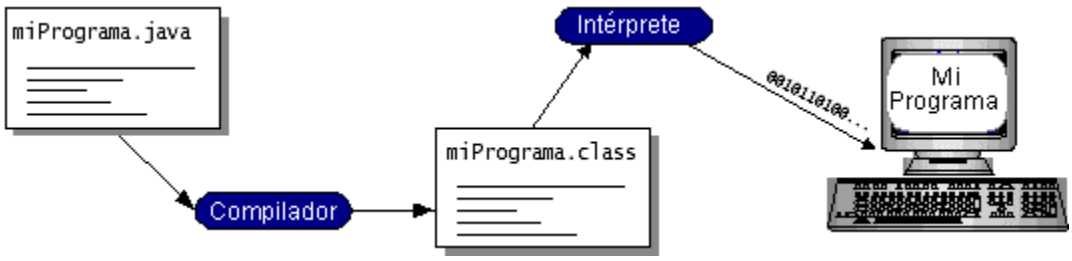

Figura 3.1. Estructura de funcionamiento de Java

Los *bytecodes* generados en la compilación ejercen de instrucciones en código máquina para la Máquina Virtual Java (*Java VM*). Cualquier intérprete de Java, ya sea una herramienta de desarrollo o un navegador que pueda ejecutar *applets* es una implementación de la Máquina Virtual Java. Estos *bytecodes* hacen posible el hecho de que un código sea compilado en cualquier plataforma y ejecutado allá donde haya una implementación de la *Java VM*, ya sea un ordenador que funcione bajo Windows, una estación Solaris o un iMac (véase la figura 3.2).

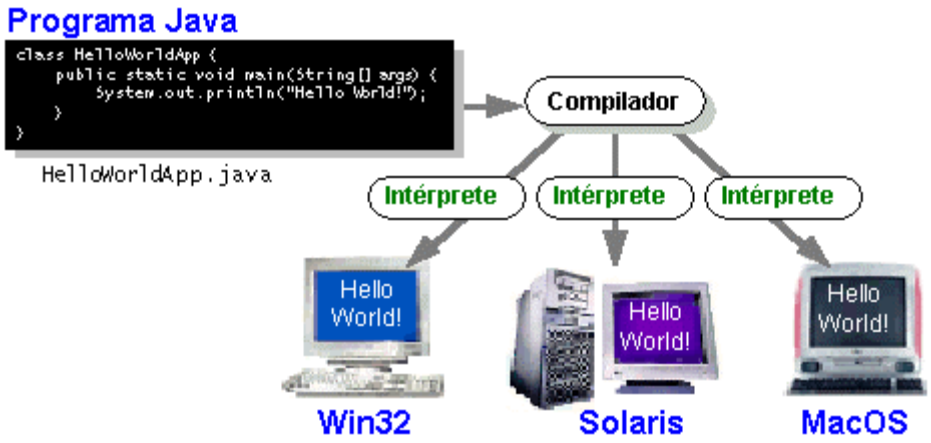

Figura 3.2. Un único código para múltiples plataformas

3.1.1.2. JAVA COMO PLATAFORMA

Una plataforma es el entorno hardware o software en el que se ejecuta un programa. La mayoría de ellas son una combinación de un sistema operativo y un equipo físico. La plataforma Java se distingue de las demás en que es sólo software ejecutándose sobre otras plataformas basadas en hardware. Consta de dos componentes: la Máquina Virtual Java y

el Interfaz de Programación de Aplicaciones (*API*). La primera ha ido descrita con anterioridad. El *API* es un conjunto de componentes software que proporcionan múltiples utilidades, como por ejemplo elementos para crear interfaces gráficos de usuario. Está agrupado en librerías que contienen elementos relacionados entre sí, denominadas 'paquetes'.

En la siguiente figura se expone un programa que se ejecuta sobre la plataforma Java. El *API* y la Máquina Virtual Java aíslan al programa del equipo físico.

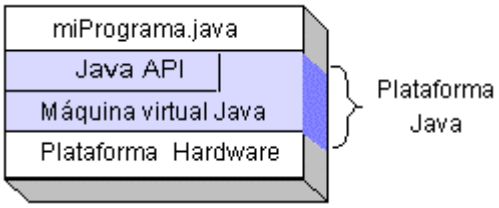

Figura 3.3. La Plataforma Java aísla el código de la máquina

El código denominado "nativo" es compilado específicamente para una determinada plataforma. Esto puede hacer que este tipo de código sea más rápido que el *bytecode* de Java, al ser este último un código universal. Se están realizando logros en compiladores avanzados, intérpretes afinados y otros elementos que acercan el rendimiento de un programa Java al de un código completamente nativo sin amenazar su universalidad.

Una de las librerías de más reciente aparición es la específica para el diseño de interfaces de usuario y tiene el nombre de *Swing*.

3.1.2. INTERFAZ DE USUARIO: SWING

Swing [Swing'01] es el nombre en clave de un conjunto de clases llamado JFC (*Java Foundation Classes*), que vieron la luz con el propósito de sentar una base sólida para crear interfaces de usuario en Java. Su código está escrito íntegramente en Java, con lo cual se garantiza la independencia de la plataforma sobre la que se utilice.

Swing es una poderosa herramienta que cuenta con las siguientes características, entre las más importantes: **Componentes GUI** (*Graphical User Interface*). Introduce un conjunto de librerías de elementos para la creación de interfaces de usuario pensadas para funcionar en colaboración unas con otras. No contienen código específico de ningún sistema para el manejo de ventanas, facilitando así la creación de aplicaciones sin utilizar el entorno de ventanas del sistema en uso. **Aspecto adaptativo al sistema utilizado**. Permite cambiar el aspecto de los componentes visuales de una aplicación al instante, sin reiniciarla y sin tener que variar el código. Además, puede hacerse que una misma aplicación escoja el aspecto del sistema en el que se esté utilizando, para adquirir mayor familiaridad con el entorno. **Accesibilidad.** Proporciona un interfaz indicado para personas con discapacidades. ***API* Java 2D.** Incluye facilidades para la mejora estética de los programas. **Arrastrar y Soltar.** Facilita el intercambio de elementos con aplicaciones no realizadas en Java.

3.1.3. EXTENSIÓN DEL SERVIDOR: JAVA SERVLETS

3.1.3.1. SERVIDOR *WEB*

Un servidor *web* es simplemente un proceso que se ejecuta en una máquina y maneja peticiones de clientes remotos, que suelen ser los ya mencionados navegadores, enviándoles páginas u otro tipo de archivos a través de la red. Las peticiones y las respuestas que se intercambian el servidor y los clientes están definidas por el protocolo HTTP, que está implementado en la mayoría de los casos sobre una red TCP/IP.

Cuando un cliente se pone en contacto con un servidor, desea llevarse un archivo de él, que puede ser desde una página escrita en HTML (*HyperText Markup Language*), empleado en las páginas *web*, hasta un fichero de sonido, una imagen o un *clip* de vídeo. Para ello envía una petición HTTP al servidor en la que se indica una serie de parámetros que caracterizan la conexión existente entre ambos. En estos parámetros va incluido el método a utilizar, el nombre del archivo que se desea descargar, así como otros valores del cliente que el servidor puede o no utilizar.

Una vez recibida la petición y comprobado que existe el fichero que el cliente desea y que no hay restricciones de seguridad o de acceso, el servidor envía una respuesta HTTP

donde indica, entre otras cosas, el tipo de documento que devuelve y su tamaño, para que el navegador del cliente lo reciba correctamente.

Gracias a la disponibilidad de los programas navegadores, muchas aplicaciones que se desarrollan en extremos diferentes de una red, como pueden ser el comercio electrónico, bases de datos, telecontrol y otras, han adoptado para sus comunicaciones el protocolo HTTP. Estas aplicaciones necesitan generar contenidos y realizar determinadas acciones dependiendo de los parámetros que se envíen con la petición, con lo cual, el simple hecho de acceder a una página o un archivo que ya existe se ha vuelto insuficiente. Por consiguiente, se ha hecho necesario aumentar la funcionalidad de los servidores *web* de modo que sean capaces de generar dinámicamente nuevos contenidos a partir de los parámetros que el usuario les proporciona en su petición.

El servidor debe ser capaz de distinguir aquellas peticiones que tienen como objetivo contenidos estáticos de aquellas que proporcionan unos valores para obtener resultados generados de forma dinámica. Si bien esta distinción se puede hacer de formas diferentes, por lo general, el servidor debe ser capaz de saber cuándo se trata de una petición de este último tipo con sólo mirar la URL (*universal resource locator*), que es la dirección del recurso solicitado. Los parámetros irán codificados según el método empleado, en la URL si se trata de un método GET, o como contenido de la petición si se trata de un POST.

3.1.3.2. SERVLETS

- **INTRODUCCIÓN Y VENTAJAS**

Una vez que hemos apuntado que los servidores *web* pueden distinguir entre las peticiones que contienen parámetros y las que no, vamos a describir la técnica escogida para implementar el aumento de funcionalidad necesario en el servidor. La tecnología *servlet* de Java [Servlet'01] es un entorno de trabajo basado en Java que proporciona un mecanismo simple y consistente para aumentar la funcionalidad de los servidores *web*, haciendo uso de la capacidad de Java para cargar clases (ver inicio del capítulo 4) de forma dinámica. Esto se puede hacer funcionar de dos maneras, siendo una el uso de un servidor *web* basado en Java y otra la utilización de ampliaciones para servidores convencionales.

La ventaja que posee el empleo de *servlets* con relación a otros tipos de extensiones de servidor es que hace suyas todas las características del lenguaje Java, con todo lo que ello implica. Entre sus rasgos más destacables se encuentran los siguientes:

- Los *servlets* son independientes tanto del servidor como de la plataforma sobre la que se les haga funcionar. Esto permite que puedan ser reutilizados en cualquier máquina sin necesidad de modificar el código o recompilar los programas.

- Puesto que Java permite ejecutar código dentro de unas medidas de seguridad que restringen el acceso al sistema de archivos, a la red o a métodos comprometidos, cualquier programa puede ejecutarse sin hacer peligrar la integridad del servidor.

- Los *servlets* se cargan una única vez, al iniciar el servidor, creando una nueva hebra de ejecución cada vez que son invocados. Esto, unido a que su tamaño es mucho menor que el que ocupan soluciones con la misma funcionalidad implementadas en otras tecnologías, gracias a que tienen acceso a la Máquina Virtual Java y no necesitan elementos adicionales que dependan del sistema, garantiza un coste computacional y un uso de recursos mucho más reducidos.

- **EL CICLO DE VIDA DE UN SERVLET**

Un *servlet* pasa por tres fases a lo largo de su vida. En la primera, **inicialización**, el servidor carga en memoria el *servlet* y lleva a cabo las tareas que necesita para comenzar su funcionamiento. En la siguiente, **interacción con el cliente**, ninguna, una o más peticiones del extremo remoto son atendidas. Por último, en la **desinicialización** o finalización, se descarga el *servlet* y se liberan los recursos que había venido ocupando. En la figura 3.4. se pueden observar las tres fases.

  - **INICIALIZACIÓN**

Cuando el servidor carga un *servlet*, llama a su método `init()`. Una vez realizada esta tarea, el *servlet* queda registrado y se le considera preparado para atender peticiones de los clientes. Dependiendo del servidor, unos *servlets* se inicializan durante el arranque y otros esperan a ser llamados por primera vez.

- **INTERACCIÓN CON EL CLIENTE**

Una vez que el *servlet* ha sido inicializado, puede atender peticiones mediante su método `service()`. Para cada petición, se le pasan a ese método dos objetos: `ServletRequest`, que contiene los datos procedentes del cliente, y `ServletResponse`, que contiene la respuesta que se envía al cliente.

Dentro del cuerpo de este método es donde se lleva a cabo todo el procesamiento que aumenta la funcionalidad del servidor, debiendo el programador garantizar el correcto funcionamiento de todos los casos cuando se produzcan peticiones simultáneas.

- **FINALIZACIÓN**

Cuando el servidor quiere descargar el *servlet*, llama a su método `destroy()`. Este método debe deshacer la inicialización y ajustar los valores que permanecerán almacenados en el servidor de forma permanente.

Una vez realizado este paso, el *servlet* no podrá ser llamado hasta que no se le vuelva a inicializar. El punto en el que se evacua el *servlet* de la memoria depende del servidor y puede ser al apagar el sistema, de forma periódica, o a petición del administrador.

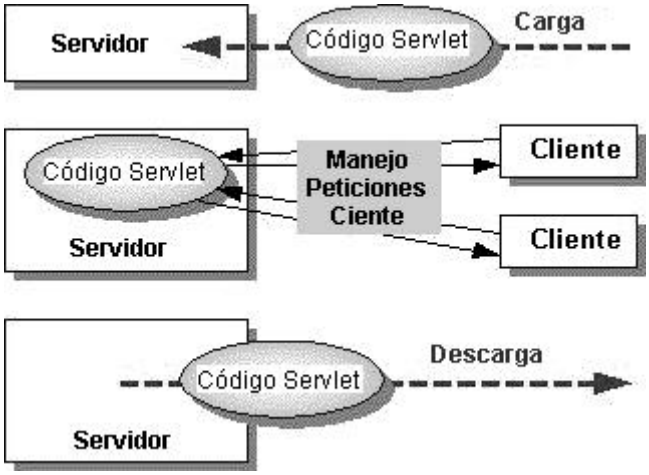

Figura 3.4. El ciclo de vida de un servlet

3.1.4. EXTENSIÓN DEL CLIENTE: JAVA APPLETS

Dado que el lenguaje HTML fue diseñado para transmitir páginas estáticas, la funcionalidad como interfaz de usuario de un navegador se ha vuelto muy escasa para las necesidades actuales. Para ello nacieron los *applets* [Applet'01], que no son más que ampliaciones de la funcionalidad del cliente, del mismo modo que los *servlets* lo son del servidor.

Los *applets* disfrutan de las mismas ventajas que los demás programas escritos en Java, que han sido ya abordadas en este documento, con la diferencia de que tienen ciertas restricciones de seguridad. Un *applet* es descargado automáticamente del sitio *web* en el que está situado al solicitar el cliente la página que lo contiene. Otorgar control total sobre la máquina del cliente a cualquier programa que un desaprensivo pudiera colocar en una página sería una imprudencia, con lo cual hay una serie de acciones que están prohibidas para los *applets* descargados a través de una red. Entre ellas están el acceso al sistema de archivos local, el establecimiento de conexiones de red con otro *host* que no sea aquél del que ha sido descargado y efectuar determinadas llamadas al sistema.

Estas medidas de precaución otorgan una gran tranquilidad y confianza a aquellos que utilizan *applets*, ya que saben que la plataforma Java les protege de sufrir determinados daños. Existe la posibilidad de levantar estas restricciones mediante el uso de *applets* "de confianza" (*trusted applets*) que llevan la firma del autor certificada por una autoridad competente y hacen responsable al firmante de los daños que pudiera ocasionar con un mal uso del lenguaje.

3.2. DISEÑO GENERAL DE LA APLICACIÓN

3.2.1. CONCEPTO

La aplicación desarrollada pretende generar un determinado tráfico telemático de forma sintética siguiendo el esquema que se describe a continuación:

Se pone en marcha un servidor *web* con capacidad de ejecutar *servlets*. Este servidor puede estar funcionando tanto en una red de área local como en la propia Internet. Desde un ordenador conectado a la misma red que el servidor mencionado, ya sea en un lugar con acceso a la misma red de área local como en cualquier lugar del mundo si se trata de Internet, se pone en marcha el programa navegador y se solicita la página donde se hospeda el *servlet* elegido. En nuestro caso se conduce al cliente a la opción elegida a través de una serie de páginas *web* previas.

Al acceder a la página deseada, se descarga un *applet* que se encarga de solicitar los parámetros de la simulación que se va a llevar a cabo. Una vez dada la orden oportuna pulsando el botón correspondiente, se realiza una petición al servidor con la solicitud de una URL donde se halla hospedado el *servlet* asociado al *applet* que está funcionando. En esta solicitud se encapsulan los datos que ha proporcionado el usuario a través del interfaz gráfico que muestra el *applet*. Una vez que el *servlet* recibe los parámetros a modo de objeto de configuración, envía una respuesta al cliente en la que le indica que el proceso se ha llevado a cabo correctamente.

En este momento, el servidor abre un *socket* para atender una petición del cliente. Una vez aceptada esta petición, queda establecida una conexión TCP que denominaremos “de control”, que permanecerá activa durante todo el tiempo que dure la simulación.

Seguidamente, cada extremo de la conexión se encarga de lanzar su mitad del simulador, es decir, la servidora y la cliente cada una en su parte.

La simulación finalizará bien por la pulsación del botón de cancelación o bien por que se haya agotado el tiempo señalado como duración de la misma.

En la siguiente figura se muestra un esquema simbólico del funcionamiento general de la aplicación. En ella se pueden observar los niveles de comunicación que se establecen a lo largo del proceso de simulación completo, comenzando por la llegada al sitio *web* donde reside la totalidad del programa (Comunicación HTTP), continuando con la carga del *applet* en el cliente y el *servlet* en el servidor, la comunicación entre ellos (HTTPServletRequest y HTTPServletResponse), donde se envían los parámetros de los modelos y el acuse de recibo de éstos, el establecimiento de la conexión de control y finalizando con la consecución de la simulación propiamente dicha, mediante la generación del tráfico sintético deseado.

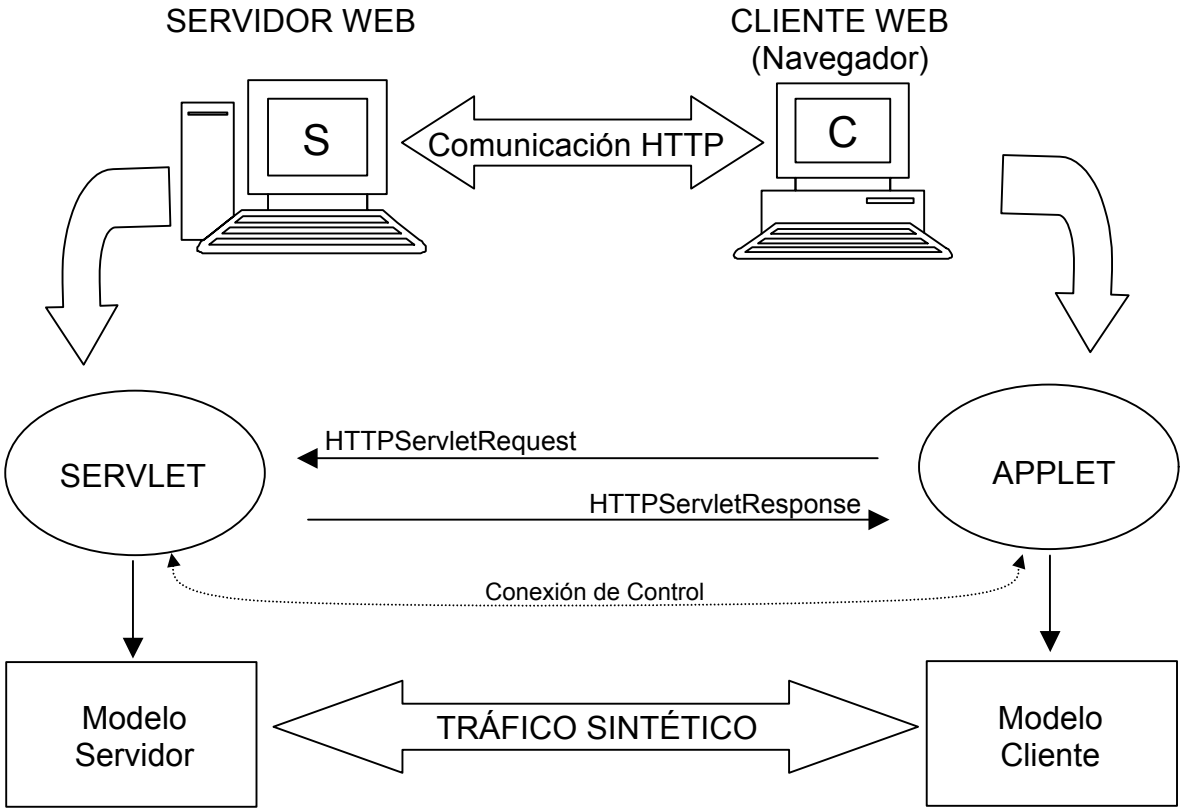

Figura 3.5. Esquema general de la aplicación

3.2.2. HERRAMIENTAS EMPLEADAS

Para la puesta en funcionamiento de este sistema se utilizaron los siguientes elementos:

- Un **PC** Pentium III a 733 MHz con 128 MB de memoria RAM que se empleó tanto para el desarrollo de la aplicación como para el hospedaje de la misma.

- Sistema operativo Microsoft **Windows 2000 Advanced Server**, equipado con el programa servidor *web* Microsoft **Internet Information System**.

- Programa para permitir al servidor manejar *servlets* Allaire **JRun**.

- Una **red de área local** con salida a Internet.

- Un **PC** Pentium III a 600 MHz con 128 MB de RAM conectado a la misma red de área local, que se empleó para hacer pruebas con la aplicación.

- Sistema operativo Microsoft **Windows Millennium Edition** con programas navegadores Microsoft **Internet Explorer 5.5** y Netscape **Navigator 6**.

- Entorno de desarrollo Borland **JBuilder 3**, basado en el *kit* de desarrollo de Java JDK 1.3.0.

3.2.3. REQUISITOS DEL SISTEMA

Para montar la aplicación es necesario disponer de una red de área local, que puede o no tener salida a Internet, y dos ordenadores equipados con cualquier sistema operativo siempre y cuando éste permita la instalación de un servidor *web* con soporte para *servlets* en el servidor y un programa navegador con el '*Java Plug-in*' instalado (ver apartado 5.3) en el cliente. No es necesario mencionar que ambos sistemas operativos pueden ser distintos.

Como requisitos hardware de las máquinas a emplear se han de tomar aquellos que marquen los programas utilizados; es decir, se tomarán los más elevados que necesite cualquiera de las combinaciones de aplicaciones mencionadas. Sirva como ejemplo el sistema detallado en el apartado anterior.

Hay que recalcar que en determinadas redes de área local están instalados *firewalls* o cortafuegos, que actúan como barrera de seguridad para proteger la red del exterior. En estos casos, es imposible establecer las conexiones de los simuladores desde fuera del área local, con lo que el administrador de la red deberá liberar de la protección mencionada a los puertos que se utilizan si se quiere emplear la aplicación desde fuera de dicha red.