Title: COMPUTER SYSTEM FOR MHEG APPLICATIONS

Abstract: A computer system for an MHEG level 6 application (130) comprising MHEG instructions and non-MHEG instructions, the system comprising a Java Virtual Machine (100), having a first memory heap (140), and an MHEG engine (120), the MHEG engine receiving instructions from the MHEG level 6 application (130) for executing those MHEG instructions, and a second Java Virtual Machine (110), having a second memory heap (150), for executing non-MHEG instructions of the MHEG level 6 application, wherein data objects created by the MHEG engine are stored in the first memory heap and data objects created by the execution of non-MHEG instructions are stored in the second memory heap.
COMPUTER SYSTEM FOR MHEG APPLICATIONS

The present invention relates to a computer system for MHEG applications. In particular, the present invention relates to a computer system supporting an MHEG engine and an MHEG-6 application.

Originally, multimedia was only one of many elements of the World Wide Web's design. It was never meant as a support technology for the presentation of technical content or for interaction with the outside world. The advent of technologies such as Java (© Sun Microsystems Corporation) and ActiveX has increased the functionality of multimedia and, in turn, the World Wide Web but these have in turn made access to the World Wide Web a memory and CPU resource hungry activity. In a desire to produce low-cost graphical terminals for accessing the World Wide Web, the MHEG (multimedia and hypermedia experts group) standard was developed to define a basic Graphical User Interface (GUI) and multimedia library for use in kiosk information systems, TV sets, or in video-on-demand servers for use in communicating with set-top boxes.

The MHEG standard was developed to support the distribution of interactive multimedia applications in a client/server architecture across platforms of different types. MHEG level 5 defines a final form representation for applications such that the appearance of an application is common across all platform types. It is the responsibility of the client terminal to have a runtime system that interprets MHEG commands, to present the application to the user and to handle local interaction with the user. MHEG level 6 is an extension to MHEG level 5 that defines a set of mechanisms allowing MHEG applications and Java applications to inter-operate thereby permitting communication with the external environment. MHEG level 6 is mainly intended to support distribution of interactive retrieval (client/server) applications running on limited resource terminals. This extension was needed because previous MHEG levels did not support any data processing
functionality or interaction with an external environment. The Digital Audio-
Visual Council (DAVIC) has adopted MHEG level 6 as the basis for the
specification of set-top units for interactive Television.

MHEG applications are delivered to the user's terminal as a series of
MHEG objects which are interpreted on the user's terminal by an MHEG
engine responsible for turning the MHEG objects into graphical and multimedia
output at the terminal. Each MHEG level is built on the preceding level. An
MHEG engine capable of running an MHEG level 6 application comprises an
MHEG level 5 engine interfaced with a Java Virtual Machine (VM). Typically,
the MHEG level 5 engine and the Java VM used to be written in a
programming language such as C or C++. However, due to the now
widespread availability and use of Java, and also due to the fact that a single
Java application can be run on many different computer platforms, many
MHEG level 6 engines are now implemented in Java so that a single MHEG
ingine can operate on different Java-enabled platforms such as Apple-
Macintosh and Microsoft Windows.

A Java-enabled platform conventionally has a single Java VM running
on the platform which is responsible for running all Java programs requested
of it. Java VM's conventionally run Java programs concurrently. Normally, a
Java VM will only have one memory heap from which all memory objects are
created.

When a conventional Java Virtual Machine running Java application(s)
runs out of memory when trying to meet a processing request, it executes a
complete garbage collection cycle before trying again to meet the request
(synchronous garbage collection). Whilst this is a good thing for the
implementor of Java applications, such as an MHEG level 6 engine, as they do
not have to concern themselves with memory management, it results in
random pauses in the MHEG engine and also in any MHEG application
running on the engine as the garbage collector performs a synchronous
garbage collection cycle during their execution and therefore affects interactive
performance.
If garbage collection occurs only when the developer states, or at the end of an MHEG scene or application, the behaviour of the engine and application would become more predictable - memory used could be predicted more accurately and developers would know exactly (and could choose) when applications would pause due to garbage collection. However many, if not all, implementations of Java VM's do not allow the developer to prevent garbage collection from taking place. Furthermore, many Java VM's perform garbage collection as a synchronous process, concurrent with executing the program, thereby having different CPU loads at differing stages of execution, introducing a further performance variable.

Two problems arise due to the inability of the Java VM to partition its memory heap, forcing multiple applications to use the same address space. Any synchronous garbage collection, either that which is invoked by a call to the Java VM system or when the garbage collection cycle is started due to the Java VM running out of memory, will operate across the entire Java VM memory heap including any memory used by the MHEG engine and associated applications. This will slow the system down and result in pauses in applications being longer than the developer envisaged.

Furthermore, having the MHEG engine and application program share address space makes it very difficult to monitor the engine (and application). There is no way of ensuring the amount of memory specified as available to the MHEG application is actually available. Similarly there is no way to tell what amount and which areas of memory are being used by the MHEG engine and what are being used by the application.

According to an aspect of the present invention, there is provided a computer system for an MHEG level 6 application comprising MHEG instructions and non-MHEG instructions, the system comprising a Java Virtual Machine, having a first memory heap, and an MHEG engine, the MHEG engine receiving instructions from the MHEG level 6 application for executing the MHEG instructions, and a second Java Virtual Machine, having a second memory heap, for executing the non-MHEG instructions of the MHEG level 6
application, wherein data objects created by the MHEG engine are stored in the first memory heap and data objects created by the execution of non-MHEG instructions are stored in the second memory heap.

Preferably, non-MHEG instructions are Java instructions or objects but they may well be other types of instructions capable of being run by a Java VM.

In the system of the present invention the distribution of processes between a number of Virtual Machines separates the memory heaps for the respective processes and permits the separate garbage collection of each processes heap. Each separate garbage collection routine may be tailored to the particular characteristics of the process. Furthermore, an MHEG engine implemented as a process in the system of the present invention is isolated from side effects of other application processes which may be badly-written or written with malicious intent.

Preferably, the computer system comprises a first garbage collection system for reclaiming redundant data objects from the first memory heap and a second garbage collection system for reclaiming redundant data objects from the second memory heap. Preferably, each garbage collection system operates according to a different algorithm. Preferably, the first and second processing systems comprise Java Virtual Machines.

An example of the present invention will now be described, by way of example only, with reference to the accompanying drawings in which:

Figure 1 is a schematic diagram of a conventional system running an MHEG engine and an MHEG level 6 application; and

Figure 2 is a schematic diagram of the system of the present invention for running an MHEG engine and an MHEG level 6 application.

Figure 1 is a schematic diagram of a conventional system comprising a Java Virtual Machine (JVM) 10 running an MHEG engine (ENG) 20 written in Java and processing Java objects from an MHEG level 6 application (APP) 30. Data objects for both the MHEG engine 20 and the MHEG application 30 are
created by the Java VM 10 in a common memory heap (MH) 40. As and when required, garbage collection is performed by the Java VM 10 across the whole heap 40.

Figure 2 is a schematic diagram of the system of the present invention for running an MHEG engine and an MHEG level 6 application. The system comprises a first Java VM 100 and a second Java VM 110. The first Java VM 100 runs the MHEG engine 120 which is written in Java. The engine 120 runs the MHEG level 6 application 130 in cooperation with the second Java VM 110.

The MHEG level 6 application 130 communicates with the MHEG engine 120 for its multimedia, interaction and display functions which appear in the application 130 as MHEG objects and communicates with the second Java VM 110 for executing Java code and interaction with the external environment. Data objects created by the first Java VM in response to requests from the MHEG engine are allocated memory resources from a first heap memory 140. Data objects created by the second Java VM 110 in response to Java code run by the MHEG level 6 application 130 are allocated memory resources from a second heap memory 150. Garbage collection is performed separately on each of the first and second heaps (140, 150) by respective garbage collection processes (GC) 160 and 170. As one example, garbage collection process 160 performs highly aggressive asynchronous garbage collection to reduce pauses in the processing performed by the MHEG engine 120. This reduces pauses in the execution of MHEG objects from the MHEG application 130 and ensuring graceful degradation in performance. Garbage collection process 170, on the other hand performs no asynchronous garbage collection and only collects garbage when explicitly invoked by the Java application developer or when the MHEG application 130 ends.

Whilst the present invention has been described in the context of Java Virtual Machines, it is equally applicable to other programming environments and also to Java Actual Machines. Furthermore, the garbage collection systems discussed could readily be replaced with other systems known in the art.
CLAIMS

1. A computer system for an MHEG level 6 application comprising MHEG instructions and non-MHEG instructions, the system comprising a Java Virtual Machine, having a first memory heap, and an MHEG engine, the MHEG engine receiving instructions from the MHEG level 6 application for executing the MHEG instructions, and a second Java Virtual Machine, having a second memory heap, for executing the non-MHEG instructions of the MHEG level 6 application, wherein data objects created by the MHEG engine are stored in the first memory heap and data objects created by the execution of non-MHEG instructions are stored in the second memory heap.

2. A computer system according to claim 1, further comprising a first garbage collection system for reclaiming redundant data objects from the first memory heap and a second garbage collection system for reclaiming redundant data objects from the second memory heap.

3. A computer system according to claim 2, in which each garbage collection system operates according to a different algorithm.

4. A program storage device readable by a machine and encoding one or more programs of instructions for executing the computer system of a specified one of claims 1 through 4.

5. A computer program for executing the computer system of any one of claims 1 to 3.