Docket No.: 154892800400
(PATENT)

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Ex Parte Reexamination of:
Nedim FRESKO et al.

Control No.: 90/011,492

Reexamination Filing Date: February 15, 2011

Patent No.: 5,966,702

Issue Date: October 12, 1999

For: METHOD AND APPARATUS FOR PRE-PROCESSING AND PACKAGING CLASS FILES

Examiner: M. Steelman

Group Art Unit: 3992

Confirmation No.: 8223

RESPONSE TO FIRST OFFICE ACTION

MS Ex Parte Reexam
Central Reexamination Unit
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Sir:

INTRODUCTORY COMMENTS

In response to a first Office Action issued on June 6, 2011 ("the Action"), for which a response was due on August 6, 2011, and for which an extension of time to September 6, 2011 was requested and granted, Patent Owner provides the following remarks and requests reconsideration.

Listing of the Claims begins on page 2 of this paper.

Examiner Interview Summary begins on page 7.

Listing of the Exhibits begins on page 9.

Remarks begin on page 10 of this paper.

LISTING OF THE CLAIMS

Claim 1 (issued): A method of pre-processing class files comprising:

determining plurality of duplicated elements in a plurality of class files;

forming a shared table comprising said plurality of duplicated elements;

removing said duplicated elements from said plurality of class files to obtain a plurality of reduced class files; and

forming a multi-class file comprising said plurality of reduced class files and said shared table.

Claim 2 (issued): The method of claim 1, further comprising:

computing an individual memory allocation requirement for each of said plurality of reduced class files;

computing a total memory allocation requirement for said plurality of class files from said individual memory allocation requirement of each of said plurality of reduced class files; and

storing said total memory allocation requirement in said multi-class file.

Claim 3 (issued): The method of claim 2, further comprising:

reading said total memory allocation requirement from said multi-class file;

allocating a portion of memory based on said total memory allocation requirement; and

loading said reduced class files and said shared table into said portion of memory.

Claim 4 (issued): The method of claim 3, further comprising:

accessing said shared table in said portion of memory to obtain one or more elements not found in one or more of said reduced class files.

Claim 5 (issued): The method of claim 1, wherein said step of determining a plurality of duplicated elements comprises:

determining one or more constants shared between two or more class files.

Claim 6 (issued): The method of claim 5, wherein said step of forming a shared table comprises:

forming a shared constant table comprising said one or more constants shared between said two or more class files.

Claim 7 (issued): A computer program product comprising:

a computer usable medium having computer readable program code embodied therein for pre-processing class files, said computer program product comprising:

computer readable program code configured to cause a computer to determine a plurality of duplicated elements in a plurality of class files;

computer readable program code configured to cause a computer to form a shared table comprising said plurality of duplicated elements;

computer readable program code configured to cause a computer to remove said duplicated elements from said plurality of class files to obtain a plurality of reduced class files; and

computer readable program code configured to cause a computer to form a multi-class file comprising said plurality of reduced class files and said shared table.

Claim 8 (issued): The computer program product of claim 7, further comprising:

computer readable program code configured to cause a computer to compute an individual memory allocation requirement of each of said plurality of reduced class files;

computer readable program code configured to cause a computer to compute a total memory allocation requirement of said plurality of class files from said individual memory allocation requirement of each of said plurality of reduced class files; and

computer readable program code configured to cause a computer to store said total memory allocation requirement in said multi-class file.

Claim 9 (issued): The computer program product of claim 8, further comprising:

computer readable program code configured to cause a computer to read said total memory

allocation requirement from said multi-class file;

computer readable program code configured to cause a computer to allocate a portion of memory based on said total memory allocation requirement; and

computer readable program code configured to cause a computer to load said reduced class files and said shared table into said portion of memory.

Claim 10 (issued): The computer program product of claim 9, further comprising:

computer readable program code configured to cause a computer to access said shared table in said portion of memory to obtain one or more elements not found in one or more of said reduced class files.

Claim 11 (issued): The computer program product of claim 7, wherein said computer readable program code configured to cause a computer to determine said plurality of duplicated elements comprises:

computer readable program code configured to cause a computer to determine one or more constants shared between two or more class files.

Claim 12 (issued): The computer program product of claim 11, wherein said computer readable program code configured to cause a computer to form said shared table comprises:

computer readable program code configured to cause a computer to form a shared constant table comprising said one or more constants shared between said two or more class files.

Claim 13 (issued): An apparatus comprising:

a processor;

a memory coupled to said processor;

a plurality of class files stored in said memory;

a process executing on said processor, said process configured to form a multi-class file comprising:

a plurality of reduced class files obtained from said plurality of class files by removing one

or more elements that are duplicated between two or more of said plurality of class files; and

a shared table comprising said duplicated elements.

Claim 14 (issued): The apparatus of claim 13, wherein said multi-class file further comprises a memory requirement, said memory requirement being computed by said process.

Claim 15 (issued): The apparatus of claim 13, wherein said duplicated elements comprise elements of constant pools of respective class files, said shared table comprising a shared constant pool.

Claim 16 (issued): The apparatus of claim 13, further comprising:

a virtual machine having a class loader and a runtime data area, said class loader configured to obtain and load said multi-class file into said runtime data area.

Claim 17 (issued): The apparatus of claim 16, wherein said class loader is configured to allocate a portion of said runtime data area based on said memory requirement in said multi-class file.

Claim 18 (issued): The apparatus of claim 17, wherein said class loader is configured to load said plurality of reduced class files and said shared table into said portion of said runtime data area.

Claim 19 (issued): The apparatus of claim 16, wherein said virtual machine is configured to access said shared table when a desired element associated with a first class file is not present in a corresponding one of said plurality of reduced class files.

Claim 20 (issued): A memory configured to store data for access by a virtual machine executing in a computer system, comprising:

a data structure stored in said memory, said data structure comprising:

a plurality of reduced class files associated with a plurality of corresponding classes, said plurality of reduced class files configured to be loaded by the virtual machine for execution of said plurality of classes;

a shared table comprising one or more elements that are duplicated between two or more of said plurality of classes, said shared table configured to be loaded into the virtual machine to be accessed for said duplicated elements; and

a memory requirement value configured to be read by a class loader of the virtual machine to allocate a portion of a runtime data area for loading said plurality of reduced class files and said shared table.

Claim 21 (issued): The memory of claim 20, wherein said duplicated elements are removed from said plurality of reduced class files.

Claim 22 (issued): The memory of claim 20, wherein said duplicated elements comprise constants and said shared table comprises a shared constant pool.

Claim 23 (issued): The memory of claim 20, wherein said memory requirement value is computed from individual memory requirements of said plurality of reduced class files and a memory requirement of said shared table.

Interview Summary Pursuant to 37 C.F.R. 1.560(b)

Patent Owner thanks the Examiners for the courtesy of an in-person interview to discuss the Action on August 4, 2011. In attendance for the interview were Examiners Mary Steelman, Eric Kiss, and Fred Ferris, and for the Patent Owners, Christopher Eide (48,375), Julie Akhter (59,570), George Simion (47,089), Benjamin Goldberg (technical expert), Tracy Druce (35,493), and Lissi Mojica (63,421). Claims 1, 7, 13, and 16 and the Tock and Palay references were discussed during the interview. The following recitations of claim 1 were primarily discussed: "plurality of class files," "plurality of reduced class files," and "multi-class file." Additionally, claim 16 and its recitation of a "class loader" configured to load classes into a virtual machine were discussed.

Patent Owner presented that Palay discloses "object files," not "class files," where "class files" comprise machine-independent bytecode and "object files" comprise machine code. (*See also*, the Examiner Interview Agenda accompanying the Examiner's Interview summary mailed August 4, 2011.)

Patent Owner further presented that Tock fails to disclose the recited "multi-class file," which is clearly defined in the '702 Patent as a machine-independent file or package comprising reduced class files. ('702 Patent, 1:29-31, 2:66-3:1, 4:63-5:28, and 10:25-51.) Instead, Tock discloses producing an "executable module," which is a runtime representation (i.e., a memory representation of a loaded class) of the classes for a particular client device, and is not in a machine-independent format. (Tock, 1:41-44, 5:38-50, and 7:36-8:13.) (*See also*, the Examiner Interview Agenda.)

In response to the Patent Owner's presentation, the Examiners inquired regarding the requirement that "class files" include bytecode. Patent Owner directed the Examiners to the clear definition provided in the specification that class files comprise bytecode, and further that bytecode is machine-independent (in particular, pointing to the '702 Patent specification that states, "[c]lass files contain bytecode instructions," and that bytecode instructions include code and data in a machine-independent format ('702 Patent, 1:34, 2:62-3:1). The proper scope of the claims will be discussed in detail in the response to the Office Action.

The Examiners appeared to appreciate the distinctions between the claimed invention and the cited references as presented by the Patent Owner.

LISTING OF EXHIBITS

The following Exhibits are submitted herewith.

Exhibit	Description
A	Declaration of Prof. Benjamin Goldberg ("Goldberg Declaration")
B	Curriculum Vitae of Prof. Benjamin Goldberg

REMARKS

I. Introduction

Claims 1, 5-7, 11-13, 15, and 16 are pending in the present reexamination of U.S. Patent No. 5,966,702 ("the '702 Patent"). The '702 Patent is directed to a method, computer program product, and apparatus for pre-processing and packaging a plurality of class files and forming a multi-class file. This is achieved through the following unique combination of features as set forth in exemplary claim 1:

- **"determining a plurality of duplicated elements in a plurality of class files"**[1]
- **"forming a shared table comprising said plurality of duplicated elements"**
- **"removing said duplicated elements from said plurality of class files to obtain a plurality of reduced class files"** and
- **"forming a multi-class file comprising said plurality of reduced class files and said shared table."**

The Office, during the original examination, found this combination of features—specifically **"forming a multi-class file comprising said plurality of reduced class files and said shared table"**—to be patentable over the prior art. (Office Action dated Jan. 29, 1999, page 3.)

The art of the present reexamination fails to compel a different conclusion. Neither of the cited references—Tock nor Palay—teaches or suggests the above claimed combination of features. In fact, and as stated by Professor Goldberg in his declaration, "the cited references fail to disclose an important feature of the claimed invention of the '702 Patent, overlooked by the original request, namely that the **'reduced class files'** obtained by, and the **'multi-class file'** resulting from the claimed method and apparatus are—like Java class files or other class files containing bytecode—machine-independent, allowing them to be downloaded from a server to a variety of hardware systems." (Goldberg Decl., ¶ 13.) For example, the '702 Patent states, "[t]o accommodate the

[1] Throughout this Response, recited claim language is in quotes and boldface.

variety of hardware systems used by clients, applications or applets are distributed in a platform- or machine-independent format such as the Java® class file format. Object-oriented applications are formed from multiple class files that are accessed from servers and downloaded individually as needed" ('702 Patent, 1:29-35); and "[w]hen a virtual machine wishes to load the classes in the multi-class file, the location of the multi-class file is determined and the multi-class file is downloaded from a server, if needed." ('702 Patent, 5:18-22.) Furthermore, the '702 Patent is clear that the multi-class files of the claimed invention serve as a "single package for efficient storage" of a plurality of class files,[2] addressing the storage inefficiencies of JAR files containing (machine-independent) class files. ('702 Patent, 4:44-5:5.)

Tock's disclosure is quite different than the claimed features of the '702 Patent. "Tock generally discloses a method and system for statically loading classes in read-only memory (ROM) of a client device. Tock's system became known as 'ROMizer.'" (Goldberg Decl., ¶ 16.) Tock describes that the goal of the disclosed offline class loader is to determine which methods and variables associated with each class can be stored in ROM and which must be stored in a random access memory (RAM) device. (Tock, 1:41-67.) According to Professor Goldberg's declaration "[t]he package output by the class loader of Tock is a machine-dependent, 'executable module,' and is formatted as 'data definitions, where each definition specifies a bytecode and an offset indicating a memory location'—that is, the output of the class loader is a runtime representation (i.e., a memory representation of a loaded class) of the classes for a particular client device, and is no longer in a machine-independent format. (*Id.* at 1:41-44, 5:38-50, and 7:36-8:13.)" (Goldberg Decl., ¶ 17.) "Thus, Tock's 'executable module' contains a machine-dependent (i.e., machine-specific) memory representation of the class information and is not a **'multi-class file'** as recited." (Goldberg Decl., ¶ 17.)

[2] The '702 Patent makes clear that "[c]lass files contain bytecode instructions," where bytecode instructions include code and data in a machine-independent format. ('702 Patent, 1:34, 2:62-3:1.)

Palay's disclosure is also very different than the claimed features of the '702 Patent. Palay describes systems and methods for compiling C++ source code files into object files and linking the object files into an executable file, in a manner that allows the linker to resolve class definitions and perform class relocation when creating the executable file. (Palay, Abstract, 6:1-27.) According to Prof. Goldberg's declaration, "[t]he '702 Patent makes clear that '[c]lass files contain bytecode instructions,' and further that such bytecode instructions of the **'class files'** are machine-independent. ('702 Patent, 1:29-38; 2:62-3:15.)" (Goldberg Decl., ¶ 30.) Further, Palay fails to disclose class files (or an equivalent thereto) as object files do not contain bytecode instructions or a machine-independent format. "In fact, the very nature of object files—being compiled from source code (e.g., C++ source code) into machine code—is that they are machine-dependent." (Goldberg Decl., ¶ 31.) Therefore, object files are not, and do not disclose, **"class files"** as recited. Moreover, the described manipulation of the object files in Palay fails to disclose a **"multi-class file"** as recited. As such, basic features of the claims are deficient in Palay.

Accordingly, Patent Owner submits that all of the rejections should be withdrawn.

II. Brief Summary of the Invention

As stated in Professor Goldberg's declaration, "[A]n important feature of the claimed invention of the '702 Patent, overlooked by the original Request, [is] that the **'reduced class files'** obtained by, and the **'multi-class file'** resulting from the claimed method and apparatus are—like Java class files and other class files containing bytecode—machine-independent, allowing them to be downloaded from a server to a variety of hardware systems." (Goldberg Decl., ¶ 13.) For example, the '702 Patent states, "[t]o accommodate the variety of hardware systems used by clients, applications or applets are distributed in a machine-independent format such as the Java® class file format. Object-oriented applications are formed from multiple class files that are accessed from servers and downloaded individually as needed" ('702 Patent, 1:29-35); and "[w]hen a virtual machine wishes to load the classes in the multi-class file, the location of the multi-class file is

determined and the multi-class file is downloaded from a server, if needed." ('702 Patent, 5:18-22.) Furthermore, the '702 Patent is clear that the **"multi-class file"** of the claimed invention serves as a "single package for efficient storage," addressing the storage inefficiencies of JAR files containing (machine-independent) class files. ('702 Patent, 4:44-5:5.) "Because a multi-class file as used in the '702 Patent is in a machine-independent format, the **'reduced class files'** it contains are also machine-independent, just like the **'class files'** from which they were obtained." (Goldberg Decl., ¶ 13.) The '702 Patent makes "clear that the **'multi-class file'** and the plurality of reduced **'class files'** it contains are to be loaded by a virtual machine, further indicating that they *must* be machine-independent (i.e., the reduced class files and multi-class file contain machine-independent bytecode). ('702 Patent, 1:29-31, 2:66-3:1, 4:63-5:28, and 10:25-51.)" (Goldberg Decl., ¶ 13.)

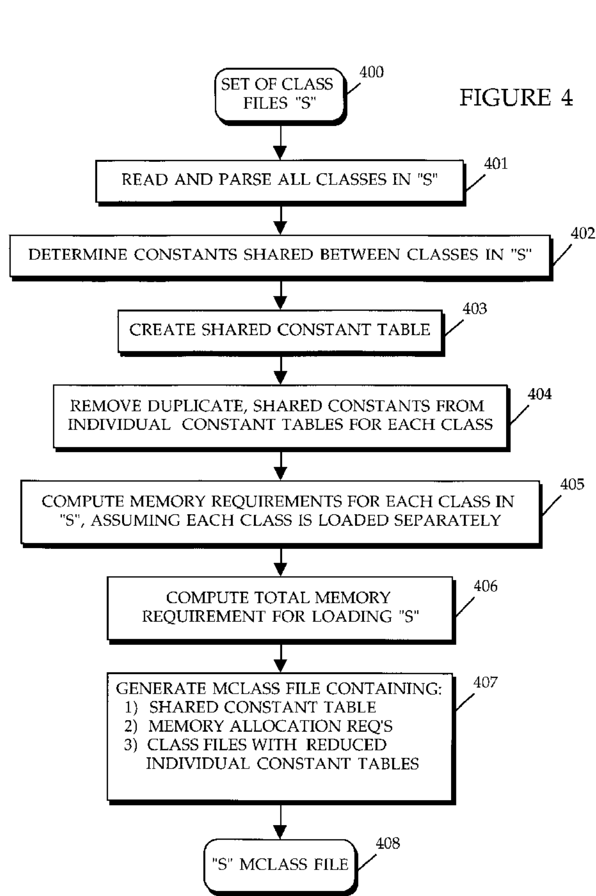
Prior to the filing of the '702 Patent, an issue with the class file format and the class loading process was that class files often contained duplicated data. ('702 Patent, 1:38-48.) The storage, transfer, and processing of the individual class files were generally inefficient due to the redundancy of the information. Also, an application may contain many class files, all of which are loaded and processed in separate transactions. This slowed down the application and degraded memory allocator performance. Further, a client was typically required to maintain a physical connection to the server for the duration of the application transfer process in order to access class files on demand. (*Id.*)

A Java archive (JAR) format had been developed to group class files together in a single transportable package known as a JAR file. (*Id.* at 4:45-60.) Broadly speaking, JAR files encapsulate Java classes in archived, compressed format. A JAR file can be identified in an HTML document within an applet tag, and when a browser application reads the HTML document and finds the applet tag, the JAR file is downloaded to the client computer and decompressed. Thus, a group of class files may be downloaded from a server to a client in one download transaction. After downloading and decompressing, the archived class files are available on the client system for

individual loading as needed in accordance with standard class loading procedures. The archived class files remain subject to storage inefficiencies, however, due to duplicated data between files, as well as memory fragmentation due to the performance of separate memory allocations for each class file. (*Id.*)

A breakthrough of the '702 Patent included a new approach for pre-processing and packaging class files in a manner to reduce memory requirements. The approach employed a **"multi-class file,"** which is a machine-independent package containing a set of reduced class files and related items from a set of class files. ('702 Patent, 1:29-31, 2:66-3:1, 4:63-5:28, and 10:25-51.) (Goldberg Decl., 13.) The general idea of the **"multi-class file"** is to eliminate code and data duplicated among a plurality of class files (e.g., the plurality of class files associated with a particular application). Also, only one copy of duplicated code and data is placed in the **"multi-class file"** along with the non-duplicated code and data. (*Id.* at Abstract, 5:6-17.) An aspect of the claimed invention broadly includes forming a shared table comprising duplicated elements of the class files, obtaining reduced class files (what remains of the class files with the duplicated elements removed), and forming a multi-class file containing the reduced class files and the shared table of duplicated elements. This breakthrough is expressly reflected in the claims, for example, in claim 1 (and similarly in other independent claims) as **"determining . . . duplicated elements in a plurality of class files," "forming a shared table"** of the duplicated elements, **"removing said duplicated elements from said plurality of class files to obtain a plurality of reduced class files,"** and **"forming a multi-class file comprising said plurality of reduced class files and said shared table."**

An exemplary method for pre-processing and packaging a set of class files into a multi-class file is illustrated in the flow diagram of Figure 4 (shown below) and described at least in column 9, lines 9-65. Additionally, an exemplary embodiment of the multi-class file format is depicted in Figure 5 (shown below) and described at least in column 9, lines 66-67.



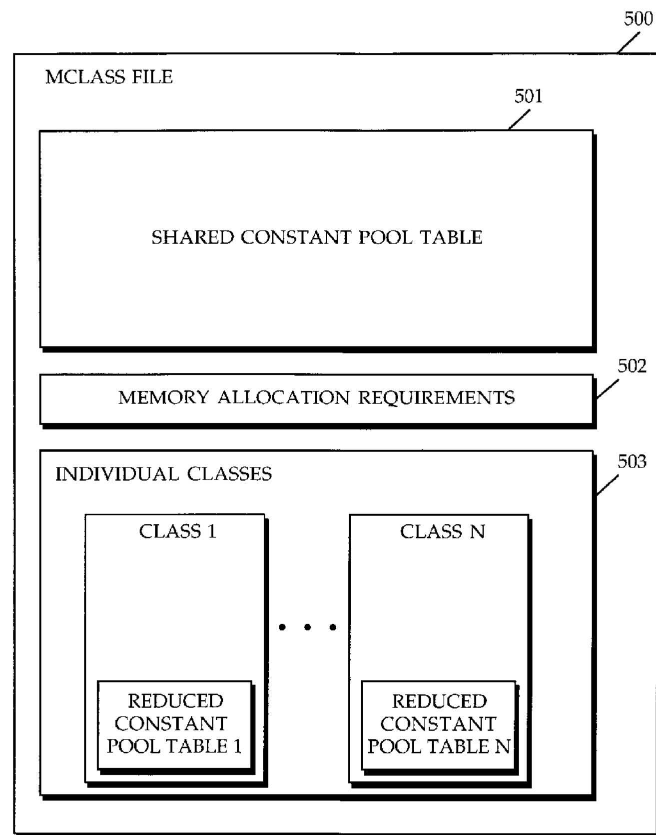


FIGURE 5

The exemplary method for forming a multi-class file begins at 400 with a set of class files "S" (typically part of one application), where the class files contain machine-independent bytecode. At 401, the pre-processor reads and parses each class in "S." At 402, the pre-processor examines the constant pool tables of each class to determine the set of class file constants (e.g., strings and numerics, as well as others specific to the class file format) that can be shared between classes in "S." A shared constant pool table is created at 403, with all duplicate constants determined from 402. At 404, the pre-processor removes the duplicate, shared constants from the individual constant pool tables of each class. At 405, the pre-processor may compute the in-core memory requirements of each class in "S," as would normally be determined by the class loader for the given virtual machine. This is the amount of memory the virtual machine would allocate for each class, if the

virtual machine were to load each class separately. After considering all classes in "S," and the additional memory requirement for the shared constant pool table, the total memory requirement for loading "S" is computed at 406. At 407, the pre-processor produces a **"multi-class file"** (mclass) file that contains the shared constant pool table created at 403, information about memory allocation requirements determined at 405 and 406, and all classes in "S," with their respective reduced constant pool tables. The mclass file for the class set "S" is output at 408.

Figure 5 is a simplified block diagram of an embodiment of the multi-class file format output by the method of Figure 4. Mclass file 500 comprises shared constant pool table 501, memory allocation requirements 502, and the set of individual classes 503. The set of individual classes 503 comprises the class file structures for classes 1 through N (N being the number of classes in the set), along with the corresponding reduced constant pool tables 1 through N. The size of the shared constant pool table 501 is dependent on the number of duplicate constants found in the set of classes. The memory allocation requirements 502 may be represented as a single value indicating the total memory needed to load all class structures (classes 1 through N) in individual classes 503, as well as the shared constant pool table 501.

The **"multi-class file"** is typically considerably smaller than the sum of the sizes of the individual class files from which it was derived because the classes therein share information. It can be loaded by the virtual machine during or prior to the execution of an application, instead of having to load each contained class on demand. The virtual machine is also able to take advantage of the allocation requirements information to pre-allocate all required memory for the multi-class set. This solves many of the problems associated with class loading. (*Id.* at 10: 16-24.)

The smaller size of the **"multi-class file"** (relative to the plurality of class files before processing) results in the classes taking up less space on servers or storage devices, less network or file transfer time to read, and less memory when loaded, as well as faster execution (in part, because shared constants are resolved at most once). (*Id.* at 10: 25-35.)

As further described in the ’702 Patent (e.g., at column 10, lines 36-51), a **“multi-class file”** further consolidates the loading of required classes instead of loading the classes one by one. Using allocation information, only one dynamic memory allocation is needed instead of multiple allocation operations. This results in less fragmentation, less time spent in the allocator, and less waste of memory space. Because the class files are consolidated in a single **“multi-class file,”** only a single transaction is typically needed to perform a network or file system search, to set up a transfer session (e.g., HTTP), and to transfer the entire set of classes. This minimizes pauses in the execution that can result from such transactions and provides for deterministic execution, with no pauses for class loading during a program run. Also, once the multi-class file is loaded and parsed, there is no need for the computer executing the program to remain connected to the source of the classes.

Accordingly, the approach employed by the ’702 Patent includes determining a plurality of duplicated elements in a plurality of **“class files”** (which contain machine-independent bytecode) and forming a **“multi-class file,”** which is a package containing a set of reduced class files and other related items in a machine-independent format similar to the class files from which the package was formed. Neither of the cited references discloses or reasonably suggests such features.

III. Claims 1, 5-7, 11-13, 15, and 16 Are Not Anticipated by Tock (Ground 1)

A. Tock fails to disclose or suggest a “multi-class file,” a machine-independent package of processed “class files.”

Tock generally discloses a method and system for statically loading classes in read-only memory (ROM) of a client device. Tock’s system became known as ‘ROMizer’.” (Goldberg Decl., ¶ 16.) Tock teaches that the goal of the offline class loader is to determine which methods and variables associated with each class can be stored in a ROM and which must be stored in a random

access memory (RAM) device. (Tock, at 1:8-21.) Methods that invoke Java interfaces or utilize non-static instance variables need to reside in random access memory. This is because the bytecodes that implement interfaces are determined at runtime and non-static instance variables are altered for each instantiation of the associated class. The offline class loader finds these methods and variables and flags them by inserting a special indicator that specifies that they are to be loaded in a RAM device. "The offline class loader performs a number of optimizations in order to produce a more compact representation of the executable code." (Goldberg Decl., ¶ 18.) For example, "the constant pool that is associated with each class is combined for all the classes residing in the application. In addition, the offline class loader performs additional processing to tailor the class files that were originally structured for dynamic loading for a preloaded class environment." (Goldberg Decl., ¶ 18.)

Further, "Tock discloses that the offline class loader performs two other transformations to the class files to output the executable module that renders the offline class loader's output machine-<u>dependent</u>). (Tock, Fig. 3; 6:37-44.)" (Goldberg Decl., ¶ 19.) First, a "static initializer is created, which performs class initialization for the classes that are preloaded." (Tock, 6:37-41.) Second, "the offline class loader performs bytecode quickening of the bytecodes, which renders them machine-<u>dependent</u>: 'bytecodes using a non-quick instruction that symbolically reference methods are recoded in a quick instruction format that references the methods directly.' (Tock, 6:41-44.)" (Goldberg Decl., ¶ 19.) According to Prof. Goldberg's declaration, "[t]he altered bytecodes that result from these steps are no longer the bytecodes from the class files after duplicate removal. As such, the output of Tock, a runtime representation of processed class files for a client device, is not, and does not include, a file or other data structure that includes machine-independent bytecode." (Goldberg Decl., ¶ 19.) "In contrast to Tock, the '702 Patent makes clear that the structure of the recited **'class file'** must contain machine-independent bytecode. ('702 Patent at least at 4:63-5:28.)" (Goldberg Decl., ¶ 19.)

B. Tock fails to disclose or suggest "a multi-class file" as recited by claims 1, 5-7, 11-13, 15, and 16.

Claim 1 recites,[3] *inter alia*:

> **"forming a multi-class file comprising said plurality of reduced class files and said shared table."**

As described above and by Professor Goldberg's declaration, "a **'multi-class file,'** read in light of the specification of the '702 Patent, is a machine-independent package that includes a set of reduced class files and a shared table of duplicated elements determined from the set of reduced class files. ('702 Patent, 5:6-17.)" (Goldberg Decl., ¶ 14.) Additionally, "the '702 Patent makes clear that the recited **'class files'** and **'multi-class file'** must contain machine-independent bytecode. (*Id.* at 1:29-35, 2:66-3:1, 4:63-5:28, and 10:25-51)." (Goldberg Decl., ¶ 14.) Further, the **"multi-class file"** consolidates the loading of a plurality of classes associated with an application, for example, and provides for more efficient storage, transfer, and processing of the plurality of machine-independent class files. (*Id.* at 1:29-35 and 4:63-5:5.) As such, and according to Professor Goldberg's declaration, "one of ordinary skill in the art would *necessarily* read the recited 'multi-class file' as requiring a machine-independent package." (Emphasis added; Goldberg Decl., ¶ 14.)

The Office Action states "Tock discloses (FIG. 8B; 10: 29-32: 38-50) a multi-class file comprising said plurality of reduced class files." (Action, page 6). The Office appears to primarily rely on Tock's disclosure of an offline class loader that can output a constant pool file and updated class files as "a single file" for disclosing the recited **"multi-class file."** (Action, page 6.) As described below, however, the output of Tock's offline class loader, i.e., the "single file," is an "'executable module' [that] contains a machine-dependent (i.e., machine-specific) memory representation of the class information and is not a **'multi-class file'** as required by the claims." (Goldberg Decl., ¶ 17.)

[3] Throughout this Response, Patent Owner refers to independent claim 1. The analysis as to claim 1 applies to the other independent claims (claims 7 and 13) unless otherwise noted.

In particular, Patent Owner submits that Tock is quite different than the claimed invention of the ’702 Patent. Tock discloses a method and system for statically loading classes in read-only memory (ROM) of a client device. As described above, and according to Prof. Goldberg’s declaration, “[t]he output of Tock, a runtime representation of processed class files for a client device, is not, and does not include, a file or other data structure that includes machine-independent bytecode.” (Goldberg Decl., ¶ 19.) More specifically, Professor Goldberg states “I conclude that the Requester and Office have overlooked that the package output by the offline class loader of Tock is formatted as ‘data definitions, where each definition specifies a bytecode and an offset indicating a memory location’—that is, the output of the class loader is a runtime representation (i.e., a memory representation of a loaded class) of the classes for a particular client device, and is no longer in a device independent format.” (Goldberg Decl. 17.) (Tock, 5:38-50 and 7:36-8:13.) Further to this point, Tock states that the offline class loader “is used to produce an executable module whose classes are preloaded into memory without requiring runtime dynamic loading.” (Tock, 1:41-44) “Tock’s ‘executable module’ contains a machine-dependent (i.e., machine-specific) memory representation of the class information and is not a **‘multi-class file’** as required by the claims.” (Goldberg Decl., ¶ 17.)

Tock further discloses that the offline class loader performs two other transformations that render its output machine-dependent. (Tock, Fig. 3; 6:37-44.) (Goldberg Decl., ¶ 19.) First, a “static initializer is created, which performs class initialization for the classes that are preloaded.” (Tock, 37-41.) Second, the offline class loader performs bytecode quickening of the bytecodes, which renders them machine-dependent: “bytecodes using a non-quick instruction that symbolically reference methods are recoded in a quick instruction format that references the methods directly.” (Tock, 41-44.) “The altered bytecodes that result from these steps are no longer the bytecodes from the class files after duplicate removal.” (Goldberg Decl., ¶ 19.) “As such, the output, a runtime representation of processed class files for a client device, is not, and does not include, a file or other

data structure that includes machine-independent bytecode, and therefore would not have been considered a **'multi-class file'**" as required by the claims of the '702 Patent. (Goldberg Decl., ¶ 19.)

The Office further cites to FIG. 8B and column 10, lines 29-32 in support of the rejection. (Action, page 6.) These portions, however, do not cure the deficiency of Tock discussed above. For example, "FIG 8B and column 10, lines 29-32 discloses that the universal constant pool file and updated class files are output by the class loader, and include an indicator specifying the memory storage requirements and a boot time initiator. This description fails to disclose or suggest that the output is a **'multi-class'** file as recited; rather, it is further made clear that the output is machine-dependent because it includes a boot time initiator for a particular client device." (Goldberg Decl., ¶ 20.)

Patent Owner also submits that no intermediate form of the processed class files, e.g., the updated class files, in Tock satisfies the recited **"multi-class file."** For example, as indicated in Fig. 3 of Tock, class files 128 are processed by offline class loader 132, which outputs the updated class file and constant pool 302. (Tock, 5:8-50.) In addition to formatting the class files as a memory representation of loaded classes as discussed above, "Tock further indicates at column 6, lines 37-44 that the offline class loader 132 itself performs the transformation of bytecode quickening, rendering the updated class files machine-dependent." (Goldberg Decl., ¶ 21.) "As such, there is no intermediate updated class, having duplicate entries removed, that can be considered in a machine-independent format." (Goldberg Decl., ¶ 21.)

Accordingly, for at least these reasons, Tock fails to disclose or suggest the concept of a **"multi-class file"** or package as required by independent claims 1, 7, and 13 at least because the output of the offline class loader does not include a **"multi-class file."**

Furthermore, Tock clearly teaches away from a machine-independent multi-class file (teaching a machine-dependent "executable module" for a client device), and necessary modifications of Tock to meet the features of claim 1 would impermissibly modify the principle of

operation of Tock (e.g., modifying the output of the offline class loader, which includes a machine-dependent runtime representation of the loaded classes, to include a machine-independent multi-class file) and therefore cannot render the claims obvious. (MPEP §§ 2143.01, 2143.01(VI).)

For at least these reasons, Tock does not teach or suggest **"a multi-class file"** as recited in independent claims 1, 7, and 13. Accordingly, the rejection to claims 1, 5-7, 11-13, 15, and 16 should be withdrawn.

C. Tock fails to disclose or suggest the features of dependent claim 16.

Claim 16, which depends from claim 13, also cannot be anticipated by Tock. Claim 16 adds additional features not found in Tock: **"a virtual machine having a class loader configured to obtain and load said multi-class file into said runtime data area."** In particular, Tock fails to disclose a class loader configured as recited by claim 16 **"to obtain and load said multi-class file into said runtime data area."** (Goldberg Decl., ¶ 23.)

The Office Action states "Tock discloses (FIG. 12; 3: 46-52; 10: 29-43) a virtual machine having a class loader and a runtime data area . . . . Tock describes a boot time initiator that loads a multi-class file into the runtime data area." (Action, page 9.)

Initially, Patent Owner submits that Tock fails to disclose a **"virtual machine"** or a **"class loader"** as recited. As stated in Professor Goldberg's declaration: "Portions of Tock cited in the Office Action at page 9 (e.g., relating to the offline class loader and the Java browser [i.e., 3:46-52 and 10:29-43]), do not disclose a class loader as recited. For instance, Tock discloses a Java browser and a class loader associated therewith, but the class loader is described (e.g., at 4:36-39) for loading classes to a user's address space. This discloses neither a '**virtual machine'** nor a **'class loader'** configured as recited in claim 16." (Goldberg Decl., ¶ 24.) Moreover, as stated previously, Tock does not disclose a machine-independent **"multi-class file,"** and as such, this disclosure fails

to disclose **"a virtual machine having a class loader configured to obtain and load said multi-class file into said runtime data area"** as recited.

Additionally, and according to Professor Goldberg's declaration, "the Java interpreter mentioned at column 3, lines 46-52 of the Action, is not described as receiving the output of the offline class loader. Instead it is described as used for executing a Java application, and therefore does not disclose or suggest the **'virtual machine'** as recited." (Goldberg Decl., ¶ 25.)

The Office Action and Requestor further cite to Figure 12 of Tock for these features of claim 16. However, "Tock's Figure 12 shows the contents of the read-only memory of a client computer, and no class loader is shown." (Goldberg Decl., ¶ 26.) Figure 12 illustrates a boot time initiator 1220, which copies over data from ROM to RAM when the device is powered up. There is no disclosure, however, that boot time initiator 1220 is a class loader configured to obtain a multi-class file and load it into the runtime data area. In fact, Tock specifically discloses that "the boot time initiator copies all methods and data that must be resident in random access memory during execution to the random access memory locations assigned to them by the linker" (Tock, 10:37-41), where "[t]he output from the linker 136 is a preloadable executable module 306 containing the methods and data" (Tock, 5:65-67). At an earlier step in Tock, the output of the offline class loader (i.e., the updated class files) was "transmitted to an assembler 134 which produces an object module having the required format for the linker . . . . Once the memory layout is determined, the linker 136 resolves all symbolic references and replaces them with direct addresses." (Tock, 5:51-60.) "That is, the updated class files are assembled into object modules (i.e., machine code modules) and linked by the linker into a preloadable executable module (again, comprising machine-dependent code having no symbolic references)." (Goldberg Decl., ¶ 26.) Subsequently, "the preloadable executable module and boot time initiator 1220 are permanently stored in the read-only memory of a client computer." (Tock, 10:33-35.) "Thus, the boot time initiator in Tock does not load the executable module with a class loader; it merely copies the code ("methods") that is already in (loaded)

machine code format, not byte code format, into 'random access memory locations.'" (Goldberg Decl., ¶ 26.) Accordingly, "Tock's offline class loader is a 'preloader' that produces a 'preloaded' output file containing classes that do not require runtime dynamic loading." (Goldberg Decl., ¶ 26.)

For at least these reasons, Tock fails to disclose a class loader configured to load a multi-class file into a virtual machine as required by claim 16.

D. Tock fails to disclose or suggest the features of dependent claims 5, 6, 11, 12, and 15.

Tock further fails to disclose or suggest the features of claims 5, 6, 11, 12, and 15 at least because Palay fails to disclose the features of independent claims 1, 7, and 13, from which claims 5, 6, 11, 12, and 15 depend respectively. Accordingly, the rejection to claims 5, 6, 11, 12, and 15 should be withdrawn.

IV. Claims 1, 5-7, 11-13, and 15 Are Not Anticipated by Palay (Ground 2)

A. Palay fails to disclose or suggest "class files," comprising bytecode.

Palay generally describes systems and methods for compiling and linking source files that include methods for merging class information from object files. (Palay, Abstract, 6:1-27.) For example, a compiler generates class information pertaining to object-oriented classes referenced in the source file, where the class information is sufficient to enable a linker to resolve class definitions and to perform class relocation operations. The compiler also generates an object file from the source file, where the object file includes the class information. The compiler generates the object file such that resolution of class definitions and performance of class relocation operations are delayed until operation of the linker. A linker links the object file potentially with at least one other object file or shared library to thereby generate an executable file or shared library. The linker uses

the class information contained in the object file to resolve class definitions and to perform class relocation operations. (*Id.*)

Palay describes operation of the linker as follows:

> In step 606, the linker 112 *reads in the object files* 106, 108 and the shared libraries 110 and merges together the class information 406 contained in these files. In particular, the linker 112 merges together the class symbol tables 408 contained in these files to generate a merged class symbol table. Similarly, the linker 112 merges together the class relocation tables 412, class instance tables 416, class reference tables 410, class definition tables 414, and class information variable tables 418 to generate a merged class relocation table, a merged class instance table, a merged class reference table, a merged class definition table, and a merged class information variable table.
>
> Two issues that must be addressed when doing this merge operation are what to do about duplicate members and the mapping of references from the relocations to the merged class instance table and the merged class symbol table. *When merging the class definition tables 414, duplicate non-dynamic and internal dynamic class definitions are removed. Duplicate definitions of dynamic classes are considered to be an error. When merging class instance tables 416, duplicate entries are considered to be an error. Duplicate entries in the class symbol tables 408 are removed.* There are no duplicate entries in the class relocation tables 412.

(Col. 28, lines 39-61.) (Emphasis added.)

As such, Palay generally describes compilers and linkers for C++ that operate on object files to delete duplicate class definitions (e.g., "duplicate non-dynamic and internal dynamic class definitions are removed"), where other items that are detected as duplicates are "considered to be an error" (e.g., "[d]uplicate definitions of dynamic classes are considered to be an error"). (*See*, e.g., 28:51-61.)

B. Palay fails to disclose or suggest "a plurality of class files," let alone "determining [a] plurality of duplicated elements in a plurality of class files," or "removing said duplicated elements from said plurality of class files to obtain a plurality of reduced class files" as recited by claim 1 (and similarly 7 and 13).

Claim 1 recites, *inter alia*:

"determining [a] plurality of duplicated elements in a plurality of class files." The '702 Patent specifically states that "[c]lass files contain bytecode instructions." ('702 Patent, 1:34.) The '702 Patent further states that Java classes are complied into machine-independent bytecode class files, where each class contains code and data in a machine-independent format called the class file format. (*Id.* at 2:62-3:1.) As such, and according to Professor Goldberg's declaration, "one of ordinary skill in the art would necessarily read the recited 'class files' as requiring machine-independent bytecode." (Goldberg Decl., ¶ 30.)

The Office Action, in adopting the Requester's analysis, states "Palay discloses (FIG. 6; 28: 39-61) determining a plurality of duplicated elements in a plurality of class files . . . ." (Action, page 10.) As described in greater detail below, Patent Owner submits that Palay discloses methods and systems relating to "object files," which are not equivalent to or the same as **"class files."** (Goldberg Decl., ¶ 31.)

In contrast to **"class files,"** which comprise bytecode instructions, Palay describes a compiler and linker for C++ that operate with object files; Palay's object files are not **"class files."** In particular, "one of ordinary skill in the art, reading the specification and the definition of the class file format found there, would necessarily conclude that **'class files'** include machine-independent bytecodes, and further recognize that Palay's C++ object files include machine-<u>dependent</u> executable code." (Goldberg Decl., ¶ 32.) "Palay's disclosure of merging class information found in object files (see, e.g., '702 Patent, 28:39-61; Fig. 6), where 'object files 106' are what is being read in and having duplicated entries removed therefrom, fails to disclose **'class files'** that comprise bytecode instructions." (Goldberg Decl., ¶ 31.) "In fact, the very nature of object files—being compiled from source code (e.g., C++ source code) into machine code—is that they are machine-<u>dependent</u>." (Goldberg Decl., ¶ 31.) Therefore, object files are not, and do not disclose, **"class files"** as recited by independent claims 1, 7, and 13. Accordingly, this basic feature of the claims is

deficient in Palay, such that claimed features derived therefrom are also deficient, and the rejection should be withdrawn.

For at least these reasons Palay fails to disclose or suggest the recited **"plurality of class files," "removing said duplicated elements from said plurality of class files to obtain a plurality of reduced class files,"** or **"removing said duplicated elements from said plurality of class files,"** as recited by claim 1 (and similarly claims 7 and 13). Accordingly, the rejection of claims 1, 5-7, 11-13, and 15 should be withdrawn.

C. Palay fails to disclose or suggest the features of dependent claims 5, 6, 11, and 12.

Claim 5 further adds that determining a plurality of duplicated elements includes **"determining one or more constants shared between two or more class files,"** and claim 6 further adds that forming a shared table includes **"forming a shared constant table comprising said one or more constants shared between said two or more class files."** These features are not disclosed or reasonably suggested by Palay. Claims 11 and 12 recite similar features as claims 5 and 6 respectively.

The Office Action states that Palay discloses the features of claims 5 and 6 in column 28, lines 54-56, stating: "[w]hen merging the class definition tables 414, duplicate non-dynamic and internal dynamic class definitions (constants) are removed." (Action, page 12.) (Parenthetical added to citation of Palay by Requester and adopted by the Office.) It is unclear to the Patent Owner what the relevance of this portion of Palay is to the recited features of claims 5 and 6. In particular, "Palay is silent as to the removal of duplicate 'constants.' At best, Palay describes the removal of duplicate 'class definitions' shared between two or more classes, however, this is not the same or equivalent to removal of duplicate constants of class files." (Goldberg Decl., ¶ 36.) Furthermore, removing duplicate class definitions per the operation of Palay's linker 112 does not teach or

suggest modifying Palay's invention to remove shared constants of class files. (Goldberg Decl., ¶ 36.)

Accordingly, claims 5, 6, 11, and 12 should be confirmed for these additional reasons.

D. Palay fails to disclose or suggest the features of dependent claim 15.

Palay further fails to disclose or suggest the features of claim 15 at least because Palay fails to disclose the features of independent claim 13 from which claim 15 depends. Accordingly, the rejection to claim 15 should be withdrawn.

V. Supplemental Information Disclosure Statement

Patent Owner further draws the Examiner's attention to the Supplemental Information Disclosure Statement filed herewith. The Supplemental Information Disclosure Statement cites further information produced during the pending litigation since the mailing of the Office Action.

VI. Conclusion

For at least these reasons, Patent Owner requests reconsideration and withdrawal of the rejections in the Action and confirmation of the patentability of claims 1, 5-7, 11-13, 15, and 16 of the '702 Patent.

Patent Owner notes that the Request and the Action include a number of assertions and allegations, including those concerning the '702 Patent specification and claims, and the cited references. Patent Owner has addressed the references and the assertions and allegations to the extent understood. Patent Owner does not subscribe to any assertion or allegation in the Request and the Action regardless of whether it is specifically addressed herein.

In the event the U.S. Patent and Trademark office determines that an extension and/or other

relief is required, Patent Owner petitions for any required relief including extensions of time and authorizes the Commissioner to charge the cost of such petitions and/or other fees due in connection with the filing of this document to Deposit Account No. 03-1952 referencing docket no. 158492800400.

Dated: September 6, 2011

Respectfully submitted,

By Electronic Signature /Christopher B. Eide/
Christopher B. Eide
Registration No.: 48,375
MORRISON & FOERSTER LLP
755 Page Mill Road
Palo Alto, California 94304-1018
(650) 813-5720