UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
95/001,560	03/01/2011	7426720	13557.105125	8687

25226 7590 11/18/2011
MORRISON & FOERSTER LLP
755 PAGE MILL RD
PALO ALTO, CA 94304-1018

EXAMINER	
STEELMAN, MARY J	
ART UNIT	**PAPER NUMBER**
3992	

MAIL DATE	DELIVERY MODE
11/18/2011	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

UNITED STATES PATENT AND TRADEMARK OFFICE

Commissioner for Patents
United States Patents and Trademark Office
P.O.Box 1450
Alexandria, VA 22313-1450
www.uspto.gov

DO NOT USE IN PALM PRINTER

THIRD PARTY REQUESTER'S CORRESPONDENCE ADDRESS
KING & SPALDING
1180 PEACHTREE STREET, N.E.
ATLANTA, GA 30309-3521

Date: **MAILED**

NOV 18 2011

CENTRAL REEXAMINATION UNIT

Transmittal of Communication to Third Party Requester
Inter Partes Reexamination

REEXAMINATION CONTROL NO. : 95001560
PATENT NO. : 7426720
TECHNOLOGY CENTER : 3999
ART UNIT : 3992

Enclosed is a copy of the latest communication from the United States Patent and Trademark Office in the above identified Reexamination proceeding. 37 CFR 1.903.

Prior to the filing of a Notice of Appeal, each time the patent owner responds to this communication, the third party requester of the inter partes reexamination may once file written comments within a period of 30 days from the date of service of the patent owner's response. This 30-day time period is statutory (35 U.S.C. 314(b)(2)), and, as such, it cannot be extended. See also 37 CFR 1.947.

If an ex parte reexamination has been merged with the inter partes reexamination, no responsive submission by any ex parte third party requester is permitted.

All correspondence relating to this inter partes reexamination proceeding should be directed to the Central Reexamination Unit at the mail, FAX, or hand-carry addresses given at the end of the communication enclosed with this transmittal.

ACTION CLOSING PROSECUTION (37 CFR 1.949)	Control No. 95/001,560	Patent Under Reexamination 7426720
	Examiner MARY STEELMAN	Art Unit 3992

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address. --

Responsive to the communication(s) filed by:
Patent Owner on 05 July, 2011
Third Party(ies) on 04 August, 2011

Patent owner may once file a submission under 37 CFR 1.951(a) within 1 month(s) from the mailing date of this Office action. Where a submission is filed, third party requester may file responsive comments under 37 CFR 1.951(b) within 30-days (not extendable- 35 U.S.C. § 314(b)(2)) from the date of service of the initial submission on the requester. **Appeal cannot be taken from this action.** Appeal can only be taken from a Right of Appeal Notice under 37 CFR 1.953.

All correspondence relating to this inter partes reexamination proceeding should be directed to the **Central Reexamination Unit** at the mail, FAX, or hand-carry addresses given at the end of this Office action.

PART I. THE FOLLOWING ATTACHMENT(S) ARE PART OF THIS ACTION:

1. ☒ Notice of References Cited by Examiner, PTO-892
2. ☒ Information Disclosure Citation, PTO/SB/08
3. ☐ _____

PART II. SUMMARY OF ACTION:

1a. ☒ Claims 1-8,10-17 and 19-22 are subject to reexamination.
1b. ☒ Claims 9 and 18 are not subject to reexamination.
2. ☐ Claims _____ have been canceled.
3. ☐ Claims _____ are confirmed. [Unamended patent claims]
4. ☐ Claims _____ are patentable. [Amended or new claims]
5. ☐ Claims 1-8, 10-17, and 19-22 are rejected.
6. ☐ Claims _____ are objected to.
7. ☐ The drawings filed on _____ ☐ are acceptable ☐ are not acceptable.
8 ☐ The drawing correction request filed on _____ is: ☐ approved. ☐ disapproved.
9 ☐ Acknowledgment is made of the claim for priority under 35 U.S.C. 119 (a)-(d). The certified copy has: ☐ been received. ☐ not been received. ☐ been filed in Application/Control No _____
10. ☐ Other _____

This office action addresses the reexamination of claims 1-8, 10-17, and 19-22 of USPN 7,426,720 B1 to Fresko. Per Non Final Office Action mailed 05/05/2011, claims 1-8, 10-17, and 19-22 are rejected. This Office Action is responsive to Patent Owner Remarks (07/05/2011) and Requester Comments (08/04/2011).

Information Disclosure Statement

IDS received 04/27/2011 and 07/05/2011 have been entered into prosecution. "With respect to the Information Disclosure Statement (PTO/SB/08A and 08B or its equivalent) considered with this action, the information cited has been considered as described in the MPEP. Note that MPEP 2256 and 2656 indicate that degree of consideration to be given to such information will be normally limited by the degree to which the party filing the information citation has explained the content and relevance of the information by way of a concise explanation of the relevance, as it is presently understood by the individual designated in § 1.56(c) most knowledgeable about the content of the information, of each patent, publication, or other information listed that is not in the English language. The concise explanation may be either separate from applicant's specification or incorporated therein.

Information that does not appear to be "patents or printed publications" as identified in 35 U.S.C. 301 has been considered to the same extent (unless otherwise noted), but has been lined through and will not be printed on any resulting reexamination certificate."
Undated NPL has been lined through and not considered.

Examiner has entered into prosecution the NPL documents, submitted as Appendices / Exhibits by Patent Owner (07/05/2011) and Third Party Requester (08/04/2011) by way of Form PTOL 892. Copies of the NPL are found in the reexamination prosecution file.

1.132 Declarations

Dr. Benjamin Goldberg, in support of Patent Owner (07/05/2011)

Dr. Goldberg has provided an opinion directed to patentability over the cited references and secondary considerations of nonobviousness. Dr. Goldberg has presented credentials (Exhibit A, Curriculum Vitae of Dr. Benjamin Goldberg) that support his experiences and knowledge as one skilled in the art. Dr. Goldberg is being compensated for his opinions and such compensation is not conditioned on the outcome of the reexamination.

Dr. Goldberg asserts (paragraph 9) the '720 Patent provides a new approach to virtual machine memory management and startup by using copy-on-write with process cloning of virtual machines. As a result, a master virtual machine (e.g., the claimed master runtime system process) preloaded to a prewarmed state can be cloned to provide a child virtual machine (e.g., the claimed child runtime system process) that inherits the prewarmed state, thereby reducing the startup time for the child virtual machine. Additionally, copy-on-write cloning can be applied to the memory space of the master virtual machine so as to defer copying of the memory space until the child virtual machine needs to modify that memory space, thereby reducing the memory footprint of the child virtual machine and further reducing startup time.

Dr. Goldberg rebuts the rejections based on the Webb, Kuck and Bach combination (paragraphs 10-14), the Dike and Steinberg combination (paragraph 15), the Bryant and Bach combination (paragraphs 16-18), the Bryant and Traut combination (paragraphs 19-21), the Srinivasan and Bach combination (paragraphs 22-23), the Sexton and Bugnion combination (paragraphs 24-27) and the Sexton and Johnson combination (paragraphs 28-29).

Dr. Goldberg asserts secondary considerations (paragraphs 30-31) noting a need felt for efficient use of memory between multiple virtual machine processes, while providing a robust environment for executing the multiple virtual machine processes concurrently. Dr. Goldberg asserts that '720 satisfied this need with a new approach to virtual machine memory management and startup that used process cloning with copy-on-write technology to share memory between a master virtual machine and a cloned virtual machine until the cloned virtual machine needed to modify the shared memory. '720 shares common libraries between processes, does not have to repeat initialization costs for each cloned virtual machine, and it shares memory between processes by default. It reduces startup time by cloning a prewarmed state of a master virtual machine onto a child virtual machine.

The 1.132 Declaration of Dr. Goldberg has been weighed and considered in full. In assessing the probative value of an expert opinion, the examiner must consider the nature of the matter sought to be established, the strength of any opposing evidence, the interest

of the expert in the outcome of the case, and the presence or absence of factual support for the expert's opinion. Ashland Oil, Inc. v. Delta Resins & Refractories, Inc., 776 F.2d 281, 227 USPQ 657 (Fed. Cir. 1985), cert. denied, 475 U.S. 1017 (1986).

In view of the foregoing, regarding the Declaration of Dr. Goldberg, under 37 CFR 1.132, based on consideration of the entire record by a preponderance of the evidence, weighted by the relevance, when all of the evidence is considered, the totality of the rebuttal evidence pertaining to secondary considerations fails to outweigh the evidence presented and considered by Examiner. The Goldberg Declaration under 37 CFR 1.132, filed 07/05/2011, is insufficient to overcome the rejections under 35 USC 102(b) and 103(a) for the reasons detailed in the following paragraphs, claim rejections, and responses to arguments below.

The record has established a strong cause of obviousness with respect to combinations including Webb, Kuck, Bach, Dike, Steinberg, Bryant, Srinivasan, Sexton and Bugnion. Patent Owner's evidence is deemed insufficient to rebut the strong prima facie case of obviousness. Assertions related to rejections are aligned with Patent Owner arguments and addressed below.

Dr. Jason Flinn, in support of Third Party Requester (08/04/2011)

Dr. Flinn has provided an opinion directed to the validity and obviousness embodied by the scope of the claims of '720. Dr. Flinn has presented credentials that support experience and knowledge of one skilled in the art. Dr. Flinn is being compensated for

his work on this reexamination and states that such compensation is not conditioned on the outcome of the reexamination. Dr. Flinn notes (paragraph 10) that copy-on-write was a common low level operating system optimization that reduces the overhead associated with the instantiation of a new process, by sharing memory between the parent and child (created new processes) until the memory is modified by either process. The optimization is transparent to the application. Dr. Flinn provides an understanding of (paragraphs 11-12) the limitation "runtime environment" to be the underlying environment managing the master runtime system process, and managing the cloning process, for example by implementing the fork() system call. The master runtime environment will typically include the underlying operating system to perform functions like memory allocation and process cloning. Dr. Flinn provides an understanding of the level of skill of one having ordinary skill in the art at the time of the invention as required by an obviousness statement, (paragraphs 13-14) in agreement with Requester.

Dr. Flinn provides rebuttals to Patent Owner arguments that are mimicked by Third Party Requester. See analysis in Argument section below. The 1.132 Declaration of Dr. Flinn has been fully weighed and considered in its entirety. In view of the foregoing, regarding the Declaration of Dr. Flinn under 37 CFR 1.132, based on consideration of the entire record by a preponderance of the evidence, weighted by the relevance, when all of the evidence is considered, the totality of the rebuttal evidence is found to be persuasive by Examiner, for the reasons detailed in the following paragraphs, claim rejections, and responses to arguments below.

Patent Owner Exhibits (PO Remarks 07/052011, p. 6)

Ex. A Declaration of Dr. Benjamin Goldberg ("Goldberg Declaration")

Ex. B "CDC Porting Guide for the Sun Java Connected Device Configuration Application

Management System," Version 1.0, Sun Microsystems, Inc., November 2005 ("Porting Guide")

Ex. C "CDC Runtime Guide for the Sun Java Connected Device Configuration Application Management System," Version 1.0, Sun Microsystems, Inc., November 2005 ("Runtime Guide")

Ex. D "Anatomy and Physiology of an Android, Google I/O 2008," by Patrick Brady, http ://sites.google.com/site/io/anatomy-phvsiology-of-an-android/Android-Anatomy-GoogleIO.pdf ("Android Presentation," last visited July, 5, 2011)

Ex. E "Dalvik Virtual Machine Internals, Google I/O 2008," by Dan Bornstein, http://sites.google.com/site/io/dalvik-vm-internals/2008-O5-29-Presentation-Of-Dalvik-VM-Internals.pdf ("Dalvik Presentation," last visited July 5, 2011)

Ex. F "Bug ID: 4416624 multiple JVM runtimes do not share memory between themselves," http://bugs.sun.com/bugdatabase/viewbug.do?bug id=4416624 ("Shared Memory Blog," last visited July 5,2011)

Ex. G "Bug ID: 4469557 Faster startup/reduced footprint for subsequent VMs," http://bugs.sun.com/bugdatabase/viewbug.do?bugid=44679557 ("Reduced Footprint Blog," last visited July 1,2011)

Ex. H "Complaint for Patent and Copyright Infringement," Oracle America, Inc. v.

Google, Inc., N.D. Cal., Case No. cv10-03561 ("Complaint")

Ex. I "Zygote" reference page, http://59. 61.88.234/android-help/reference/dalvik/system /Zygote.html ("Zygote," last visited July 5, 2011)

Ex. J "Evidence of Copying by Google of US 7,426,720" Chart ("Copy Chart")

Ex. K "Multitasking VMs: More Performance, Less Memory," by Kyle Buza, et al., JavaOne Conference, June 2005 ("JavaOne Presentation")

Ex. L "Cloneable JVM: A New Approach to Start Isolated Java Applications Faster," by Kiyokuni Kawachiya, et al., VEE'07, June 2007 ("Kawachiya Paper")

Requester Exhibits (Third Party Comments, 08/04/2011, p. 3)

Ex. 1. Declaration of Dr. Jason Flinn ("Flinn Declaration")

Ex. 2. Curriculum Vitae of Jason Flinn

Ex. 3. ABRAHAM SILBERSCHATZ & I. PETERSON, OPERATING SYSTEMS CONCEPTS (Alt. ed. 1988) ("Silberschatz 1")

Ex. 4. CURT SCHIMMEL, UNIX SYSTEMS FOR MODERN ARCHITECTURES (1994) ("Schimmel")

Ex. 5. DANIEL P. BOVET & MARCO CESATI, UNDERSTANDING THE LINUX KERNEL (2001) ("Bovet")

Ex. 6. HP-UX MEMORY MANAGEMENT WHITE PAPER (Version 1.3, Apr. 7, 1997) ("HP-UX White Paper")

Ex. 7. ABRAHAM SILBERSCHATZ & PETER GALVIN, OPERATING SYSTEMS CONCEPTS (5^{th} ed. 1998) ("Silberschatz 2")

Ex. 8. JOHN R. LEVINE, LINKERS & LOADERS (Rev. 2.3, Jun. 30, 1999) ("Linkers

& Loaders")

Ex. 9. Janice J. Heiss, The Multi-Tasking Virtual Machine: Building a Highly Scalable JVM, published Mar. 22, 2005 ("Heiss")

Ex. 10. "Bug ID: 4416624 multiple JVM runtimes do not share memory between themselves," available at: http://bugs.sun.com/bugdatabase/viewbug.do?bug_id=4416624 (last visited Aug. 3,2011) ("Bug Report")

Ex. 11. New Java Language Features in J2SE 1.5, published Jul. 29, 2003 ("Java Live Dialog")

Arguments

Requester asserts (Third Party Comments, 08/04/2011, Introductory Comments, pp. 4-7) that "copy-on-write" technology (copy-on-write version of the fork() system call) was widely known in the art both at the time of, and even long before the work described in the '720 patent. **Requester** notes the ('720, 4: 66-5:6) credit given to Bach (See Bach, "The Design of the Unix Operating System, disclosure of Chapter 7, incorporated by reference within the '720 patent) as an affirmation of a prior art teaching of the copy-on-write process. **Requester** points to additional citations (Bach at Chapter 9) in support of the assertion that the copy-on-write version of the fork() [vfork] system call was known.

> Bach at page 289-290, "As explained in Section 7.1, the kernel duplicates every region of the parent process during the *fork* system call and attaches it to the child process. Traditionally, the kernel of a swapping system makes a physical copy of the parent's address space, usually a wasteful operation... On the System V paging system, the kernel avoids copying the page by manipulating...The page can now

be referenced through both regions, which share the page until a process writes to it. The kernel then copies the page so that each region has a private version. To do this, the kernel turns on the "copy on write" bit for every page table entry in private regions of the parent and child processes during fork. If either process writes the page, it incurs a protection fault, and in handling the fault, the kernel makes a new copy of the page for the faulting process. The physical copying of the page is thus deferred until a process really needs it."

"Figure 9.15 shows the data structures when a process forks. The processes share access to the page table of the shared text region... The kernel allocates a new child data region... The implementation of the fork system call in the BSD system makes a physical copy of the pages of the parent process. Recognizing the performance improvement gained by not having to do the copy, however, the BSD system also contains the vfork system call, which assumes that a child process will immediately invoke exec on return from the vfork call. Vfork does not copy page tables so it is faster than the System V fork implementation."

Requester cites to Dike at § 2.1; Steinberg at 21; Traut at ¶[0026]; Bugnion at 6:29-36; Johnson at 18:34-44 evidencing known copy-on-write versions of the fork() system call. Additional references, entered into prosecution via PTO Form 892 by Examiner, disclose known copy-on-write versions:

Ex. 3 –Silberschatz 1 at 481-82 (disclosing that "[a]n alternative is to share all pages by duplicating the page table, but to mark the entries of both page tables as *copy-on-write")* (emphasis in original).

Ex. 4 - Schimmel at 10 (A 1994 advanced UNIX text describing the fork() system call explained that "nearly all [UNIX] implementations [of fork()] use a technique called *copy-on-write* to avoid having to copy the bulk of the remaining portions of the address space.")

Ex. 5 - Bovet at 225 (A text on Linux, a popular open-source version of UNIX, described this optimization as follows: First-generation Unix systems implemented process creation in a rather clumsy way: when a fork() system call was issued, the kernel duplicated the whole parent address space in the literal sense of the word and assigned the copy to the child process... Modem Unix kernels, including Linux, follow a more efficient approach called Copy On Write, or COW.)

Ex. 6 - HP-UX White Paper at 96 "HP-UX now implements copy-on-write of EXEC_MAGIC processes, to enable the system to manipulate processes more efficiently...Copy-on-write means that pages in the parent's region are not copied to

the child's region until needed... sharing the same page...as soon as either parent or child writes to the page, a new copy is written, so that the other process retains the original view of the page."

Third Party Requester cites (Third Party Comments 08/04/2011, pp. 5-7), to the Flinn Declaration, paragraph 10 for support that "by the December 22, 2003 filing date of the application for the '720 patent, copy-on-write was a well-known, low-level operating system technique that expedited the fork() system call and allowed shared memory between parent and child processes and did so in a manner that was transparent to programmers.

Examiner again reviews the prosecution history for application 10/745,023 that matured into the '720 patent. Applicant's Remarks (12/18/2007, pp. 9-10) after a Non Final Office action assert, "Both Webb and Kuck involve reducing the number of instructions needed when allocating a new virtual machine from scratch, but do not attempt to reduce the memory usage for these virtual machines."

"In contrast, the present invention reduces both the allocation overhead and the memory space usage of virtual machines by using copy-on-write cloning (see page 4, line 28 to page 5, line 11, of the instant application). The present invention enables copying the master runtime system process context to create a child runtime system by creating a logical set of references to the master runtime system process context so that referenced segments are lazily copied at modification time. Deferring the copying of the memory space of the master runtime system process until the child runtime system process needs to modify the referenced memory space reduces the time and space needed to initialize and execute the child runtime system (see page 12, lines 2 to 17, of the instant

application). Nothing in Webb or Kuck discloses reducing memory usage for virtual machines by using copy-on-write cloning."

"Accordingly, Applicant has amended independent claims 1, 12, and 24 to include limitations from dependent claims 7 and 18, to clarify that the runtime environment is configured to clone the memory space of a child runtime system process using a copy-on-write process cloning mechanism. This cloning mechanism instantiates the child runtime system process by copying references to the memory space of the master runtime system process into a separate memory space for the child runtime system process. This process cloning mechanism defers copying the memory space of the master runtime system process until the child runtime system process needs to modify the reference memory space of the
master runtime system process."

Additionally as noted in Applicant's interview agenda (12/21/2007), "My understanding is that Webb (USPN 6,823,509) discloses re-using allocated virtual machines, and Kuck (US Pub. No. 2003/0088604) discloses copying a pre-initialized process-attachable virtual-machine to reduce overhead. While both Webb and Kuck involve reducing the number of instructions needed when allocating a new virtual machine from scratch, they do not attempt to reduce the memory usage for those virtual machines. In contrast, the present invention reduces both the allocation overhead and the memory space usage of virtual machines using copy-on-write cloning (see page 4, line 28 to page 5, line 11, of

the instant application). Nothing in Webb or Kuck discloses reducing memory usage for - virtual machines by using copy-on-write cloning."

An Examiner's Amendment (05/20/2008) to add "a processor" and "a memory" to the system and apparatus (presumably to add hardware system / apparatus elements) preceded the Examiner's statement of reasons for allowance. The Examiner stated, "As pointed out by applicant, the prior art of record fails to teach and/or suggest a runtime environment to clone the memory space as a child runtime system process responsive to a process request and to execute the child runtime system process and a copy-on-write process cloning mechanism to instantiate the child runtime system process by copying references to the memory space of the master runtime system process into a separate memory space for the child runtime system process, and to defer copying of the memory space of the master runtime system process until the child runtime system process needs to modify the referenced memory space of the master runtime system process as recited in independent claims."

Prosecution history suggests that the Applicant alleged the novelty of '120 included reducing both the allocation overhead and the memory space usage of virtual machines using copy-on-write cloning, in contrast to prior art teachings that involve reducing the number of instructions needed when allocating a new virtual machine.

Examiner agrees with Requester that evidence exists of known copy-on-write versions of the fork() system call, prior to the file date of '720. **Examiner** opines that regardless of Patent Owner's prior statement, a copy-on-write version of the fork() system call (as taught by Bach) would necessarily reduce the allocation overhead and the memory space usage of virtual machines. Until a point in time where the child virtual machine attempts to modify shared memory locations of the parent virtual machine (the non-copied memory segments), any unmodified parent memory space is shared between the parent and child process.

Support in the '720 Specification for Claim Language

Patent Owner asserts (Remarks 07/05/2011, pp. 17-18, 27-28) that a Linux environment executing C and C++ programs lacks the class preloader. **Patent Owner** cites to the **Goldberg Declaration**, paragraph 15 and asserts that Dike does not disclose or even suggest a "class preloader" or any other preloader that "obtain[s] a representation of at least one class from a source definition provided as object-oriented program code," because Dike discloses a Linux environment, which executes C and C++ programs that do not provide a representation that can be interpreted and instantiated as a class definition. Rather C and C++ programs are simply compiled into machine code that is then executed. On the other hand, the "class preloader" provides "a representation of at least one class from a source definition provided as object-oriented program code" that is interpreted and instantiated as a class definition. A Linux virtual machine in machine code compiled from C and C++ has no need to be prewarmed, i.e., preloaded with the

class data. Thus, the combination's Linux virtual machine does not lead to a "class preloader."

At §2.2, Dike's disclosure of the initialization of a system in preparation for booting up a user-mode kernel does not involve a "class preloader." Dike's mention of the initialization being "analogous to the boot loader on a physical machine" is not the same as the "class preloader." The boot loader loads the programs that run the physical machine, without also performing preloading that "obtain[s] a representation of at least one class from a source definition provided as object-oriented program code." In §3, Dike discloses various applications, including daemons and services, that are started when the user-mode kernel boots up. However, none of the various applications is started by a "class preloader" or includes "a representation of at least one class from a source definition

provided as object-oriented program code."

Steinberg's brief disclosures of C++ code and glibc are not associated with a "class preloader." (Goldberg Declaration, paragraph 15.)

Regarding claim interpretations, **Requester** points out that (Third Party Comments 08/04/2011, pp. 7-9) the broadest possible construction consistent with the specification is applied to claim language in the context of a reexamination. *See In re Icon Health & Fitness, Inc.*, 496 F.3d 1374, 1379 (Fed. Cir. 2007).

Requester disputes (p. 8) Patent Owner's assertion that the term "class preloader" requires only a Java language class preloader. **Requester** cites to '720, 5: 12-15: "[a]lthough described with specific reference to classes, other forms of structured static data could also be preloaded, including data structures, processes, functions, subroutines, interfaces, and the like." **Requester** asserts that the dynamic library compilation of a C++ program's object-oriented classes, which indisputably contains classes, and the packages of an assembled Perl program, are directly analogous to the Java class, and easily fall within the definition of a class preloader provided by the '720 patent specification when applying a broadest reasonable construction.

Requester asserts (p. 9-10) that the master runtime system process reads on the parent virtual machine. **Requester** cites to '720, 2: 49-51; 5: 24-26 and to the **Flinn Declaration**, paragraphs 11-12 for support. **Requester** asserts that a [master] runtime environment is simply the underlying environment managing the master runtime system process, and managing the cloning process, for example by implementing the fork() system call...will typically include the underlying operating system to perform functions like memory allocation and process cloning.

Examiner agrees with Requester that the '720 specification broadly provides for '(720, 5: 7-21: "By way of example, the system is described with reference to the Java operating environment, although other forms of managed code platforms that execute applications preferably written in an object oriented programming language, such as the Java

programming language, could also be used.") object oriented languages, only reciting Java as an example. Therefore the invention is not limited to the use of the Java language class preloader. Object oriented code representing a class may be interpreted by a virtual machine.

Examiner cites to the Specification for claim construction:

'720, 2: 66 – 3: 6 "class preloader" During initialization, the master runtime system process **preloads classes and interfaces likely to be required** by user application at runtime. The **classes and interfaces are identified through profiling by ranking a set of classes according to a predetermined criteria** such as described in commonly-assigned U.S. patent application Ser. No. 09/970,661, filed Oct. 5, 2001, pending, the disclosure of which is incorporated by reference. 6: 38-43, **preloads classes 36 and classes defined in the class libraries 37 that are likely to be required by applications** at runtime...classes and interfaces are **identified through profiling by ranking a set of classes** according to a predetermined criteria...

'720, 3: 12-20 "dynamic preloading of classes" dynamic **preloading of classes through memory space cloning of a master runtime system process**. A master runtime system process is executed. A representation of at least one class is obtained from a source definition provided as object-oriented program code. The representation is **interpreted and instantiated as a class definition in a memory space of the master runtime**

system process. The **memory space is cloned as a child runtime system process** responsive to a process request and the child runtime system process is executed.

'720, 5: 11-14, Although described with specific reference to classes, **other forms of structured static data could also be preloaded, including data structures, processes, functions, subroutines, interfaces, and the like**.

'720, 5: 7-21, describes a system for **preloading classes**. "By way of example, the system is described with reference to the Java operating environment, **although other forms of managed code platforms that execute applications** preferably written in an object oriented programming language, such as the Java programming language, could also be used."

'720, 5: 48-58 – Upon **initialization**, the master JVM process 33 reads an executable process image from the storage device 35 and performs **bootstrapping operations**. These operations **include preloading the classes 36 and classes defined in the class libraries 37**...upon completion of initialization, the memory image of the master JVM process 33 resembles that of an initialized, primed and warmed up JVM process with key classes stored in the master JVM process context as prewarmed state 41.

'720, 6: 49-54 – **Class loading requires** identifying **a binary form of a class type as identified by specific name**...[and] can include retrieving a binary representation from source and constructing a class object to represent the class in memory.

'720, 8: 47-50 - **preloading classes** involves executing the **bootstrap class loader 39 and system application class loader 40** to **create and resolve classes likely required** by one or more of the applications.

'720, 9: 29-62, routine 150 for **preloading a class** 36 for use in the routine 120...one purpose... is to **find and instantiate prewarmed instances of classes 36 and classes defined in the class libraries 37 as specified in the bootstrap class loader 39 and system application class loader 40 as prewarmed state 41 in the master** JVM process 33 **for inheritance by a cloned** JVM process 34.

'720, 6: 64-67, Thus, the **prewarmed** state 41 includes the **class loading** for applications [every class likely to be requested by the applications] **prior to actual execution** and the **initialized and loaded classes are inherited by each cloned** JVM process 34 **as** the inherited **prewarmed** state 42.

'720, 2: 49-50 - "master runtime system process" such as a virtual machine, to interpret machine-portable code defining compatible applications

'720, 5: 22-32, The exemplary runtime environment 31 includes an application manager 32, master Java virtual machine (JVM) process 33 and zero or more cloned JVM processes 34. The master JVM process 33 and cloned JVM processes 34 respectively correspond to a master runtime system process and child runtime system processes. The master runtime system process, **preferably provided as a virtual machine**, interprets machine-portable code defining compatible applications. The runtime environment 31 need not execute cloned JVM processes 34, which are only invoked upon request by the application manager 32.

'720: 2: 51- 60- "child runtime system process" An application manager executing within the application framework, communicatively interfaced to the master runtime system process through an inter-process communication mechanism logically **copies the master runtime system process context upon request** by the application framework **to create a child runtime system process through process cloning. The context of the master runtime system process stored in memory is inherited by the child runtime system process as prewarmed state and cached code.**

'720, 2: 60-66 - "copy-on-write process cloning mechanism" **When implemented** with copy-on-write semantics, the process cloning **creates a logical copy of references to the master runtime system process context**. [i.e., initially, all master runtime system process context is not copied into child runtime system process] **Segments** of the referenced master runtime system process context **are lazily copied** [into child] only **upon an attempt by the child runtime system process to modify the referenced context**.

'720, 8: 4-8 - Through **copy-on-write** semantics, the overall footprint of the runtime **environment 31 is maintained as small as possible** and only **grows** until, and **if**, each **cloned JVM process 34 actually requires additional memory space** for application specific context.

'720, 3: 6-10 - "runtime environment" An example of a suitable managed code platform and runtime system process are the Java operating environment and Java virtual machine (JVM) architecture...

'720, 5: 33-36 - The runtime environment 31 executes an application framework that **spawns** multiple independent and isolated user application processes instances **by preferably cloning the memory space** of a master runtime system process.

'720, 5: 36-40 - The example of an application framework suitable for use in the present invention is the Unix **operating system**...

'720, 3: 21-40; 4: 50-5: 6 – "process cloning mechanism" provided by the underlying application framework [operating system], resolves the need for efficient concurrent application execution of machine portable code. The **inheritance of prewarmed state through the cloning of the master runtime process context** provides inter-process sharing of preloaded classes. Similarly, each child runtime system process executes in isolation of each other process, thereby providing strong resource control through the system level services of the application framework. Isolation, reliable process invocation and termination, and resource reclamation are available and cleanly provided at an operating system level. In addition, **process cloning provides fast user application initialization** and deterministic runtime behavior, particularly for environments providing process cloning with copy-on-write semantics. Finally, **for non-shareable segments** of the master runtime system process context, actual **copying is deferred until required through copy-on-write semantics**, which avoids impacting application performance until, and if, the segment is required.

Each operating system supports a process cloning mechanism that spawns multiple and independent isolated user applications **by cloning the memory space** of specifiable

processes. **An example of a process cloning mechanism** suitable for use in the present invention is the **fork( ) system call provided by the Unix or Linux operating systems,** such as described in M. J. Bach, "The Design Of The Unix Operating System," Ch. 7, Bell Tele. Labs., Inc. (1986), the disclosure of which is incorporated by reference. **The process invoking the fork( ) system call is known as the parent process and the newly created process is called the child process.** The operating system assigns a separate process identifier to the child process, which executes as a separate process. The operating system also creates a logical copy of the context of the parent process by copying the memory space of the parent process into the memory space of the child process. In **a copy-on-write variant of the fork( ) system call, the operating system only copies references to the memory space and defers actually copying individual memory space segments until, and if, the child process attempts to modify the referenced data of the parent process context**. The copy-on-write fork( ) system call is faster than the non-copy-on-write fork( ) system call and **implicitly shares any data not written into** between the parent and child processes. ['721, 7: 61-64, until child cloned process attempts to modify the master process context, the memory space is treated a read only data, which can be shared by other processes]

'720 5: 59 – 6: 19 - Following initialization, the master JVM process 33 idles, that is, "sleeps"...[and] awakens in response to requests received from the application manager 32 to execute applications...The application manager 32 sends a request to the master JVM process 33, including standard command line parameters, such as application name,

class path, and application arguments. The master JVM process 33 awakens and **creates a cloned** JVM process 34 as a new cloned process instance of the master JVM process 33 **using the process cloning mechanism of the underlying operating system**. The **context of the master** JVM process 33 **stored in memory as prewarmed state 41 is inherited by the cloned** JVM process 34 as inherited prewarmed state 42, thereby saving initialization and runtime execution times and providing deterministic execution behavior...the master JVM process 33 records the launched application in an applications launched list 38 and return to an inactive sleep state.

When implemented with copy-on-write semantics, the process cloning creates a logical copy of only the reverences to the master JVM process contest. Segments of the referenced master JVM process context are lazily copied only upon an attempt by the cloned JVM process to modify the referenced context. Therefore, as long as the cloned JVM process does not write into a **memory segment**, the segment **remains shared** between parent and child processes.

Rejections based on the combination of Webb, Kuck, and Bach

Patent Owner asserts (Remarks 07/05/2011, pp. 12-14 & 15) the Webb, Kuck, and Bach combination does not disclose the **"runtime environment."** In particular, Kuck, cited for this element, lacks the requisite cloning, i.e., "to clone," in a **"runtime environment."** Kuck's initialization merely copies initialization data from the master PAVM to the new PAVM. Kuck thus cannot be considered to disclose that the new PAVM is cloned. Because Kuck does not disclose cloning, Kuck does not disclose the

"runtime environment to clone." **Patent Owner** cites to the Goldberg Declaration paragraph 10.

Patent Owner argues (Remarks 07/05/2011, pp. 14-16) the combination of Webb, Kuck, and Bach and asserts: (a) Webb teaches away from the combination regarding cloning; (b) Kuck teaches away from the combination regarding copy-on-write; (c) the combination changes the principle of operation of Webb from a single reusable virtual machine to multiple independent virtual machines; (d) the combination changes the principle of operation of Kuck from simple shared memory to copy-on-write memory; and (e) the reasons for the combination are insufficient to support *a prima facie* case of obviousness.

Patent Owner asserts (Remarks 07/05/2011, p. 14) that Webb discloses the desirability of running successive applications on the same Java virtual machine (Webb, 1:39-41; 2:5-25; 4:8-11). This is fundamentally different from Kuck (Abstract, directed to creating multiple Java machines), and Bach (Chapter 7, directed to creating multiple processes). Patent Owner concludes therefore, that Webb teaches away from using any process, such as Bach's cloning, that would create multiple Java virtual machines. **Patent Owner** cites to the Goldberg Declaration, paragraph 11(attesting Webb's reuse of a virtual machine). **Patent Owner** quotes, " 'Teaching away' does not require that the prior art foresaw the specific invention that was later made, and warned against taking that path." *Spectralytics, Inc., v. Cordis Corp.* (Fed. Cir. June 13,2011). Rather, the

design of the prior art device itself can teach away from the invention. *(Id.)* See also, e.g., MPEP 2145(X)(D), which describes teaching away as an improper rationale for combining references.

Patent Owner asserts (Remarks 07/05/2011, pp. 14-15) that Kuck teaches away. Kuck discloses the allocation of a memory block for a new PAVM: "a PAVM is generated and initialized for that user session (502). That can include allocating a block of memory for the PAVM...After a block of memory has been allocated to the PAVM, the PAVM can be stored in the memory block." (Kuck, [0062]-[0063].) This is fundamentally different from Bach's copy-on-write process, in which no allocation of a memory block is performed when a new process is forked. As such, Kuck teaches away from using any process, such as copy-on-write, that would avoid or defer until necessary memory block allocation for a new virtual machine. **Patent Owner's** position is supported by the Goldberg Declaration, paragraph 13.

Patent Owner asserts (Remarks 07/05/2011, p. 15) that such a combination would change the principle of operation of Webb, and the principle of operation of Kuck. The modification of Webb, would provide separate Java virtual machines, with no need to reinitialize classes on the same virtual machine. Modifying Kuck to implement cloning would obviate the need to generate and reuse persistent PAVMs. A copy-on-write would replace Kuck's simple shared memory with a copy-on-write memory. **Patent Owner** cites to the Goldberg Declaration, paragraphs 13-14.

Patent Owner disagrees with (Remarks 07/05/2011, pp.15-16) the given reasons to combine Webb, Kuck, and Bach. **Patent Owner** asserts that Kuck does not disclose the "runtime environment" to clone a memory space and even if the "runtime environment" were added to Kuck, the skilled person reading Webb would not be motivated to modify Webb to include cloning because Webb specifically teaches the desirability of using a single Java virtual machine (1:39-41; 2:5-25; 4: 8-11; claim 1), thereby eliminating the need for cloning to create multiple virtual machines. Adding cloning would increase system overhead and increase the likelihood of crashing processes because of the increased number of virtual machines running the processes, negating any alleged benefit from Kuck.

Patent Owner disagrees with the given motivation to combine (Non Final Office Action, 05/05/2011, p. 10) Bach with Webb and Kuck "to reduce memory usage and overhead impact when instantiating and executing virtual machines." "...the combination of Webb and Kuck does not provide all the claimed elements, i.e., the "runtime environment," as described above... implementing cloning, even with copy-on-write, would increase the memory usage and overhead impact of Webb' s operation by creating and using multiple Java virtual machines, rather than reusing the same virtual machine. Lastly, because Kuck's operation already performs efficiently and effectively by creating and storing PAVMs for later use in the disclosed manner, there would have been no need to implement copy-on-write.

As stated during prosecution of the '720 Patent (December 18, 2007 Amendment), Webb and Kuck were concerned with reducing the overhead needed when executing a virtual machine (e.g., Webb, 2:5-16; Kuck, [0064]-[0065]), but did not attempt to reduce the memory usage for the virtual machine, as provided by copy-on-write.

Moreover, given that copy-on-write technology was known as early as 1986, when Bach was published and at least fourteen years before Webb and Kuck were filed as patent applications, it clearly was not obvious to use copy-on-write "to reduce memory usage and overhead impact when instantiating and executing virtual machines." Otherwise, either Webb or Kuck would have done so. **Patent Owner** cites to Goldberg, paragraphs 10-14 for support.

Regarding dependent claims 4 and 13, **Patent Owner** asserts (Remarks 07/05/2011, pp.16-17) patentability for the same reasons given for their respective independent claims. Regarding dependent claims 6 and 15, Patent Owner asserts that Kuck's disclosure of the initialization of a new PAVM based on an already initialized master PAVM does not disclose that the PAVM is created as the result of "cloning." Regarding dependent claims 21 and 22, Patent Owner asserts that Kuck's PAVMs cannot be considered to be cloned and therefore cannot provide the requisite "child runtime system process" on which the "resource controller" operates.

Requester asserts (Third Party Comments, 08/04/2011, pp. 16-20) that the prior rejection [during the original examination of 10/745,023] relied on only the Webb and Kuck references. In response to Patent Owner Remarks (07/05/2011, p. 22), **Requester** disagrees (p. 17) that Webb, Kuck, and Bach were "specifically indicated as not being combinable." **Requester** opines (p. 18) that Kuck's step-by-step description of creating a new virtual machine and providing that child virtual machine with the parent virtual machine's full set of initialization data does disclose cloning ¶¶[0062]-[0064]: "the new PAVM to start running in an already-initialized state." Kuck's cloning includes "copying" of the entire set of initialization data from the master virtual machine to the child virtual machine.

Further supporting this point, the Kuck process contains no values; it starts with a blank canvas. After creating this blank canvas, the memory block of the master PAVM can simply be copied into the child. This is the same as the "cloning" mechanism cited by Patent Owner, just broken down into more clear steps. Kuck discloses all of the parts of the cloning process, including the integral copying aspect, in clear detail. **Requester** cites to the Flinn Declaration, paragraphs 31-34: Kuck discloses the allocation of memory space for a child virtual machine...starts as a 'blank canvas...copying of the master process attachable virtual machine into the child memory...child virtual machine to inherit preloaded classes. Flinn opines that Bach's (pp. 193-195 & FIG. 7.2) allocation followed by copying are an integral part of cloning.

Requester asserts (p. 18) that Patent Owner's related arguments concerning a "runtime environment to clone" are based on the same spurious argument that attempts to find a

distinction between "copying" and "cloning", *see* Response at 13, and fail for the same reasons.

Requester disagrees with Patent Owner's arguments related to "teaching away" and changing the "principle of operation." **Requester** opines that Patent Owner's claim that Webb disclosed the desirability of running applications on a single JVM, while Kuck was directed to creating multiple JVMs, does not support a "teaching away" argument. **Requester** asserts that Webb and Kuck, as well as the Bach reference, are directed to the reduction of overhead: thus, a person having ordinary skill in the art at the time of the invention, seeking to reduce system overhead, would be very likely to combine the two approaches disclosed in these references, e.g., running multiple applications on multiple JVMs. See Flinn Declaration at ¶35: references are technically and thematically consistent, seeking to reduce system overhead.

Requester cites to the Flinn Declaration at ¶36: "...Kuck and Bach disclosures work seamlessly with one another using only basic code instructions and system calls. Kuck describes child process attachable virtual machines that are copies of a master process attachable virtual machine. To employ copy-on-write, Kuck would share memory pages between the master and newly cleated child process attachable virtual machines. These pages would be mapped read-only into any process to which one of these process attachable virtual machines is attached. When the [read only] memory page is modified [written to], a page fault would occur, during which Kuck would allocate a new page for the faulting process attachable virtual machine, copy the data from the faulted page to the

new page, map the page(s) read-write, and restart the faulting process. This is the same method used by a copy-on-write fork."

Requester asserts (p. 19) that the principles of operation of the Webb, Kuck, & Bach disclosures are consistent. Kuck and Bach both disclose making a copy of the original state, where the copy becomes the master process, and then creating child versions based on this saved original state. Webb simply re-runs an initialization sequence (restores a virtual machine to an initialized state); Kuck restores the initialized state from a snapshot (e.g., the master virtual machine). These are just two ways of doing the same thing; both are well-known techniques. See Flinn Declaration at ¶37. The motivation, function and outcome are the same: returning to the original pre-warmed state for consistent reference information. Combining these references, a person having ordinary skill in the art could start with Kuck's process-attached virtual machine. See Flinn Declaration at ¶36. Following the Kuck disclosure, the person having ordinary skill in the art would create new virtual machines in response to new sessions. *See id.*

Within a Java system, the Webb disclosure provides a roadmap for the loading of classes into a JVM. One of ordinary skill in the art, seeking to implement the Kuck system in a Java system, would know from Webb to preload the Java classes into Kuck's parent virtual machine, so that the newly created virtual machines would have a copy of the pre-warmed, pre-initialized state of the parent. *See id.* at ¶38.

As the Examiner found during the original prosecution, the motivation here would be "avoiding the overhead" and "enabling the server to run robustly." *See* Non-Final

Rejection at 10. Further, implementation of this process on any number of operating systems utilizing copy-on-write, such as the UNIX system described by Bach, would result in a system with all of the limitations of the claimed '720 system. The use of a copy-on-write fork() is also consistent with the motivation to reduce the overhead of cloning new virtual machines by using the well-known copy-on-write optimization. *See* Flinn Declaration at ¶40.

Requester asserts that Patent Owner's argument that Webb would somehow preclude multiple virtual machines is without merit. Where multiple processors are present, Webb would inevitably lead to using multiple virtual machines, each applying the Webb technique. *See* Flinn Declaration at ¶41.

Requester asserts (p. 20) that Patent Owner's dependent claim arguments are conclusory and without support. Requester responds to these arguments by referring to the particularized citations to the prior art submitted with the original Request for Reexamination show that the cited prior art references render the dependent claims unpatentable.

Examiner is persuaded by Requester's rebuttal. Upon review, there appears to be no prosecution history suggesting that Webb, Kuck, and Bach (or Webb and Kuck) were not combinable. **Examiner** notes that Webb, Kuck, and Bach disclose cloning in a "runtime environment" embodied as virtual machines. **Examiner** asserts that Webb does not

"teach away" from the combination regarding multiple virtual machines and cloning, as Webb is directed to reducing overhead. **Examiner** asserts that Kuck does not "teach away" from the combination regarding "copy-on-write." Kuck allocates memory for a new virtual machine and copies initialization data, but does not copy values [Kuck shares some memories]. Kuck copies and initializes data from a master PAVM to a new PAVM. Allocating memory and copying are steps of a clone process.

Examiner disagrees with Patent Owner's comments that Bach's copy-on-write process does not involve an allocation of a memory block when a new process is forked. **Examiner** asserts that the forked child process does require memory, but copying of certain portions of the parent process memory storage are deferred /not immediately copied. The Webb / Kuck / Bach modification reduces both the system resources allocated to overhead and the memory space usage of the child virtual machines by using copy-on-write cloning.

Examiner asserts that the combination does not change the principle of operation of Kuck from simple shared memory to copy-on-write memory. To modify Kuck's process attached virtual machines, by preloading classes, as taught by Webb into Kuck's parent process attached virtual machine, thereby enables newly created child virtual machines to have a copy of the parent virtual machine in a pre-warmed state, thus reducing system overhead. Shared memory in a child process attached virtual machine, when allocated by an operating system optimization fork() call, is maintained until the child attempts to

modify write protected memory. Many operating systems use a copy-on-write fork() optimization to reduce overhead of cloning new virtual machine processes, as known in the art, as acknowledged by Patent Owner, and as disclosed by Bach.

Kuck's "runtime environment" is found in the process attachable virtual machines. Webb is relied upon to teach the preloading of classes. **Examiner** agrees with the Flinn Declaration, paragraph 41, Webb does not preclude multiple virtual machines. Bach is relied upon for teaching the copy-on-write optimization when cloning. While certainly the act of cloning to create child virtual machine process increases overhead, cloning using copy-on-write methods optimizes the increased overhead. There are many benefits achieved when creating multiple child virtual machines as recognized by the '720 patent (3: 21-40, ...several benefits...efficient concurrent application execution...sharing of preloaded classes...each child runtime system process executes in isolation of each other process...resource control...fast user application initialization and deterministic runtime behavior..." **Examiner** asserts that it was obvious to use copy-on-write "to reduce memory usage and overhead impact when instantiating and executing virtual machines" because many operating systems included this feature.

Regarding dependent claims 6 and 15, Examiner notes that the memory allocation and copying methods used when creating a PAVM are steps of the cloning process. Dependent claims 21 and 22 read on generated PAVMs / "child runtime system process"

which are provided with a copy the parent virtual machine's full set of initialization data on a "blank canvas."

Examiner agrees with Requester and maintains the rejections based on the combination of Webb, Kuck. and Bach.

Rejections based on the combination of Dike and Steinberg

Regarding the anticipation rejections based on the combination of Dike and Steinberg (Remarks 07/05/2011, p.17), **Patent Owner** asserts that neither the Office nor Requester describes the circumstances of a proper multiple reference anticipation rejection.

Patent Owner asserts that Dike is directed to a different technology than the '720 Patent. Dike discloses a Linux environment, which executes C and C++ programs that do not provide a representation that can be interpreted and instantiated as a class definition...C and C++ programs are simply compiled into machine code that is then executed. On the other hand, the "class preloader" provides "a representation of at least one class from a source definition provided as object-oriented program code" that is interpreted and instantiated as a class definition. A Linux virtual machine in machine code compiled from C and C++ has no need to be prewarmed, i.e., preloaded with the class data. Thus, the combination's Linux virtual machine does not lead to a "class preloader." **Patent Owner** cites to the Goldberg Declaration, paragraph 15.

Dike discloses (§2.2) the initialization of a system in preparation for booting up a user-mode kernel. However, there is nothing in this system initialization that involves a "class preloader." Dike's mention of the initialization being "analogous to the boot loader on a physical machine" is not the same as the "class preloader." The boot loader loads the programs that run the physical machine, without also performing preloading that "obtain[s] a representation of at least one class from a source definition provided as object-oriented program code."

Dike's disclosure (§2.2, §2.3, §3, Non Final Office Action 05/05/2011, p. 16) does not suggest that a parent process in the user-mode kernel has the ability "to interpret and to instantiate the representation [of at least one class] as a class definition" in the parent's address space. There can be no "master runtime system process" if there is no "class preloader" to provide the "representation [of at least one class] as a class definition" for interpreting and instantiation.

Dike (§2.3) simply discloses how the fork or clone system call is implemented. It has nothing to do with a "class definition." Dike also does not disclose that the new process is the "child runtime system process" whose copied address space is a copy of the "memory space [of the master runtime system process]" that includes a "class definition." Rather Dike makes no mention of the contents of that address space in this regard.

Dike does not disclose the "copy-on-write process cloning mechanism." Dike (§2.1) describes the address space having a shared memory segment that is converted from a copy-on-write memory segment in order to share kernel data. However, this is a disclosure not of using a "copy-on-write process cloning mechanism," but of the opposite. Dike's shared segment does not "defer copying" of the parent's memory space until the new process "needs to modify" the memory space. Dike prevents such deferring by making the address space a shared one. Moreover, Dike's shared segment is not the "memory space of the master runtime system process" because the address space's kernel data does not include a "class definition." Dike does not disclose the kernel data contents. As such, the skilled person reading Dike would have been led toward shared memory and away from copy-on-write memory.

Patent Owner asserts that the deficiencies of Dike are not corrected by Steinberg. In Steinberg, Fiasco-UX can include C++ code and can link to glibc, a C library for Linux (p. 14). However, the C++ code does not provide the "class preloader to obtain a representation of at least one class from a source definition provided as object-oriented program code." Steinberg's brief disclosures of C++ code and glibc are not associated with a "class preloader" or with a "master runtime system process." **Patent Owner** further asserts (Remarks 07/05/2011, p.19) that Dike and Steinberg are further deficient in disclosing the claimed "runtime environment": a runtime environment to clone the memory space [of the master runtime system process] as a child runtime system process responsive to a process request and to execute the child runtime system process.

Patent Owner asserts (Remarks 07/05/2011, pp.20-21) that Dike / Steinberg do not disclose the claimed copy-on-write process cloning mechanism: a copy-on-write process cloning mechanism to instantiate the child runtime system process by copying references to the memory space of the master runtime system process into a separate memory space for

the child runtime system process, and to defer copying of the memory space of the master runtime system process until the child runtime system process needs to modify the referenced memory space of the master runtime system process.

Patent Owner asserts (Remarks 07/05/2011, pp.21-22) that Dike and Steinberg are "directed to virtual machines in a Linux environment." As such, the skilled person reading Dike and Steinberg would have been led toward a Linux environment, which executes C and C++ programs that do not provide representations of a class that can be interpreted and instantiated as a class definition, and away from the '720 Patent.

Patent Owner asserts that dependent claims 4 and 13, directed to a "class resolver" are patentable because Dike fails to disclose the "class preloader" that obtains a "class definition" and therefore cannot disclose the "class resolver" to resolve the absent class definition. Claims 6 and 15 directed to a "process cloning mechanism" are also patentable because Dike's address space is a shared memory space, not a copied "memory space." Claims 21 and 22, directed to a "resource controller" are further patentable

because Dike does not disclose the requisite "child runtime system process," as described above, on which the "resource controller" operates.

Regarding the anticipation rejection, **Requester** opines (Third Party Comments, 08/04/2011, pp. 20-23) that the Dike reference provides an enabling disclosure; the Steinberg reference is provided merely to explain the meaning of certain of the Dike terms and to show that the copy-on-write characteristic is inherent in the Dike reference. These rationales support combining multiple references under 35 U.S.C. § 102. See MANUAL OF PATENT EXAMINING PRACTICE § 2131.01 (Rev. 7, Jul. 2008).

Requester notes that Dike discloses that the process cloning mechanism, e.g., the Linux fork() call, is implemented as copy-on-write, as evidenced by suggesting a further approach of creating a shared data segment from the copy-on-write segment containing kernel data, where such data needs to be shared across processes. Steinberg (p. 21) supports this by explaining that "[t]he normal means by which a Linux process can create new tasks is the fork() system call. This system call creates an exact copy of the calling process, which then becomes the calling process' child process. The created child process inherits copies of the parent process data space, heap, and stack, which are copied on demand using a copy-on-write mechanism."

Requester opines that Dike's disclosure of the claim elements with respect to C++ code is directly analogous to the '720 patent's claim elements with respect to Java. **Requester**

opines that Patent Owner's entire argument rests on the differences between C++ and Java code, where Patent Owner asserts that "Dike discloses a Linux environment, which executes C and C++ programs that do not provide a representation that can be interpreted and instantiated as a class definition." See Patent Owner Response at 18.

Requester asserts that the preferred embodiment of the '720 patent relies on a "class preloader." As recited above, the '720 patent explains the class preloader by noting that "[a]lthough described with specific reference to classes, other forms of structured static data could also be preloaded, including data structures, processes, functions, subroutines, interfaces, and the like. See '720 patent at 5:12-15. The broadest reasonable interpretation of the claims, especially in light of the directions provided in the specification, must extend beyond Java code to include code derived from other object-oriented and functional computer languages. **Requester** asserts (p. 22) that a class preloader is merely a tool that loads the code and data associated with object-oriented programming before the program needs the code and/or data, for example, before a first process request. See the Flinn Declaration at ¶42.

Dike's disclosed C++ programs undeniably contain classes, which are an abstraction of underlying code and related data structures. See Flinn Declaration at ¶43. C++ classes are compiled into dynamic libraries which are typically loaded before the code is executed or the data is referenced by the program. See Flinn Declaration at ¶44; see also Linkers & Loaders at, e.g., 11 (attached as Exhibit 8).

Requester asserts that the Dike / Steinberg combination discloses that the dynamic libraries preloaded into the master virtual machine are copied into the child virtual machine in their pre-loaded state. See Flinn Declaration at ¶44. Thus, the act of preloading a class (Java) is directly analogous to the act of linking and loading a dynamic library (C++). See id. at ¶44. **Requester** summarizes Patent Owner arguments: that C++ code does not involve a class definition, that the Dike in view of Steinberg combination does not disclose a class loader, a master runtime system process, a claimed runtime environment, or a copy-on-write process cloning mechanism.

Requester opines that a person having ordinary skill in the art seeking to improve Java performance would have immediately looked to other object-oriented languages, such as C++. See Flinn Declaration at ¶46. Patent Owner's own documents support this point. For example, Patent Owner's articles discussing scalable JVMs note that "Home programming tasks related to systems programming cannot be performed solely using the standard Java libraries. Developers might have to use native code, scripting languages, and C or C++ before returning to Java code." *See* Heiss at 1 (provided as Exhibit 9).

Requester asserts that Dike discloses both traditional copy-on-write memory segments as well as shared segments, contrary to Patent Owner's assertions at 22. *See* Dike at § 2.1.

Requester asserts that Patent Owner's dependent claim arguments are conclusory and without support. **Requester** responds to these arguments by referring to the particularized citations to the prior art that were submitted with the original Request for Reexamination.

Examiner agrees that the purpose of the Steinberg reference in the anticipation rejection was to properly evidence the teachings of Dike. **Examiner** agrees that the '720 specification and claim language broadly references object oriented program code, which includes C++. As an example, Dike (5.3.3) discloses the mm_struct object processed by a virtual machine, where an object is an instance of a class from a source definition provided as object-oriented program code. **Examiner** agrees with Requester and the Flinn Declaration that Dike's disclosed C++ programs contain classes, which are an abstraction of underlying code and related data structures, that C++ classes are compiled into dynamic libraries are typically loaded before the code is executed or the data is referenced by the program.

Examiner asserts that Dike (at 2.3) provides a master runtime system process (a set of Linux processes / a virtual machine, as a virtual kernel running on Linux) to interpret and instantiate the representation of a class definition in a memory space of the master runtime system process. "A Linus virtual machine (master runtime system process) is capable of running nearly all of the applications and services available on the host architecture (broadly loading and resolving a representation of a class, such as object

instances derived from a C++ class source definition)." A virtual machine interprets. Dike's fork or clone process (responsive to a process request) duplicates virtual machine system processes, creating a child runtime system process in a child runtime environment. "...the new address space (separate memory space for child's runtime system process / cloned child runtime environment) to be a copy of the parent address space, and the new process has the same registers as the old one... (share some memory space, until a modification is requested as confirmed by Steinberg's copy-on-write teachings)" See Dike at 2.1. "When a new process is created (child runtime environment)...creates a new process in the host for each new process in the virtual machine...system call in the forking process (clone) returns the pid (process identification / child process virtual machine) of the new process." (2.3) At 4.2 Dike explains that there are several motivations to want isolation (by use of multiple virtual machines)...to protect the physical machine and its resources from a potentially hostile process...to allocate machine resources...Running many virtual machines on a large server offers the advantages of a dedicated machine together with the administrative conveniences of having everything running on a single machine (conserving overhead).

Examiner opines that it would have been obvious to one of ordinary skill in the art at the time of the invention to combine Dike's disclosure, directed to virtual machines in a Linux environment, making using of a fork( ) system call, with Steinberg's disclosure of the Linux fork( ) system call and the explicit use of the copy-on-write mechanism. The Steinberg reference describes the copy-on-write mechanism of the Linux fork( ) system call that is disclosed by Dike: "The normal means by which a Linux process can create new tasks is the fork system call. This system call creates an exact copy (clone) of the

calling process, which then becomes the calling process' child process. The created child process inherits copies of the parent process data space, heap and stack, which are copied on demand using a copy-on-write mechanism." Steinberg at 21; see also Steinberg at 16 (describing the Linux clone functionality).

Examiner agrees with Requester's position and maintains the rejections based on Dike and Steinberg.

Rejections based on Bryant in view of Bach

Patent Owner asserts (Remarks 07/05/2011, pp. 21-22) that Bryant teaches away from or at least fails to provide any motivation to combine Bryant with any process, such as copy-on-write, that would reduce memory usage. Bryant discloses a Java server preloading all the classes that would be potentially needed by requesting applications. (2:46-54.) Bryant also discloses a child Java server selecting a subset of the preloaded classes when a particular Java application executes. (Id.) Clearly, by loading all potentially needed classes, Bryant is not concerned about memory usage reduction, having been implemented in a web server environment having substantial memory resources. Rather, Bryant is directed to faster startup at the expense of memory usage. (Bryant, 2:32-36, 46-63; 7:36-40) **Patent Owner** cites to the Goldberg Declaration, paragraph 16.

Patent Owner opines that the description in Bach (pp. 289-290) of the motivation for copy-on-write teaches away from a combination with Bryant. Bach notes that a call to exec soon after the fork call wastes address space (when not using a copy-on-write mechanism). **Patent Owner** asserts that Bryant's fork call is not followed by a call to an exec (thus there is no need for a copy-on-write mechanism). **Patent Owner** asserts that modifying Bryant to implement a copy-on-write memory would change the principle of operation of Bryant. The modification would require the operating system to change how it implements the fork system call. **Patent Owner** cites to the Goldberg Declaration, paragraphs 16-17.

Patent Owner asserts that Bryant already discloses streamlining and accelerating a Java machine by loading all potentially needed classes for faster startup and then selecting the actually needed classes therefrom. Given that copy-on-write technology was known as early as 1986, when Bach was published and twelve years before Bryant was filed as a patent application, it clearly was not obvious to use copy-on-write "to streamline and accelerate a Java machine." Otherwise, Bryant would have done so. **Patent Owner** cites to the Goldberg Declaration, paragraphs 16-18.

Patent Owner asserts (Remarks 07/05/2011, p. 24) that Bryant teaches away from copy-on-write as stated above. Second, computer scientists at the time did not think about using copy-on-write technology with process cloning for Java virtual machines. **Patent Owner** cites to the Goldberg Declaration, paragraph 18. Rather, as evidenced by the

cited references, other schemes were used to manage memory for multiple Java virtual machines or multiple Java processes. Thus, rather than cast its mind back to the time of invention to be guided by what the references teach, the Office has simply used the claimed elements as a blueprint or road map to find them in the prior art. This is impermissible. (MPEP 2145(X)(A).) Interconnect Planning Corp. v. Feil, 774 F.3d 1132, 1139.

Patent Owner asserts that dependent claims are patentable over the combination for at least the same reasons as their respective independent claims. Claims 4 and 13 directed to a "class resolver" are patentable because Bryant merely discloses that classes are preloaded and does not disclose that the classes are resolved. Claims 6 and 15 directed to a "process cloning mechanism" are patentable at least by virtue of the patentability of their independent claims. Claims 21 and 22 directed to a "resource controller" are also patentable because Bryant' s disclosure of the child Java server receiving information though a pipe connection from the Java server has nothing to do with a resource controller "to set operating system level resource management parameters."

Requester (Third Party Requester Comments 08/04/2011, pp. 12-15) notes that Patent Owner concedes that Bryant discloses a class preloader, a master runtime system process, and a runtime environment. **Requester** notes that Patent Owner concedes that Bach discloses a copy-on-write process cloning mechanism. Regarding Patent Owner's arguments asserting that Bryant "teaches away," **Requester** opines (p. 13) that "if the system of Bryant were built on one of the many UNIX operating systems implementing

copy-on-write (including Linux), then it would necessarily benefit from the copy-on-write technique. (For support Requester cites to Flinn, paragraphs 16-19; Schimmel at 10; Bovet at 225.) **Requester** opines that systems with substantial memory resources face significant resource constraints as they are designed to support the large and unrelenting workloads one would expect in a web server environment like the one addressed by Bryant. In a web server environment, more efficient memory management results in the ability to save expense by both providing fewer servers for a given demand for a service and by provisioning less memory per server...web server would still benefit from decreased startup times and overhead impacts. · "Thus, a person having ordinary skill in the art at the time of the invention would be inclined to combine the references to increase performance by reducing startup times and overhead impacts."

Requester opines (p. 14) that Bryant was filed in 1998 by engineers at the Hewlett Packard Company, which at leach as early as April 1997, had implemented copy-on-write technology in its commercial UNIX operating system, called HP-UX. Even if HP-UX was not the operating environment of the Bryant system, there would have motivation to make use of a copy-on-write enabled operating system environment to enable the system to manipulate processes more efficiently. **Requester** cites to the HP-US White Pater (at p.96) and the Flinn Declaration, paragraph 21 for support.

Requester asserts (p. 14) that because copy-on-write is an OS-level variant of the fork() command, the disclosure of Bryant could function within a copy-on-write enabled environment without changing a single line of its code. **Requester** asserts that Patent

Owner's noted difficulties involved in replacing (the Goldberg Declaration, paragraph 17) a "standard fork" with a "copy-on-write fork are completely without merit. HP was developing the Bryant system just after it had finished updating its HP-UX operating system environment to include copy-on-write technology. HP engineers developing the Bryant system had every reason to implement their improvements on the most current HP operating system so as to ensure compatibility and extend the useful life of their improvements. The fact that the HP-UX operating system had implemented copy-on-write technology more than a year before the filing data of the application disclosing the Bryant system shows the combination is not and cannot be based on impermissible hindsight. **Requester** cites to the Flinn Declaration, paragraphs 22-23 for support. Regarding the dependent claim arguments, **Requester** asserts that arguments are conclusory and without support. The citations in the Request render the dependent claims unpatentable.

Examiner notes the conflict in Patent Owner's argument: "...by loading all potentially needed classes (broadly resolves classes by loading referenced classes), Bryant is not concerned about memory usage reduction." In fact the '720 Specification ('720, 2: 66 – 3: 3: 1) recites "During initialization, the master runtime system process preloads classes and interfaces likely to be required by user applications at runtime." Bryant is as concerned about memory usage (memory access usage) as Patent Owner. Bryant discloses (2: 50-47-49) "...initialization of the Java virtual machine to speed up the actual execution ..." and (7: 38-40), "...it is faster to connect up to the Java server 160 and have it fork a child and execute the correct code than to load and execute the correct code."

Examiner asserts that Bryant is silent regarding whether his invention calls exec immediately following a fork() call. **Examiner** agrees that the assignee of Bryant's '367 patent, Hewlett-Packard Company, likely implemented the Bryant HP apparatus and method on an HP UNIX (HP-UX) operating system that by default includes copy-on-write technology as an operating system variant of the fork() system call (which addresses the quantity of memory used). When Bach was published and when Bryant was filed as a patent application, it clearly was obvious to use copy-on-write "to streamline and accelerate" a cloned process, because it was a default operating system call. Bach is relied upon for teaching copy-on-write as a default operating system call. Regarding the arguments directed towards a "resource controller," **Examiner** notes that environmental arguments (as cloned from parent process in system operating system) are received on the pipe connection. **Examiner** agrees with Requester's rebuttal and maintains the rejections based on the Bryant / Bach combination.

Rejections based on Bryant in view of Traut

Regarding the rejections based on the combination of Bryant and Traut, **Patent Owner** asserts (Remarks 07/05/2011, pp. 25-26) that Bryant and Traut are not combinable because: (a) Bryant teaches away from the combination regarding copy-on-write cloning; (b) the combination changes the principle of operation of Bryant regarding how the fork system call is implemented; (c) the reasons for the combination are insufficient to support a *prima facie* case of obviousness; (d) the combination is based on impermissible

hindsight that uses the '720 Patent as a road map to find these elements in the prior art; (e) Traut teaches away from the "copy-on-write process cloning mechanism" and toward a copy-on-access cloning mechanism for its child virtual machine; and (f) Traut and Bryant's virtual machines are too different to reasonably teach the skilled person how to transfer Traut's teachings to Bryant's virtual machine. The combination also does not disclose the dependent claims.

Patent Owner asserts that Traut describes its own system as using copy-on-write when the parent process modifies a page, not when the child process does so. ([0026], [0030]) Traut describes the child using an advantageous copy-on-access technique. ([0031], [0034]-[0035]; FIG. 3.) It is clear that Traut leads the skilled ...away from using copy-on-write for the child virtual machine. This is because Traut's goal is to copy all memory to the child either immediately or over time ([0032], [0039]) and copy-on-access achieves that goal. This is fundamentally different from the '720 Patent, in which the goal is to copy as needed. **Patent owner** cites to the Goldberg Declaration, paragraph 20. Had Traut intended that copy-on-write be used with the child virtual machine, Traut would certainly have done so, particularly in view of the parent virtual machine using copy-on-write.

Patent Owner asserts that Traut's virtual machine emulates a hardware machine ([0006]), whereas Bryant's Java virtual machine emulates a software machine. (1:61-65.) As an example, Traut's virtual machine interprets machine code, whereas Bryant's Java virtual machine interprets Java bytecodes. The skilled person looking to implement a

"copy-on-write process cloning mechanism" in a Bryant Java virtual machine would not have looked to Traut to do so. Patent Owner cites to Goldberg, paragraph 21.

For the reasons that claims 4, 6, 13, 15, 21, and 22 are not disclosed in Bryant and Bach, they are also not disclosed in Bryant and Traut. The remaining dependent claims are patentable over the combination for at least the same reasons as their respective independent claims.

Requester (Third Party Comments, 08/04/2011, pp. 15-16) rebuts Patent Owner's "teaches away" argument. Traut discusses [0026-0029] 'copy-on-write' and the combination of forking with the concept of a virtual machine to improve conversion of shared resources, when both virtual machines [parent and child] are running on the same host. Traut discloses an embodiment where the parent and child are located on the same host computer, where resource sharing is possible and so memory is kept as copy-on-write.

"Traut plainly points one of skill in the art towards use of the copy-on-write fork, and even discloses that this functionality is 'often' already included in the operating systems current in and around Traut's July 2002 filing date." **Requester** cites to the Flinn Declaration, paragraphs 25-28.

Requester asserts (p. 16) that Bryant and Traut are both directed to the emulation of a machine, a software machine and hardware machine, respectively. Requester opines the emulation of the two types of machines is so functionally similar that it is often taught in

successive sections with the same chapter of computer science textbooks. Requester cites to Silberschatz 2 at §§3.6.1-3.6.3 for supporting evidence and to the Flinn Declaration, paragraph 29 for support.

Requester asserts (p. 16) that Patent Owner's arguments directed towards dependent claims are conclusory and without support. **Requester** points to the citations submitted in the original Request.

Examiner opines that Traut teaches [0003-0006] a virtual machine, guest computer system that emulates the software instruction set mapped to the host processor. **Examiner** notes that Traut is directed towards [0011] "increasing the efficiency of virtual machine processing" and expressly discloses forking ("to create a child virtual machine at a new location without a least a first portion of the stored data," i.e., share some data). Citing an exemplary UNIX process, Traut explains [0026] that a fork copy on write technique shares pages that are not modified, thereby saving memory resources, an making the forking process faster. Traut discloses "demand paging." See FIG. 2 & [0026]: When either the parent virtual machine process or the child virtual machine process attempts an access to write to a write protected shared memory, a copy of the data is made for each virtual machine process (thus modifications to certain data are isolated within particular virtual machine environments). "Pages that are not modified can continue to be shared between the two processes." The type of code that a virtual machine executes is not narrowly construed. Claim language recites the code source as

object oriented program code. **Examiner** agrees with Requester and maintains the rejections based on the combination of Bryant / Traut.

Rejections based on the Srinivasan / Bach combination

Regarding the rejections based on the combination of Srinivasan and Bach, **Patent Owner** asserts (Remarks 07/05/2011, pp. 27-32) Srinivasan and Bach are not combinable because: (a) the combination fails to disclose any of the claimed elements, as the combination merely discloses simple Perl program code that describes how to create and use a Perl package, without disclosing the base element, i.e., the "class preloader," and hence the remaining claimed elements devised therefrom; and (b) the reasons for the combination are insufficient to provide a *prima facie* case of obviousness. Even if the claimed elements were added to Srinivasan, the skilled person would have no idea how to cobble them together to achieve the claimed invention, except by impermissibly using the '720 Patent as a road map.

Patent Owner asserts that Srinivasan is basically a programming manual to show the skilled person how to program various discrete concepts in Perl, a scripting language. These disparate sections mark a sum total of 10 pages scattered throughout different and sometimes "stand alone" chapters of a 400-page manual, with no teaching or suggestion how each of the separate elements should be used together. The combination also does not disclose the dependent claims.

Srinivasan (pp. 389-390, 98, 99, 101, 323- 324, 370) does not disclose or suggest a "class preloader to obtain a representation of at least one class from a source definition provided as object-oriented program code." Srinivasan, pp. 389- 390, merely discloses an example of Perl program code used to create and use an object package. Srinivasan does not disclose that the Perl program code includes a "class preloader." Srinivasan, p. 98, discloses differences between a Perl package and a Java package; p. 99, that Perl can be used to build objects; and p. 101, that Perl objects of certain types can belong to a class. Srinivasan, pp. 323-324, discloses that a Perl translator can convert a Perl script into opcodes that can be executed by a virtual machine. Srinivasan, p. 370, discloses that a Perl compiler can translate a Perl script into C code. Srinivasan's example Perl program code that executes a Unix fork system call (pp. 193-194) has nothing to do with a "representation as a class definition." The parent "environment" and "open file descriptors" in the parent's memory space are not a "representation as a class definition." Srinivasan does not disclose (pp. 321 & 323) how the multiple interpreters are created or that any of the interpreters is a "master" interpreter "to interpret and to instantiate the representation as a class definition in a memory space" of that interpreter. The interpreters' main and loaded packages are not the "memory space" contents. Because the above elements are absent from Srinivasan, the "master runtime system process" that recites them is also absent.

Patent Owner asserts that the combination of Srinivasan and Bach fail to disclose a "runtime environment" as a child runtime system process request and to execute the child runtime system process. Srinivasan does not disclose (pp. 193-195) the "class preloader,"

the "master runtime system process," and its "memory space," e.g., its memory space contents, as described above. **Patent Owner** cites to the Goldberg Declaration, paragraph 22. Srinivasan's disclosure of Perl program code to execute a fork system call has nothing to do with a "class definition" that would be contained in the forked memory space. That is, this section of Srinivasan does not disclose that the program code "clone[s] the memory space [of the master runtime system process having an interpreted and instantiated representation of a class definition]." Srinivasan's shows forking a process containing a socket, as written in Perl. Srinivasan and Bach fail to disclose the copy-on-write process cloning mechanism to instantiate the child runtime system process by copying references to the memory space of the master runtime system process [having an interpreted and instantiated representation of a class definition provided as object-oriented program code] into a separate memory space for the child runtime system process, and to defer copying of the memory space of the master runtime system process until the child runtime system process needs to modify the referenced memory space of the master runtime system process.

Patent Owner opines that Bach (pp. 192, 287,289-290) is directed to the forking process, not to the contents of the forked memory space. Because previously argued elements ("class preloader," the "master runtime system process," its "memory space," e.g., its memory space contents, and the "runtime environment") are not taught by the Srinivasan / Bach combination, the "copy-on-write process cloning mechanism" that recites them is also absent.

Patent Owner asserts (Remarks 07/05/2011, pp. 30-31) that the cited sections of Srinivasan would require a "leap" to cobble together and could not be done without guidance from the '720 Patent. **Patent Owner** cites to the Goldberg Declaration, paragraph 23. For example, the Action picks piecemeal from a Srinivasan section that describes Perl syntax (e.g., Appendix B, pp. 389-390), a section that describes object orientation (e.g., Chapter 7, pp. 99, 101), and a section that describes a Perl system (e.g., Chapter 19, pp. 323-324) in an attempt to find a disclosure of the "class preloader." The Action similarly picks from a Srinivasan section that describes Perl program code for networking (e.g., Chapter 12, pp. 193-194) and a section that describes a Perl system (e.g., Chapter 19, pp. 323-324) in an attempt to find a disclosure of the "master runtime system process." Similar attempts are made regarding the "runtime environment" and the "copy-on-write process cloning mechanism" as well. The skilled person could only know to combine the cited Srinivasan sections to achieve the claimed invention from Patent Owner's disclosure...impermissible hindsight and is not allowed as the basis for rejection.

Patent Owner asserts that there are insufficient reasons to establish a *prima facie* case of obviousness. Given that copy-on-write technology was known as early as 1986, when Bach was published and eleven years before Srinivasan was published, it clearly was not obvious to use copy-on-write "to further streamline the impact on system memory" (Non Final Office Action, 05/05/2011, p. 37) in Perl systems. Otherwise, Srinivasan would have done so. Moreover, there is no disclosure in either Srinivasan or Bach to suggest that the fork system call "commonly used in conjunction with the copy-on-write

mechanism" could be used "to clone the memory space of a master runtime system process [having an interpreted and instantiated representation of a class definition provided as object-oriented program code]." **Patent Owner** cites to the Goldberg Declaration, paragraphs 22-23.

Regarding the dependent claims, **Patent Owner** asserts that claims 4 and 13, directed to a "class resolver," are patentable because the Srinivasan/ Bach combination fails to disclose the "class preloader" that obtains a "class definition" (therefore cannot disclose the "class resolver" to resolve the absent class definition), claims 6 and 15, directed to a "process cloning mechanism" are patentable because Srinivasan fails to disclose the "memory space" (e.g., its interpreted and instantiated representation of a class definition and therefore cannot disclose the "process cloning mechanism" that copies the absent memory space), and claims 21 and 22, directed to a "resource controller" are patentable because Srinivasan does not disclose the requisite "child runtime system process," on which the "resource controller" operates.

Requester asserts (Third Party Comments 08/04/2011, pp. 23-26) that Patent Owner seeks to narrow the field of relevant prior art to only those disclosures dealing with Java computer code. **Requester** asserts that a person having ordinary skill in the art at the time of the invention would have certainly looked to Perl. "Srinivasan is an instructional computer science text; it merely expects you to know the essentials of Perl. See Srinivasan at xi. **Requester** posits that one of ordinary skill in the art at the time of the

invention, *i.e.,* December of 2003, would have been already in possession of or easily capable of putting together the basic Perl concepts outlined in the Srinivasan text. See the Flinn Declaration at paragraph 52. **Requester** and Flinn note that Perl is an object oriented computer language. Perl modules (analogous to classes) are loaded (as an example preload a Perl socket module into parent process) using "use" or "require" commands. Srinivasan discloses (pp. 194-195) the creation of a child process from a parent process using the fork() system call (to clone). Srinivasan (p. 87), "[t]he advantage of use is that when a program starts executing, there's a guarantee that all required modules have been successfully loaded (classes loaded and resolved)..."

Requester opines that a person having ordinary skill in the art seeking to improve Java performance would have looked to other object oriented languages, such as Perl. **Requester** cites to the Flinn Declaration, paragraphs 48-49 and Hess, p. 1 for support.

Examiner agrees with Requester, that the combination of Srinivasan and Bach provide an obvious teaching that maps to the claimed '720 invention and maintains the rejections.

Rejections based on the combination of Sexton and Bugnion

Regarding the rejections based on the combination of Sexton and Bugnion, **Patent Owner** asserts (Remarks 07/05/2011, pp. 32-36) the combination does not disclose, (a) the "runtime environment" (b) the "copy-on-write process cloning mechanism" because Bugnion, cited for this element, lacks the requisite cloning, i.e., "to clone," in a "runtime environment" and copy-on-write cloning.

Patent Owner submits that the Office has mistakenly equated "spawned" with "to clone." Sexton clearly describes "spawned" to mean instantiation, in which a virtual machine instance is created from a virtual machine class (i.e., a "VM data structure"). (8:3-5.) Sexton does not disclose that the "spawned" Java virtual machines are either all clones of an undesignated master Java virtual machine, or one of the virtual machines is the undesignated master and the others are clones. Sexton does not disclose virtual machine cloning. **Patent Owner** cites to the Goldberg Declaration, paragraph 24. Because the combination does not disclose the above elements, the "runtime environment" that recites them is also not disclosed.

The Sexton / Bugnion combination fails to teach: a copy-on-write process cloning mechanism to instantiate the child runtime system process by copying references to the memory space of the master runtime system process into a separate memory space for the child runtime system process, and to defer copying of the memory space of the master runtime system process until the child runtime system process needs to modify the referenced memory space of the master runtime system process.

While Bugnion discloses (6:29-36; 14:55-15:35; 15:66-16:1) each virtual machine referencing shared data, e.g., the code and buffer cache, in the machine memory and the machine memory having copy-on-write capabilities (14:55-15:25; FIG. 4), Bugnion does not disclose that any of the virtual machines is the result of cloning. **Patent Owner** cites to the Goldberg declaration, paragraph 25.

Bugnion does not disclose the "master runtime system process" that owns the machine memory or the "child runtime system process" that is cloned and that references and/or needs to modify the memory space of another virtual machine. Bugnion does not disclose that the machine memory is the "memory space" of any one of the virtual machines. The machine memory is an independent shared memory that each virtual machine can reference (or map to), but not own. (6:29-36.) Neither is a virtual machine's physical memory the claimed "memory space" because it is not accessible by any other virtual machine for referencing thereto and/or copying therefrom. (FIG. 4.) Bugnion further does not disclose the "copy-on-write process cloning mechanism." Bugnion merely discloses using copy-on-write disks for machine memory, with no indication that the copy-on-write disks are associated with cloning of the virtual machines. (14:55-15:35; 15:66- 16:1.) Because Bugnion does not disclose the above elements, Bugnion also does not disclose the "copy-on-write process cloning mechanism" that recites them. This deficiency is not corrected by Sexton, as acknowledged by the Action. (Action at 44.)

Sexton discloses a segmented memory model for the virtual machines, where session and call memories of each virtual machine are set up to store data to be modified by the virtual machine, and shared memory is set up to store data not to be modified by any virtual machine. (6:59-67; 7:1- 11, 31-45; FIGs. 2, 3.) Sexton leads the skilled person to provide individual memories, e.g., session memories, upon instantiation of a virtual machine so as to store data that the virtual machine intends to modify and, therefore, away from a shared memory that uses copy-on-write to store such data, fundamentally different from Bugnion, which initially sets up all the data (modifiable and unmodifiable)

in a global machine memory. Sexton did not consider sharing modifiable data between virtual machines, having set up the session and call memories. Accordingly, Sexton teaches away from the combination. Modifying Sexton to implement copy-on-write, would replace a segmented memory model having a read-only shared memory with a copy-on-write shared memory, thereby changing the principle of operation of Sexton. Patent Owner cites to the Goldberg Declaration, paragraphs 26-27.

Patent Owner asserts that Requester's reasons for the combination of Sexton and Bugnion are insufficient to establish a *prima facie* case of obviousness. Patent Owner disagrees with Non Final Office Action, (05/05/2011, p. 46) reasons for combining the prior art. The combination does not disclose all of the claimed elements, as described above. The skilled person reading Sexton would not consider modifying Sexton to include copy-on-write because Sexton has already set up a memory model to avoid unnecessary copying by initializing a shared memory area to store "necessary files" of Java classes. There need not be any further copying because the session memories store any data specific to that session's virtual machine. Sexton has reduced the session memory to the extent necessary, such that copy-on-write would be ineffective and unneeded.

It clearly was not obvious to use copy-on-write for "reducing session memory by sharing data between multiple Virtual Machines" to "only copying necessary files." Otherwise, Sexton would have done so. **Patent Owner** cites to the Goldberg Declaration,

paragraphs 24-27.

Regarding dependent claims 4 and 13 (directed to a "class resolver"), Sexton merely discloses that class data is shared and does not disclose that classes are resolved. Regarding dependent claims 6 and 15 (directed to a "process cloning mechanism"), Sexton does not disclose its Java virtual machines being the result of cloning, but merely the result of instantiating a virtual machine data structure. Bugnion does not disclose how its virtual machines are created. Regarding dependent claims 21 and 22 (directed to a "resource controller"), said claims are patentable at least by virtue of the patentability of their independent claims.

Requester notes (Third Party Comments 08/04/2011, p. 38) that Patent Owner admits (Patent Owner Remarks, p. 34) the Sexton & Bugnion prior art combination discloses "each virtual machine referencing shared data," which undercuts Patent Owner's assertion of a long felt need for a shared memory space. The Flinn Declaration does not address rejections based on the Sexton / Bugnion combination.

Examiner asserts that Sexton discloses "instantiating separate Java virtual machines for each session established by a server. Virtual machine instances can be created and run in separate units of execution that are managed by the operating system." (Abstract) Sexton, teaches ('114, 3: 21-23), "...the server may respond to the request by causing the Java virtual machine (master process) to spawn (allocate space and copy/instantiate) a second thread for executing the second Java program (child process). This action broadly

teaches "cloning" and "resolving" classes of a Java Virtual Machine operating on a server runtime environment. Broadly Examiner opines that the terms read on Sexton's teachings. **Examiner** cites to an Applicant response (12/18/2007, pp. 9-10) in reference to cloning / instantiation/ and copying and the '720 specification: "This cloning mechanism instantiates the child runtime system process by copying references to the memory space of the master runtime system process into a separate memory space for the child runtime system process."

Sexton ('114, 12: 1-23) discloses a first virtual machine instance (master runtime) and a second virtual machine instance (child runtime) that are "distinct instances of the same type of virtual machine" (i.e., cloned). Sexton discloses ('114, 12: 66-67) that as part of "spawning" "storing a pointer within said data structure (data structure of first virtual machine is spawned) to provide access to the data stored in the shared state area. Sexton discloses ('114, 13: 22-27) virtual machine instances share data stored in said shared state area allocated in volatile memory within said server. ('114, 14: 4-18), Virtual machines shares resources / stored values associated with an object class. Sexton discloses ('114, 14, 30-32) "...scheduling, for execution (by interpreting) in a system thread, the particular virtual machine instance..." Notably the '720 specification recites ('720, 4: 50-53), "Each operating system supports a process cloning mechanism that spawns multiple and independent isolated user applications by cloning the memory space of specifiable processes. An example ...fork() system call..." and ('720, 7: 58-61), "Initialization and execution of the application associated with the cloned JVM process 34 requires less time, as only the page table entries 62 are copied to clone the master JVM process

context." Sexton discloses ('114, 5: 34-37), "The separate VM instances can be created and run, for example, in separate units of execution that are managed by the operating system of the platform on which the server is executing." As discussed in prior arguments above, known operating system calls include clone() and fork(). Except for the copy-on-write feature, **Examiner** opines that Sexton provides an obvious disclosure of claim limitations.

Regarding Patent Owner's arguments asserting that session memories storing data as teaching away, **Examiner** notes that Sexton recognized the need to separate memory segments by providing for a shared state area read only memory [write protected, database instance memory 220, stores read-only data and instructions, e.g., bytecodes of the JAVA classes] ('114, 8: 40-64, to store methods, method table and fields) and a data structure within each individual [virtual machine session] session space is used to store session specific values ['114, 7: 10-11, i.e., Java class variables].

Regarding the arguments directed towards a "class resolver," **Examiner** notes that by virtue of creating VM instances, necessary classes are loaded, thus resolved.

Sexton recognized the need for shared and non-shared memory spaces. Given the goal of the reduction of overhead of Sexton (Sexton, 5: 53-65; 8: 40-64), it would be obvious to one of ordinary skill in the art to combine Sexton with the well-known copy on write technology, thereby placing the artisan in possession of the invention. Bugnion discloses that "[t]he virtual subnet and networking interfaces of Disco also use copy-on-write

mappings to reduce copying and to allow for memory sharing." (Bugnion, 6: 29-36; 14: 55-64; 14: 66- 15: 35; 15: 66 – 16: 1)

Rejections base on the combination of Sexton and Johnson

Requester and the Flinn declaration did not address Patent Owner's comments (Remarks 07/05/2011, pp. 36-39) regarding the rejections based on the combination of Sexton and Johnson.

Examiner notes that while Johnson (18: 34-44) appears to disclose shared address space (SAS) copy on write storage, the SAS shared storage is only shared among instances of a class running within a particular JVM. Johnson is silent regarding the sharing of copy on write memory between virtual machines. Johnson does not appear to provide an obvious teaching of:

> "a copy-on-write process cloning mechanism to instantiate the child runtime system process by copying references to the memory space of the master runtime system process into a separate memory space for the child runtime system process, and to defer copying of the memory space of the master runtime system process until the child runtime system process needs to modify the referenced memory space of the **master runtime system process." (similarly recited in all independent claims)**

Examiner withdraws the prior rejections of claims 1-8, 10-17, and 19-22 under 35 U.S.C. § 103(a) as obvious over Sexton in view of Johnson.

Secondary Considerations

Patent Owner asserts (Remarks 07/05/2011, pp. 39-41) that evidence of secondary considerations demonstrates the nonobviousness of the claimed invention. When

evaluating the nonobviousness of a claimed invention, such evidence must be considered, and "may often be the most probative and cogent evidence in the record." Stratoflex, Inc. v. Aeroquip Corp., 713 F.2d 1530, 1538 (Fed. Cir. 1983); Transocean Offshore Deepwater Drilling, Inc. v. Maersk Contractors USA, Inc., 2010 U.S. App. LEXIS 17181 at "14-'15 (Fed. Cir. August 18, 2010).

Patent Owner developed a commercial embodiment of the '720 Patent called Connected Device Configuration-Application Management System (CDC AMS). CDC AMS is a distributed system for launching and managing multiple applications that includes, inter alia, virtual machine instances, each application having its own instance, and copy-on-write technology for memory management of the virtual machine instances. (Appendix C, Runtime Guide at 5-1 through 5-12; Appendix B, Porting Guide at 12-1 through 12-12.) Patent Owner also presented CDC AMS at JavaOne, a premier Java conference. (Appendix J, JavaOne Presentation.)

Some customers requested a product embodying the claimed invention, e.g., CDC AMS, because of the performance and memory efficiencies therein. These customers wanted memory usage reduction, as provided by the "copy-on-write process cloning mechanism," for such applications as multi-task printing, e-reader technology, and VoIP telephony, because this reduction was a good fit for their multi-process environments.

Patent Owner asserts that since the implementation of Java virtual machines, there has been a need among system developers to have efficient use of memory between multiple

virtual machine processes, while providing a robust environment for executing the multiple virtual machine processes concurrently. Patent Owner cites to the Goldberg Declaration, paragraph 30, Appendix F, Shared Memory Blog and Appendix G, Reduced Footprint Biog.

The naive approach that was developed involved starting multiple instances of a Java virtual machine, each process having and executing its own virtual machine instance and each virtual machine instance having a distinct address space that is physically separate from that of other virtual machines. There were several drawbacks to this approach, mainly due to the dynamic nature of Java virtual machines. When each process got its own virtual machine, the initialization cost was repeated for each virtual machine, there was no shared memory between the processes, and common libraries were duplicated among processes. As a result, memory usage was inefficient. Patent Owner cites to the Goldberg Declaration, paragraph 30.

A later approach was developed to reduce the memory footprint associated with running concurrent Java virtual machines. In this approach, the virtual machines shared some memory between the processes, while maintaining individual memory spaces. This approach had its own drawbacks. Initialization costs were still repeated for each virtual machine and the individual memory spaces were not always necessary. Memory usage improved, but was still inefficient. (Id.) **Patent Owner** cites to Appendix L, Kawachiya Paper.

As smaller devices having limited or otherwise constrained memory resources emerged, these approaches were no longer viable because their memory usage requirements were greater than the device memory resources available. **Patent Owner** cites to the Goldberg Declaration, paragraph 30.

Recognizing this unsatisfied need for efficient memory usage, **Patent Owner** developed and patented a new approach to virtual machine memory management, as in the claimed invention of the '720 Patent, that used copy-on-write with process cloning to share memory between a master virtual machine and a cloned virtual machine until the cloned virtual machine needed to modify the shared memory. As a result, Patent Owner's approach was well suited for smaller devices having limited memory resources because it shared common libraries between processes, it did not have to repeat initialization costs for each cloned virtual machine, and it shared memory between processes by default. **Patent Owner** cites to the Goldberg declaration, paragraph 31.

Patent Owner opines (Remarks 07/05/2011, p. 41) that Google copied the claimed invention, presumably in order to come up with its competing Android software without having to invest the substantial time and resources that Patent Owner did in the claimed invention. **Patent Owner** cites to Appendix H, Complaint for Patent and Copyright Infringement Demand for Jury Trial, Oracle America, Inc v. Google, Inc, cv 10 – 03561 at 5-6, 8-9. Google's own public disclosures, e.g., websites, presentation videos and

slides, and open source code, describe Android software that has the functionality of the '720 patent, further evidencing copying of the claimed invention. Indeed, a side-by-side comparison (Appendix J) of Patent Owner' s CDC AMS description and Google's Android description and source code, with respect to claim 1 of the '720 Patent, demonstrates copying by Google.

Patent Owner additionally cites to (Remarks 07/05/2011, footnotes at p. 41):

Appendix I, Zygote. See also, generally, *http://www.developer.android.com.*

Appendix D, Android Presentation, and corresponding Video, "Google I/O 2008 - Anatomy and Physiology of an Android," by Patrick Brady, *http://developer.android, com/videos/index.html* (follow "Google I/O Sessions" tab; follow "Google I/O 2008 - Anatomy and Physiology of an Android" hyperlink; last visited June 29, 2011).

Appendix E, Dalvik Presentation, and corresponding Video, "Google I/O 2008 - Dalvik Virtual Machine Internals," by Dan Bornstein, *http://developer.android.com/videos/index.html* (follow "Google I/O Sessions" tab; follow "Google I/O 2008 - Dalvik Virtual Machine Internals" hyperlink; last visited June 29, 2011).

Appendix J, Copy Chart. See also, generally, *http://android.git.kernel.org.*

Requester (Third Party Comments 08/04/2011, pp. 26-30) opines that Dr. Goldberg's opinions (Goldberg Declaration, paragraph 30) concerning the knowledge and needs of

developers (the need among developers for efficient use of memory and the awareness among developers of disadvantages associated with Java virtual machines) is defective because Dr. Goldberg fails to define what level of ordinary skill in the art he employed in developing his opinions. **Requester** also noted Dr. Goldberg's references (Goldberg Declaration, paragraphs 18, 23, 27) to the "mindset at the time of the claimed invention" and opinions regarding the numerous prior art references. **Requester** states, "Because Dr. Goldberg fails to set forth his definition of the level of ordinary skill, it is impossible to evaluate whether his opinions have merit."

Requester opines that Patent Owner's invention was a "clear, obvious, and even trivial implementation as evidenced by Patent Owner's Appendix F (Bug ID: 4416624 report), noting a following developer comment/discussion (Exhibit 11) regarding the forking of new virtual machines as a trivial implementation.

Requester asserts that Patent Owner's admission (Patent Owner Remarks, 07/05/2011, p. 34) that the Sexton / Bugnion combination discloses "each virtual machine referencing shared data," undercuts the assertion of a long-felt need for a shared memory space.

Requester opines (p. 29) that Patent Owner's copying arguments are unsupported and conflict with their previous arguments and interpretations of the claim elements. **Requester** asserts that Google developed the accused technology independently, leveraging well-known techniques the prior art such as the Linux fork() system call. Whereas Patent Owner argues that the Google I/O 2008 NPL is evidence of copying

(Remarks 07/05/2011; p. 41), **Requester** notes that the Google I/O 2008 presentation was originally released prior to the issue date of the '720 patent (where the '023 application had been subject to a non-publication request). **Requester** asserts (p. 29) that Patent Owner's unsupported copying allegations provide none of the actual, direct evidence of copying required by Federal Circuit case law. *See Wyers v. Master Lock Co.,* 616 F.3d 1231, 1246 (Fed. Cir. 2010) (copying requires evidence such as "internal company documents, direct evidence such as disassembling a patented prototype...and using [it] as a blueprint to build a replica, or access to the patented product").

Requester questions Patent Owner's interpretation of the term "class preloader" as encompassing only a "Java class preloader," noting Patent Owner's Exhibit J cites to standard Java functionality. **Requester** notes that Patent Owner does not dispute that the accused Dalvik Virtual machine does not run Java bytecode. **Requester** notes that Patent Owner points to zygote bytecode that initializes a Dalvik virtual machine using copy-on-write (a non Java embodiment) as evidence that Requestor copied Patent Owner's invention. **Requester** asserts (p. 30) that the Dalvik virtual machine cannot fall within the claim scope and thus cannot demonstrate copying.

Examiner notes, in support of the assertion of a long felt need, that Patent Owner has submitted Appendix J to show a nexus between all independent claims, limitation by limitation with Patent Owner' s CDC AMS description and Google's Android description. **Examiner** agrees with Requester that "copying" evidence requires extensive

documentation (such as "internal company documents, direct evidence such as disassembling a patented prototype...and using [it] as a blueprint to build a replica, or access to the patented product). **Examiner** agrees with Requester that there is little support (considering the Sexton / Bugnion comments) for an argument based on a long felt need.

In addition, notably, an argument asserting a "long felt need" is improper against an anticipation rejection (Dike in view of Steinberg; Non Final Office Action 05/05/2011, beginning at p. 16). Evidence of secondary considerations, such as unexpected results or commercial success, is irrelevant to 35 U.S.C. 102 rejections and thus cannot overcome a rejection so based. *In re Wiggins*, 488 F.2d 538, 543, 179 USPQ 421, 425 (CCPA 1973).

Examiner agrees with Requester and Dr. Flinn regarding the Patent Owner's lack of description of one of ordinary skill in the art, at the time of the invention, with respect to the claimed scope of the invention. Thus, Patent Owner arguments, based on what one of ordinary skill in the art at the time of the invention would consider to be obvious, are difficult to access.

Examiner asserts that regarding the forking of new virtual machines (indicated as a trivial implementation), using the copy-on-write mechanism (which had already been known and, by default, in use in many operating systems), and the fact that it was known for multiple virtual machines to reference shared data, supports a conclusion of obviousness that a person of ordinary skill in the art (a person educated and / or with

work experience in programming languages, and operating system programming) would consider cloning virtual machines and sharing copy-on-write memory space to reduce system overhead and memory resources. **Examiner** agrees with Requester that a modification to implement a fork() copy-on-write mechanism would be trivial.

Examiner has weighed Patent Owner's and the Goldberg Declaration, but is not persuaded given the abundance of evidence presented by Requester's rebuttal.

In summary, the following rejections are maintained and incorporated by reference to the Non Final Office Action 05/05/2011:

1. Claims 1-8, 10-17, and 19-22 are unpatentable under 35 U.S. C. § 103(a) as rendered obvious by Webb in view of Kuck and further in view of APA-Bach

2. Claims 1-7, 10-16, and 19-22 are unpatentable under 35 U.S. C. § 102(b) as anticipated by Dike in view of Steinberg

3. Claims 1-7, 10-16, and 19-22 are unpatentable under 35 U.S. C. § 103(a) as rendered obvious by Dike in view of Steinberg

4. Claims 1-8, 10-17, and 19-22 are unpatentable under 35 U.S. C. § 103(a) as rendered obvious by Bryant in view of APA-Bach

5. Claims 1-8, 10-17, and 19-22 are unpatentable under 35 U.S. C. § 103(a) as rendered obvious by Bryant in view of Traut

6. Claims 1-8, 10-17, and 19-22 are unpatentable under 35 U.S. C. § 103(a) as rendered obvious by Srinivasan in view of APA-Bach

7. Claims 1-8, 10-17, and 19-22 are unpatentable under 35 U.S. C. § 103(a) as rendered obvious by Sexton in view of Bugnion

Examiner has withdrawn the following rejection:
Claims 1-8, 10-17, and 19-22 are unpatentable under 35 U.S. C. § 103(a) as rendered obvious by Sexton in view of Johnson

Claims 9 and 18 have not been requested for reexamination and have not been reexamined.

This is an ACTION CLOSING PROSECUTION (ACP); see MPEP § 2671.02.
(1) Pursuant to 37 CFR 1.951 (a), the patent owner may once file written comments limited to the issues raised in the reexamination proceeding and/or present a proposed amendment to the claims which amendment will be subject to the criteria of 37 CFR 1.116 as to whether it shall be entered and considered. Such comments and/or proposed amendments must be filed within a time period of 30 days or one month (whichever is longer) from the mailing date of this action. Where the patent owner files such comments

and/or a proposed amendment, the third party requester may once file comments under 37 CFR 1.951 (b) responding to the patent owner's submission within 30 days from the date of service of the patent owner's submission on the third party requester.

(2) If the patent owner does not timely file comments and/or a proposed amendment pursuant to 37 CFR 1.951 (a), then the third party requester is precluded from filing comments under 37 CFR 1.95 l(b).

(3) Appeal cannot be taken from this action, since it is not a final Office action.

Any comments considered necessary by PATENT OWNER regarding the above statement must be submitted promptly to avoid processing delays. Such submission by the patent owner should be labeled: "Comments on Statement of Reasons for Patentability and/or Confirmation" and will be placed in the reexamination file.

Extensions of time under 37 CFR 1.136(a) will not be permitted in *inter partes* reexamination proceedings because the provisions of 37 CFR 1.136 apply only to "an applicant" and not to the patent owner in a reexamination proceeding. Additionally, 35 U.S.C. 314(c) requires that inter partes reexamination proceedings "will be conducted with special dispatch" (37 CFR 1.937). Patent owner extensions of time in inter partes reexamination proceedings are provided for in 37 CFR 1.956. Extensions of time are not available for third party requester comments, because a comment period of 30 days from service of patent owner's response is set by statute. 35 U.S.C. 314(b)(3).

Any paper filed with the USPTO, i.e., any submission made, by either the Patent Owner or the Third Party requester must be served on every other party in the reexamination

proceeding, including any other Third Party requester that is part of the proceeding due to merger of the reexamination proceedings. As proof of service, the party submitting the paper to the Office must attach a Certificate of Service to the paper, which sets forth the name and address of the party served and the method of service. Papers filed without the required Certificate of Service may be denied consideration. 37 CFR 1.903; MPEP 2666.06.

The Patent Owner is reminded that any proposed amendment to the specification and/or claims in this reexamination proceeding must comply with 37 CFR 1.530(d)-(j), must be formally presented pursuant to 37 CFR 1.52(a) and (b), and must contain any fees required by 37 CFR 1.20(c).

Amendments in an *inter partes* reexamination proceeding are made in the same manner that amendments in an *ex parte* reexamination are made. MPEP 2666.01. See MPEP 2250 for guidance as to the manner of making amendments in a reexamination proceeding.

The Patent Owner is reminded of the continuing responsibility under 37 CFR 1.985(a), to apprise the Office of any litigation activity, or other prior or concurrent proceeding, involving the instant Patent Under Reexamination or any related patent throughout the course of this reexamination proceeding. The Third Party requester is also reminded of the ability to similarly inform the Office of any such activity or proceeding throughout the course of this reexamination proceeding. See MPEP §§ 2686 and 2286.04.

All correspondence relating to this Inter Partes reexamination proceeding should be directed:

By EFS: Registered users may submit via the electronic filing system EFS-Web, at https://efs.uspto.gov/efile/myportal/efs-registered

By Mail to: Mail Stop Inter Partes Reexam
Attn: Central Reexamination Unit
Commissioner for Patents United States Patent & Trademark Office
P.O. Box 1450
Alexandria, Virginia 22313-1450

By FAX to: (571) 273-9900
Central Reexamination Unit

By hand: Customer Service Window
Attn: Central Reexamination Unit
Randolph Building, Lobby Level
401 Dulany Street
Alexandria, VA 22314

Any inquiry concerning this communication or earlier communications from the examiner, or as to the status of this proceeding, should be directed to the Central Reexamination Unit at telephone number (571) 272-7705.

/Mary Steelman/

Mary Steelman, Primary Examiner

Central Reexamination Unit - Art Unit 3992

(571) 272-3704

Conferees:

/EBK/

MARK J. REINHART
CRU SPE-AU 3992