(12) PATENT
(19) AUSTRALIAN PATENT OFFICE

(11) Application No. AU 200115632 B2
(10) Patent No. 768181

(54) Title
A method for garbage collection of unused methods

(51)6 International Patent Classification(s)
G06F 012/02

(21) Application No: 200115632

(22) Application Date: 2000.10.27

(87) WIPO No: WO01/31455

(30) Priority Data

(31) Number	(32) Date	(33) Country
9903890	1999.10.28	SE

(43) Publication Date : 2001.05.08
(43) Publication Journal Date : 2001.07.26
(44) Accepted Journal Date : 2003.12.04

(71) Applicant(s)
BEA Systems Stockholm Engineering AB

(72) Inventor(s)
Joakim Dahlstedt

(74) Agent/Attorney
Davies Collison Cave,1 Little Collins Street,MELBOURNE VIC 3000

(56) Related Art
WO 99/32978
US 5915255
US 5088036

AU 200115632

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
3 May 2001 (03.05.2001)

PCT

(10) International Publication Number
WO 01/31455 A1

(51) International Patent Classification?: G06F 12/02

(21) International Application Number: PCT/SE00/02096

(22) International Filing Date: 27 October 2000 (27.10.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
9903890-3 28 October 1999 (28.10.1999) SE

~~(71) Applicant (for all designated States except US): APPLAL VIRTUAL MACHINES AB [SE/SE]; Box 2189, S-103 15 Stockholm (SE).~~

(72) Inventor; and
(75) Inventor/Applicant (for US only): **DAHLSTEDT, Joakim** [SE/SE]; Flintbacken 10, S-118 42 Stockholm (SE).

(74) Agents: **ÖRTENBLAD, Bertil** et al.; Noréns Patentbyrå AB, Box 10198, S-100 55 Stockholm (SE).

(71) BCA Systems Stockholm Engineering AB
Box 4213, 102 65 Stockholm, Sweden

AUSTRALIAN PATENT OFFICE SEC 104

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *With international search report.*
— *Before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments.*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

WO 01/31455 A1

(54) **Title:** A METHOD FOR GARBAGE COLLECTION OF UNUSED METHODS

(57) **Abstract:** A method for improving the effectiveness of a data processing application when using a virtual machine, wherein the program includes a large number of methods, i.e. program sections, that are stored in the memory of the computer used, and wherein the program uses a garbage collecting procedure. The invention is characterised by analysing in a first step all so called thread stacks with respect to the procedures required thereby; causing each of said requisite methods to be regenerated in a second step, wherein occurring references to a method prior to the regeneration are replaced with references to regenerated methods; and by erasing all non-regenerated methods in a third step and placing corresponding memory space at the disposal of the program.

A METHOD FOR GARBAGE COLLECTION OF UNUSED METHODS

The present invention relates to a method of improving the effectiveness of a data processing application.

More specifically, the invention is concerned with increasing the data processing rate in virtual machines, and then particularly with respect to the Java program language.

The invention is not restricted to Java, but can be applied with many program languages, although the invention is described below primarily with reference to Java.

The method is intended for use with adaptive optimisation of a program. In adaptive optimisation, the program is restructured and different parts of the program are optimised as the program is run. The general problem of increasing data processing capacity resides in the rapid creation of new memory sites, since the longer the program is run, the more memory space is required.

Java and other dynamic program languages include an automatic memory management. This means that the programmer need not keep an account of those parts of the memory that are used. The virtual machine carries out a so-called garbage collection from time to time, meaning, in principle, that the virtual machine scans the entire memory and finds which objects have been stored in the memory and which the program can no longer address. These parts of the memory are returned for later use.

Java also includes methods for so called thread management methods. Thus, Java incorporates a system for supporting or simulating the simultaneous processing of two or more programs. The thread management can be divided into two parts. One part concerns the manner in which different threads are structured in a controlled manner. Another part is concerned with which threads shall be run and which threads shall be passive and wait to be run.

In order to further increase effectiveness and place occupied memory space at the disposal of the program, it is not sufficient to solely optimise the memory with respect to the objects.

The present invention attempts to address or ameliorate this problem, or to at least provide a useful alternative to the prior art.

The reference to any prior art in this specification is not, and should not be taken as, an acknowledgment or any form of suggestion that that prior art forms part of the common general knowledge in Australia.

The present invention provides a method for improving the effectiveness of an application program when using a virtual machine or runtime environment, wherein said application program includes a large number of methods that are stored in the memory of the computer used, and wherein said application program uses a garbage collecting procedure, characterised by

analyzing in a first step all so called thread stacks with respect to the methods required thereby;

causing each of said requisite methods to be regenerated in a second step, wherein occurring references to a method prior to regeneration are replaced with references to regenerated methods; and

erasing all non-regenerated methods in a third step and placing corresponding memory space at the disposal of the application program.

The present invention further provides a process for optimizing the distribution of methods within the memory space of a computer system or virtual machine, comprising the steps of:

analyzing each method to determine a selection of methods to be regenerated;

creating regenerated methods, and replacing references to each method with references to the regenerated method; and,

erasing all non-regenerated methods and releasing the memory space occupied by those non-regenerated methods.

The present invention further provides a system for optimizing the distribution of methods within the memory space of a computer system, including a virtual machine or runtime environment operating thereupon, wherein the virtual machine or runtime environment provides a plurality of methods, running in threads, that can be accessed by other methods, comprising:

computer readable instructions which when executed perform the steps of

analyzing thread stacks associated with each method to determine a selection of methods to be regenerated;

creating regenerated methods, and replacing references to each method with references to the regenerated method, and,

erasing all non-regenerated methods and releasing the memory space occupied by those non-regenerated methods.

The present invention further provides a method for use with a computer for improving the effectiveness of a data processing application or program using a Java Virtual Machine or runtime environment, wherein the data processing application or program uses a large number of methods that are stored in the memory of the computer, and wherein the method uses a garbage collecting procedure, comprising the steps of:

analyzing all thread stacks with respect to the methods required thereby;

causing each of said requisite methods to be regenerated, wherein occurring references to a method prior to regenerations are replaced with references to regenerated methods; and,

erasing all non-regenerated methods and releasing the corresponding memory space.

The present invention further provides a system for improving the effectiveness of a data processing application or program using a Java Virtual Machine or runtime environment, wherein the data processing application or program uses a large number of methods that are stored in the memory of the computer, comprising:

a processor; and,

processor-readable instructions which when executed cause the processor to

perform the steps of

analyzing all thread stacks with respect to the methods required thereby;

causing each of said requisite methods to be regenerated, wherein occurring references to a method prior to regeneration are replaced with references to regenerated methods; and,

erasing all non-regenerated methods and releasing the corresponding memory space.

The present invention also provides a processor readable medium having instructions thereon which when executed cause the processor to perform the steps of

analyzing each of a plurality of methods within the memory space of a computer system or virtual machine to determine a selection of methods to be regenerated;

creating regenerated methods, and replacing references to each method with references to the regenerated method; and,

erasing all non-regenerated methods and releasing the memory space occupied by those non-regenerated methods.

Embodiments of the present invention will now be described in more detail partly with reference to an exemplifying embodiment of the invention shown in the accompanying drawings, in which

Figure 1 is a block diagram;

Figure 2 illustrates old methods; and

Figure 3 illustrates methods regenerated in accordance with an embodiment of the invention.

Figure 1 shows that a Java virtual machine, JVM, can be used to run different data programs 1, 2, 3, regardless of whether the operative system is Win NT, LINUX, Solaris or some other system. As mentioned above, although Java is a very popular program

language, the present invention is not restricted to this language but can be applied to all object-orientated and platform-independent corresponding program languages.

Embodiments of the present invention thus relate to a method of improving the effectiveness of a data processing application when using a virtual machine, wherein the program includes a large number of methods, i.e. program sections, that are stored in the memory of the computer used, and wherein a garbage collecting process is used by the program.

It is previously known to garbage collect objects and therewith erase objects that are no longer in current use thereby placing corresponding memory capacity at disposal.

In large systems, many methods, i.e. program sections, are used one or a few times, or methods are applied for a short period of time and then left unused.

In the case of Java and corresponding programs, new methods are loaded and old methods left unused.

Furthermore, adaptive optimisation results in the optimisation and re-optimisation of methods placed in the memory, where old methods are left unused.

When optimising lock mechanism selection and garbage collection selection, it is necessary to replace all used methods that use old mechanisms with new mechanisms.

According to the invention, all so called thread stacks are analysed with respect to the methods required, in a first step of the inventive method. In a second step, each of the methods required is regenerated, where occurrent references to a method are replaced with references to regenerated methods prior to said regeneration. In a third step, all non-regenerated methods are erased and the corresponding memory space placed at the disposal of the program.

This procedure does not only clean-out unused methods, but also results in a reorganisation between those methods that have been regenerated, so as to direct references of the methods immediately to a regenerated method instead of proceeding via an old method that is no longer used.

This is illustrated in Figures 2 and 3, of which Figure 2 illustrates old methods and Figure 3 illustrates used regenerated methods. Three methods foo, apa and bar are shown in Figure 2. Foo starts on the memory address 4711. Apa starts on the address 4714 and bar starts on the address 4720.

Analysis of the thread stacks shows that only the methods foo and bar are used, and consequently foo and bar have not been referenced to the method apa.

The method foo and bar are regenerated to those methods illustrated in Figure 3. In this case, the methods foo and bar are recreated precisely, although with the difference that the methods obtain new addresses and that then the foo reference to bar points to the new bar address 4903.

All old methods, i.e. the methods in Figure 2, are erased and the memory spaces previously occupied by these methods are vacated for further use.

When garbage collection of objects takes place, running of the program normally stops while garbage collection takes place. Running of the program is restarted subsequent to the garbage collection and to the erasure of objects that are not in use.

Such a method can be used when applying the present invention.

However, it is very much preferred to use the following method instead.

When practicing the inventive method, one thread is stopped at a time whilst the program is running, wherewith methods used for a stopped thread are transferred to a list and the thread then restarted. The methods in the list are then regenerated and stored. All threads are later caused to be stopped at the same time, subsequent to having treated all threads in this way, namely so that all used methods relating to the threads concerned have been regenerated. All methods that have not been regenerated are erased and all threads are restarted with the regenerated methods.

This method obviates the need to stop running the program, since the regeneration takes place intermittently.

As before mentioned, lock mechanisms are used in Java and corresponding languages. Different lock mechanisms can be selected. The important thing is to select the lock mechanism that is the most effective in preventing more than one thread having access to a given object at the same time as another thread.

A synchronisation problem exists when several threads desire access to one and the same object or source. In order to solve this problem in Java, each thread endeavours to reach the source lock. The source lock mechanism can be used in various ways. The effectiveness of different lock mechanisms will depend on how threads endeavour to obtain access to synchronised sources.

According to a preferred embodiment, when locking mechanisms are used the most effective locking mechanisms are identified in a step prior to said first step, and the methods that use a thus identified locking mechanism are regenerated.

With respect to garbage collecting algorithms, these also need to be selected. Many object orientated languages use garbage collection. This means that the programmer need not instruct the system explicitly that a certain object is no longer required. The system is responsible for this detection, and reclaims the part of the memory occupied by the object. A number of different algorithms have been proposed for effective implementation of this detection and reclaim. It has been found that different algorithms are best for different applications. The choice of the best garbage collecting algorithm for the program application being run is highly significant in achieving maximum execution rate in respect of the program concerned.

According to another preferred embodiment of the invention, when different garbage collecting algorithms are used the allocation and length of life of the various objects are determined in a step prior to said first method step, whereafter the most effective garbage collecting algorithm is caused to be identified and the methods constituting the requisite garbage collecting algorithms are regenerated and remaining garbage collecting algorithms then erased.

Application of the preferred embodiments provides a highly effective method for optimising codes, threads and memory management, where a generic feature resides in the identification and regeneration of methods so as to not load the system with unused methods.

Throughout this specification and the claims which follow, unless the context requires otherwise, the word "comprise", and variations such as "comprises" and "comprising", will be understood to imply the inclusion of a stated integer or step or group of integers or steps but not the exclusion of any other integer or step or group of integers or steps.

5

THE CLAIMS DEFINING THE INVENTION ARE AS FOLLOWS:

1. A method for improving the effectiveness of an application program when using a virtual machine or runtime environment, wherein said application program includes a large number of methods that are stored in the memory of the computer used, and wherein said application program uses a garbage collecting procedure, characterised by

analyzing in a first step all so called thread stacks with respect to the methods required thereby;

causing each of said requisite methods to be regenerated in a second step, wherein occurring references to a method prior to regeneration are replaced with references to regenerated methods; and

erasing all non-regenerated methods in a third step and placing corresponding memory space at the disposal of the application program.

2. A method according to claim 1, further including causing one thread at a time to be stopped as the application program is run; transferring methods used for a stopped thread to a list, and thereafter restarting the thread; regenerating and storing the methods in said list, causing all threads to be stopped simultaneously subsequent to having treated all threads in said fashion; erasing all methods that have not been regenerated and restarting all threads with the regenerated methods.

3. A method according to claim 1 or 2, further including, when locking mechanisms are used, identifying the most effective locking mechanism in a step prior to said first method step; and regenerating those methods that use a thus identified locking mechanism.

4. A method according to claim 1, 2 or 3, further including, when different garbage collecting algorithms are used, determining the allocation and length of life of the various objects in a step prior to said first method step, and then identifying the most effective garbage collecting algorithm; regenerating the methods constituting the requisite garbage collecting algorithms; and erasing remaining garbage collecting algorithms.

5. A process for optimizing the distribution of methods within the memory space of a computer system or virtual machine, comprising the steps of:

analyzing each method to determine a selection of methods to be regenerated;

creating regenerated methods, and replacing references to each method with references to the regenerated method; and,

erasing all non-regenerated methods and releasing the memory space occupied by those non-regenerated methods.

6. The process of claim 5 for optimizing the distribution of methods within the memory space of a computer system or virtual machine, wherein said step of analyzing includes the substeps of, for each of a selection or all of the threads

stopping each thread one at a time,

transferring methods used for the stopped thread to a regenerate list, and

restarting the stopped thread; and,

wherein said step of creating regenerated methods includes a step of

regenerating the methods in said regenerate list; and,

wherein said step of erasing includes the substeps of

stopping all threads simultaneously,

erasing all methods that have not been regenerated, and,

restarting those threads with the regenerated methods.

7. The process of claim 5 for optimizing the distribution of methods within the memory space of a computer system or virtual machine, including, when locking mechanisms are used, the steps of:

identifying the most effective locking mechanism; and,

regenerating those methods that use the identified locking mechanism.

8. The process of claim 6 for optimizing the distribution of methods within the memory space of a computer system or virtual machine, including, when locking mechanisms are used, the steps of:

identifying the most effective locking mechanism; and,

regenerating those methods that use the identified locking mechanism.

9. The process of claim 5 for optimizing the distribution of methods within the memory space of a computer system or virtual machine, including, when different garbage collecting algorithms are used, the steps of:

determining the allocation and lifetime of the objects in use;

identifying the most effective garbage collecting algorithm;

regenerating the methods that use the identified garbage collecting algorithm; and,

erasing the remaining garbage collecting algorithms.

10. The process of claim 6 for optimizing the distribution of methods within the memory space of a computer system or virtual machine, including, when different garbage collecting algorithms are used, the steps of:

determining the allocation and lifetime of the objects in use;

identifying the most effective garbage collecting algorithm;

regenerating the methods that use the identified garbage collecting algorithm; and,

erasing the remaining garbage collecting algorithms.

11. The process of claim 7 for optimizing the distribution of methods within the memory space of a computer system or virtual machine, including, when different garbage collecting algorithms are used, the steps of:

determining the allocation and lifetime of the objects in use;

identifying the most effective garbage collecting algorithm;

regenerating the methods that use the identified garbage collecting algorithm; and,

erasing the remaining garbage collecting algorithms.

12. The process of claim 8 for optimizing the distribution of methods within the memory space of a computer system or virtual machine, including, when different garbage collecting algorithms are used, the steps of:

determining the allocation and lifetime of the objects in use;

identifying the most effective garbage collecting algorithm;

regenerating the methods that use the identified garbage collecting algorithm; and,

erasing the remaining garbage collecting algorithms.

13. A system for optimizing the distribution of methods within the memory space of a computer system, including a virtual machine or runtime environment operating thereupon, wherein the virtual machine or runtime environment provides a plurality of methods, running in threads, that can be accessed by other methods, comprising:

computer readable instructions which when executed perform the steps of

analyzing thread stacks associated with each method to determine a selection of methods to be regenerated;

creating regenerated methods, and replacing references to each method with references to the regenerated method, and,

erasing all non-regenerated methods and releasing the memory space occupied by those non-regenerated methods.

14. The system of claim 13, wherein said computer readable instructions further include instructions which when executed perform the additional steps of, for each of a selection or all of the threads:

stopping each thread one at a time,

transferring methods used for the stopped thread to a regenerate list,

restarting the stopped thread, and,

regenerating the methods in said regenerate list; and,

subsequent to having treated all threads in said fashion, the additional steps of

stopping all threads simultaneously,

erasing all methods that have not been regenerated, and,

restarting those threads with the regenerated methods.

15. The system of claim 13, including, when locking mechanisms are used, computer readable instructions which when executed perform the additional steps of:

identifying the most effective locking mechanism; and

regenerating those methods that use the identified locking mechanism.

16. The system of claim 14, including, when locking mechanisms are used, computer readable instructions which when executed perform the additional steps of:

identifying the most effective locking mechanism; and,

regenerating those methods that use the identified locking mechanism.

17. The system of claim 13, including, when different garbage collection mechanisms are used, computer readable instructions which when executed perform the additional steps of:

determining the allocation and lifetime of the objects in use;

identifying the most effective garbage collecting algorithm;

regenerating the methods that use the identified garbage collecting algorithm; and,

erasing the remaining garbage collecting algorithms.

18. The system of claim 14, including, when different garbage collection mechanisms are used, computer readable instructions which when executed perform the additional steps of:

determining the allocation and lifetime of the objects in use;

identifying the most effective garbage collecting algorithm;

regenerating the methods that use the identified garbage collecting algorithm; and,

erasing the remaining garbage collecting algorithms.

19. The system of clam 15, including, when different garbage collection mechanisms are used, computer readable instructions which when executed perform the additional steps of:

determining the allocation and lifetime of the objects in use;

identifying the most effective garbage collecting algorithm;

regenerating the methods that use the identified garbage collecting algorithm; and,

erasing the remaining garbage collecting algorithms.

20. The system of claim 16, including, when different garbage collection mechanisms are used, computer readable instructions which when executed perform the additional steps of:

determining the allocation and lifetime of the objects in use;

identifying the most effective garbage collecting algorithm;

regenerating the methods that use the identified garbage collecting algorithm; and,

erasing the remaining garbage collecting algorithms.

21. A method for use with a computer for improving the effectiveness of a data processing application or program using a Java Virtual Machine or runtime environment, wherein the data processing application or program uses a large number of methods that are stored in the memory of the computer, and wherein the method uses a garbage collecting procedure, comprising the steps of:

analyzing all thread stacks with respect to the methods required thereby;

causing each of said requisite methods to be regenerated, wherein occurring references to a method prior to regenerations are replaced with references to regenerated methods; and,

erasing all non-regenerated methods and releasing the corresponding memory space.

22. The method according to claim 21, including the steps of:

causing one thread at a time to be stopped;

transferring methods used for a stopped thread to a list;

restarting the stopped thread;

regenerating and storing the methods in said list;

causing all threads to be stopped simultaneously subsequent to having treated all threads in said fashion; and

erasing all methods that have not been regenerated, and restarting all threads with the regenerated methods.

23. The method according to claim 21, including, when locking mechanisms are used,

the steps of:

identifying the most effective locking mechanism; and,

regenerating those methods that use the identified locking mechanism.

24. The method according to claim 23, including, when locking mechanisms are used, the steps of:

identifying the most effective locking mechanism; and,

regenerating those methods that use a thus identified locking mechanism.

25. The method according to claim 21, including, when different garbage collecting algorithms are used, the steps of:

determining the allocation and length of life of the various objects;

identifying the most effective garbage collecting algorithm;

regenerating the methods constituting the requisite garbage collecting; and,

erasing the remaining garbage collecting algorithms.

26. The method according to claim 22, including, when different garbage collecting algorithms are used, the steps of:

determining the allocation and length of life of the various objects;

identifying the most effective garbage collecting algorithm;

regenerating the methods constituting the requisite garbage collecting; and,

erasing the remaining garbage collecting algorithms.

27. The method according to claim 23, including, when different garbage collecting algorithms are used, the steps of:

determining the allocation and length of life of the various objects;

identifying the most effective garbage collecting algorithm;

regenerating the methods constituting the requisite garbage collecting; and,

erasing the remaining garbage collecting algorithms.

28. The method according to claim 24, including, when different garbage collecting

algorithms are used, the steps of:

determining the allocation and length of life of the various objects;

identifying the most effective garbage collecting algorithm;

regenerating the methods constituting the requisite garbage collecting; and,

erasing the remaining garbage collecting algorithms.

29. The process of claim 5 for optimizing the distribution of methods within the memory space of a computer system or virtual machine, wherein said step of analyzing includes analyzing thread stacks associated with each method.

30. A system for improving the effectiveness of a data processing application or program using a Java Virtual Machine or runtime environment, wherein the data processing application or program uses a large number of methods that are stored in the memory of the computer, comprising:

a processor; and,

processor-readable instructions which when executed cause the processor to perform the steps of

analyzing all thread stacks with respect to the methods required thereby;

causing each of said requisite methods to be regenerated, wherein occurring references to a method prior to regeneration are replaced with references to regenerated methods; and,

erasing all non-regenerated methods and releasing the corresponding memory space.

31. The system according to claim 30, wherein the processor-readable instructions further include instructions which when executed cause the processor to perform the steps of

causing one thread at time to be stopped;

transferring methods used for a stopped thread to a list;

restarting the stopped thread;

regenerating and storing the methods in said list;

causing all threads to be stopped simultaneously subsequent to having treated all threads in said fashion; and,

erasing all methods that have not been regenerated, and restarting all threads with the regenerated methods.

32. The system according to claim 30, wherein the processor-readable instructions further include instructions which when executed cause the processor to perform the steps of

identifying the most effective locking mechanism; and,

regenerating those methods that use the identified locking mechanism.

33. The system according to claim 31, wherein the processor-readable instructions further include instructions which when executed cause the processor to perform the steps of

identifying the most effective locking mechanism; and,

regenerating those methods that use a thus identified locking mechanism.

34. The system according to claim 30, wherein the processor-readable instructions further include instructions which when executed cause the processor to perform the steps of

determining the allocation and length of life of the various objects;

identifying the most effective garbage collecting algorithm;

regenerating the methods constituting the requisite garbage collecting; and,

erasing the remaining garbage collecting algorithms.

35. The system according to claim 31, wherein the processor-readable instructions further include instructions which when executed cause the processor to perform the steps of

determining the allocation and length of life of the various objects;

identifying the most effective garbage collecting algorithm;

regenerating the methods constituting the requisite garbage collecting; and,

erasing the remaining garbage collecting algorithms.

36. The system according to claim 32, wherein the processor-readable instructions further include instructions which when executed cause the processor to perform the steps of

determining the allocation and length of life of the various objects;

identifying the most effective garbage collecting algorithm;

regenerating the methods constituting the requisite garbage collecting; and,

erasing the remaining garbage collecting algorithms.

37. The system according to claim 33, wherein the processor-readable instructions further include instructions which when executed cause the processor to perform the steps of

determining the allocation and length of life of the various objects;

identifying the most effective garbage collecting algorithm;

regenerating the methods constituting the requisite garbage collecting; and, erasing the remaining garbage collecting algorithms.

38. A processor readable medium having instructions thereon which when executed cause the processor to perform the steps of

analyzing each of a plurality of methods within the memory space of a computer system or virtual machine to determine a selection of methods to be regenerated;

creating regenerated methods, and replacing references to each method with references to the regenerated method; and,

erasing all non-regenerated methods and releasing the memory space occupied by those non-regenerated methods.

39. A method for improving the effectiveness of a data processing application or program substantially as hereinbefore described with reference to the accompanying drawings.

40. A process for optimizing the distribution of methods within the memory space of a computer system substantially as hereinbefore described with reference to the accompanying drawings.

41. A system for optimizing the distribution of methods within the memory space of a computer system substantially as hereinbefore described with reference to the accompanying drawings.

42. A method for use with a computer substantially as hereinbefore described with reference to the accompanying drawings.

43. A system for improving the effectiveness of a data processing application or program substantially as hereinbefore described with reference to the accompanying drawings.

44. A processor readable medium having instructions thereon which when executed cause a processor to perform a method substantially as hereinbefore described with reference to the accompanying drawings.

DATED this 1st day of October, 2003

Appeal Virtual Machines AB

by DAVIES COLLISON CAVE
Patent Attorneys for the Applicant

Fig 1

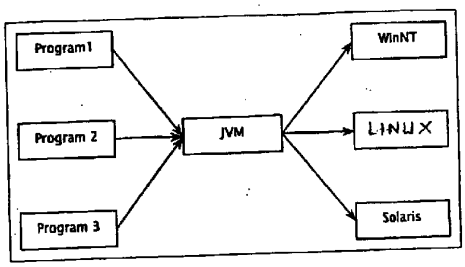


Fig 2

```
foo()
{
   x;              4711: x
   bar();          4712: call 4720
   y;              4713: y
}

apa()
{
                   4714: .
                   4715: .
                   4716: .
                   4717: .
                   4718: .
}                  4719: .

bar()
{
   .
}                  4720: .
```

Fig 3

```
foo_av()
{
   x;              4900: x
   bar();          4901: call 4903
   y;              4902: y
}

bar()
{
   .               4903: .
}
```