Title: AUTOMATIC NATIVE GENERATION

Abstract: Various technologies and techniques are disclosed for automatically generating native images for a virtual machine environment. A virtual machine environment is provided where application libraries are distributed in an intermediate language format and then compiled at a later point in time. An automatic native generation service is provided that monitors the application libraries and generates a native image for a particular one or more of the application libraries when the service determines that native generation is appropriate. Invalid native images are automatically detected and re-generated. If a load attempt for a particular native image is unsuccessful, then the native image is determined to be invalid. The particular native image is then regenerated automatically.
AUTOMATIC NATIVE GENERATION

BACKGROUND

[001] Many applications and libraries are distributed in an intermediate format, such as MICROSOFT® Intermediate Language (MSIL). These intermediate language binaries (also known as managed assemblies in the case of MICROSOFT® .NET) are typically compiled dynamically at runtime in a virtual machine environment using a Just-in-Time (JIT) compiler. An alternative to dynamic compilation is pre-compilation via Native Generation (NGen). NGen generates machine code and runtime data structures from the intermediate language and persists them in files on disk. The images produced by NGen are called Native or NGen images. Unlike JIT-compiled code, code and data structures in NGen images can be shared across processes. For libraries and frameworks that are typically shared across multiple processes, NGen is extremely useful since it minimizes the working set of each managed process. NGen therefore reduces the overall memory utilization of the system. NGen is also very useful for minimizing start up time of client-side applications.

[002] Several managed platforms/applications are using NGen. Unfortunately, however, it is quite difficult to use NGen in these current platforms. Since NGen images need to be created on the end-user machine, NGen commands need to be chained through the framework/application's installer. Typically, that involves writing a custom action (such as a MICROSOFT® WINDOWS® Installer action) that invokes a command-line tool (ngen.exe in the case of MICROSOFT® .NET). Custom actions are not trivial to write. Moreover, NGen images may become invalidated for a variety of reasons (such as when the corresponding libraries/assemblies are serviced), and need to be regenerated each time that happens by issuing explicit commands through the command line tool.

SUMMARY

[003] Various technologies and techniques are disclosed for automatically generating native images for a virtual machine environment. A virtual machine environment is provided where application libraries are distributed in an intermediate language format and then compiled at a later point in time. An
automatic native generation service is provided that monitors the application libraries and generates a native image for a particular one or more of the application libraries when the service determines that native generation is appropriate.

[004] In one implementation, native generation is determined to be appropriate by monitoring how frequently a particular application library is used, and then regenerating the application libraries in order of frequency of usage. In another implementation, native generation is determined to be appropriate by reading an identifier associated with a particular application library indicating it has been opted into native generation.

[005] Invalid native images are automatically detected and re-generated. For example, if a load attempt for a particular native image is unsuccessful, then the native image is determined to be invalid. The particular native image is then regenerated automatically.

[006] This Summary was provided to introduce a selection of concepts in a simplified form that are further described below in the Detailed Description. This Summary is not intended to identify key features or essential features of the claimed subject matter, nor is it intended to be used as an aid in determining the scope of the claimed subject matter.

**BRIEF DESCRIPTION OF THE DRAWINGS**

[007] Figure 1 is a diagrammatic view of a computer system of one implementation.

[008] Figure 2 is a diagrammatic view of an automatic native generation application of one implementation operating on the computer system of Figure 1.

[009] Figure 3 is a high-level process flow diagram for one implementation of the system of Figure 1.

[010] Figure 4 is a process flow diagram for one implementation of the system of Figure 1 illustrating the stages involved in providing an automatic native generation service that uses load counters to determine an order to generate the native images.

[011] Figure 5 is a process flow diagram for one implementation of the system of Figure 1 illustrating the stages involved in providing an automatic native generation service that uses heuristics to help determine which native images to generate.
[012] Figure 6 is a process flow diagram for one implementation of the system of
Figure 1 illustrating the stages involved in providing an automatic native generation
service that automatically re-generates native images that are determined to be
invalid.

[013] Figure 7 is a process flow diagram for one implementation of the system of
Figure 1 that illustrates the stages involved in managing and interacting with a data
file to help track which native images should be generated by the native generation
service.

[014] Figure 8 is a logical diagram for one implementation of the system of Figure
1 that illustrates the interaction of various libraries and/or subsystems during an
automatic native generation process.

DETAILED DESCRIPTION

[015] For the purposes of promoting an understanding of the principles of the
invention, reference will now be made to the embodiments illustrated in the
drawings and specific language will be used to describe the same. It will
nevertheless be understood that no limitation of the scope is thereby intended. Any
alterations and further modifications in the described embodiments, and any further
applications of the principles as described herein are contemplated as would
normally occur to one skilled in the art.

[016] The system may be described in the general context as an application that
automatically generates native images (NGen images) in a virtual machine
environment, but the system also serves other purposes in addition to these. In one
implementation, one or more of the techniques described herein can be
implemented as features within a framework program such as MICROSOFT®
.NET Framework, Java Virtual Machine, or from any other type of program or
service that deals with an intermediate language and then needs to compile that
intermediate language into machine code for execution on a target machine.

[017] As shown in Figure 1, an exemplary computer system to use for
implementing one or more parts of the system includes a computing device, such as
computing device 100. In its most basic configuration, computing device 100
typically includes at least one processing unit 102 and memory 104. Depending on
the exact configuration and type of computing device, memory 104 may be volatile (such as RAM), non-volatile (such as ROM, flash memory, etc.) or some combination of the two. This most basic configuration is illustrated in Figure 1 by dashed line 106.

[018] Additionally, device 100 may also have additional features/functionality. For example, device 100 may also include additional storage (removable and/or non-removable) including, but not limited to, magnetic or optical disks or tape. Such additional storage is illustrated in Figure 1 by removable storage 108 and non-removable storage 110. Computer storage media includes volatile and nonvolatile, removable and non-removable media implemented in any method or technology for storage of information such as computer readable instructions, data structures, program modules or other data. Memory 104, removable storage 108 and non-removable storage 110 are all examples of computer storage media. Computer storage media includes, but is not limited to, RAM, ROM, EEPROM, flash memory or other memory technology, CD-ROM, digital versatile disks (DVD) or other optical storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or any other medium which can be used to store the desired information and which can accessed by device 100. Any such computer storage media may be part of device 100.

[019] Computing device 100 includes one or more communication connections 114 that allow computing device 100 to communicate with other computers/applications 115. Device 100 may also have input device(s) 112 such as keyboard, mouse, pen, voice input device, touch input device, etc. Output device(s) 111 such as a display, speakers, printer, etc. may also be included. These devices are well known in the art and need not be discussed at length here. In one implementation, computing device 100 includes automatic native generation application 200. Automatic native generation application 200 will be described in further detail in Figure 2.

[020] Turning now to Figure 2 with continued reference to Figure 1, an automatic native generation application 200 operating on computing device 100 is illustrated. Automatic native generation application 200 is one of the application programs that
reside on computing device 100. However, it will be understood that automatic native generation application 200 can alternatively or additionally be embodied as computer-executable instructions on one or more computers and/or in different variations than shown on Figure 1. Alternatively or additionally, one or more parts of automatic native generation application 200 can be part of system memory 104, on other computers and/or applications 115, or other such variations as would occur to one in the computer software art.

[021] Automatic native generation application 200 includes program logic 204, which is responsible for carrying out some or all of the techniques described herein. Program logic 204 includes logic for providing an automatic native generation service for a virtual machine environment 206; logic for monitoring application libraries used in the virtual machine environment 208; logic for generating and/or re-generating a native image from an intermediate language for one or more of the application libraries when the service determines that native generation is appropriate 210; logic for determining that native generation is appropriate at least in part by reading a native generation attribute associated with each of the application libraries 212; logic for determining that native generation is appropriate at least in part by tracking a load counter for each of the application libraries, sorting the load counter for each of the application libraries, and using the sorted load counters to determine an order in which to generate the native image for some or all of the application libraries 214; logic for determining that native generation is appropriate for a given one or more libraries by using heuristics (e.g. tracking which libraries are used simultaneously by multiple processes and would benefit from native generation, etc.) 216; and other logic for operating the application 220.

In one implementation, program logic 204 is operable to be called programmatically from another program, such as using a single call to a procedure in program logic 204.

[022] Turning now to Figures 3-8 with continued reference to Figures 1-2, the stages for implementing one or more implementations of automatic native generation application 200 are described in further detail. Figure 3 is a high level process flow diagram for automatic native generation application 200. In one form,
the process of Figure 3 is at least partially implemented in the operating logic of computing device 100. The procedure begins at start point 240 with providing a virtual machine environment where application libraries are distributed in an intermediate language format (such as MICROSOFT® intermediate language [MSIL]) and then compiled at a later point in time (e.g. to machine-executable code) (stage 242). The term application library as used herein is meant to include code libraries that can be compiled into machine code and executed and/or accessed by a computer as a dynamic link library (DLL), executable program (EXE), and/or another type of program or library.

[023] An automatic native generation service is provided that monitors the application libraries and generates a native image (e.g. synchronously before image load is completed, asynchronously immediately following the image load, or asynchronously when the machine is idle) for a particular one or more of the application libraries when the service determines that native generation and/or re-generation is appropriate (stage 244). One or more combinations of heuristics, tracking load counters, determining the old native image is invalid, reading a native generation attribute specified by a developer, and/or other rules of logic can be used to determine that native generation is appropriate (stage 244). As one non-limiting example, a developer can set a native generation attribute for a particular library/assembly, and the system can then read that attribute to determine that the developer wishes to opt-in to native generation. Some additional non-limiting examples are described in further detail in Figures 4-6. The process ends at end point 246.

[024] Figure 4 illustrates one implementation of the stages involved in providing an automatic native generation service that uses load counters to determine an order to generate the native images. In one form, the process of Figure 4 is at least partially implemented in the operating logic of computing device 100. One or more combinations of load counters can be used instead of or in addition to the other techniques described herein for determining when automatic native generation is appropriate. The procedure begins at start point 270 with providing an automatic native generation service that monitors application libraries on the machine (stage
Each time one of the application libraries is loaded, a load counter is incremented for the particular library to indicate it has been loaded again (stage 274). The load counter for each application library is recorded in a data file (or other data storage means) (stage 276). The load counters are sorted in a particular order (e.g. descending numerical order so that the application library that has been loaded the most appears first in the list) (stage 278). The sorted list is then used to determine the order in which to generate the respective native image (stage 280). In one implementation, the native image for the library with the highest counter is generated before others that have a lower counter and were thus not used as much (stage 280). The process ends at end point 282.

[025] Figure 5 illustrates one implementation of the stages involved in providing an automatic native generation service that uses heuristics to help determine which native images to generate. One or more combinations of heuristics can be used instead of or in addition to the other techniques described herein for determining when automatic native generation is appropriate. In one form, the process of Figure 5 is at least partially implemented in the operating logic of computing device 100. The procedure begins at start point 290 with providing an automatic native generation service that monitors the application libraries on the machine (stage 292). One or more heuristics are used to determine that native generation is appropriate (stage 294). In one implementation, one heuristic includes monitoring which of the application libraries are used simultaneously by multiple processes (and would thus benefit from native image generation) (stage 294). The system automatically generates/re-generates native images for the application libraries that the heuristics determine are appropriate (stage 296). The process ends at end point 298.

[026] Figure 6 illustrates one implementation of the stages involved in providing an automatic native generation service that automatically re-generates native images that are determined to be invalid. One or more combinations of re-generation techniques for invalid native images can be used instead of or in addition to the other techniques described herein for determining when automatic native generation is appropriate. In one form, the process of Figure 6 is at least
partially implemented in the operating logic of computing device 100. The procedure begins at start point 310 with providing an automatic native generation service that monitors the application libraries on the machine (stage 312). The system attempts to load a particular native image (stage 314), and after unsuccessfully loading the particular native image, determines that the particular native image is invalid (stage 316). An identifier is then stored for the particular native image in a data file or other data storage location (e.g. in a data file where various identifiers are stored for native images that need to be re-generated) (stage 318). The data file is accessed (e.g. asynchronously) to automatically re-generate the particular native image (e.g. from a background process) (stage 320). Other native images referenced in the data file can also be re-generated at this or a later time (stage 320). The process ends at end point 322.

[027] Figure 7 illustrates one implementation of the stages involved in managing and interacting with a data file to help track which native images should be generated by the native generation service. In one form, the process of Figure 7 is at least partially implemented in the operating logic of computing device 100. The procedure begins at start point 340 with using a data file to record information about application libraries for use by the automatic native generation service (stage 342). Data is read from the data file at particular intervals of time to see if there are any entries representing new application libraries for which native images should be generated and/or re-generated (stage 344). When there are entries that need to be processed, a request is placed in the native generation queue so those native images can be regenerated (e.g. when that machine is idle) (stage 346). The corresponding entry is then deleted from the data file (stage 348). Once the machine detects that the machine is idle, the entries in the queue are processed to regenerate the respective native images (stage 350). The data file can optionally be further cleaned after particular intervals of time to ensure it does not grow too large (stage 352). The process ends at end point 354.

[028] Figure 8 is a logical diagram for one implementation of the system of Figure 1 that illustrates the interaction of various libraries and/or subsystems during an automatic native generation process. In one form, the process of Figure 8 is at least
partially implemented in the operating logic of computing device 100. The names
of libraries/assemblies that have opted into native generation along with data on
how many times each library/assembly was used in the last T hours are stored
(370). A loader (372), such as a MICROSOFT® Common Language Runtime
(loader, or another loader can access this usage data (370). This loader (372)
interacts with managed application (376). The usage data (370), along with the
names of libraries/assemblies that have opted into the native generation service that
have stale native images (378) and that have no native images (380) are provided to
a runtime optimization service (374). The runtime optimization service (374) then
generates the native images (NGen images) (382).

[029] Although the subject matter has been described in language specific to
structural features and/or methodological acts, it is to be understood that the subject
matter defined in the appended claims is not necessarily limited to the specific
features or acts described above. Rather, the specific features and acts described
above are disclosed as example forms of implementing the claims. All equivalents,
changes, and modifications that come within the spirit of the implementations as
described herein and/or by the following claims are desired to be protected.

[030] For example, a person of ordinary skill in the computer software art will
recognize that the client and/or server arrangements, and/or data layouts as
described in the examples discussed herein could be organized differently on one or
more computers to include fewer or additional options or features than as portrayed
in the examples.
What is claimed is:

1. A method for automatically generating native images for a virtual machine comprising the steps of:
   - providing a virtual machine environment where a plurality of application libraries are distributed in an intermediate language format and then compiled at a later point in time (242); and
   - providing an automatic native generation service, the service being operable to monitor the plurality of application libraries, and to generate a native image for a particular one or more of the application libraries when the service determines that native generation is appropriate (244).

2. The method of claim 1, wherein the automatic native generation service runs asynchronously (244).

3. The method of claim 1, wherein the automatic native generation service runs when a machine is idle (244).

4. The method of claim 1, wherein the automatic native generation service is further operable to regenerate the native image for the particular one or more of the application libraries when the service determines the native image is invalid (244).

5. The method of claim 1, wherein the automatic native generation service monitors the plurality of application libraries by incrementing a respective counter each time a respective one of the application libraries is loaded (274), and wherein the automatic native generation service uses the respective counter for each of the respective application libraries to determine an order in which to generate a respective native image (280).

6. The method of claim 5, wherein each of the respective counters are sorted in a descending order, and wherein the descending order is used to determine an order in which to generate the respective native image (278).

7. The method of claim 6, wherein a particular application library having a highest respective counter will have the respective native image generated first (280).

8. The method of claim 1, wherein the service determines that native generation is appropriate by using heuristics (294).
9. The method of claim 8, wherein the heuristics used to determine that native
generation is appropriate includes monitoring which of the plurality of application
libraries are used simultaneously by multiple processes (294).

10. The method of claim 1, wherein the intermediate language is Microsoft
Intermediate Language (242).

11. A computer-readable medium having computer-executable instructions for
causing a computer to perform the steps recited in claim 1 (200).

12. A computer-readable medium having computer-executable instructions for
causing a computer to perform steps comprising:

   provide an automatic native generation service for a virtual machine
   environment (206), the service being operable to monitor a plurality of
   application libraries (208), and to generate a native image from an intermediate
   language for a particular one or more of the application libraries when the
   service determines that native generation is appropriate (210).

13. The computer-readable medium of claim 12, wherein the native generation
   service is further operable to generate the native image asynchronously (244).

14. The computer-readable medium of claim 12, wherein the native generation
   service is further operable to determine that native generation is appropriate at least
   in part by reading a native generation attribute associated with each of the
   application libraries (212).

15. The computer-readable medium of claim 12, wherein the native generation
   service is further operable to determine that native generation is appropriate at least
   in part by tracking a load counter for each of the application libraries, sorting the
   load counter for each of the application libraries, and then using the sorted load
   counters to determine an order in which to generate the native image for some or all
   of the application libraries (214).

16. The computer-readable medium of claim 12, wherein the native generation
   service is further operable to determine that native generation is appropriate at least
   in part by using heuristics (216).
17. A method for automatically detecting invalid native images and automatically regenerating native images for a virtual machine comprising the steps of:

- attempting to load a particular native image (314);
- unsuccessfully loading the particular native image because the particular native image is determined to be invalid (316); and
- automatically regenerating the particular native image (320).

18. The method of claim 17, wherein after unsuccessfully loading the particular native image, an identifier for the particular native image is stored in a data file (318).

19. The method of claim 18, wherein the data file is accessed asynchronously to regenerate the particular native image in a background process (320).

20. A computer-readable medium having computer-executable instructions for causing a computer to perform the steps recited in claim 17 (200).
<table>
<thead>
<tr>
<th>AUTOMATIC NATIVE GENERATION APPLICATION</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>200</strong></td>
</tr>
<tr>
<td>PROGRAM LOGIC</td>
</tr>
<tr>
<td><strong>204</strong></td>
</tr>
<tr>
<td>LOGIC FOR PROVIDING AN AUTOMATIC NATIVE</td>
</tr>
<tr>
<td>GENERATION SERVICE FOR A VIRTUAL MACHINE</td>
</tr>
<tr>
<td>ENVIRONMENT</td>
</tr>
<tr>
<td><strong>206</strong></td>
</tr>
<tr>
<td>LOGIC FOR MONITORING APPLICATION LIBRARIES USED IN THE VIRTUAL MACHINE ENVIRONMENT</td>
</tr>
<tr>
<td><strong>208</strong></td>
</tr>
<tr>
<td>LOGIC FOR GENERATING AND/OR RE-GENERATING</td>
</tr>
<tr>
<td>A NATIVE IMAGE FROM AN INTERMEDIATE</td>
</tr>
<tr>
<td>LANGUAGE FOR ONE OR MORE OF THE</td>
</tr>
<tr>
<td>APPLICATION LIBRARIES WHEN THE SERVICE</td>
</tr>
<tr>
<td>DETERMINES THAT NATIVE GENERATION IS</td>
</tr>
<tr>
<td>APPROPRIATE</td>
</tr>
<tr>
<td><strong>210</strong></td>
</tr>
<tr>
<td>LOGIC FOR DETERMINING THAT NATIVE</td>
</tr>
<tr>
<td>GENERATION IS APPROPRIATE AT LEAST IN</td>
</tr>
<tr>
<td>PART BY READING A NATIVE GENERATION</td>
</tr>
<tr>
<td>ATTRIBUTE ASSOCIATED WITH EACH OF THE</td>
</tr>
<tr>
<td>APPLICATION LIBRARIES</td>
</tr>
<tr>
<td><strong>212</strong></td>
</tr>
<tr>
<td>LOGIC FOR DETERMINING THAT NATIVE</td>
</tr>
<tr>
<td>GENERATION IS APPROPRIATE AT LEAST IN</td>
</tr>
<tr>
<td>PART BY TRACKING A LOAD COUNTER FOR</td>
</tr>
<tr>
<td>EACH OF THE APPLICATION LIBRARIES,</td>
</tr>
<tr>
<td>SORTING THE LOAD COUNTER FOR EACH OF</td>
</tr>
<tr>
<td>THE APPLICATION LIBRARIES, AND USING</td>
</tr>
<tr>
<td>THE SORTED LOAD COUNTERS TO DETERMINE</td>
</tr>
<tr>
<td>AN ORDER IN WHICH TO GENERATE THE NATIVE</td>
</tr>
<tr>
<td>IMAGE FOR SOME OR ALL OF THE APPLICATION</td>
</tr>
<tr>
<td>LIBRARIES</td>
</tr>
<tr>
<td><strong>214</strong></td>
</tr>
<tr>
<td>LOGIC FOR DETERMINING THAT NATIVE</td>
</tr>
<tr>
<td>GENERATION IS APPROPRIATE FOR A GIVEN</td>
</tr>
<tr>
<td>ONE OR MORE LIBRARIES BY USING</td>
</tr>
<tr>
<td>HEURISTICS (E.G. TRACKING WHICH</td>
</tr>
<tr>
<td>LIBRARIES ARE USED SIMULTANEOUSLY BY</td>
</tr>
<tr>
<td>MULTIPLE PROCESSES AND WOULD BENEFIT</td>
</tr>
<tr>
<td>FROM NATIVE GENERATION, ETC.)</td>
</tr>
<tr>
<td><strong>216</strong></td>
</tr>
<tr>
<td>OTHER LOGIC FOR OPERATING THE APPLICATION</td>
</tr>
<tr>
<td><strong>220</strong></td>
</tr>
</tbody>
</table>
START 240

PROVIDE A VIRTUAL MACHINE ENVIRONMENT WHERE APPLICATION LIBRARIES ARE DISTRIBUTED IN AN INTERMEDIATE LANGUAGE FORMAT (SUCH AS MICROSOFT INTERMEDIATE LANGUAGE [MSIL]) AND THEN COMPILLED AT A LATER POINT IN TIME (E.G. TO MACHINE-EXECUTABLE CODE) 242

PROVIDE AN AUTOMATIC NATIVE GENERATION SERVICE THAT MONITORS THE APPLICATION LIBRARIES AND GENERATES A NATIVE IMAGE (E.G. SYNCHRONOUSLY BEFORE IMAGE LOAD IS COMPLETED, ASYNCHRONOUSLY IMMEDIATELY FOLLOWING THE IMAGE LOAD, OR ASYNCHRONOUSLY WHEN THE MACHINE IS IDLE) FOR A PARTICULAR ONE OR MORE OF THE APPLICATION LIBRARIES WHEN THE SERVICE DETERMINES THAT NATIVE GENERATION AND/OR RE-GENERATION IS APPROPRIATE (E.G. USING HEURISTICS, TRACKING LOAD COUNTERS, WHEN DETERMINING THE OLD NATIVE IMAGE IS INVALID, AND/OR UPON READING A NATIVE GENERATION ATTRIBUTE SPECIFIED BY A DEVELOPER, ETC.) 244

END 246

FIG. 3
PROVIDE AN AUTOMATIC NATIVE GENERATION SERVICE THAT MONITORS APPLICATION LIBRARIES ON THE MACHINE

EACH TIME ONE OF THE APPLICATION LIBRARIES IS LOADED, INCREMENT A LOAD COUNTER FOR THE PARTICULAR LIBRARY

RECORD THE LOAD COUNTERS FOR EACH APPLICATION LIBRARY IN A DATA FILE (OR OTHER DATA STORAGE MEANS)

SORT THE LOAD COUNTERS IN A PARTICULAR ORDER (E.G. DESCENDING NUMERICAL ORDER SO THAT THE APPLICATION LIBRARY THAT HAS BEEN LOADED THE MOST APPEARS FIRST IN THE LIST)

USE THE SORTED LIST TO DETERMINE THE ORDER IN WHICH TO GENERATE THE RESPECTIVE NATIVE IMAGE (E.G. GENERATE THE NATIVE IMAGE FOR THE LIBRARY WITH THE HIGHEST COUNTER BEFORE OTHERS THAT HAVE A LOWER COUNTER AND WERE THUS NOT USED AS MUCH)

END
START

PROVIDE AN AUTOMATIC NATIVE GENERATION SERVICE THAT MONITORS THE APPLICATION LIBRARIES ON THE MACHINE

USE ONE OR MORE HEURISTICS TO DETERMINE THAT NATIVE GENERATION IS APPROPRIATE (E.G. BY MONITORING WHICH OF THE APPLICATION LIBRARIES ARE USED SIMULTANEOUSLY BY MULTIPLE PROCESSES, ETC.)

AUTOMATICALLY GENERATE / RE-GENERATE NATIVE IMAGES FOR THE APPLICATION LIBRARIES THAT THE HEURISTICS DETERMINE ARE APPROPRIATE

END

FIG. 5
6 / 8

START

310

PROVIDE AN AUTOMATIC NATIVE GENERATION SERVICE THAT MONITORS THE APPLICATION LIBRARIES ON THE MACHINE

312

ATTEMPT TO LOAD A PARTICULAR NATIVE IMAGE

314

UNSUCCESSFULLY LOAD THE PARTICULAR NATIVE IMAGE BECAUSE THE PARTICULAR NATIVE IMAGE IS DETERMINED TO BE INVALID

316

STORE AN IDENTIFIER FOR THE PARTICULAR NATIVE IMAGE IN A DATA FILE OR OTHER DATA STORAGE LOCATION (E.G. IN A DATA FILE WHERE VARIOUS IDENTIFIERS ARE STORED FOR NATIVE IMAGES THAT NEED TO BE RE-GENERATED)

318

ACCESS THE DATA FILE (E.G. ASYNCHRONOUSLY) TO AUTOMATICALLY REGENERATE THE PARTICULAR NATIVE IMAGE (E.G. FROM A BACKGROUND PROCESS) AND/OR OTHER NATIVE IMAGES REFERENCED IN THE DATA FILE

320

FIG. 6

END

322
START

340

USE A DATA FILE TO RECORD INFORMATION ABOUT APPLICATION LIBRARIES FOR USE BY THE AUTOMATIC NATIVE GENERATION SERVICE

342

READ DATA FROM THE DATA FILE AT PARTICULAR INTERVALS OF TIME TO SEE IF THERE ARE ANY ENTRIES REPRESENTING NEW APPLICATION LIBRARIES FOR WHICH NATIVE IMAGES SHOULD BE GENERATED AND/OR RE-GENERATED

344

WHEN THERE ARE ENTRIES THAT NEED PROCESSED, A REQUEST IS PLACED IN THE NATIVE GENERATION QUEUE SO THOSE NATIVE IMAGES CAN BE REGENERATED (E.G. WHEN THAT MACHINE IS IDLE)

346

THE CORRESPONDING ENTRY IS THEN DELETED FROM THE DATA FILE

348

ONCE THE MACHINE DETECTS THAT THE MACHINE IS IDLE, THE ENTRIES IN THE QUEUE ARE PROCESSED TO REGENERATE THE RESPECTIVE NATIVE IMAGES

350

THE DATA FILE CAN OPTIONALLY BE FURTHER CLEANED AFTER PARTICULAR INTERVALS OF TIME TO ENSURE IT DOES NOT GROW TOO LARGE

352

FIG. 7

END

354
FIG. 8

1. Loaders (e.g., Common Language Runtime [CLR] or other loader) 372
2. Runtime Optimization Service (NGen Service) 374
3. Managed application 376
4. Names of libraries/assemblies opted into NGen along with data on how many times each library/assembly was used in the last T hours 370
5. NGen images 382
6. Names of libraries/assemblies opted into NGen with stale native images 378
7. Names of libraries/assemblies opted into NGen with no native images 380
INTERNATIONAL SEARCH REPORT

International application No
PCT/ISA/2007/080451

A. CLASSIFICATION OF SUBJECT MATTER

G06F 17/24(2006.01)i, G06F 17/00(2006.01)i

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

IPC 8 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Korean Utility models and applications for Utility models since 1975
Japanese Utility models and applications for Utility models since 1975

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

PAJ, FPD, USPAT, eKIPASS, IEEE, YAHOO, GOOGLE. Keyword "native image, just-in-time compilation, NGen, Intermediate Language"

C. DOCUMENTS CONSIDERED TO BE RELEVANT

<table>
<thead>
<tr>
<th>Category</th>
<th>Citation of document, with indication, where appropriate, of the relevant passages</th>
<th>Relevant to claim No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>REID WILKES, 'SPEED NGen Revs Up Your Performance with Powerful New Features', MSDN Magazine, April 2005. See following website: <a href="http://msdn.microsoft.com/msdnmag/issues/05/04/NGen/default.aspx">http://msdn.microsoft.com/msdnmag/issues/05/04/NGen/default.aspx</a> (See 1st paragraph, Section 'Enter NGen 2.0')</td>
<td>1-20</td>
</tr>
<tr>
<td>Y</td>
<td>US 2006-0101438 A1 (RICO MARIANI) 11 May 2006 (See Abstract, paragraph [0005],[0012]-[0048])</td>
<td>1-20</td>
</tr>
<tr>
<td>A</td>
<td>US 2006-0101446 A1 (RICO MARIANI et al) 11 May 2006 (See Abstract, paragraph [0002]-[0009],[0026]-[0061])</td>
<td>1-20</td>
</tr>
<tr>
<td>A</td>
<td>EP 0997816 A2 (SUN MICROSYSTEMS, Inc) 03 May 2000 (See Abstract, paragraph [0111]-[0016])</td>
<td>1-20</td>
</tr>
<tr>
<td>A</td>
<td>US 06151703 A (REGIS CRELIER) 21 November 2000 (See Abstract, column 1-4)</td>
<td>1-20</td>
</tr>
</tbody>
</table>

Further documents are listed in the continuation of Box C

See patent family annex

*T* later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

*X* document of particular relevance, the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

*Y* document of particular relevance, the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

&* document member of the same patent family

Date of the actual completion of the international search
21 January 2008 (21.01.2008)

Date of mailing of the international search report
21 January 2008 (21.01.2008)

Name and mailing address of the ISA/KR
Korean Intellectual Property Office
920 Dunsan-dong, Seo-gu, Daejeon 302-701, Republic of Korea
Facsimile No 82-42-472-7140

Authorized officer
Sun Dong Guk
Telephone No 82-42-481-8248

Form PCT/ISA/210 (second sheet) (April 2007)
<table>
<thead>
<tr>
<th>Patent document cited in search report</th>
<th>Publication date</th>
<th>Patent family member(s)</th>
<th>Publication date</th>
</tr>
</thead>
<tbody>
<tr>
<td>US20060 10 1438A 1</td>
<td>11.05.2006</td>
<td>NONE</td>
<td></td>
</tr>
<tr>
<td>US20060 10 1446A 1</td>
<td>11.05.2006</td>
<td>NONE</td>
<td></td>
</tr>
<tr>
<td>EP0997816A2</td>
<td>03.05.2000</td>
<td>CA2287746A 1</td>
<td>30.04.2000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CN 1255674A</td>
<td>07.06.2000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>JP2000181724A2</td>
<td>30.06.2000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>US06298477B 1</td>
<td>02.10.2001</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EP9997816A3</td>
<td>27.08.2003</td>
</tr>
<tr>
<td>US06 15 1703A</td>
<td>21.11.2000</td>
<td>NONE</td>
<td></td>
</tr>
</tbody>
</table>

Form PCT/ISA/210 (patent family annex) (April 2007)