PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 9/445	A1	(11) International Publication Number: WO 98/37486
		(43) International Publication Date: 27 August 1998 (27.08.98)

(21) International Application Number: PCT/IB97/00135

(22) International Filing Date: 18 February 1997 (18.02.97)

(71) Applicant *(for all designated States except US)*: INTERNATIONAL BUSINESS MACHINES CORPORATION [US/US]; Old Orchard Road, Armonk, NY 10504 (US).

(72) Inventor; and
(75) Inventor/Applicant *(for US only)*: EIRICH, Thomas [DE/CH]; Zopfstrasse 16, CH–8804 Au (CH).

(74) Agent: HEUSCH, Christian; International Business Machines Corporation, Säumerstrasse 4, CH–8803 Rueschlikon (CH).

(81) Designated States: JP, US, European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
With international search report.
With amended claims.

(54) Title: METHOD FOR LOOKUP OF PACKAGES AND CLASSES IN JAVA, AND DEVICES MAKING USE OF THIS METHOD

(57) Abstract

The present invention concerns a client system comprising a Java interpreter for the execution of a Java program (Java applet) and a set of Java classes for interaction with the client system. The client system only holds a reduced set of Java classes, and comprises means for loading Java classes which do not belong to the reduced set of Java classes into the client system only if needed during execution of said Java program. It may also include means for erasing such a loaded class and/or means for storing such a loaded class permanently, i.e., adding it to the reduced set of Java classes.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

DESCRIPTION

Method for lookup of packages and classes in Java, and devices making use of this method

TECHNICAL FIELD

The present invention relates to a method for loading Java classes or packages into a client system where these classes or packages could not be found otherwise ("Java" is a programming language developed by and a trademark of SUN Microsystems Inc., USA). Furthermore, the present invention relates to a new group of Java-enabled devices making use of the method for loading Java classes or packages.

BACKGROUND OF THE INVENTION

Current networks, such as the Internet are heterogeneous systems linking computers of various manufacturers. There are many different hardware platforms - based on Intel processors, RISC processors, and the like - each of which has a specific architecture. In addition, there are various operating systems which are installed on these platforms. In order to access the world-wide-web (WWW) part of the Internet for example, one also needs a so-called browser which is able to fetch information from a remote server. Such information is then stored in the client computer, i.e. the computer where the browser is located, before it is displayed on the local display. Such a browser is able to invoke a suitable editor for viewing information.

Currently, the hypertext markup language (HTML) is used on many network servers. HTML is mainly suited for loading information into a client computer where this information is then displayed. Interaction between client and server is only possible if the browser makes use of the Common

Gateway Interface (CGI). CGI enables a user to insert information in an entry panel. Based on the information inserted by the user, certain tasks can be carried out by the server. A typical example is a search engine which supports the user in searching for a particular word or combination of words.

Real interactivity is possible if one uses Java programs (also called Java applets). It is an important feature of Java that the client computer instead of the server is the computer which carries out certain task upon request of the user. The time a task takes does no longer depend on the load of the server or on the load of the network linking the client to the server. Once a Java program is loaded into the client, it runs on the client and is completely under control of the user.

A Java applet is a program written in the Java language. By means of a Java compiler it is transformed into Java code. This Java code is a byte code being platform-independent. Usually, the byte code is made available on a server. If a user wants to use a Java applet, he fetches the applet from the server, i.e. the respective byte code is transmitted via the network from the server to the client and loaded into the client computer. This is done under control of a Java interpreter being located at the client. The Java interpreter checks the process of loading the byte code and performs some verification tasks to ensure that the byte code received conforms to a required specification, the Java Byte Code specification.

Next, the Java interpreter executes the byte code step-by-step. Because this interpreter is required for the execution of the byte code, a Java applet usually is much slower than a comparable applet written in a native programming language which can be directly understood and executed by the processor of the client.

From time to time, the interpreter needs to interact with the client system directly, e.g. if there is information on a local disk to be loaded. In order to

enable the interaction between the Java interpreter and the client system, a so-called Java system API (application program interface) is provided. The API is client-specific. In today's systems, a built-in Java class loader is provided which searches for a class or package needed and loads the respective package or class. All these packages and classes have to be installed on the client before Java applets are executed.

It is a disadvantage of this approach that the Java packages and classes need to be made available by the client, i.e., memory is occupied by the packages and classes kept in the client. It is another disadvantage that new or updated packages and classes can only be added by loading them into all client computers.

It is an object of the present invention to provide low-cost client systems, such as computers, set-top boxes, telephones and the like, which support Java, which do not require the memory space identified above.

It is another object of the present invention to provide a method which allows to execute Java applets on low-cost systems, such as small network computers, set-top boxes, telephones and the like.

A further object of the present invention is to minimize the size of the pre-loaded functionality, especially the number of Java classes on client systems.

A still further object is to avoid loading the network with bulk transmissions of notifications, updates and/or modifications of Java classes from servers to a large plurality of clients.

SUMMARY OF THE INVENTION

This is achieved by providing a new and inventive approach which allows to reduce the number of classes to be kept in a client system. In order to be able to execute Java code using a Java interpreter, a special mechanism for loading those classes which are not found on the local client system is provided. This special class loader, referred to as extended class loader, enables the loading of unavailable classes into a client system, before they are loaded into the runtime system via the existing API.

Thus, the number of classes in a client system can be reduced, such that a client system according to the present invention needs far less storage space than a conventional Java client. In other words, smaller and cheaper systems can be made. Network computers, a new category of machines designed to cut the cost of personal computing, can be realized, which just provide the functionally that is permanently needed. Classes which are not available at the client are loaded only upon demand by the extended class loader according to the present invention.

It is another advantage of the present invention that all classes, except for those being stored in the client system, remain under the control of the server. These classes can be modified and updated from time to time without having to notify the clients and/or transmit modifications and updates automatically to a large number of clients.

GENERAL DESCRIPTION

The expression 'client system' is used hereinafter as a synonym for any kind of system enabled to execute a Java applet. Typical examples of such client systems are: computers, in particular network computers, set-top boxes, telephones, cellular phones, pagers, printers and the like. Also included are browsers and other software which is able to execute a Java applet.

The network which is used to interconnect the client system with a server from where a Java applet can be fetched, may be a conventional network or a wireless IR or RF network. The expression network is also meant to cover a simple link (peer-to-peer, for example) between a client system and a server. Also included are telephone, video on demand and other networks.

A client system comprises a Java interpreter which is mainly employed to control the loading of a Java applet from the network. This Java interpreter checks the process of loading the respective byte code and performs verification tasks to ensure that the byte code received was transmitted properly.

Either automatically, or upon request of the user, the Java interpreter now executes the byte code step-by-step. During the execution of a Java applet, the interpreter needs to interact with the client system directly, e.g. if there is information on a local disk to be loaded, or information to be displayed on a display. In order to enable the exchange of information between the Java interpreter and the client system, the Java system API is provided. This API is client-specific.

In today's systems, the Java interpreter comprises a built-in Java class loader which handles the binding of said Java classes with the Java applet executed. It initiates the access to a class referred to by a string in the byte code of said applet. Then, it looks up the respective class to be accessed

and loads it into the Java interpreter. A pointer is returned which indicates where said respective class is located in said client system's memory.

In current client systems, all these packages and classes have to be pre-installed before a Java applet can be executed by the interpreter.

Packages and classes in Java are organized in a tree, where the inner nodes of the tree are package components and the leaves are class names. A class is specified by a possibly empty sequence of package components followed by a class name. These components are separated by dots such as for example : java.util.Date. In this example 'Date' is the class name and 'util' and 'java' are package components and the whole string is referred to as qualified class name. The class name specifications are translated into path names of the client system's operating system (e.g. java/util/Date) and are expected to refer to a file containing the definition of a Java class, i.e. a pointer is provided which indicates where the respective file is to be found in the client's memory.

It has been realized that these packages and classes can be divided up into classes and packages which are required by the client system to ensure proper operation (hereinafter referred to as essential classes), and other classes and packages.

The underlying idea of the present invention is to reduce the burden of a client system by pre-installing only those classes and packages which are essential for the execution of most Java applets. In addition to these essential classes and packages, one may pre-install those classes and packages which are expected to be needed frequently during the execution of specific Java applets the respective client system is designed for. It is to be noted that the number and kind of essential classes may differ from client system to client system.

According to the present invention, means are provided for loading a package, package component, or class which is not present in the file system held in the client system's memory. Instead of indicating to the Java runtime interpreter that a particular class was not found (simply because this class is not stored in the client) a special class loader may be provided which handles the loading of classes from another system. This special class loader receives the qualified class name from the Java runtime interpreter. It handles the lookup of the package components until it reaches the requested class. The class then is loaded from the other system into the client's runtime system via the existing Java API. While parsing the package components, the extended class loader may hand over to yet another class loader which is responsible for a subtree.

According to the present invention, a class loader may be employed which does not require to change the Java runtime interpreter. The extended class loader simply interacts with a standard Java runtime interpreter, for example.

First, the extended class loader, according to the present invention, may check if there exists a special Java class file instead of the missing directory related to a package component. The name is derived from the package name in such a way that it will not interfere with any legal class or package name, e.g. a dash may be used in the name of a class file replacing a missing directory. The class file could be called 'util-loader', for example, if the 'util' directory is missing in the client. This class is automatically loaded to handle requests for the missing directory. A second possibility to implement the extended class loader is to explicitly register those classes which do not belong to the reduced set of classes. This requires an initial setup before user applications can be executed but gives more flexibility.

In a client system, according to the present invention, only the essential classes are pre-installed. For execution, the byte code of a Java applet needs to be interpreted. Only those classes which are available in the client

system can be accessed directly using the built-in class loader. All other classes are not available in the client system and are loaded using the extended class loader described above.

It is important to note that the implementation of the extended class loader according to the present invention does not require any modification of existing Java application sources code and compiled Java code. The Java applets need not to be aware of the fact that a client system operates only on a reduced set of classes (namely at least the essential classes). It is conceivable that the present invention can also be implemented by modification of the built-in class loader of the Java runtime system.

To sustain a homogenous network platform, it is necessary that all Java platforms have the same set of Java APIs available, i.e. all these platforms connected to one and the same network should have the same set of Java classes. For small Java client systems the required disk space to hold all Java classes might be a problem. With the present class loading mechanism, it is possible to fetch all classes which are not available at the client (APIs) on demand over the network.

The present invention has the advantage that the newest class versions can be loaded from the server, i.e. the operator needs to replace an old version of a class by a newer version only in the server. Each time a client system asks for a class which is not stored in the client, the inventive class loader automatically fetches the newest version provided by the server.

A further arrangement of the invention includes an automatic erasing function in the client system. Such a function would erase any class fetched by the class loader according to the invention after it was used. This erase function may be executed either immediately after the loaded class was used or may be conditioned on the lapse of a predetermined time or the non-use for a predetermined time or any other condition, e.g. the remaining space of the storage/memory in the client system.

A still further embodiment adds a function that permanently stores a fetched class in the client system whenever a predetermined condition is met. Such a condition may be the repeated use of a fetched class within a predetermined time space.

The present invention is of particular importance for network computers which are equipped with a storage/memory of limited capacity to keep them as simple and inexpensive as possible. Such a network computer still may execute any Java applet. If a particular class is needed which is not stored in the client's storage, the inventive class loader fetches this respective class or classes from another system, such as a server. It is conceivable that the loading of a class or set of classes is billed using an electronic cash billing system. Furthermore, it is possible that certain classes are only available for down-loading for a specific set of customers, e.g. system administrators.

The present class loader can be further modified such that a message is returned to the user which allows him to stop the process of loading a class not belonging to the reduced set of classes being held in the client system, i.e. the user may control the loading of classes from remote systems. The user may even be provided with information concerning the lengths of a class file to be loaded, the duration the loading will take, and the price.

The present class loader can also be used as template mechanism. the class loader can create and compile a Java class on-the-fly, depending on the passed class specification. Since Java presently lacks a template mechanism, and the Java language is intended to be kept simple, this is a good way of introducing a template mechanism. A class loader, according to the present invention, need not to load an existing class file from a remote system, but can create such a class file itself using a template mechanism. The information given in a qualified class name may be used to create a respective class file.

CLAIMS

1. Client system comprising
   - a Java interpreter for the execution of a Java program, and
   - a set of Java classes for interaction with the client system,

   said client system being characterized in that said set of Java classes is a reduced set of Java classes, and in that said client system comprises means for loading Java classes which do not belong to said reduced set of Java classes into said client system only if needed during execution of said Java program.

2. The client system of claim 1, wherein the Java interpreter returns a string to the means for loading Java classes, said string being used to enable the loading of a class not belonging to the reduced set of classes.

3. The client system of claim 1, wherein the means for loading Java classes fetches a class not belonging to the reduced set of classes from another system holding at least those classes which do not belong to said reduced set of classes.

4. The client system of claim 3, wherein the other system is a remote system, especially a server, connected to said client system via a network.

5. The client system of claim 1, wherein the class not belonging to the reduced set of classes is loaded into a memory of said client system or a storage device connected to said client system.

6. The client system according to any of the preceding claims, further including an erase function for automatically erasing any fetched class not belonging to the reduced set of classes, in particular erasing it whenever one or more predetermined first conditions are met, and/or

an addition function adding any fetched class to said reduced set of classes whenever one or more predetermined second conditions are met.

7. The client system of claim 1, wherein the means for loading Java classes returns a message to the user which allows the user to stop the process of loading a class not belonging to the reduced set of classes.

8. The client system of claim 1 being a computer, in particular a network computer, a set-top box, a telephone, in particular a cellular telephone, a pager, or a printer.

9. The client system of claim 1 being a browser or software tool.

10. Method for executing the byte code of a Java program in a client system, said method comprising the step of:
    - executing said byte code using a Java interpreter,
    - initiating the access to a class referred to by a string in said byte code,
    - looking up the respective class to be accessed,
    - if said respective class belongs to a reduced set of classes found in said client system, loading said respective class into said Java interpreter and returning a pointer which indicates where said respective class is located in said client system's memory,
    - else loading said respective class from another system into said client system's memory using said string and returning a pointer which indicates where said respective class was put in said client system's memory.

11. The method of claim 10, whereby the respective class not belonging to the reduced set of classes is automatically erased whenever a predetermined first condition is met, and/or is added to said reduced set of classes whenever a predetermined second condition is met.

12. The method of claim 10, whereby either the steps for loading classes from another system are repeated such that several classes are loaded sequentially, or wherein several classes are loaded together.

13. The method of claim 10, whereby a network connection to the other system is established before a class is loaded from said other system.

14. The method of claim 10, whereby a message is returned to the user allowing the latter to stop the process of loading a class not belonging to the reduced set of classes.

AMENDED CLAIMS

[received by the International Bureau on 19 February 1998 (19.02.98);
original claims 1, 6, 8 and 10-11 amended;
remaining claims unchanged (3 pages)]

1. Client system comprising a Java interpreter for the execution of a Java program and a stored set of Java classes for interaction with the client system,

said client system being characterized in that

- said stored set is a reduced set of Java classes and said client system comprises
- means for loading an additional Java class, which does not belong to said reduced set of Java classes, into said client system whenever said additional Java class is needed during execution of said Java program, and
- *means for automatically erasing said additional Java class, preferably after is has been used.*

2. The client system of claim 1,
wherein the Java interpreter returns a string to the means for loading an additional Java class, said string being used to enable the loading of an additional class not belonging to the reduced set of classes.

3. The client system of claim 1,
wherein the means for loading an additional Java class fetches a class not belonging to the reduced set of classes from another system holding at least those classes which do not belong to said reduced set of classes.

4. The client system of claim 3,
wherein the other system is a remote system, especially a server, connected to said client system via a network.

5. The client system of claim 1,
wherein the class not belonging to the reduced set of classes is loaded into a memory of said client system or a storage device connected to said client system.

6. The client system of claim 1,
wherein *the means for automatically erasing an additional class* erases the latter whenever one or more predetermined first conditions are met, *preferably after said additional class has been used, and,* whenever one or more predetermined second conditions are met, *maintains it and may add it to to said reduced set of classes.*

7. The client system of claim 1,
wherein the means for loading an additional Java class returns a message to the user which allows the user to stop the process of loading a class not belonging to the reduced set of classes.

8. The client system of claim 1 being a computer, in particular a *small* network computer, *a smartcard*, a set-top box, a telephone, in particular a cellular telephone, a pager, or a printer.

9. The client system of claim 1 being a browser or software tool.

10. Method for executing the byte code of a Java program in a client system, said method comprising the step of:
- executing said byte code using a Java interpreter,
- initiating the access to a class referred to by a string in said byte code,
- looking up the respective class to be accessed,
- if said respective class belongs to a reduced set of classes found in said client system, loading said respective class into said Java interpreter and returning a pointer which indicates where said respective class is located in said client system's memory,
- else loading said respective class from another system into said client system's memory using said string and returning a pointer which indicates where said respective class was put in said client system's memory, and
- *automatically erasing said respective class, preferably after it has been used.*

11. The method of claim 10,
whereby the respective class not belonging to the reduced set of classes is automatically erased if a predetermined first condition is met, and *is maintained and, preferably, added to said reduced set of classes* whenever a predetermined second condition is met.

12. The method of claim 10,
whereby either the steps for loading classes from another system are repeated such that several classes are loaded sequentially or wherein several classes are loaded together.

13. The method of claim 10,
whereby a network connection to the other system is established before a class is loaded from said other system.

14. The method of claim 10,
whereby a message is returned to the user allowing the latter to stop the process of loading a class not belonging to the reduced set of classes.

INTERNATIONAL SEARCH REPORT

Interna al Application No
PCT/IB 97/00135

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 G06F9/445

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 6 G06F

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

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

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	"JAVA PERKS DEVELOPER INTEREST FROM IS TO EMBEDDED SYSTEMS" COMPUTER DESIGN, vol. 35, no. 6, pages 32-34, 37, XP000593778 see page 33, right-hand column, line 2 - line 51 --- -/--	1-14

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"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

1

Date of the actual completion of the international search	Date of mailing of the international search report
8 October 1997	23.10.97

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Fonderson, A

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Internal Application No
PCT/IB 97/00135

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	JAMES GOSLING & HENRY MCGILTON: "The Java Language Environment: A White Paper" SUN MICROSYSTEMS, INC., October 1995, MOUNTAIN VIEW, CA, USA, pages 1-85, XP002042922 (ftp.javasoft.com/docs/papers/whitepaper.A4.ps.zip) see page 16, paragraph 2 see page 52, line 1 - page 53, paragraph 3 see page 55, paragraph 2 see page 72, line 1 - last line see page 75, line 1 - page 81, paragraph 2 ---	1-14
X	EP 0 718 761 A (SUN MICROSYSTEMS INC) 26 June 1996 see page 1, line 13 - page 7, line 5; figures 1-3 ---	1-14
X	"JAVA DYNAMIC CLASS LOADER" IBM TECHNICAL DISCLOSURE BULLETIN, vol. 39, no. 11, page 107/108 XP000679837 see the whole document ---	1-14
X	JAMES GOSLING ET AL.: "The Java Language Specification" August 1996 , ADDISON-WESLEY , USA XP002042923 ISBN:0-201-63451-1 (ftp.javasoft.com/docs/specs/langspec-1.0.ps.zip) see page 215, line 1 - page 217, line 18 see page 218, line 13 - page 219, last line see page 221, line 24 - line 33 see page 235, line 15 - line 26 -----	1-14

1

Form PCT/ISA/210 (continuation of second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Information on patent family members

Intern ial Application No
PCT/IB 97/00135

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
EP 0718761 A	26-06-96	US 5630066 A	13-05-97
		JP 8263447 A	11-10-96