

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28

Katherine Kelly Lutton (CAB # 194971)  
lutton@fr.com  
Robert J. Kent (CAB # 250905)  
RJKent@fr.com  
FISH & RICHARDSON P.C.  
500 Arguello Street, Suite 500  
Redwood City, California 94063  
Telephone: (650) 839-5070  
Facsimile: (650) 839-5071

Attorneys for Plaintiffs  
SUN MICROSYSTEMS INCORPORATED

ORIGINAL  
FILED  
JAN 15 2009  
RICHARD W. WIEKING  
CLERK, U.S. DISTRICT COURT,  
NORTHERN DISTRICT OF CALIFORNIA

E-filing

UNITED STATES DISTRICT COURT  
NORTHERN DISTRICT OF CALIFORNIA

JL

SUN MICROSYSTEMS INCORPORATED, a  
Delaware corporation,

Plaintiff,

v.

IMPLICIT NETWORKS, INC., a Washington  
corporation,

Defendant.

CV No. 09

0201

COMPLAINT FOR DECLARATORY  
JUDGMENT OF NON-INFRINGEMENT,  
INVALIDITY AND  
UNENFORCEABILITY

JURY TRIAL DEMANDED

1. Sun Microsystems Incorporated ("Sun") hereby brings this action for declaratory judgment against Implicit Networks, Inc. ("Implicit"). Specifically, Sun seeks, amongst other things, declaratory judgment of non-infringement, invalidity and unenforceability of U.S. Patent Nos. 6,324,685 ("the '685 patent"), entitled "Applet server that provides applets in various forms," and 6,976,248 ("the '248 patent"), entitled "Application server facilitating with client's computer for applets along with various formats."

2. Upon information and belief, Implicit is a patent holding company that makes no products and whose principal and only business is licensing and enforcing patents. Implicit claims to own the '685 and '248 patents. True and correct copies of the '685 and '248 patents are attached as Exhibits A and B (respectively).

1           3.       Upon information and belief, as part of its patent licensing business strategy, Implicit  
2 offers licenses to, and files patent infringement suits against, prominent high technology companies.  
3 Upon information and belief, through either these arms-length license negotiations or as a result of  
4 litigation settlements, Implicit has in fact licensed various patents to high technology companies.  
5 Such companies include two high technology corporations headquartered in this judicial district.

6           4.       Implicit's public website consists of one page enumerating its technology that is  
7 "currently available for licensing." A true and correct copy of the website is attached as Exhibit C.  
8 At the bottom of the webpage, Implicit solicits requests for licenses over the enumerated  
9 technology, and includes an interactive link allowing website viewers throughout the country  
10 including in this judicial district to send electronic mail to Implicit.

11           5.       On July 15, 2008, Implicit filed a lawsuit against Adobe Systems Incorporated,  
12 International Business Machines Corporation, Oracle Corporation, and SAP America, Inc. in the  
13 United States District Court for the Middle Western District of Washington, Civil Action No. C08-  
14 01080, alleging that various products and services of those respective companies infringed both the  
15 '685 and '248 patents. Implicit seeks damages.

16           6.       On January 14, 2008, Implicit sent a letter to Sun's litigation counsel amongst others.  
17 A true and correct copy of this letter is attached as Exhibit D. In the letter, Implicit states its  
18 intention to imminently add Sun as a defendant in the aforementioned Washington litigation, or in  
19 the alternative file a new complaint against Sun.

20           7.       As the above-stated facts indicate, Implicit has asserted rights under the '685 and  
21 '248 patents based on certain as-yet unidentified ongoing activity of Sun. Based on Implicit's  
22 expressed intention to commence litigation against Sun, an actual, substantial, and continuing  
23 justiciable controversy exists between Sun and Implicit, requiring a declaration of rights by this  
24 Court.

25           8.       In terms of venue, Sun has its headquarters in this judicial district and conducts  
26 substantial business and maintains extensive facilities in this district. It is likely that a great deal of  
27 evidence concerning Sun's allegedly infringing conduct is located in this district. It is also likely  
28

1 that many of the witnesses who will be asked to testify in this matter will be located in this judicial  
2 district. The relevant facts are set forth below.

3 **THE PARTIES**

4 9. Sun is a corporation organized and existing under the laws of Delaware with its  
5 corporate headquarters and principal place of business at 4150 Network Circle, Santa Clara, CA  
6 95054.

7 10. Upon information and belief, Implicit is incorporated under the laws of the state of  
8 Washington with a registered agent for service of process located at 701 Fifth Avenue, # 7000,  
9 Seattle, Washington, 98104.

10 **JURISDICTION AND VENUE**

11 11. This is a declaratory judgment action brought pursuant to the Federal Declaratory  
12 Judgment Act, 28 U.S.C. § 2201 et seq., for patent non-infringement, invalidity and  
13 unenforceability arising under the patent laws of the United States, Title 35, United States Code.  
14 This Court has subject matter jurisdiction over the causes of action stated herein pursuant to 28  
15 U.S.C. §§ 1331, 1338(a) and 2201, because this action concerns a federal question arising under the  
16 patent laws of the United States.

17 12. Implicit is subject to personal jurisdiction in this District because Implicit has  
18 purposefully availed itself of the privilege of doing business in California, including pursuing its  
19 licensing efforts directly and through its website efforts in this judicial district and actually licensing  
20 its various patents to at least two entities headquartered in this judicial district. Further, upon  
21 information and belief, Implicit has substantial contacts with California because of its ongoing  
22 attempts to license patents to other entities in the state of California.

23 13. Venue is proper in this District pursuant to 28 U.S.C. §§ 1391 and 1400(b) because  
24 Implicit has engaged in significant activity in this district including seeking licensing arrangements  
25 in this district.

26 **COUNT I: NON-INFRINGEMENT OF THE '685 AND '248 PATENTS**

27 14. Sun incorporates by reference and realleges paragraphs 1 through 13, above, as if  
28 fully set forth herein.

1 15. The products and services provided by Sun and utilized by its customers do not  
2 infringe and have not infringed, either directly, indirectly, or by equivalents, any claim of the '685  
3 patent or the '248 patent.

4 **COUNT II: INVALIDITY OF THE '685 AND '248 PATENTS**

5 16. Sun incorporates by reference and realleges paragraphs 1 through 15, above, as if  
6 fully set forth herein.

7 17. The claims of the '685 and '248 patents are invalid for failure to satisfy the  
8 requirements of patentability specified by Title 35 of the United States Code, including §§ 101, 102,  
9 103 and/or 112.

10 **COUNT III: DECLARATORY RELIEF REGARDING EQUITABLE ESTOPPEL**

11 18. Sun incorporates by reference and realleges paragraphs 1 through 17, above, as if  
12 fully set forth herein.

13 19. The first of Implicit's '685 and '248 patents issued on November 27, 2001—over  
14 seven years ago and more than six years before the institution of this action and Sun's first notice of  
15 Implicit's claims. Since that time, Implicit has unreasonably delayed asserting its claims with  
16 prejudice to Sun. During this delay, Sun relied on its detriment to Implicit's affirmative actions and  
17 representations (in the claims, patent application and prosecution history) about what its patents did  
18 and did not cover.

19 20. Pursuant to the Federal Declaratory Judgment Act, 28 U.S.C. §§ 2201 et seq., Sun  
20 requests the declaration of the Court that the '685 and '248 patents are unenforceable by the  
21 doctrine of equitable estoppel.

22 **COUNT IV: DECLARATORY RELIEF REGARDING LACHES**

23 21. Sun incorporates by reference and realleges paragraphs 1 through 20, above, as if  
24 fully set forth herein.

25 22. The first of Implicit's '685 and '248 patents issued on November 27, 2001—over  
26 seven years ago. Since that time, Implicit has unreasonably and inexcusably delayed bringing any  
27 action and now threatens to imminently file suit.

28 ///

1           23. Pursuant to the Federal Declaratory Judgment Act, 28 U.S.C. §§ 2201 et seq., Sun  
2 requests the declaration of the Court that the '685 and '248 patents are unenforceable by the  
3 doctrine of laches.

4                           **COUNT V: DECLARATORY RELIEF REGARDING MARKING**

5           24. Sun incorporates by reference and realleges paragraphs 1 through 23, above, as if  
6 fully set forth herein.

7           25. Upon information and belief, prior to the filing of this action, Sun was not given  
8 constructive notice of Implicit's claimed patent rights through marking or other appropriate means.

9           26. Pursuant to the Federal Declaratory Judgment Act, 28 U.S.C. §§ 2201 et seq., Sun  
10 requests the declaration of the Court that Implicit failed to mark or otherwise provide proper notice  
11 of the '685 and '248 patents pursuant to 35 U.S.C. § 287 and therefore any potential damages are  
12 accordingly limited.

13                           **COUNT VI: DECLARATORY RELIEF REGARDING**  
14                           **PROSECUTION HISTORY ESTOPPEL**

15           27. Sun incorporates by reference and realleges paragraphs 1 through 26, above, as if  
16 fully set forth herein.

17           28. During prosecution of the '685 and '248 patents, Implicit made statements and  
18 representations to the Patent Office, without limitation relating to the definition of the term  
19 "applet," to which the applicant must now be bound. Implicit made these statements in attempt to  
20 seek allowance of the patents and cannot now try to recapture that which the application gave up.

21           29. Pursuant to the Federal Declaratory Judgment Act, 28 U.S.C. §§ 2201 et seq., Sun  
22 requests the declaration of the Court that Implicit is barred, under the doctrine of Prosecution  
23 History Estoppel, from construing the claims of either the '685 patent or the '248 patent in such a  
24 way as to cover any of Sun's products or processes.

25                           **COUNT VII: EXCEPTIONAL CASE**

26           30. Sun incorporates by reference and realleges paragraphs 1 through 29, above, as if  
27 fully set forth herein.

28           ///

1 31. On information and belief, prior to filing its complaint, Implicit knew, or reasonably  
2 should have known, that the patent claims were invalid and/or not infringed by Sun or that its  
3 claims were barred in whole or in part. Implicit's filing of the complaint and continuing to pursue  
4 its present claims in view of this knowledge makes this case exceptional within the meaning of  
5 35 U.S.C. § 285.

6 **COUNT VIII: DECLARATORY RELIEF REGARDING DAMAGES**

7 32. Sun incorporates by reference and realleges paragraphs 1 through 29, above, as if  
8 fully set forth herein.

9 33. Upon information and belief, some of the products accused of infringing the '685 or  
10 '248 patents are used by or manufactured for the United States as provided in 28 U.S.C. § 1498(a).

11 34. Pursuant to the Federal Declaratory Judgment Act, 28 U.S.C. §§ 2201 et seq., SAP  
12 requests the declaration of the Court that Implicit's claims for relief and prayer for damages may be  
13 limited due by 28 U.S.C. § 1498(a).

14 **PRAYER FOR RELIEF**

15 WHEREFORE, Sun prays for judgment against Implicit as follows:

16 (a) Declare that Sun's products do not infringe, either directly or indirectly, literally or  
17 by equivalents, any claim of the '685 patent or the '248 patent;

18 (b) Declare that the '685 and '248 patents are invalid;

19 (c) Declare that the '685 and '248 patents are unenforceable;

20 (d) Issue an injunction preventing Implicit and those in concert with Implicit from  
21 harassing Sun or Sun customers, seeking licenses from Sun or Sun customers, and accusing Sun or  
22 Sun customers of infringing any claim of Implicit's '685 or '248 patents;

23 (e) Issue an Order awarding Sun its costs, expenses and reasonable attorney fees as  
24 provided by law; and

25 (f) Award Sun any other and further relief as this Court may deem just and proper.

26 ///

27 ///

28 ///

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28

**DEMAND FOR JURY TRIAL**

Plaintiff requests a trial by jury on all matters appropriately tried to a jury.

Dated: January 15, 2009

FISH & RICHARDSON P.C.

By:   
Robert J. Kent

Attorneys for Plaintiffs  
SUN MICROSYSTEMS INCORPORATED

50630413.doc

# **EXHIBIT A**





US006324685B1

(12) **United States Patent**  
**Balassanian**

(10) **Patent No.:** **US 6,324,685 B1**

(45) **Date of Patent:** **\*Nov. 27, 2001**

(54) **APPLET SERVER THAT PROVIDES APPLET IN VARIOUS FORMS**

(75) Inventor: **Edward Balassanian**, Kirkland, WA (US)

(73) Assignee: **BeComm Corporation**, Redmond, WA (US)

(\* ) Notice: This patent issued on a continued prosecution application filed under 37 CFR 1.53(d), and is subject to the twenty year patent term provisions of 35 U.S.C. 154(a)(2).

Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/040,972**

(22) Filed: **Mar. 18, 1998**

(51) **Int. Cl.**<sup>7</sup> ..... **G06F 9/45**

(52) **U.S. Cl.** ..... **717/5; 717/1**

(58) **Field of Search** ..... **395/705, 701, 395/200.33, 200.32, 188.01; 717/5, 1; 709/203, 202; 713/202**

(56) **References Cited**

**U.S. PATENT DOCUMENTS**

|           |   |         |              |            |
|-----------|---|---------|--------------|------------|
| 5,805,829 | * | 9/1998  | Cohen et al. | 395/200.32 |
| 5,828,840 | * | 10/1998 | Cowan et al. | 395/200.33 |
| 5,848,274 | * | 12/1998 | Hamby et al. | 395/705    |
| 5,872,915 | * | 2/1999  | Dykes et al. | 395/188.01 |
| 5,884,078 | * | 3/1999  | Faustini     | 395/701    |

**OTHER PUBLICATIONS**

"Eliminating Unnecessary Synchronization," <http://kimera.cs.washington.edu/synch/index.html> [Accessed Oct. 4, 2000].

Sirer, Emin Gün, "Kimera Paper Trail," <http://kimera.cs.washington.edu/papers/index.html> [Accessed Oct. 4, 2000].

Sirer, Emin Gün, "Java, Extensibility and Security Related Links," <http://kimera.cs.washington.edu/related/index.html> [Accessed Oct. 4, 2000].

Sirer, Emin Gün, "Java-Relevant Articles in the Press," <http://kimera.cs.washington.edu/press/index.html> [Accessed Oct. 4, 2000].

"Project Members" <http://kimera.cs.washington.edu/members.html> [Accessed Oct. 4, 2000].

Emin Gün Sirer, et al., "Distributed Virtual Machines: A System Architecture for Network Computing," Dept. of Computer Science & Engineering, University of Washington, Seattle, Washington <http://kimera.cs.washington.edu> Feb. 26, 1998.

(List continued on next page.)

*Primary Examiner*—Mark R. Powell

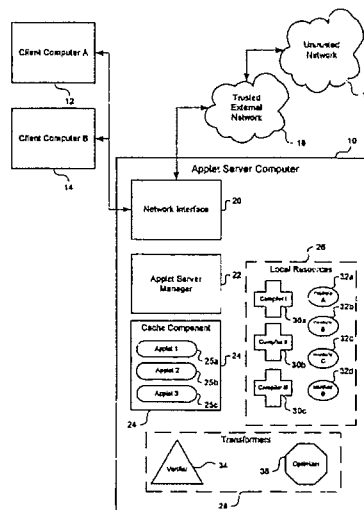
*Assistant Examiner*—Hoang-Vu Antony Nguyen-Ba

(74) *Attorney, Agent, or Firm*—Perkins Coie LLP

(57) **ABSTRACT**

The present invention is an applet server which accepts requests for applets from client computers. A request specifies the format in which an applet is to be delivered to the requesting client computer. The applet server has a cache which it uses to store applets for distribution to client computers. If the specified form of the requested applet is available in the cache, the applet server transmits the applet to the requesting client. If the applet is not available in the cache, the server will attempt to build the applet from local resources (program code modules and compilers) and transformer programs (verifiers and optimizers). If the applet server is able to build the requested applet, it will then transmit the applet to the requesting client computer. If the applet server is unable to build the requested applet, it will pass the request to another applet server on the network for fulfillment of the request.

**106 Claims, 3 Drawing Sheets**



US 6,324,685 B1

Page 2

---

OTHER PUBLICATIONS

Emin Gün Sirer, et al., "Design and Implementation of a Distributed Virtual Machine for Networked Computers," University of Washington, Department of Computer Science and Engineering, Seattle Washington, 17<sup>th</sup> ACM Symposium on Operating system Principles, Dec. 1999.

Sirer, Emin Gün, "A System Architecture for Next Generation Network Computing," Dept. of Computer Science & Engineering, University of Washington, Seattle, Washington <http://www.dyncorp-is.com/darpa/meetings/gradmeet98/Whitepapers/darpa-wp.html> Jun. 26, 1998.

Sirer, Emin Gün, <http://www.cs.washington.edu/homes/egs/> [Accessed Oct. 4, 2000].

Sirer, Emin Gün, "Kimera—A System Architecture for Networked Computers," <http://kimera.cs.washington.edu/> [Accessed Oct. 4, 2000].

Emin Gün Sirer and Brian Bershad, "Kimera Architecture," <http://kimera.cs.washington.edu/overview.html> [Accessed Oct. 4, 2000].

Sirer, Emin Gün, "Security Flaws in Java Implementations," <http://kimera.cs.washington.edu/flaws/index.html> [Accessed Oct. 4, 2000].

Sirer, Emin Gün, "Kimera Bytecode Verification," <http://kimera.cs.washington.edu/verifier.html> [Accessed Oct. 4, 2000].

Sirer, Emin Gün, "Kimera Test Suite," <http://kimera.cs.washington.edu/testsuite.html> [Accessed Oct. 4, 2000].

Sirer, Emin, Gün, "Kimera Disassembler," <http://kimera.cs.washington.edu/disassembler.html> [Accessed Oct. 4, 2000].

\* cited by examiner

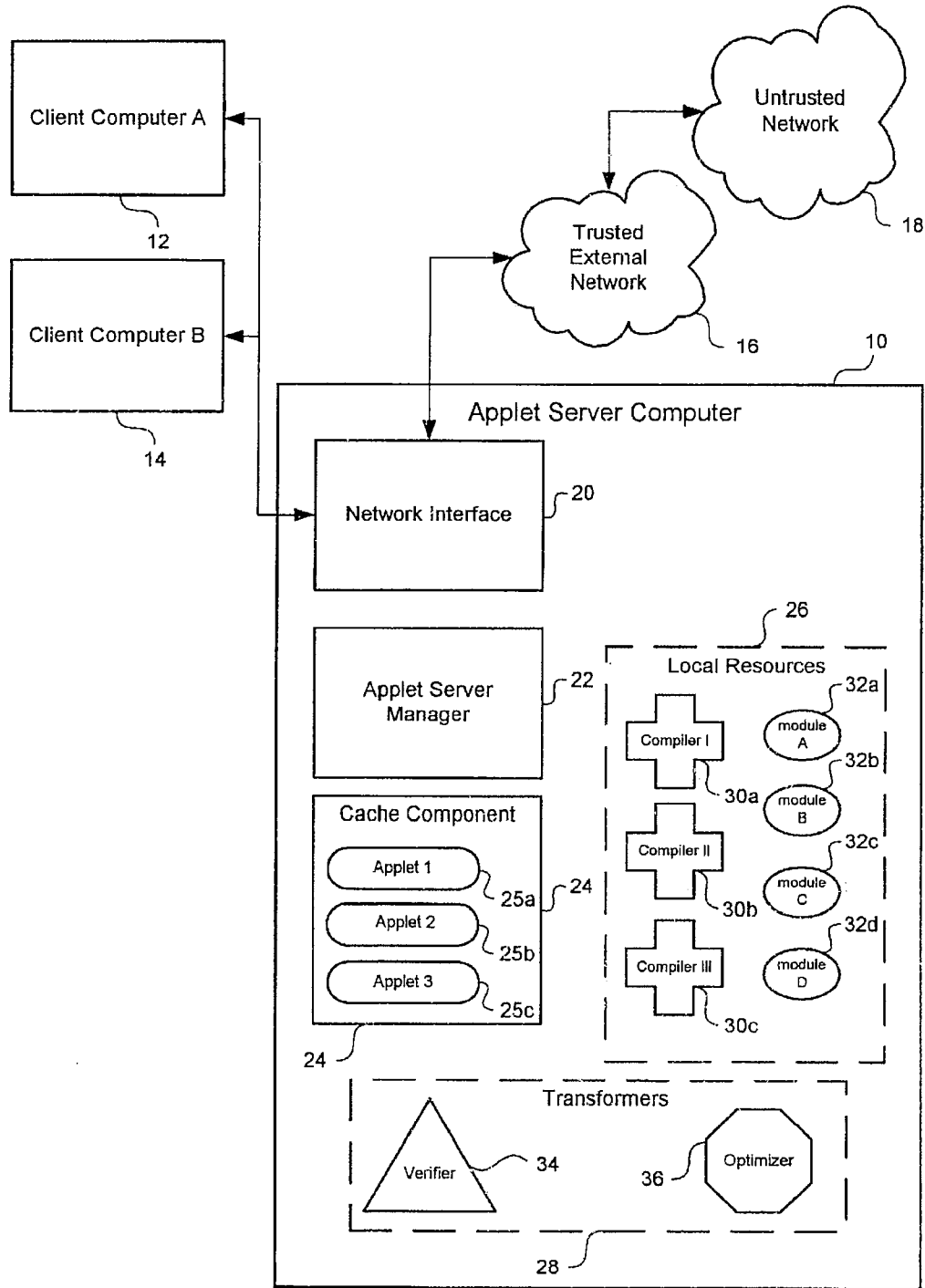


Fig. 1

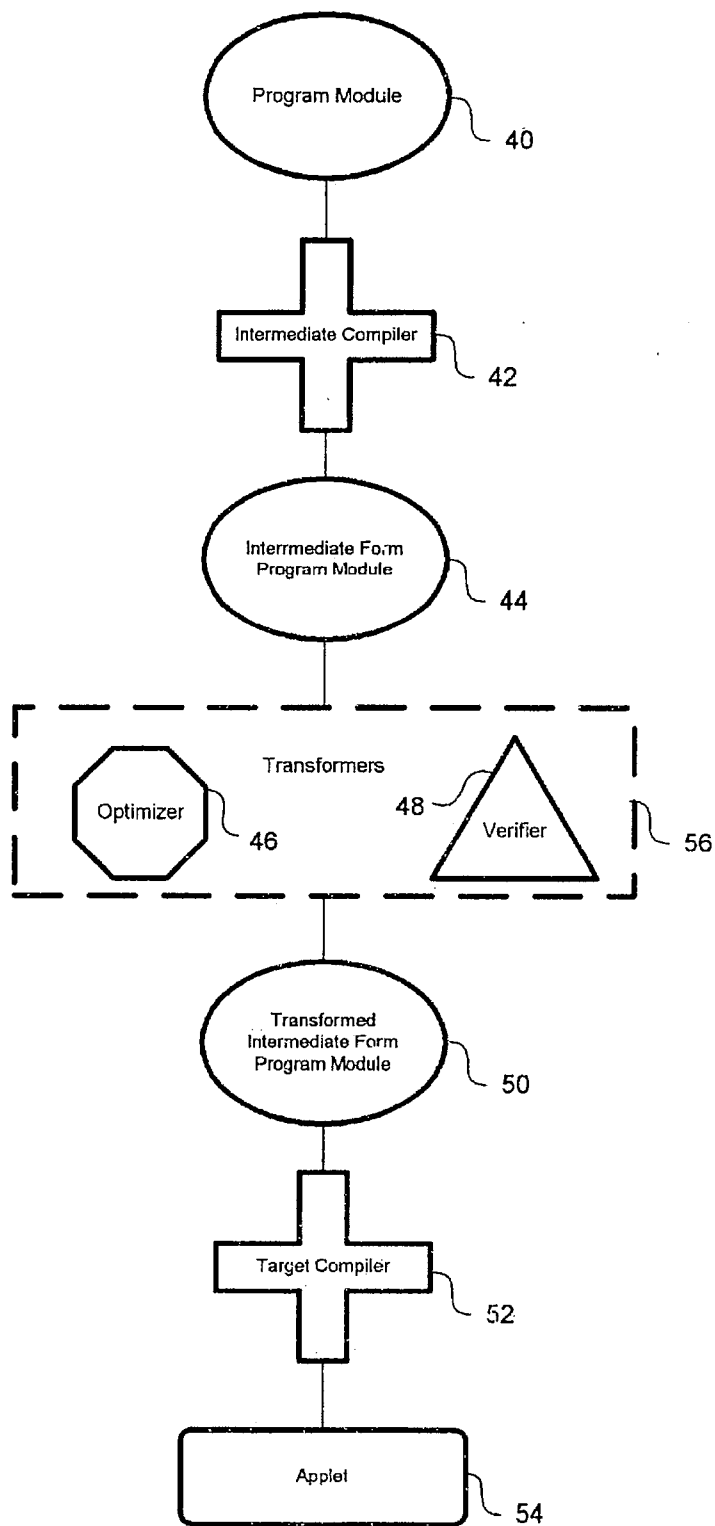
**Request Data Type**

| Tag                | Value  |
|--------------------|--|
| Applet-URL         | (String) specifies the name of the requested applet  |
| Code-Type          | (Source/Intermediate/Binary) specifies the format the applet is to be delivered to the requesting client in. A request for binary would specify the CPU of the requesting client (e.g., x86) |
| Verification-Level | (0-100) specifies the degree of verification to be performed. 0 = no/minimal verification, 100 = maximum verification (highest level of security).   |
| Optimization-Level | (0-100) specifies the degree of optimization to be performed. 0 = no/minimal optimization, 100 = maximum optimization.   |

**Fig. 2A****Code Data Type**

| Tag                | Value  |
|--------------------|--|
| Applet-URL         | (String) specifies the name of the requested applet  |
| Code-Type          | (Source/Intermediate/Binary) specifies the format the applet is to be delivered to the requesting client in. A request for binary would specify the CPU of the requesting client (e.g., x86) |
| Verification-Level | (0-100) specifies the degree of verification to be performed. 0 = no/minimal verification, 100 = maximum verification (highest level of security).   |
| Optimization-Level | (0-100) specifies the degree of optimization to be performed. 0 = no/minimal optimization, 100 = maximum optimization.   |
| Applet Length      | (0-2 <sup>32</sup> ) specifies the size of the requested applet.   |
| Applet Code        | The Requested Applet in the form specified by the request data type.   |

**Fig. 2B**



*Fig. 3*

US 6,324,685 B1

1

## APPLET SERVER THAT PROVIDES APPLETS IN VARIOUS FORMS

### FIELD OF THE INVENTION

The present invention relates to computer operating systems and, in particular, to a server architecture providing application caching and security verification.

### BACKGROUND OF THE INVENTION

The growth of the Internet's importance to business, along with the increased dependence upon corporate networks, has created a demand for more secure and efficient computer systems. The traditional solution to this problem has been to depend upon improvements in hardware performance to make up for the performance penalty that is typically incurred when a computer system is made more secure and stable. Increased interoperability has also created a need for improved interoperability amongst a variety of computers that are now connected to one another. One solution to the problem of the variety of computers interconnected via the Internet and corporate networks has been the development of portable architecture neutral programming languages. The most widely known of these is Java, though, there are numerous other architecture neutral languages.

Architecture neutral programming languages allow programs downloaded from a server computer to a client computer to be interpreted and executed locally. This is possible because the compiler generates partially compiled intermediate byte-code, rather than fully compiled native machine code. In order to run a program, the client machine uses an interpreter to execute the compiled byte-code. The byte-codes provide an architecture neutral object file format, which allows the code to be transported to multiple platforms. This allows the program to be run on any system which implements the appropriate interpreter and run-time system. Collectively, the interpreter and runtime system implement a virtual machine. This structure results in a very secure language.

The security of this system is premised on the ability of the byte-code to be verified independently by the client computer. Using Java or some other virtual machine implementing technology, a client can ensure that the downloaded program will not crash the user's computer or perform operations for which it does not have permission.

The traditional implementations of architecture neutral languages are not without problems. While providing tremendous cross platform support, the current implementations of architecture neutral languages require that every client performs its own verification and interpretation of the intermediate code. The high computation and memory requirements of a verifier, compiler and interpreter restrict the applicability of these technologies to powerful client computers.

Another problem with performing the verification process on the client computer is that any individual within an organization may disable some or all of the checks performed on downloaded code. The current structure of these systems makes security management at the enterprise level almost impossible. Since upgrades of security checking software must be made on every client computer, the cost and time involved in doing such upgrades makes it likely that outdated or corrupt copies of the verifier or interpreter exist within an organization. Even when an organization is diligent in maintaining a client based security model, the size of the undertaking in a large organization increases the likelihood that there will be problems.

2

There is a need for a scalable distributed system architecture that provides a mechanism for client computers to request and execute applets in a safe manner without requiring the client machines to have local resources to compile or verify the code. There is a further need for a system in which the applets may be cached in either an intermediate architecture neutral form or machine specific form in order to increase overall system performance and efficiency.

### SUMMARY OF THE INVENTION

In accordance with one embodiment of the invention, an applet server architecture is taught which allows client computers to request and execute applets in a safe manner without requiring the client to have local resources to verify or compile the applet code. Compilation and byte-code verification in the present invention are server based and thereby provide more efficient use of resources and a flexible mechanism for instituting enterprise-wide security policies. The server architecture also provides a cache for applets, allowing clients to receive applet code without having to access nodes outside the local network. The cache also provides a mechanism for avoiding repeated verification and compilation of previously requested applet code since any client requesting a given applet will have the request satisfied by a single cache entry.

Machine specific binary code is essentially interpreted code since the processor for a given computer can essentially be viewed as a form of an interpreter, interpreting binary code into the associated electronic equivalents. The present invention adds a level of indirection in the form of an intermediate language that is processor independent. The intermediate language serves as the basis for security verification, code optimizations, or any other compile time modifications that might be necessary. The intermediate form allows a single version of the source to be stored for many target platforms instead of having a different binary for each potential target computer. Compilations to the target form can either be done at the time of a cache hit or they can be avoided all together if the target machine is able to directly interpret the intermediate form. If the compilation is done on the server, then a copy of the of the compiled code as well as the intermediate form can be stored in the cache. The performance advantage derived from caching the compiled form as well as the intermediate depends upon the number of clients with the same CPU.

The novel features believed characteristic of the invention are set forth in the appended claims. The invention itself, however, as well as other features and advantages thereof will best be understood by reference to the detailed description which follows, when read in conjunction with the accompanying drawings.

### BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is diagram showing the major components which may be used to implement an applet server in one embodiment of the present invention;

FIG. 2a is a table which illustrates the structure of the request format data type;

FIG. 2b is a table which illustrates the structure of the returned code data type.

FIG. 3 is a diagram showing the compilation and transformation of a program module into an applet in a particular form.

### DETAILED DESCRIPTION OF THE INVENTION

Referring to FIG. 1, an applet server architecture according to one embodiment of the invention is based on an applet



US 6,324,685 B1

3

server computer 10 which in turn is connected to client computer A12, client computer B14, an external network 16 and an untrusted network 18. The applet server computer 10 connects to client computers 12 and 14, an external network 16, and an untrusted network 18 by means of a network interface 20. Typically this connection will involve one or more of the computers or networks having a connection to the Internet.

The applet server computer 10 accomplishes its objectives by manipulating computer programs in several formats. An applet (e.g. applets 1-3, 25a-25c) is any form of program instructions, whether in binary, source or intermediate format. In the case of this architecture, the applet code can either be a self contained program, or it can be a code fragment associated with a larger application.

Binary format refers to processor specific machine instructions suitable for running natively on a given computing platform (also referred to as "target" because of the concept of "targeting" a compiler to produce binary code for a given processor type).

Source refers to non-binary applet code, generally in the form of higher level languages (i.e. C, C++, Java, Visual Basic, ActiveX, Fortran, and Modula).

Intermediate format refers to a common intermediate byte-code that is produced by compiling a given source code input. The intermediate byte-code need not necessarily be Java byte-code.

Treating applets in this general sense allows client computers 12 and 14 to request not only applications, but portions of applications. Client computers 12 and 14 are thus able to use applet server computer 10 as the equivalent of a loader, loading in appropriate parts of the application in the form of applets. In turn client computers 12 and 14 can run large applications without requiring that the client computers 12 and 14 have the resources to store the entire application in memory at once.

Having the applets delivered from applet server computer 10 allows code in intermediate form to be verified, optimized, and compiled before being transmitted to client computers 12 and 14. This reduces the amount of work the client computers 12 and 14 have to do and provides a convenient way to impose global restrictions on code.

In operation, client computer A 12 transmits a request to an applet server computer 10 requesting an applet in a particular form. The form may be selected from a large matrix of many possible forms that can be recognized by the system. The request specifies the format (source, intermediate, or binary) in which the client wishes to receive the applet. The request may also specify that the applet be verified or have some other transformation operation performed upon it. Verification, optimization and compression are examples of types of transformation operations. The request is received by the network interface 20 of the applet server computer 10 which passes the request onto the applet server manager 22.

After interpreting the request, the applet server manager 22 checks to see if the requested applet is available in the cache 24. The cache 24 stores applets in a variety of formats (source, intermediate, or binary). If the requested form of the applet is available in the cache 24 (applet 1 25a, applet 2 25b, or applet 3 25c in this example) the applet server manager 22 instructs the network interface 20 to transmit the applet to requesting client computer A 12. If the requested applet is not available in the cache 24, then the applet server manager 22 will attempt to build the requested applet from local resources 26 and one or more transformation opera-

4

tions performed by one or more of the transformers 28. Local resources 26 are comprised of compilers 30a, 30b and 30c and program code modules 32a, 32b, 32c and 32d. The requested applet is built by selecting one or more program code modules 32 and compiling them with one or more compilers 30. Transformer operations may be performed by the verifier 34 or the optimizer 36. After the applet server manager 22 builds the applet, the network interface 20 transmits the applet to the requesting client computer A 12.

If the request can not be satisfied by building the applet from local resources 26 and transformers 28, the applet server manager 22 will pass a request for the requested applet to external network 16 and/or untrusted network 18. The applet server manager 22 may request the applet in intermediate form or in executable form or it may request the local resources 26 and transformers 28 it needs to complete building the applet itself.

The cache 24 is capable of responding to the following commands: GET, PUT, and FLUSH. GET is used to retrieve a given applet from the cache. PUT is used to store an applet in the cache. FLUSH is used to clear the cache of one or more entries. When the cache is unable to locate an item in response to a GET operation, it returns a cache miss. The program which issued the GET command is then responsible for locating the desired form of the applet by other means and optionally storing it in the cache when it is retrieved (using the PUT operation). The FLUSH command will clear the cache of one or more entries and any subsequent GETs for the FLUSHed applet code will result in a cache miss. This is useful if a particular applet needs to be updated from a remote server on a periodic basis. When using PUT, the program issuing the command specifies a time to live (TTL) in the cache. When the TTL expires, the cache entry is removed by means of a FLUSH operation.

Local resources 26 are comprised of program modules 32 (applets in source form, not the requested form) and compilers 30. The program modules 32 are run through the compilers 30 in order to produce applets in the requested form. The applet server manager 20 may also direct the modules 32 to be processed by a verifier 34 or another transformer such as an optimizer 36. Program modules 32 are program code used to build applets. Program modules 32 may be stored in local resources 26 in source, binary, or intermediate formats. When an applet is built it may require the operation of one or more compilers 30 upon one or more program modules 32. The program modules 32 may be combined and recompiled with previously cached applets and the resulting applet may be also cached for use at a future time. Additionally, program modules 32, compilers 30 and transformers 28 (including verifiers 34 and optimizers 36) may be distributed across a network. The applet server manager 22 may pass requests for the components it needs to build a particular applet back to the network interface 20 which in turn passes the request onto the rest of the network and may include external network 16 and untrusted network 18.

FIG. 3 provides further illustration of how an applet is produced from local resources and transformers. In this illustration the request is for an optimized and verified applet compiled to a machine specific form. A program module 40 is compiled into an intermediate form program module 44 by an intermediate compiler 42. The intermediate form program module 44 is then transformed by an optimizer 46 or a verifier 48. The resulting transformed intermediate form program module 50 is then compiled by target compiler 52 into machine specific code applet 54.

There are two types of compilers used to build applets: intermediate compilers 42 and target compilers 52. The

US 6,324,685 B1

5

intermediate compiler 42 compiles program modules (source applet code) 40 and produces a common intermediate pseudo-binary representation of the source applet code (intermediate form program module 44). The word pseudo is used because the intermediate form 44 is not processor specific but is still a binary representation of the source program module 40. This intermediate form can be re-targeted and compiled for a particular processor. Alternatively, the intermediate form 44 can be interpreted by an interpreter or virtual machine that understands the internal binary representation of the intermediate form. A target compiler 52 compiles intermediate applet code 44 into an applet 54 in a processor specific format (binary) suitable for running natively on a given computing platform.

Transformers 56 are programs that take in intermediate code and put out intermediate code. Transformers 56 are generally used for things like verification and optimization. Other transformers might include compressors that identify portions of code that can be replaced with smaller equivalents. Transformers can be matched up to any other component that takes in intermediate code as an input. These include the cache 24 and the target compilers 52. Global policies for transformers 56 can be implemented which ensure that all applets are run through some set of transformers before being returned to the client.

A verifier 48 is a type of transformer that is able to analyze input code and determine areas that might not be safe. The verifier 48 can determine the level of safety. Some verifiers 48 look for areas where unsafe or protected memory is being accessed, others might look for accesses to system resources such as I/O devices. Once a verifier 48 determines the portion of unsafe applet code several steps can be taken. The offending code portion can be encased with new code that specifically prevents this unsafe code section from being executed. The unsafe code can be modified to be safe. The unsafe code can be flagged in such a way that a user can be warned about the possible risks of executing the code fragment. The verifier's role can therefore be summarized as determining where unsafe code exists and possibly altering the offending code to render it harmless. Verifiers 48 can operate on any format of input code, whether in source, intermediate or binary form. However, since intermediate code is a common format, it is most efficient to have a single verifier that will operate on code in this format. This eliminates the need to build specific knowledge of various source languages into the verifier. Verifiers 48 are a form of a transformer. Verifiers 48 take in intermediate code and put out verified intermediate code. Verifiers 48 are responsible for identifying non-secure portions of code in the intermediate code and modifying this code to make it secure. Security problems generally include access to memory areas that are unsafe (such as system memory, or memory outside the application space of the applet).

The choice of adding in the verification step can be left up to the client computer 12, the applet server computer 10 (see FIG. 1), or can be based on the network that the applet originated from. Server managers can institute global policies that affect all clients by forcing all applets to be run through the verifier 48. Alternatively, verification can be reserved for un-trusted networks (18 in FIG. 1), or it can be left up to the client to determine whether the verification should be performed. In the preferred embodiment, verification level is determined by the applet server 10. In this way, a uniform security policy may be implemented from a single machine (i.e., the applet server 10).

Optimizers 46 are another type of transformer program. Optimizers 46 analyze code, making improvements to well

6

known code fragments by substituting in optimized but equivalent code fragments. Optimizers 46 take in intermediate code 44 and put out transformed intermediate code 50. The transformed intermediate code 50 is functionally equivalent to the source intermediate code 44 in that they share the same structure.

Referring again to FIG. 1, policies may be instituted on the applet server 10 that force a certain set of request parameters regardless of what the client asked for.

For example, the applet server manager 22 can run the applet through a verifier 34 or optimizer 36 regardless of whether the client 12 requested this or not. Since the server 10 might have to go to an untrusted network 18 to retrieve a given applet, it will then run this applet through the required transformers 28, particularly the verifier 34 before returning it to the client 12. Since clients 12 and 14 have to go through the applet server computer 10, this ensures that clients 12 and 14 do not receive applets directly from an untrusted network 18. In addition, since the server will be dealing directly with untrusted network 18, it can be set up to institute policies based on the network. A trusted external network 16 may be treated differently than an untrusted network 18. This will provide the ability to run a verifier 34 only when dealing with an untrusted network 18, but not when dealing with a trusted external network 16. In one embodiment, all intermediate code is passed through a verifier 34 and the source of the code merely determines the level of verification applied.

The client 12 is the target computer on which the user wishes to execute an applet. The client 12 requests applets from the server 10 in a specific form. Applets can be requested in various formats including source, intermediate and binary. In addition, an applet can be requested with verification and/or other compile time operations. Optionally, the client 12 can pass a verifier to the server to provide verification. If the server 10 implements its own security, then both the client and server verifiers will be run. The verifier that is passed from the client to the server is cached at the server for subsequent verification. The client can refer to this verifier by a server-generated handle to avoid having to pass the verifier each time an applet is requested.

Client computers 12 and 14 requesting applet code in intermediate format need to have an interpreter or virtual machine capable of interpreting the binary code in the intermediate format if the applet is to be executed on the client machine.

In the preferred embodiment, requests to the applet server are in a format similar to those of an HTTP header and are comprised of tags and values. In one embodiment, an HTTP GET method is used to make the request (though use of the HTTP protocol is not necessary to implement the present invention). The request is made up of a series of tags which specify the requested applet, the platform on which it is to be run and the type of code (source/intermediate/binary), a verification level and an optimization level. New tags and values can be added to extend functionality as needed and the applet server manager 22 will discard any tag it does not recognize. When the applet server computer 10 returns the requested applet to the requesting client computer 12, it will transmit the request header followed by the applet code. In this instance, the header will additionally include a field which defines the length of the applet code. FIG. 2 provides a table which illustrates the request format and the returned code format.

While this invention has been described with reference to specific embodiments, this description is not meant to limit



US 6,324,685 B1

7

the scope of the invention. Various modifications of the disclosed embodiments, as well as other embodiments of the invention, will be apparent to persons skilled in the art upon reference to this description. It is therefore contemplated that the appended claims will cover any such modifications or embodiments as fall within the scope of the invention.

I claim:

1. A method in a server computer for providing applications to client computers, the method comprising:
  - receiving a request from a client computer, the request identifying an application and identifying a form of the application, the identified form being one of a plurality of available forms;
  - in response to receiving the request, generating the identified form of the application from another form of the application; and
  - sending the identified form of the application to the client computer; and
  - caching the identified form of the application so that when another request is received for the application in the identified form, the identified form of the application can be sent without regenerating the identified form of the application.
2. A method in a server computer for providing applications to client computers, the method comprising:
  - receiving a request from a client computer, the request identifying an application and identifying a form of the application, the identified form being one of a plurality of available forms; and
  - in response to receiving the request, when the server computer does not have the application in the other form, requesting the application in the other form from a computer other than the server computer; generating the identified form of the application from another form of the application; and
  - sending the identified form of the application to the client computer.
3. A method in a server computer for providing applications to client computers, the method comprising:
  - receiving a request from a client computer, the request identifying an application and identifying a form of the application, the identified form being one of a plurality of available forms; and
  - in response to receiving the request, generating the identified form of the application from another form of the application including when the server computer does not have the identified form of the application, requesting the application in the other form from a computer other than the server computer; and
  - sending the identified form of the application to the client computer.
4. A method in a server computer for providing applications to client computers, the method comprising:
  - receiving a request from a client computer, the request identifying an application and identifying a form of the application, the identified form being one of a plurality of available forms; and
  - in response to receiving the request, generating the identified form of the application from another form of the application; and
  - sending the identified form of the application to the client computer
 wherein the identified form is an intermediate form.
5. The method of claim 4 wherein the intermediate form is Java byte code.

8

6. The method of claim 4 wherein the intermediate form can be interpreted by an interpreter executing on the client computer.

7. A method in a server computer for providing applications to client computers, the method comprising:

receiving a request from a client computer, the request identifying an application and identifying a form of the application, the identified form being one of a plurality of available forms; and

in response to receiving the request, generating the identified form of the application from another form of the application; and sending the identified form of the application to the client computer

wherein the identified form is a target form.

8. The method of claim 7 wherein the target form is directly executable by a processor of the client computer.

9. A method in a server computer for providing applications to client computers, the method comprising:

receiving a request from a client computer, the request identifying an application and identifying a form of the application, the identified form being one of a plurality of available forms; and

in response to receiving the request, generating the identified form of the application from another form of the application; and sending the identified form of the application to the client computer

wherein the other form is a source form.

10. The method of claim 9 wherein the source form is Java source.

11. A method in a server computer for providing applications to client computers, the method comprising:

receiving a request from a client computer, the request identifying an application and identifying a form of the application, the identified form being one of a plurality of available forms; and

in response to receiving the request, transforming the application into a transformed form; generating the identified form of the application from the transformed form of the application; and sending the identified form of the application to the client computer.

12. The method of claim 11 wherein the transforming is verifying the application.

13. The method of claim 11 wherein the transforming is optimizing the application.

14. The method of claim 11 wherein the transforming is compressing the application.

15. A method in a server computer for providing applications to client computers, the method comprising:

receiving a request from a client computer, the request identifying an application and identifying a form of the application, the identified form being one of a plurality of available forms; and

in response to receiving the request, generating the identified form of the application from another form of the application and caching the generated identified form so that when the identified form of the application has been stored in a cache, the identified form of the application is retrieved from the cache; and

sending the identified form of the application retrieved from the cache to the client computer.

16. A method in a server computer for providing applications to client computers, the method comprising:

US 6,324,685 B1

9

receiving a request from a client computer, the request identifying an application and identifying a form of the application, the identified form being one of a plurality of available forms; and

in response to receiving the request,

generating the identified form of the application from another form of the application; and

sending the identified form of the application to the client computer

wherein the application is a portion of a larger application.

17. A method in a server computer for providing applications to client computers, the method comprising:

receiving a request from a client computer, the request identifying an application and identifying a form of the application, the identified form being one of a plurality of available forms; and

in response to receiving the request,

generating the identified form of the application from another form of the application; and

sending the identified form of the application to the client computer

wherein the server computer functions as a loader for the client computer.

18. A method in a server computer for providing applications to client computers, the method comprising:

receiving a request from a client computer, the request identifying an application and identifying a form of the application, the identified form being one of a plurality of available forms; and

in response to receiving the request,

generating the identified form of the application from another form of the application; and

sending the identified form of the application to the client computer wherein the application is an applet.

19. A method in a server computer for providing applications to client computers, the method comprising:

receiving a request from a client computer, the request identifying an application and identifying a form of the application, the identified form being one of a plurality of available forms; and

in response to receiving the request,

generating the identified form of the application from another form of the application; and

sending the identified form of the application to the client computer

wherein the application includes modules, wherein the generating includes generating of modules of the identified form, and wherein the generating includes combining modules of the identified form that were previously generated with modules of the identified form that are generated in response to receiving the request.

20. The method of claim 19 wherein the modules of the identified form that were previously generated are retrieved from a cache.

21. A method in a server computer for providing applications to client computers, the method comprising:

receiving a request from a client computer, the request identifying an application and identifying a form of the application, the identified form being one of a plurality of available forms; and

in response to receiving the request,

generating the identified form of the application from another form of the application;

after generating the identified form of the application,

storing the identified form of the application in a cache; and

10

sending the identified form of the application to the client computer.

22. The method of claim 21 including in response to receiving a flush request, removing the identified form of the application from the cache.

23. The method of claim 22 wherein the flush request is received from a computer other than the server computer.

24. The method of claim 21 including storing a time to live indicator with the stored identified form of the application.

25. A method in a server computer for providing applications to client computers, the method comprising:

receiving a request from a client computer, the request identifying an application and identifying a form of the application, the identified form being one of a plurality of available forms; and

in response to receiving the request,

transforming an intermediate form of the application into a transformed version of the intermediate form of the application;

generating the identified form of the application from the transformed version of the intermediate form of the application; and

sending the identified form of the application to the client computer.

26. A method in a server computer for providing applications to client computers, the method comprising:

receiving a request from a client computer, the request identifying an application and identifying a form of the application, the identified form being one of a plurality of available forms, and

in response to receiving the request,

verifying the application;

generating the identified form of the application from the verified application; and

sending the identified form of the application to the client computer.

27. The method of claim 26 wherein the verifying is specified by the client computer.

28. The method of claim 26 wherein the verifying is specified by another computer.

29. The method of claim 26 wherein the verifying is specified by the server computer.

30. The method of claim 26 including receiving a verifier from the client computer.

31. The method of claim 30 including sending to the client computer a handle for the verifier so that the client computer can subsequently identify the verifier to the server computer.

32. A method in a server computer for providing applications to client computers, the method comprising:

receiving a request from a client computer, the request identifying an application and identifying a form of the application, the identified form being one of a plurality of available forms;

transforming the application that is identified in the request using a common transformation;

in response to receiving the request,

generating the identified form of the application from the transformed application; and

sending the identified form of the application to the client computer.

33. A method in a server computer for providing applications to client computers, the method comprising:

receiving a request from a client computer, the request identifying an application and identifying a form of the application, the identified form being one of a plurality of available forms; and

US 6,324,685 B1

11

in response to receiving the request, generating the identified form of the application from another form of the application; and sending the identified form of the application to the client computer wherein the identified form indicates a processor of the client computer.

34. A method in a client computer for retrieving an application in an identified form, the method comprising: sending to a server computer a request that identifies an application and identifies a form of the application, the identified form being one of a plurality of available forms; and

in response to sending the request, receiving from the server computer the identified form of the application wherein the server computer generated the identified form of the application from another form of the application in response to receiving the request from the client computer

wherein after the server computer generated the identified form of the application, the server computer stored the identified form of the application in a cache so that when another request is received for the identified form of the application, the server computer can retrieve the identified form of the application without regenerating the identified form of the application.

35. A method in a client computer for retrieving an application in an identified form, the method comprising: sending to a server computer a request that identifies an application and identifies a form of the application, the identified form being one of a plurality of available forms; and

in response to sending the request, receiving from the server computer the identified form of the application wherein the server computer generated the identified form of the application from another form of the application in response to receiving the request from the client computer wherein when the server computer does not have the application in the other form, the server computer requests the other form of the application from a computer other than the server computer.

36. A method in a client computer for retrieving an application in an identified form, the method comprising: sending to a server computer a request that identifies an application and identifies a form of the application, the identified form being one of a plurality of available forms; and

in response to sending the request, receiving from the server computer the identified form of the application wherein the server computer generated the identified form of the application from another form of the application in response to receiving the request from the client computer; and

wherein the identified form is an intermediate form.

37. The method of claim 36 wherein the intermediate form is Java byte code.

38. The method of claim 36 wherein the intermediate form can be interpreted by an interpreter executing on the client computer.

39. A method in a client computer for retrieving an application in an identified form, the method comprising: sending to a server computer a request that identifies an application and identifies a form of the application, the identified form being one of a plurality of available forms; and

in response to sending the request, receiving from the server computer the identified form of the application

12

wherein the server computer generated the identified form of the application from another form of the application in response to receiving the request from the client computer

wherein the identified form is a target form.

40. The method of claim 34 wherein the target form is directly executable by a processor of the client computer.

41. A method in a client computer for retrieving an application in an identified form, the method comprising: sending to a server computer a request that identifies an application and identifies a form of the application, the identified form being one of a plurality of available forms; and

in response to sending the request, receiving from the server computer the identified form of the application wherein the server computer generated the identified form of the application in response to receiving the request from the client computer

wherein the other form is a source form.

42. The method of claim 41 wherein the source form is Java source.

43. A method in a client computer for retrieving an application in an identified form, the method comprising: sending to a server computer a request that identifies an application and identifies a form of the application, the identified form being one of a plurality of available forms; and

in response to sending the request, receiving from the server computer the identified form of the application wherein the server computer generated the identified form of the application from another form of the application in response to receiving the request from the client computer

wherein the server computer transforms the application before generating the identified form of the application.

44. The method of claim 43 wherein the transforming is verifying the application.

45. The method of claim 43 wherein the transforming is optimizing the application.

46. The method of claim 43 wherein the transforming is compressing the application.

47. A method in a client computer for retrieving an application in an identified form, the method comprising: sending to a server computer a request that identifies an application and identifies a form of the application, the identified form being one of a plurality of available forms; and

in response to sending the request, receiving from the server computer the identified form of the application wherein the server computer generated the identified form of the application from another form of the application in response to receiving the request from the client computer

wherein the server computer caches the identified form of the application and wherein the server computer subsequently retrieves the identified form of the application from the cache, rather than generating the identified form of the application.

48. A method in a client computer for retrieving an application in an identified form, the method comprising: sending to a server computer a request that identifies an application and identifies a form of the application, the identified form being one of a plurality of available forms; and

US 6,324,685 B1

13

in response to sending the request, receiving from the server computer the identified form of the application wherein the server computer generated the identified form of the application from another form of the application in response to receiving the request from the client computer

wherein the application is a portion of a larger application.

49. A method in a client computer for retrieving an application in an identified form, the method comprising:

sending to a server computer a request that identifies an application and identifies a form of the application, the identified form being one of a plurality of available forms; and

in response to sending the request, receiving from the server computer the identified form of the application wherein the server computer generated the identified form of the application from another form of the application in response to receiving the request from the client computer

wherein the application is an applet.

50. A method in a client computer for retrieving an application in an identified form, the method comprising:

sending to a server computer a request that identifies an application and identifies a form of the application, the identified form being one of a plurality of available forms; and

in response to sending the request, receiving from the server computer the identified form of the application wherein the server computer generated the identified form of the application from another form of the application in response to receiving the request from the client computer wherein the application includes modules, wherein the server computer generates modules of the identified form, and wherein the server computer combines modules of the identified form that were previously generated with modules of the identified form that are generated in response to receiving the request.

51. The method of claim 50 wherein the modules of the identified form that were previously generated are retrieved by the server computer from a cache.

52. A method in a client computer for retrieving an application in an identified form, the method comprising:

sending to a server computer a request that identifies an application and identifies a form of the application, the identified form being one of a plurality of available forms; and

in response to sending the request, receiving from the server computer the identified form of the application wherein the server computer generated the identified form of the application from another form of the application in response to receiving the request from the client computer

wherein the request includes an indication to verify the application.

53. The method of claim 52 including sending a verifier from the client computer to the server computer.

54. A method in a client computer for retrieving an application in an identified form, the method comprising:

sending to a server computer a request that identifies an application and identifies a form of the application, the identified form being one of a plurality of available forms; and

in response to sending the request, receiving from the server computer the identified form of the application

14

wherein the server computer generated the identified form of the application from another form of the application in response to receiving the request from the client computer

wherein the server computer transforms each application using a common transformation.

55. A computer-readable medium containing instructions for controlling a server computer to provide applets to client computers, by a method comprising:

receiving requests from client computers, each request identifying an applet and identifying a form of the applet, the identified form being one of a plurality of possible compiled forms of the applet; and

in response to receiving a request, using a compiler to compile the identified form of the applet from an un-compiled form of the applet and sending the identified form of the applet to the client computer that sent the request; and

caching the identified form of the applet so that when another request is received for the same applet in the identified form, the identified form of the applet can be sent without recompiling the identified form of the applet

whereby requests of different client computers identify different forms of the same applet.

56. A computer-readable medium containing instructions for controlling a server computer to provide applets to client computers, by a method comprising:

receiving requests from client computers, each request identifying an applet and identifying a form of the applet, the identified form being one of a plurality of possible compiled forms of the applet; and

in response to receiving the request, when the server computer does not have the applet in the un-compiled form, requesting the applet in the un-compiled form from a computer other than the server computer;

using a compiler to compile the identified form of the applet from the un-compiled form of the applet; and sending the identified form of the applet to the client computer that sent the request

whereby requests of different client computers identify different forms of the same applet.

57. The computer-readable medium of claim 56 wherein the other computer is accessible via the Internet.

58. The computer-readable medium of claim 56 wherein the server computer and client computer are connected to a local area network and the server computer and the other computer are connected via the Internet.

59. A computer-readable medium containing instructions for controlling a server computer to provide applets to client computers, by a method comprising:

receiving requests from client computers, each request identifying an applet and identifying a form of the applet, the identified form being one of a plurality of possible compiled forms of the applet; and

in response to receiving a request, using a compiler to compile the identified form of the applet from an un-compiled form of the applet and sending the identified form of the applet to the client computer that sent the request

wherein the applet is part of a web page and whereby requests of different client computers identify different forms of the same applet.

60. A computer-readable medium containing instructions for controlling a server computer to provide applets to client computers, by a method comprising:



US 6,324,685 B1

15

receiving requests from client computers, each request identifying an applet and identifying a form of the applet, the identified form being one of a plurality of possible compiled forms of the applet; and  
 in response to receiving a request, using a compiler to compile the identified form of the applet from an un-compiled form of the applet and sending the identified form of the applet to the client computer that sent the request  
 wherein the un-compiled form is an intermediate form and whereby requests of different client computers identify different forms of the same applet.

61. The computer-readable medium of claim 60 wherein the intermediate form is Java byte code.

62. A computer-readable medium containing instructions for controlling a server computer to provide applets to client computers, by a method comprising:  
 receiving requests from client computers, each request identifying an applet and identifying a form of the applet, the identified form being one of a plurality of possible compiled forms of the applet; and  
 in response to receiving a request, using a compiler to compile the identified form of the applet from an un-compiled form of the applet and sending the identified form of the applet to the client computer that sent the request  
 wherein the identified form is directly executable by a processor of the client computer that sent the request and whereby requests of different client computers identify different forms of the same applet.

63. A computer-readable medium containing instructions for controlling a server computer to provide applets to client computers, by a method comprising:  
 receiving requests from client computers, each request identifying an applet and identifying a form of the applet, the identified form being one of a plurality of possible compiled forms of the applet; and  
 in response to receiving a request, using a compiler to compile the identified form of the applet from an un-compiled form of the applet and sending the identified form of the applet to the client computer that sent the request  
 wherein the un-compiled form is a source form and whereby requests of different client computers identify different forms of the same applet.

64. The computer-readable medium of claim 63 wherein the source form is Java source.

65. A computer-readable medium containing instructions for controlling a server computer to provide applets to client computers, by a method comprising:  
 receiving requests from client computers, each request identifying an applet and identifying a form of the applet, the identified form being one of a plurality of possible compiled forms of the applet; and  
 in response to receiving a request, transforming an un-compiled form of the applet, compiling using a compiler the identified form of the applet from the transformed, un-compiled form of the applet, and sending the identified form of the applet to the client computer that sent the request  
 whereby requests of different client computers identify different forms of the same applet.

66. The computer-readable medium of claim 65 wherein the transforming is verifying the applet.

67. The computer-readable medium of claim 65 wherein the transforming is optimizing the applet.

16

68. The computer-readable medium of claim 65 wherein the transforming is compressing the applet.

69. A computer-readable medium containing instructions for controlling a server computer to provide applets to client computers, by a method comprising:  
 receiving requests from client computers, each request identifying an applet and identifying a form of the applet, the identified form being one of a plurality of possible compiled forms of the applet;  
 in response to receiving a request, using a compiler to compile the identified form of the applet from an un-compiled form of the applet and sending the identified form of the applet to the client computer that sent the request;  
 storing the identified form of the applet in a cache so that when another request is retrieved, the server computer retrieves the identified form of the applet from the cache and sends the identified form of the applet retrieved from the cache to the client computer  
 whereby requests of different client computers identify different forms of the same applet.

70. A computer-readable medium containing instructions for controlling a server computer to provide applets to client computers, by a method comprising:  
 receiving requests from client computers, each request identifying an applet and identifying a form of the applet, the identified form being one of a plurality of possible compiled forms of the applet, and  
 in response to receiving a request, using a compiler to compile the identified form of the applet from an un-compiled form of the applet and sending the identified form of the applet to the client computer that sent the request  
 wherein the applet is a portion of a computer program and whereby requests of different client computers identify different forms of the same applet.

71. A computer-readable medium containing instructions for controlling a server computer to provide applets to client computers, by a method comprising:  
 receiving requests from client computers, each request identifying an applet and identifying a form of the applet, the identified form being one of a plurality of possible compiled forms of the applet; and  
 in response to receiving a request, using a compiler to compile the identified form of the applet from an un-compiled form of the applet and sending the identified form of the applet to the client computer that sent the request  
 wherein the server computer functions as a loader for the client computer and whereby requests of different client computers identify different forms of the same applet.

72. A computer-readable medium containing instructions for controlling a server computer to provide applets to client computers, by a method comprising:  
 receiving requests from client computers, each request identifying an applet and identifying a form of the applet, the identified form being one of a plurality of possible compiled forms of the applet;  
 in response to receiving a request, using a compiler to compile the identified form of the applet from an un-compiled form of the applet and sending the identified form of the applet to the client computer that sent the request  
 wherein the applet includes multiple modules, wherein the server computer generates a module of the identi-

US 6,324,685 B1

17

filed form, and wherein the server computer compiles one of the modules into the identified form in response to receiving the request and whereby requests of different client computers identify different forms of the same applet.

73. A computer-readable medium containing instructions for controlling a server computer to provide applets to client computers, by a method comprising:

receiving requests from client computers, each request identifying an applet and identifying a form of the applet, the identified form being one of a plurality of possible compiled forms of the applet;

in response to receiving, using a compiler to compile the identified form of the applet from an un-compiled form of the applet and sending the identified form of the applet to the client computer that sent the request; and after generating the identified form of the applet, storing the identified form of the applet in a cache

whereby requests of different client computers identify different forms of the same applet.

74. The computer-readable medium of claim 73 including in response to receiving a flush request, removing the identified form of the applet from the cache.

75. The computer-readable medium of claim 74 wherein the flush request is received from a computer other than the server computer.

76. The computer-readable medium of claim 73 including a storing time to live indicator with the stored identified form of the applet.

77. A computer-readable medium containing instructions for controlling a server computer to provide applets to client computers, by a method comprising:

receiving requests from client computers, each request identifying an applet and identifying a form of the applet, the identified form being one of a plurality of possible compiled forms of the applet;

in response to receiving a request, transforming an intermediate form of the applet into a transformed version of the intermediate form of the applet and compiling using a compiler the identified form of the applet from the transformed intermediate form of the applet and sending the identified form of the applet to the client computer that sent the request; and

whereby requests of different client computers identify different forms of the same applet.

78. A computer-readable medium containing instructions for controlling a server computer to provide applets to client computers, by a method comprising:

receiving requests from client computers, each request identifying an applet and identifying a form of the applet, the identified form being one of a plurality of possible compiled forms of the applet, a request including a verifier sent from the client computer for use in verifying the applet; and

in response to receiving a request, using a compiler to compile the identified form of the applet from an un-compiled form of the applet and sending the identified form of the applet to the client computer that sent the request

whereby requests of different client computers identify different forms of the same applet.

79. A computer-readable medium containing instructions for controlling a server computer to provide applets to client computers, by a method comprising:

receiving requests from client computers, each request identifying an applet and identifying a form of the

18

applet, the identified form being one of a plurality of possible compiled forms of the applet; and

in response to receiving a request, transforming each applet that is identified in a request using a common transformation, compiling using a compiler the identified form of the applet from the transformed form of the applet, and sending the identified form of the applet to the client computer that sent the request

whereby requests of different client computers identify different forms of the same applet.

80. A computer-readable medium containing instructions for controlling a server computer to provide applets to client computers, by a method comprising:

receiving requests from client computers, each request identifying an applet and identifying a form of the applet, the identified form being one of a plurality of possible compiled forms of the applet; and

in response to receiving a request, using a compiler to compile the identified form of the applet from an un-compiled form of the applet and sending the identified form of the applet to the client computer that sent the request;

wherein the identified form indicates a processor of the client computer

whereby requests of different client computers identify different forms of the same applet.

81. A server computer for providing applets in a plurality of forms, comprising:

means for receiving from client computers requests for applets, each request identifying a form of the applet; means for retrieving the applet identified in a request, the retrieved applet being in a form other than the form identified in the request;

means for generating the identified form of the applet from the retrieved other form of the applet after receiving the request;

means for sending the identified form of the applet to the client computer that requested the applet; and

means for caching the identified form of the applet so that when another request is received for the same applet in the identified form, the identified form of the applet can be sent without regenerating the identified form of the applet.

82. A server computer for providing applets in a plurality of forms, comprising:

means for receiving from client computers requests for applets, each request identifying a form of the applet;

means for retrieving the applet identified in a request, the retrieved applet being in a form other than the form identified in the request;

means for generating the identified form of the applet from the retrieved other form of the applet after receiving the request;

means for sending the identified form of the applet to the client computer that requested the applet; and

means for when the server computer does not have the applet in the other form, requesting the other form of the applet from a computer other than the server computer.

83. The server computer of claim 82 wherein the other computer is accessible via the Internet.

84. The server computer of claim 82 wherein the server computer and client computer are connected to a local area network and the server and the other computer are connected via the Internet.

US 6,324,685 B1

19

85. A server computer for providing applets in a plurality of forms, comprising:

means for receiving from client computers requests for applets, each request identifying a form of the applet;

means for retrieving the applet identified in a request, the retrieved applet being in a form other than the form identified in the request;

means for generating the identified form of the applet from the retrieved other form of the applet after receiving the request; and

means for sending the identified form of the applet to the client computer that requested the applet

wherein the applet is part of a web page.

86. A server computer for providing applets in a plurality of forms, comprising:

means for receiving from client computers requests for applets, each request identifying a form of the applet;

means for retrieving the applet identified in a request, the retrieved applet being in a form other than the form identified in the request;

means for generating the identified form of the applet from the retrieved other form of the applet after receiving the request;

means for sending the identified form of the applet to the client computer that requested the applet; and

means for transforming the applet before generating the identified form of the applet.

87. The server computer of claim 86 wherein the transforming is verifying the applet.

88. A server computer for providing applets in a plurality of forms, comprising:

means for receiving from client computers requests for applets, each request identifying a form of the applet;

means for retrieving the applet identified in a request, the retrieved applet being in a form other than the form identified in the request;

means for generating the identified form of the applet from the retrieved other form of the applet after receiving the request;

means for sending the identified form of the applet to the client computer that requested the applet; and

means for, rather than generating the identified form of the applet, retrieving the identified form of the applet from a cache.

89. A computer-readable medium containing a data structure, the data structure including a request generated by a client computer, the request identifying an applet and identifying a form of the identified applet, the identified form being one of a plurality of compiled forms of the identified applet wherein when a server computer receives the request, the server computer uses a compiler to compile the identified form of the applet from an un-compiled form of the applet and then sends the identified form of the applet to the client computer that generated the request wherein the server computer caches the identified form of the applet so that when another request is received for the same applet in the identified form, the identified form of the applet can be sent without recompiling the identified form of the applet.

90. A computer-readable medium containing a data structure, the data structure including a request generated by a client computer, the request identifying an applet and identifying a form of the identified applet, the identified form being one of a plurality of compiled forms of the identified applet wherein when a server computer receives

20

the request, the server computer uses a compiler to compile the identified form of the applet from an un-compiled form of the applet and then sends the identified form of the applet to the client computer that generated the request wherein in response to receiving the request, the server computer requests the applet in the un-compiled form from a computer other than the server computer.

91. A computer-readable medium containing a data structure, the data structure including a request generated by a client computer, the request identifying an applet and identifying a form of the identified applet, the identified form being one of a plurality of compiled forms of the identified applet wherein when a server computer receives the request, the server computer uses a compiler to compile the identified form of the applet from an un-compiled form of the applet and then sends the identified form of the applet to the client computer that generated the request wherein the applet is part of a web page.

92. A computer-readable medium containing a data structure, the data structure including a request generated by a client computer, the request identifying an applet and identifying a form of the identified applet, the identified form being one of a plurality of compiled forms of the identified applet wherein when a server computer receives the request, the server computer uses a compiler to compile the identified form of the applet from an un-compiled form of the applet and then sends the identified form of the applet to the client computer that generated the request wherein the un-compiled form is an intermediate form.

93. The computer-readable medium of claim 92 wherein the intermediate form is Java byte code.

94. A computer-readable medium containing a data structure, the data structure including a request generated by a client computer, the request identifying an applet and identifying a form of the identified applet, the identified form being one of a plurality of compiled forms of the identified applet wherein when a server computer receives the request, the server computer uses a compiler to compile the identified form of the applet from an un-compiled form of the applet and then sends the identified form of the applet to the client computer that generated the request wherein the un-compiled form is a source form.

95. The computer-readable medium of claim 94 wherein the source form is Java source.

96. A computer-readable medium containing a data structure, the data structure including a request generated by a client computer, the request identifying an applet and identifying a form of the identified applet, the identified form being one of a plurality of compiled forms of the identified applet wherein when a server computer receives the request, the server computer uses a compiler to compile the identified form of the applet from an un-compiled form of the applet and then sends the identified form of the applet to the client computer that generated the request wherein the server computer transforms the applet before compiling the applet.

97. The computer-readable medium of claim 96 wherein the transforming is verifying the applet.

98. The computer-readable medium of claim 96 wherein the transforming is optimizing the applet.

99. A computer-readable medium containing instructions for controlling a client computer, the instructions being generated by a server computer in response to receiving a request from a client computer, the request identifying an applet and identifying a form of the applet, wherein the server computer generates the instructions by compiling a compiled form of the applet from an un-compiled form of

the applet wherein the server computer retrieves the un-compiled form of the applet from a computer other than the server computer.

100. A computer-readable medium containing instructions for controlling a client computer, the instructions being generated by a server computer in response to receiving a request from a client computer, the request identifying an applet and identifying a form of the applet, wherein the server computer generates the instructions by compiling a compiled form of the applet from an un-compiled form of the applet wherein the server computer stores the compiled form of the applet in a cache.

101. A computer-readable medium containing instructions for controlling a client computer, the instructions being generated by a server computer in response to receiving a request from a client computer, the request identifying an applet and identifying a form of the applet, wherein the server computer generates the instructions by compiling a compiled form of the applet from an un-compiled form of the applet wherein the un-compiled form of the applet is Java intermediate code.

102. A computer-readable medium containing instructions for controlling a client computer, the instructions being generated by a server computer in response to receiving a request from a client computer, the request identifying an applet and identifying a form of the applet, wherein the server computer generates the instructions by compiling a compiled form of the applet from an un-compiled form of the applet wherein the un-compiled form of the applet is Java source code.

103. A computer-readable medium containing instructions for controlling a client computer, the instructions being generated by a server computer in response to receiving a request from a client computer, the request identifying an applet and identifying a form of the applet, wherein the server computer generates the instructions by compiling a compiled form of the applet from an un-compiled form of the applet wherein the instructions are part of a web page.

104. A server computer comprising:  
local resources that include compilers and modules;  
a cache for storing applets;  
a transformer; and  
an applet server manager that  
receives requests from client computers for an applet in an identified form,  
when the identified form of the applet is stored in the cache, sends the identified form of the applet stored in the cache to the client computer that sent the request, and  
when the identified form of the applet is not stored in the cache, generates the identified form of the applet

using a compiler and a module, stores the identified form of the applet in the cache, and sends the identified form of the applet to the client computer that sent the request

wherein the applet server manager uses the transformer to transform the module before generating the identified form of the applet.

105. A server computer comprising:  
local resources that include compilers and modules;  
a cache for storing applets  
a transformer; and  
an applet server manager that  
receives requests from client computers for an applet in an identified form,  
when the identified form of the applet is stored in the cache, sends the identified form of the applet stored in the cache to the client computer that sent the request,  
when the identified form of the applet is not stored in the cache, generates the identified form of the applet using a compiler and a module, stores the identified form of the applet in the cache, and sends the identified form of the applet to the client computer that sent the request

wherein the applet server manager uses the transformer to transform the identified applet after it is generated using the compiler.

106. A server computer comprising:  
local resources that include compilers and modules;  
a cache for storing applets; and  
an applet server manager that  
receives requests from client computers for an applet in an identified form,  
when the identified form of the applet is stored in the cache, sends the identified form of the applet stored in the cache to the client computer that sent the request, and  
when the identified form of the applet is not stored in the cache, generates the identified form of the applet using a compiler and a module, stores the identified form of the applet in the cache, and sends the identified form of the applet to the client computer that sent the request

wherein the applet server manager retrieves a module from a computer other than the server computer when a module that is not another form of the applet is not stored with the local resource.

\* \* \* \* \*



UNITED STATES PATENT AND TRADEMARK OFFICE  
**CERTIFICATE OF CORRECTION**

PATENT NO. : 6,324,685 B1  
DATED : November 27, 2001  
INVENTOR(S) : Edward Balassanian

Page 1 of 1

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 3,

Line 64, after "A 12." begin a new paragraph;

Column 5,

Line 31, delete "10" and insert -- IO --;

Column 6,

Line 10, the paragraph beginning with "For" should be part of the preceding paragraph;

Column 12,

Line 6, delete "34" and insert -- 39 --;

Column 17,

Line 13, after "receiving" insert -- a request --; and

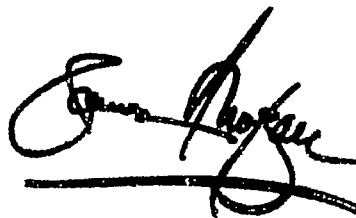
Column 19,

Line 38, delete the comma after "request" and insert a semicolon.

Signed and Sealed this

Twenty-third Day of July, 2002

*Attest:*



*Attesting Officer*

JAMES E. ROGAN  
*Director of the United States Patent and Trademark Office*

# **EXHIBIT B**



US006976248B2

(12) **United States Patent**  
Balassanian

(10) **Patent No.:** US 6,976,248 B2  
(45) **Date of Patent:** Dec. 13, 2005

(54) **APPLICATION SERVER FACILITATING WITH CLIENT'S COMPUTER FOR APPLETS ALONG WITH VARIOUS FORMATS**

(75) Inventor: Edward Balassanian, Kirkland, WA (US)

(73) Assignee: Implicit Networks, Inc., Bellevue, WA (US)

(\* ) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 478 days.

(21) Appl. No.: 09/968,704

(22) Filed: Oct. 1, 2001

(65) **Prior Publication Data**

US 2002/0100038 A1 Jul. 25, 2002

**Related U.S. Application Data**

(63) Continuation of application No. 09/040,972, filed on Mar. 18, 1998, now Pat. No. 6,324,685.

(51) **Int. Cl.**<sup>7</sup> ..... G06F 9/45

(52) **U.S. Cl.** ..... 717/148; 717/140; 709/203

(58) **Field of Search** ..... 717/116, 118, 717/136, 139, 140-142, 148, 151, 152, 165, 162, 166; 709/203, 223

(56) **References Cited**

**U.S. PATENT DOCUMENTS**

|                |         |                 |         |
|----------------|---------|-----------------|---------|
| 5,706,502 A *  | 1/1998  | Foley et al.    | 707/10  |
| 5,761,421 A *  | 6/1998  | van Hoff et al. | 709/223 |
| 5,805,829 A *  | 9/1998  | Cohen et al.    | 709/202 |
| 5,828,840 A *  | 10/1998 | Cowan et al.    | 709/203 |
| 5,848,274 A    | 12/1998 | Hamby et al.    |         |
| 5,872,915 A    | 2/1999  | Dykes et al.    |         |
| 5,884,078 A    | 3/1999  | Faustini        |         |
| 6,230,184 B1 * | 5/2001  | White et al.    | 709/201 |
| 6,282,702 B1 * | 8/2001  | Ungar           | 717/148 |
| 6,295,643 B1 * | 9/2001  | Brown et al.    | 717/148 |
| 6,321,377 B1 * | 11/2001 | Beadle et al.   | 717/148 |

|                |         |               |           |
|----------------|---------|---------------|-----------|
| 6,324,685 B1 * | 11/2001 | Balassanian   | 717/118   |
| 6,336,213 B1 * | 1/2002  | Beadle et al. | 717/136   |
| 6,446,081 B1 * | 9/2002  | Preston       | 707/104.1 |
| 6,594,820 B1 * | 7/2003  | Ungar         | 717/124   |
| 6,636,900 B2 * | 10/2003 | Abdelnur      | 719/316   |
| 6,704,926 B1 * | 3/2004  | Blandy et al. | 717/148   |
| 6,742,165 B2 * | 5/2004  | Lev et al.    | 716/1     |
| 6,745,386 B1 * | 6/2004  | Yellin        | 717/166   |
| 6,836,889 B1 * | 12/2004 | Chan et al.   | 719/310   |
| 6,842,897 B1 * | 1/2005  | Beadle et al. | 718/1     |

**OTHER PUBLICATIONS**

Yang et al, "Developing integrated web and database applications using JAVA applets and JDBC drivers", ACM SIG-SCIENCE, pp 302-306, 1998.\*

Newsome et al, "Proxy compilation of dynamically loaded Java classes with MoJo", ACM LCTES, pp 204-212, Jun. 2002.\*

Begole et al, "Transparent sharing of Java applets: a replicated approach", ACM UIST, pp 55-64, 1997.\*

(Continued)

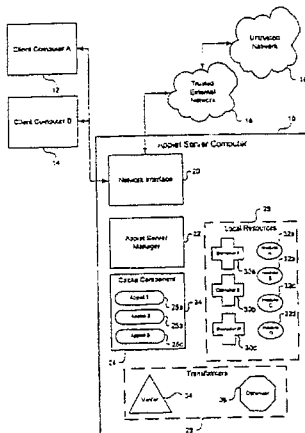
*Primary Examiner*—Anil Khatri

(74) *Attorney, Agent, or Firm*—Morgan & Finnegan, LLP

(57) **ABSTRACT**

The present invention is an applet server which accepts requests for applets from client computers. A request specifies the format in which an applet is to be delivered to the requesting client computer. The applet server has a cache which it uses to store applets for distribution to client computers. If the specified form of the requested applet is available in the cache, the applet server transmits the applet to the requesting client. If the applet is not available in the cache, the server will attempt to build the applet from local resources (program code modules and compilers) and transformer programs (verifiers and optimizers). If the applet server is able to build the requested applet, it will then transmit the applet to the requesting client computer. If the applet server is unable to build the requested applet, it will pass the request to another applet server on the network for fulfillment of the request.

13 Claims, 3 Drawing Sheets



US 6,976,248 B2

Page 2

---

OTHER PUBLICATIONS

Benton et al, "Compiling standard ML to ava bytecode", ACM ICFP, pp 129-140.\*

Sirer, Emin Gün, "Java-Relevant Articles in the Press," [Accessed Oct. 4, 2000].

"Project Members" <http://kimera.cs.washington.edu/members.html> [Accessed Oct. 4, 2000].

Emin Gün Sirer, et al., "Distributed Virtual Machines: A System Architecture for Network Computing," Dept. of Computer Science & Engineering, University of Washington, Seattle, Washington <http://kimera.cs.washington.edu> Feb. 26, 1998.

Emin Gün Sirer, et al., "Design and Implementation of a Distributed Virtual Machine for Networked Computers," University of Washington, Department of Computer Science and Engineering, Seattle, Washington, 17<sup>th</sup> ACM Symposium on Operating system Principles, Dec. 1999.

Sirer, Emin Gün, "A System Architecture for Next Generation Network Computing," Dept. of Computer Science & Engineering, University of Washington, Seattle, Washington Jun. 26, 1998.

[Accessed Oct. 4, 2000].

Sirer, Emin Gün, "Kimera—A System Architecture for Networked Computers," [Accessed Oct. 4, 2000].

Emin Gün Sirer and Brian Bershad, "Kimera Architecture," [Accessed Oct. 4, 2000].

Sirer, Emin Gün, "Security Flaws in Java Implementations," [Accessed Oct. 4, 2000].

Sirer, Emin Gün, "Kimera Bytecode Verification," [Accessed Oct. 4, 2000].

Sirer, Emin Gün, "Kimera Test Suite," [Accessed Oct. 4, 2000].

Sirer, Emin, Gün, "Kimera Disassembler," [Accessed Oct. 4, 2000].

\* cited by examiner

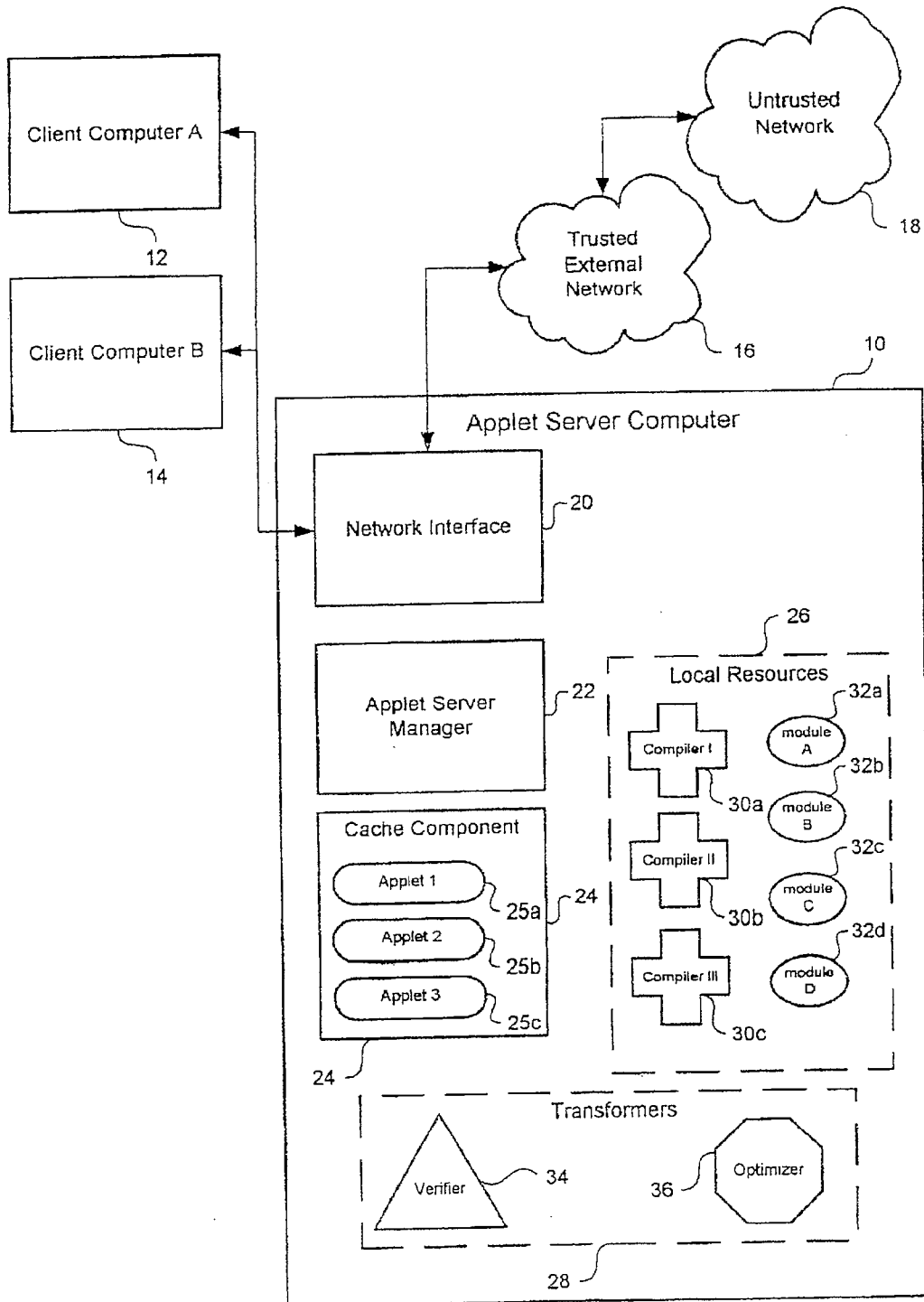


Fig. 1

**Request Data Type**

| Tag                | Value  |
|--------------------|--|
| Applet-URL         | (String) specifies the name of the requested applet  |
| Code-Type          | (Source/Intermediate/Binary) specifies the format the applet is to be delivered to the requesting client in. A request for binary would specify the CPU of the requesting client (e.g., x86) |
| Verification-Level | (0-100) specifies the degree of verification to be performed. 0 = no/minimal verification, 100 = maximum verification (highest level of security).   |
| Optimization-Level | (0-100) specifies the degree of optimization to be performed. 0 = no/minimal optimization, 100 = maximum optimization.   |

*Fig. 2A*

**Code Data Type**

| Tag                | Value  |
|--------------------|--|
| Applet-URL         | (String) specifies the name of the requested applet  |
| Code-Type          | (Source/Intermediate/Binary) specifies the format the applet is to be delivered to the requesting client in. A request for binary would specify the CPU of the requesting client (e.g., x86) |
| Verification-Level | (0-100) specifies the degree of verification to be performed. 0 = no/minimal verification, 100 = maximum verification (highest level of security).   |
| Optimization-Level | (0-100) specifies the degree of optimization to be performed. 0 = no/minimal optimization, 100 = maximum optimization.   |
| Applet Length      | (0-2 <sup>32</sup> ) specifies the size of the requested applet.   |
| Applet Code        | The Requested Applet in the form specified by the request data type.   |

*Fig. 2B*

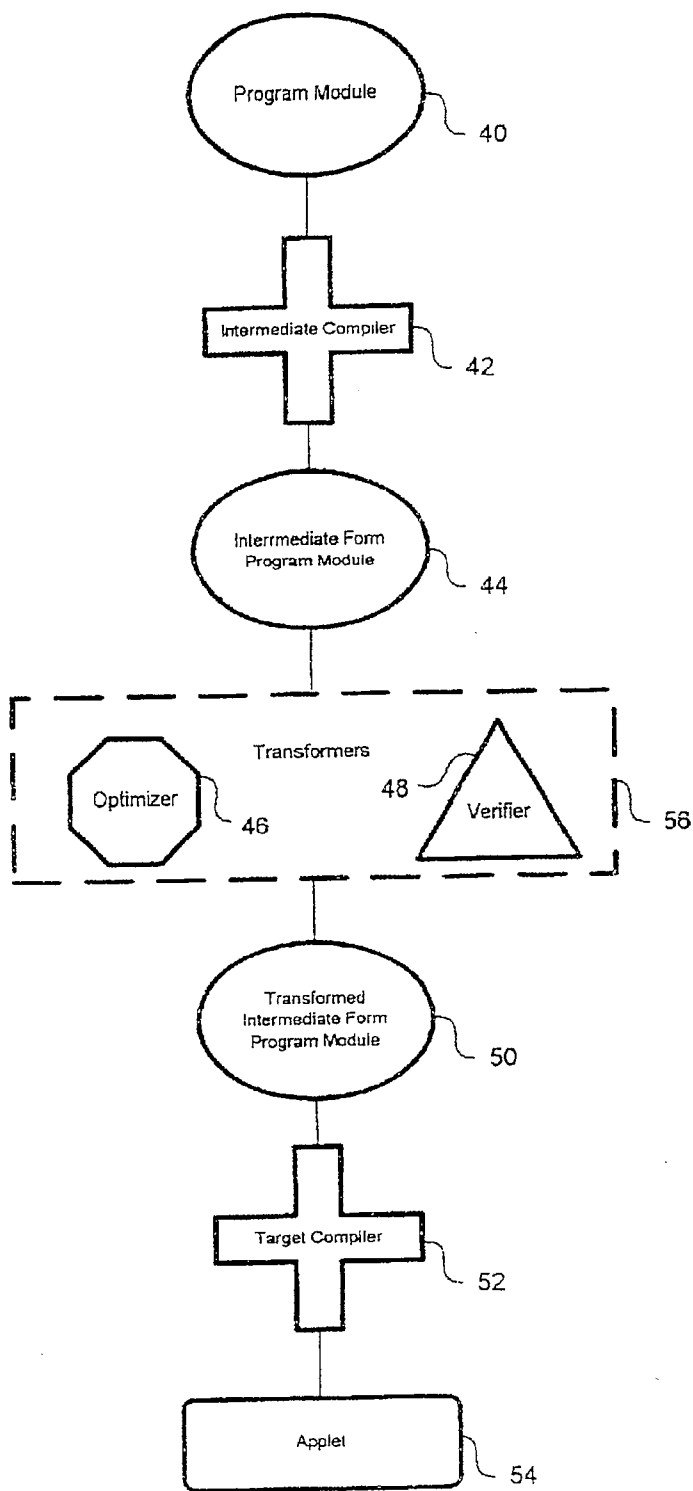


Fig. 3

US 6,976,248 B2

1

## APPLICATION SERVER FACILITATING WITH CLIENT'S COMPUTER FOR APPLETS ALONG WITH VARIOUS FORMATS

### CROSS-REFERENCE TO RELATED APPLICATION(S)

This application is a continuation of U.S. patent application Ser. No. 09/040,972, filed Mar. 18, 1998, now U.S. Pat. No. 6,324,685.

### FIELD OF THE INVENTION

The present invention relates to computer operating systems and, in particular, to a server architecture providing application caching and security verification.

### BACKGROUND OF THE INVENTION

The growth of the Internet's importance to business, along with the increased dependence upon corporate networks, has created a demand for more secure and efficient computer systems. The traditional solution to this problem has been to depend upon improvements in hardware performance to make up for the performance penalty that is typically incurred when a computer system is made more secure and stable. Increased interconnectivity has also created a need for improved interoperability amongst a variety of computers that are now connected to one another. One solution to the problem of the variety of computers interconnected via the Internet and corporate networks has been the development of portable architecture neutral programming languages. The most widely known of these is Java, though, there are numerous other architecture neutral languages.

Architecture neutral programming languages allow programs downloaded from a server computer to a client computer to be interpreted and executed locally. This is possible because the compiler generates partially compiled intermediate byte-code, rather than fully compiled native machine code. In order to run a program, the client machine uses an interpreter to execute the compiled byte-code. The byte-codes provide an architecture neutral object file format, which allows the code to be transported to multiple platforms. This allows the program to be run on any system which implements the appropriate interpreter and run-time system. Collectively, the interpreter and runtime system implement a virtual machine. This structure results in a very secure language.

The security of this system is premised on the ability of the byte-code to be verified independently by the client computer. Using Java or some other virtual machine implementing technology, a client can ensure that the downloaded program will not crash the user's computer or perform operations for which it does not have permission.

The traditional implementations of architecture neutral languages are not without problems. While providing tremendous cross platform support, the current implementations of architecture neutral languages require that every client performs its own verification and interpretation of the intermediate code. The high computation and memory requirements of a verifier, compiler and interpreter restrict the applicability of these technologies to powerful client computers.

Another problem with performing the verification process on the client computer is that any individual within an organization may disable some or all of the checks performed on downloaded code. The current structure of these

2

systems makes security management at the enterprise level almost impossible. Since upgrades of security checking software must be made on every client computer, the cost and time involved in doing such upgrades makes it likely that outdated or corrupt copies of the verifier or interpreter exist within an organization. Even when an organization is diligent in maintaining a client based security model, the size of the undertaking in a large organization increases the likelihood that there will be problems.

There is a need for a scalable distributed system architecture that provides a mechanism for client computers to request and execute applets in a safe manner without requiring the client machines to have local resources to compile or verify the code. There is a further need for a system in which the applets may be cached in either an intermediate architecture neutral form or machine specific form in order to increase overall system performance and efficiency.

### SUMMARY OF THE INVENTION

In accordance with one embodiment of the invention, an applet server architecture is taught which allows client computers to request and execute applets in a safe manner without requiring the client to have local resources to verify or compile the applet code. Compilation and byte-code verification in the present invention are server based and thereby provide more efficient use of resources and a flexible mechanism for instituting enterprise-wide security policies. The server architecture also provides a cache for applets, allowing clients to receive applet code without having to access nodes outside the local network. The cache also provides a mechanism for avoiding repeated verification and compilation of previously requested applet code since any client requesting a given applet will have the request satisfied by a single cache entry.

Machine specific binary code is essentially interpreted code since the processor for a given computer can essentially be viewed as a form of an interpreter, interpreting binary code into the associated electronic equivalents. The present invention adds a level of indirection in the form of an intermediate language that is processor independent. The intermediate language serves as the basis for security verification, code optimizations, or any other compile time modifications that might be necessary. The intermediate form allows a single version of the source to be stored for many target platforms instead of having a different binary for each potential target computer. Compilations to the target form can either be done at the time of a cache hit or they can be avoided all together if the target machine is able to directly interpret the intermediate form. If the compilation is done on the server, then a copy of the of the compiled code as well as the intermediate form can be stored in the cache. The performance advantage derived from caching the compiled form as well as the intermediate depends upon the number of clients with the same CPU.

The novel features believed characteristic of the invention are set forth in the appended claims. The invention itself, however, as well as other features and advantages thereof will best be understood by reference to the detailed description which follows, when read in conjunction with the accompanying drawings.

### BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is diagram showing the major components which may be used to implement an applet server in one embodiment of the present invention;

FIG. 2a is a table which illustrates the structure of the request format data type;



US 6,976,248 B2

3

FIG. 2b is a table which illustrates the structure of the returned code data type.

FIG. 3 is a diagram showing the compilation and transformation of a program module into an applet in a particular form.

#### DETAILED DESCRIPTION OF THE INVENTION

Referring to FIG. 1, an applet server architecture according to one embodiment of the invention is based on an applet server computer 10 which in turn is connected to client computer A 12, client computer B 14, an external network 16 and an untrusted network 18. The applet server computer 10 connects to client computers 12 and 14, an external network 16, and an untrusted network 18 by means of a network interface 20. Typically this connection will involve one or more of the computers or networks having a connection to the Internet.

The applet server computer 10 accomplishes its objectives by manipulating computer programs in several formats. An applet (e.g. applets 1-3, 25a-25c) is any form of program instructions, whether in binary, source or intermediate format. In the case of this architecture, the applet code can either be a self contained program, or it can be a code fragment associated with a larger application.

Binary format refers to processor specific machine instructions suitable for running natively on a given computing platform (also referred to as "target" because of the concept of "targeting" a compiler to produce binary code for a given processor type).

Source refers to non-binary applet code, generally in the form of higher level languages (i.e. C, C++, Java, Visual Basic, ActiveX, Fortran, and Modula).

Intermediate format refers to a common intermediate byte-code that is produced by compiling a given source code input. The intermediate byte-code need not necessarily be Java byte-code.

Treating applets in this general sense allows client computers 12 and 14 to request not only applications, but portions of applications. Client computers 12 and 14 are thus able to use applet server computer 10 as the equivalent of a loader, loading in appropriate parts of the application in the form of applets. In turn client computers 12 and 14 can run large applications without requiring that the client computers 12 and 14 have the resources to store the entire application in memory at once.

Having the applets delivered from applet server computer 10 allows code in intermediate form to be verified, optimized, and compiled before being transmitted to client computers 12 and 14. This reduces the amount of work the client computers 12 and 14 have to do and provides a convenient way to impose global restrictions on code.

In operation, client computer A 12 transmits a request to an applet server computer 10 requesting an applet in a particular form. The form may be selected from a large matrix of many possible forms that can be recognized by the system. The request specifies the format (source, intermediate, or binary) in which the client wishes to receive the applet. The request may also specify that the applet be verified or have some other transformation operation performed upon it. Verification, optimization and compression are examples of types of transformation operations. The request is received by the network interface 20 of the applet server computer 10 which passes the request onto the applet server manager 22.

4

After interpreting the request, the applet server manager 22 checks to see if the requested applet is available in the cache 24. The cache 24 stores applets in a variety of formats (source, intermediate, or binary). If the requested form of the applet is available in the cache 24 (applet 1 25a, applet 2 25b, or applet 3 25c in this example) the applet server manager 22 instructs the network interface 20 to transmit the applet to requesting client computer A 12.

If the requested applet is not available in the cache 24, then the applet server manager 22 will attempt to build the requested applet from local resources 26 and one or more transformation operations performed by one or more of the transformers 28. Local resources 26 are comprised of compilers 30a, 30b and 30c and program code modules 32a, 32b, 32c and 32d. The requested applet is built by selecting one or more program code modules 32 and compiling them with one or more compilers 30. Transformer operations may be performed by the verifier 34 or the optimizer 36. After the applet server manager 22 builds the applet, the network interface 20 transmits the applet to the requesting client computer A 12.

If the request can not be satisfied by building the applet from local resources 26 and transformers 28, the applet server manager 22 will pass a request for the requested applet to external network 16 and/or untrusted network 18. The applet server manager 22 may request the applet in intermediate form or in executable form or it may request the local resources 26 and transformers 28 it needs to complete building the applet itself.

The cache 24 is capable of responding to the following commands: GET, PUT, and FLUSH. GET is used to retrieve a given applet from the cache. PUT is used to store an applet in the cache. FLUSH is used to clear the cache of one or more entries. When the cache is unable to locate an item in response to a GET operation, it returns a cache miss. The program which issued the GET command is then responsible for locating the desired form of the applet by other means and optionally storing it in the cache when it is retrieved (using the PUT operation). The FLUSH command will clear the cache of one or more entries and any subsequent GETs for the FLUSHed applet code will result in a cache miss. This is useful if a particular applet needs to be updated from a remote server on a periodic basis. When using PUT, the program issuing the command specifies a time to live (TTL) in the cache. When the TTL expires, the cache entry is removed by means of a FLUSH operation.

Local resources 26 are comprised of program modules 32 (applets in source form, not the requested form) and compilers 30. The program modules 32 are run through the compilers 30 in order to produce applets in the requested form. The applet server manager 20 may also direct the modules 32 to be processed by a verifier 34 or another transformer such as an optimizer 36. Program modules 32 are program code used to build applets. Program modules 32 may be stored in local resources 26 in source, binary, or intermediate formats. When an applet is built it may require the operation of one or more compilers 30 upon one or more program modules 32. The program modules 32 may be combined and recompiled with previously cached applets and the resulting applet may be also cached for use at a future time. Additionally, program modules 32, compilers 30 and transformers 28 (including verifiers 34 and optimizers 36) may be distributed across a network. The applet server manager 22 may pass requests for the components it needs to build a particular applet back to the network interface 20 which in turn passes the request onto the rest of the network and may include external network 16 and untrusted network 18.

US 6,976,248 B2

5

FIG. 3 provides further illustration of how an applet is produced from local resources and transformers. In this illustration the request is for an optimized and verified applet compiled to a machine specific form. A program module 40 is compiled into an intermediate form program module 44 by an intermediate compiler 42. The intermediate form program module 44 is then transformed by an optimizer 46 or a verifier 48. The resulting transformed intermediate form program module 50 is then compiled by target compiler 52 into machine specific code applet 54.

There are two types of compilers used to build applets: intermediate compilers 42 and target compilers 52. The intermediate compiler 42 compiles program modules (source applet code) 40 and produces a common intermediate pseudo-binary representation of the source applet code (intermediate form program module 44). The word pseudo is used because the intermediate form 44 is not processor specific but is still a binary representation of the source program module 40. This intermediate form can be re-targeted and compiled for a particular processor. Alternatively, the intermediate form 44 can be interpreted by an interpreter or virtual machine that understands the internal binary representation of the intermediate form. A target compiler 52 compiles intermediate applet code 44 into an applet 54 in a processor specific format (binary) suitable for running natively on a given computing platform.

Transformers 56 are programs that take in intermediate code and put out intermediate code. Transformers 56 are generally used for things like verification and optimization. Other transformers might include compressors that identify portions of code that can be replaced with smaller equivalents. Transformers can be matched up to any other component that takes in intermediate code as an input. These include the cache 24 and the target compilers 52. Global policies for transformers 56 can be implemented which ensure that all applets are run through some set of transformers before being returned to the client.

A verifier 48 is a type of transformer that is able to analyze input code and determine areas that might not be safe. The verifier 48 can determine the level of safety. Some verifiers 48 look for areas where unsafe or protected memory is being accessed, others might look for accesses to system resources such as IO devices. Once a verifier 48 determines the portion of unsafe applet code several steps can be taken. The offending code portion can be encased with new code that specifically prevents this unsafe code section from being executed. The unsafe code can be modified to be safe. The unsafe code can be flagged in such a way that a user can be warned about the possible risks of executing the code fragment. The verifier's role can therefore be summarized as determining where unsafe code exists and possibly altering the offending code to render it harmless. Verifiers 48 can operate on any format of input code, whether in source, intermediate or binary form. However, since intermediate code is a common format, it is most efficient to have a single verifier that will operate on code in this format. This eliminates the need to build specific knowledge of various source languages into the verifier. Verifiers 48 are a form of a transformer. Verifiers 48 take in intermediate code and put out verified intermediate code. Verifiers 48 are responsible for identifying non-secure portions of code in the intermediate code and modifying this code to make it secure. Security problems generally include access to memory areas that are unsafe (such as system memory, or memory outside the application space of the applet).

The choice of adding in the verification step can be left up to the client computer 12, the applet server computer 10 (see

6

FIG. 1), or can be based on the network that the applet originated from. Server managers can institute global policies that affect all clients by forcing all applets to be run through the verifier 48. Alternatively, verification can be reserved for un-trusted networks (18 in FIG. 1), or it can be left up to the client to determine whether the verification should be performed. In the preferred embodiment, verification level is determined by the applet server 10. In this way, a uniform security policy may be implemented from a single machine (i.e., the applet server 10).

Optimizers 46 are another type of transformer program. Optimizers 46 analyze code, making improvements to well known code fragments by substituting in optimized but equivalent code fragments. Optimizers 46 take in intermediate code 44 and put out transformed intermediate code 50. The transformed intermediate code 50 is functionally equivalent to the source intermediate code 44 in that they share the same structure.

Referring again to FIG. 1, policies may be instituted on the applet server 10 that force a certain set of request parameters regardless of what the client asked for. For example, the applet server manager 22 can run the applet through a verifier 34 or optimizer 36 regardless of whether the client 12 requested this or not. Since the server 10 might have to go to an untrusted network 18 to retrieve a given applet, it will then run this applet through the required transformers 28, particularly the verifier 34 before returning it to the client 12. Since clients 12 and 14 have to go through the applet server computer 10, this ensures that clients 12 and 14 do not receive applets directly from an untrusted network 18. In addition, since the server will be dealing directly with untrusted network 18, it can be set up to institute policies based on the network. A trusted external network 16 may be treated differently than an untrusted network 18. This will provide the ability to run a verifier 34 only when dealing with an untrusted network 18, but not when dealing with a trusted external network 16. In one embodiment, all intermediate code is passed through a verifier 34 and the source of the code merely determines the level of verification applied.

The client 12 is the target computer on which the user wishes to execute an applet. The client 12 requests applets from the server 10 in a specific form. Applets can be requested in various formats including source, intermediate and binary. In addition, an applet can be requested with verification and/or other compile time operations. Optionally, the client 12 can pass a verifier to the server to provide verification. If the server 10 implements its own security, then both the client and server verifiers will be run. The verifier that is passed from the client to the server is cached at the server for subsequent verification. The client can refer to this verifier by a server-generated handle to avoid having to pass the verifier each time an applet is requested.

Client computers 12 and 14 requesting applet code in intermediate format need to have an interpreter or virtual machine capable of interpreting the binary code in the intermediate format if the applet is to be executed on the client machine.

In the preferred embodiment, requests to the applet server are in a format similar to those of an HTTP header and are comprised of tags and values. In one embodiment, an HTTP GET method is used to make the request (though use of the HTTP protocol is not necessary to implement the present invention). The request is made up of a series of tags which specify the requested applet, the platform on which it is to

US 6,976,248 B2

7

be run and the type of code (source/intermediate/binary), a verification level and an optimization level. New tags and values can be added to extend functionality as needed and the applet server manager 22 will discard any tag it does not recognize. When the applet server computer 10 returns the requested applet to the requesting client computer A 12, it will transmit the request header followed by the applet code. In this instance, the header will additionally include a field which defines the length of the applet code. FIG. 2 provides a table which illustrates the request format and the returned code format.

While this invention has been described with reference to specific embodiments, this description is not meant to limit the scope of the invention. Various modifications of the disclosed embodiments, as well as other embodiments of the invention, will be apparent to persons skilled in the art upon reference to this description. It is therefore contemplated that the appended claims will cover any such modifications or embodiments as fall within the scope of the invention.

I claim:

1. A method operating on a computer system for managing requests to a server computer for applets in a client server environment wherein each request for an applet specifies one form of the applet out of a plurality forms of the applet, comprising:

- a) receiving on said server computer a request from a client computer for an applet in a form selected from a plurality forms;
- b) compiling said applet into said selected form from a local resource comprising at least one source module and one compiler which acts on said source module to produce said selected form; and
- c) transmitting said applet in said selected form to said client computer.

2. The method of claim 1, further comprising the step of: copying said applet in said selected form to a local cache after compiling said applet from said local resource if said cache does not contain a copy of said applet in said selected form.

3. The method of claim 2, further comprising the step of: transmitting a request to an external resource for said applet in said selected form if a copy of said applet is not stored in said local cache and said applet can not be compiled from said local resource, and directing said external resource to transmit said applet in said selected form to said server computer.

4. The method of claim 1, further comprising the step of: transmitting a request to an external resource for supplying said applet in said selected form if said applet can not be compiled from said local resource and directing said external resource to transmit said applet in said selected form to said server computer.

5. A method operating on a computer system for managing requests to a server computer for applets in a client server environment wherein each request for an applet specifies one form of said applet out of a plurality of forms of said applet, comprising:

- a) receiving on said server computer a request from a client computer for an applet in a specified form selected from a plurality of forms;
- b) determining whether said applet is stored in said specified form in a local cache and, if so, transmitting said applet in said specified form to said client computer;

8

c) if said applet is not stored in said selected form in said local cache, compiling said applet into said selected form from a local resource comprising at least one source module and one compiler which acts on said source module to produce said selected form and transmitting said applet in said selected form to said client computer.

6. The method of claim 5, further comprising the step of: copying said applet in said selected form to said cache after compiling said applet from said local resource if said cache does not contain a copy of said applet in said form.

7. The method of claim 6, further comprising the step of: transmitting a request to an external resource for said applet in said selected form if a copy of said applet is not stored in said local cache and said applet can not be compiled from said local resource, and

directing said external resource to transmit said applet in said form to said server computer.

8. The method of claim 5, further comprising the step of: transmitting a request to an external resource for said applet in said selected form if a copy of said applet is not stored in said local cache and said applet can not be compiled from said local resources, and directing said external resource to transmit said applet in said selected form to said server computer.

9. A method operating on a computer system for generating an applet in response to a request by a client computer wherein each request for an applet specifies one form of the applet out of a plurality forms of the applet, comprising:

- a) receiving on a server computer a request from a client computer for an applet in a form selected from a plurality forms;
- b) compiling an applet program module into an intermediate form program module;
- c) transmitting said applet in said selected form to said client computer.

10. The method of claim 9, further comprising the step of: transforming said intermediate form program module into a transformed intermediate form program module with at least one transformer program.

11. The method of claim 10, wherein said at least one transformer program is selected from the group consisting of verifying computer programs, optimizing computer programs, compressing computer programs, debugging computer programs, usage monitoring computer programs and encrypting computer programs.

12. The method of claim 11, further comprising the step of:

compiling said transformed intermediate form program module into machine specific binary code with a target compiler.

13. The method of claim 10, further comprising the step of:

compiling said transformed intermediate form program module into machine specific binary code with a target compiler.

\* \* \* \* \*

# **EXHIBIT C**



# Implicit Networks, Inc.

## Company Overview

Implicit Networks was founded in 1996 with the goal of developing distributed computing platforms for resource oriented, distributed devices and applications. Implicit's solution allows applications to be composed on the fly from reusable resources at the intra-device level and inter-device level. This model provides a basis for interoperability, flexible processing of rich-media, and transparent access to network services for connected devices. The platform includes an operating environment for connected devices, a dynamic component delivery model, a gesture interface for media and content-centric devices, a mechanism for discovering and controlling media-rich content in a network, dataflow processing engines, and novel ways to connect devices across both the LAN and WAN.

## Technology

The following is a subset of the technology developed by Implicit and currently available for licensing:

### *Operating Environment for resource-oriented & connected devices*

Implicit's platform for distributed computing enables devices to effectively 'plug-in' to a network, discover services, and dynamically assemble application functionality through a declarative model that requires no knowledge of device characteristics, network constraints, or data formats. In addition to dramatically reducing code size and working-set size on devices, this platform enables the development of new resources and services in a reusable manner that extends device and application functionality in profound ways. This platform has been ported to Linux, Windows CE/XP, FreeBSD, and uLinux. In practice, this solution has been used for developing a diverse range of devices including residential gateways, set-top-boxes, handheld media players, and home control systems.

### *Dynamic Component Delivery Model*

Implicit's dynamic component delivery system facilitates versioning, provisioning and billing for 3rd party development of components. The system allows developers to build components at algorithm level granularity in the language that is best suited to the domain. Components can be requested at run time by devices of varying form factors and processing requirements. The system can dynamically target (including compiling, transforming, and packaging) and deliver components to client devices. Centralized management of components allows operators to administer services, while developers focus on algorithm development. In addition to provisioning software, the system also provides a billing model that can track usage of components on clients.

### *Gesture Interface for Media Rich Devices*

Implicit's gesture-based user interface maximizes usage of the screen real estate for displaying of content while improving usability in multimedia applications. The interface provides gestures for direct manipulation of content without the need for extraneous buttons or toolbars. Example gestures include support for scaling images, skipping through audio and video tracks, and rewinding and forwarding motion video. This interface has been used for improving usability on handheld devices, televisions, and other media-rich devices.

### *Content Discovery and Media Management*

Implicit's content discovery and media management platform enables dynamic discovery of devices, content, and resources within a LAN and WAN environment. This system allows devices to aggregate their resources on the fly to create virtual appliances that users can then interact with in an intuitive manner. A user can, for example, create a virtual 'home-theater' appliance by combining a networked television with a networked stereo and a wireless handheld. The user can then watch a movie on this appliance such that the video from the movie is routed to the screen of the network television, the audio routed to the speakers of the network stereo, and instant-replays and control accessed from the handheld. Resources and services on each device can be developed without premonition of being used in this manner. Likewise, the resource discovery model allows the network to manage the routing and rendering of content without requiring the user to have understanding of the nuances of media-formats, device capabilities, or network topologies.

### *Dataflow Processing Engine*

Implicit's data processing engine dynamically assembles components to process constraint-based dataflows at an intra-device and inter-device level. The engine combines stateful packet inspection with a mapping engine to provide efficient and dynamic handling of dataflow processing. This dynamic nature has proven to be far more flexible and efficient than

traditional pipeline-based media engines since it relies on inspection of content rather than file types and source descriptions. As a result dataflows can be dynamically constructed that extend from a network interface to a media end-point (e.g. from an Ethernet MAC to a screen to display video). The system is able to interpolate dataflow processing across devices so, for example, a text file can be routed to a speaker across the LAN with one or more intermediary devices serving to convert the text to speech for final rendering as audio on the target speaker. This system has been used in VoIP systems, media players, media gateways, residential gateways and similar dataflow-centric devices.

Implicit's technology is protected by the following patents: US 6,976,248; US 6,324,685; US 6,907,446; US 6,629,163; US 6,507,349; US 7,391,791; PCT 1177514

#### Licensing

For information about licensing Implicit's intellectual property portfolio and technology platform, please [contact us](#).

---

# **EXHIBIT D**



188 The Embarcadero, Suite 750  
San Francisco, California 94105  
T: 415.247.6000 F: 415.247.6001

January 14, 2009

VIA E-MAIL AND REGULAR MAIL

Katherine K. Lutton  
Fish & Richardson  
500 Arguello Street, Suite 500  
Redwood City, CA 94063  
[lutton@fr.com](mailto:lutton@fr.com)

Robert W. Stone  
Douglas W. Colt  
Quinn Emanuel Urquhart Oliver & Hedges  
555 Twin Dolphin Drive, Suite 560  
Redwood Shores, CA 94065  
[robertstone@quinnemanuel.com](mailto:robertstone@quinnemanuel.com)  
[dougcolt@quinnemanuel.com](mailto:dougcolt@quinnemanuel.com)

David A. Nelson  
Jennifer A. Bauer  
Latham & Watkins LLP  
5800 Sears Tower, 233 S. Wacker Drive  
Chicago, Illinois 60606  
[david.nelson@lw.com](mailto:david.nelson@lw.com)  
[jennifer.bauer@lw.com](mailto:jennifer.bauer@lw.com)

Ralph H. Palumbo  
Philip S. McCune  
Summitt Law Group  
315 5<sup>th</sup> Avenue S, Suite 1000  
Seattle, WA 98104  
[ralphp@summitlaw.com](mailto:ralphp@summitlaw.com)  
[philm@summitlaw.com](mailto:philm@summitlaw.com)

Jason W. Wolff  
Fish & Richardson  
12390 El Camino Real  
San Diego, CA 92130  
[wolff@fr.com](mailto:wolff@fr.com)

Re: *Implicit Networks, Inc. v. IBM, Inc. et al.*

Dear Counsel:

By this letter, I write to ask if you will stipulate to Implicit Networks filing an amended complaint which adds two new defendants, specifically, Sun Microsystems, Inc. and Microsoft Corporation.

If the existing defendants object to this request, we will need to file a contested motion for leave to amend or a new case accompanied with a related case notice.

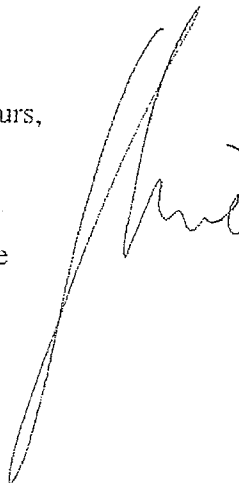


HOSIE | RICE LLP  
ATTORNEYS AT LAW

Thank you for your consideration.

Very truly yours,

Spencer Hosie

A handwritten signature in black ink, appearing to read "Spencer Hosie", written over a vertical line that extends from the signature down towards the bottom of the page.