



(51) International Patent Classification:
G06F 15/16 (2006.01) **G06F 9/30** (2006.01)
G06F 21/00 (2006.01)

Seale Avenue, Palo Alto, CA 94303 (US). **SEHR, David, C.** [US/US]; 21578 Flintshire Street, Cupertino, CA 95014 (US).

(21) International Application Number:
PCT/US2009/049679

(74) Agent: **PARK, A., Richard**; 2820 Fifth Street, Davis, CA 95616 (US).

(22) International Filing Date:
6 July 2009 (06.07.2009)

(81) Designated States (unless otherwise indicated, for every kind of national protection available): AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PE, PG, PH, PL, PT, RO, RS, RU, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
61/081,327 16 July 2008 (16.07.2008) US
12/266,602 7 November 2008 (07.11.2008) US

(71) Applicant (for all designated States except US):
GOOGLE INC. [US/US]; 1600 Amphitheatre Parkway, Bldg. 47, Mountain View, CA 94043 (US).

(72) Inventors; and

(75) Inventors/Applicants (for US only): **KOKKEVIS, Evangelos** [GR/US]; 974 Leighton Way, Sunnyvale, CA 94087 (US). **PAPAKIPOS, Matthew** [US/US]; 818

(84) Designated States (unless otherwise indicated, for every kind of regional protection available): ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE,

[Continued on next page]

(54) Title: METHOD AND SYSTEM FOR EXECUTING APPLICATIONS USING NATIVE CODE MODULES

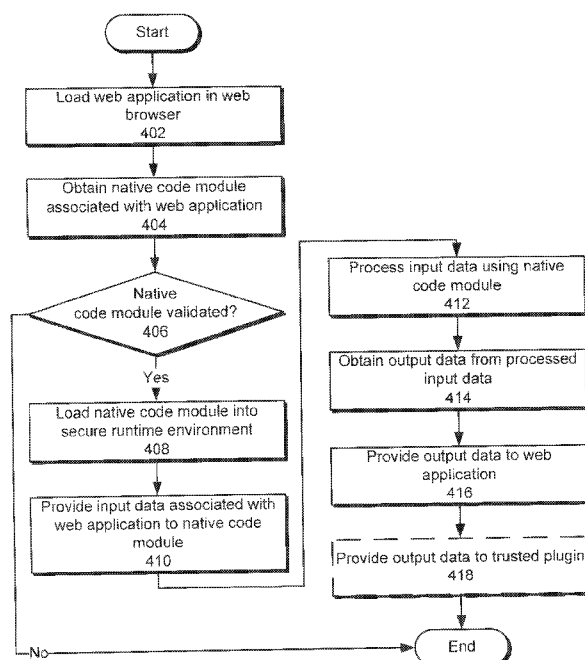


FIG. 4

(57) Abstract: Some embodiments provide a system that executes a web application. During operation, the system loads the web application in a web browser and loads a native code module associated with the web application into a secure runtime environment. Next, the system provides input data associated with the web application to the native code module and processes the input data using the native code module to obtain output data. Finally, the system provides the output data to the web application for use by the web application.



ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, **Published:**

MC, MK, MT, NL, NO, PL, PT, RO, SE, SI, SK, SM,
TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW,
ML, MR, NE, SN, TD, TG).

— *without international search report and to be republished
upon receipt of that report (Rule 48.2(g))*

METHOD AND SYSTEM FOR EXECUTING APPLICATIONS USING NATIVE CODE MODULES

Inventors: Evangelos Kokkevis, Matthew Papakipos and David C. Sehr

5

BACKGROUND

10 Field

[0001] The present embodiments relate to techniques for executing applications on computer systems. More specifically, the present embodiments relate to a method and system for executing web applications using native code modules.

15 Related Art

[0002] Computer systems may include a number of native applications that require significant processing resources to execute. Such applications may perform tasks such as image processing, audio and video decoding and mixing, mathematical modeling, and/or simulation. To run efficiently, these applications are often written in a combination of
20 general purpose programming languages such as C or C++ and low-level languages such as assembly language. Consequently, the applications may be restricted to specific platforms, involve a lengthy installation process, and may pose a security risk to the computer systems on which the applications are installed.

[0003] On the other hand, web applications, which have become more prevalent in
25 recent years, are typically written in scripting languages that are more secure and require no installation. However, web applications may lack the performance capabilities of native applications. In particular, a web application may run one or two orders of magnitude slower or more than a native application. As a result, web applications are currently unable to
30 implement computationally intensive features that are available in native applications.

SUMMARY

[0004] Some embodiments provide a system that executes a web application. During operation, the system loads the web application in a web browser and loads a native code

module associated with the web application into a secure runtime environment. Next, the system provides input data associated with the web application to the native code module and processes the input data using the native code module to obtain output data. Finally, the system provides the output data to the web application for use by the web application.

5 [0005] In some embodiments, the system also validates the native code module prior to loading the native code module into the secure runtime environment.

 [0006] In some embodiments, the output data is provided to a trusted plugin associated with the web browser for use by the trusted plugin.

10 [0007] In some embodiments, the output data is used by the trusted plugin to render graphics for the web application.

 [0008] In some embodiments, the native code module provides at least one of simulation, signal processing, artificial intelligence, modeling, and analysis for the web application.

15 [0009] In some embodiments, the secure runtime environment is implemented by a plugin associated with the web browser.

 [0010] In some embodiments, the web application and the native code module are platform independent.

20 [0011] In some embodiments, the input data is provided to the native code module using a first inter-module communication (IMC) buffer, and the output data is provided to the web application using a second IMC buffer.

 [0012] In some embodiments, processing the input data involves performing a series of mathematical calculations using the input data.

BRIEF DESCRIPTION OF THE FIGURES

25 [0013] FIG. 1 shows a schematic of an embodiment of a system.

 [0014] FIG. 2 shows the use of inter-module communication (IMC) buffers by a native code module in accordance with an embodiment of the system.

 [0015] FIG. 3 shows a system for rendering graphics for a web application.

30 [0016] FIG. 4 shows a flowchart illustrating the process of executing a web application.

DETAILED DESCRIPTION

[0017] The following description is presented to enable any person skilled in the art to make and use the disclosed embodiments, and is provided in the context of a particular application and its requirements. Various modifications to the disclosed embodiments will be readily apparent to those skilled in the art, and the general principles defined herein may be applied to other embodiments and applications without departing from the spirit and scope of the disclosed embodiments. Thus, the system is not limited to the embodiments shown, but is to be accorded the widest scope consistent with the principles and features disclosed herein.

[0018] The data structures and code described in this detailed description are typically stored on a computer-readable storage medium, which may be any device or medium that can store code and/or data for use by a computer system. The computer-readable storage medium includes, but is not limited to, volatile memory, non-volatile memory, magnetic and optical storage devices such as disk drives, magnetic tape, CDs (compact discs), DVDs (digital versatile discs or digital video discs), or other media capable of storing computer-readable media now known or later developed.

[0019] The methods and processes described in the detailed description section can be embodied as code and/or data, which can be stored in a computer-readable storage medium as described above. When a computer system reads and executes the code and/or data stored on the computer-readable storage medium, the computer system performs the methods and processes embodied as data structures and code and stored within the computer-readable storage medium.

[0020] Furthermore, the methods and processes described below can be included in hardware modules. For example, the hardware modules can include, but are not limited to, application-specific integrated circuit (ASIC) chips, field-programmable gate arrays (FPGAs), and other programmable-logic devices now known or later developed. When the hardware modules are activated, the hardware modules perform the methods and processes included within the hardware modules.

[0021] Embodiments of the system provide a method and system for executing a web application. The web application may be loaded in a web browser and executed on a computing system such as a personal computer (PC), a mobile phone, a personal digital assistant (PDA), a graphing calculator, a portable media player, a global positioning system (GPS) receiver, and/or another electronic computing device. The web application may be

obtained by the computing system from a server using a network connection with the server. For example, the web application may be downloaded over the Internet from a website.

[0022] More specifically, embodiments of the technique provide a method and system for executing the web application using a native code module. The native code module may contain native code that is executed within a secure runtime environment. In addition, the native code module may be used to process data for the web application to provide functionality associated with computationally intensive tasks such as simulation, signal processing, artificial intelligence, and/or modeling. As a result, embodiments of the system may provide a framework for combining the portability and security of web applications with the native performance of binary code.

[0023] FIG. 1 shows a schematic of a system. The system includes a computing system 102 and a set of servers (*e.g.*, server 1 104, server x 106). Computing system 102 includes a web application 116 running within a web browser 110 and a plugin 108. Each of these components is described in further detail below.

[0024] Computing system 102 may correspond to an electronic device that provides one or more services or functions to a user. For example, computing system 102 may operate as a mobile phone, personal computer (PC), global positioning system (GPS) receiver, portable media player, personal digital assistant (PDA), and/or graphing calculator. In addition, computing system 102 may include an operating system (not shown) that coordinates the use of hardware and software resources on computing system 102, as well as one or more applications (*e.g.*, web browser 110, web application 116) that perform specialized tasks for the user. For example, computing system 102 may include applications such as an email client, address book, document editor, web browser 110, and/or media player. To perform tasks for the user, applications may obtain the use of hardware resources (*e.g.*, processor, memory, I/O components, wireless transmitter, etc.) on computing system 102 from the operating system, as well as interact with the user through a hardware and/or software framework provided by the operating system, as described below.

[0025] Those skilled in the art will appreciate that computing system 102 may include functionality to execute both native applications and non-native applications. In other words, computing system 102 may include native applications, such as web browser 110, that are locally installed on computing system 102 and specific to the operating system and/or one or more hardware devices on computing system 102. Such applications may be compiled into native (*e.g.*, machine) code that executes directly on one or more central processing units

(CPUs) of computing system 102. Code execution in the applications may further be optimized by writing the applications in a combination of general-purpose programming languages (*e.g.*, C, C++, etc.) and assembly language, as well as utilizing libraries that provide hardware acceleration (*e.g.*, graphics hardware acceleration) to the applications.

5 However, the installation of native applications may compromise computing system 102 and private data stored on computing system 102.

[0026] Computing system 102 may also include functionality to execute platform independent, non-native applications. In particular, computing system 102 may obtain web application 116 from one or more servers (*e.g.*, server 1 104, server x 106) using a network
10 connection with the server(s) and load web application 116 in web browser 110. For example, web application 116 may be downloaded from an application server over the Internet by web browser 110.

[0027] Once loaded, web application 116 may provide features and user interactivity comparable to that of native applications on computing system 102. For example, web
15 application 116 may function as an email client, document editor, media player, computer-aided design (CAD) system, and/or computer game. Web application 116 may also include dynamic user interface elements such as menus, buttons, windows, sub-windows, icons, animations, and/or other graphical objects that emulate analogous user interface elements in native applications. In other words, web application 116 may correspond to a rich Internet
20 application (RIA).

[0028] Furthermore, web application 116 may execute on computing system 102 regardless of the type of platform (*e.g.*, operating system, drivers, etc.) associated with computing system 102. Though platform independent applications such as web application 116 may be more portable and secure than native applications, such cross-platform
25 applications may lack certain performance capabilities of native applications.

[0029] More specifically, non-native applications such as web application 116 may be written using scripting languages that are interpreted rather than compiled, such as Javascript (Javascript™ is a registered trademark of Sun Microsystems, Inc.). The interpretive nature of web application 116 and/or other non-native applications may result in significantly slower
30 execution times for the non-native applications than those of compiled native applications. Non-native applications may also be unable to utilize low-level libraries and/or application programming interfaces (API) that are available for use by native applications. Consequently, non-native applications may provide limited functionality in certain tasks.

[0030] In particular, web application 116 may be unable to provide computationally intensive features that are available in native applications. These features may include, but are not limited to:

- Simulation: computational fluid dynamics (CFD), rigid body dynamics, collision detection, molecular dynamics, three-dimensional (3D) animation, etc.
- Signal Processing: audio encoding and decoding, video encoding and decoding, image processing, speech signal processing, etc.
- Artificial Intelligence (AI): data mining, machine learning, game AIs, pattern recognition, etc.
- Modeling: financial modeling, mathematical modeling, scientific modeling, 3D modeling, etc.
- Analysis: numerical analysis, financial analysis, engineering analysis, cryptanalysis, statistical data analysis, etc.

In a broad sense, web application 116 may be unable to efficiently implement features that require the execution of complex mathematical operations, such as floating point operations. For example, web application 116 may execute too slowly to perform fast Fourier transforms (FFTs), Monte Carlo methods, matrix operations, conjugate gradient methods, and/or regression techniques on input data of a substantial size.

[0031] To provide computationally intensive features to the user, a native code module 118 associated with web application 116 may be used to execute computationally intensive code on behalf of web application 116. Like web application 116, native code module 118 may be obtained from one or more servers (*e.g.*, server 1 104, server x 106) by web browser 110. For example, web application 116 may provide a hyperlink to native code module 118 on the Internet. Web browser 110 may then download native code module 118 from the Uniform Resource Locator (URL) specified in the hyperlink. Alternatively, native code module 118 may be specified by the user or by an external source, such as another web application and/or a native application.

[0032] More specifically, native code module 118 may correspond to a software module containing native code that runs directly on hardware provided by computing system 102, such as a CPU. As a result, native code module 118 may be used to perform tasks that require substantial access to CPU resources on computing system 102. In particular, native code module 118 may provide one or more of the computationally intensive features listed above to a user of web application 116 by processing input data associated with web

application 116. The input data may be processed by performing a series of mathematical calculations using the input data, such as the mathematical operations described above.

[0033] Output data obtained from the processed input data may be provided to web application 116 for use by web application 116. For example, the output data may be provided to the user by web application 116 or used by web application 116 to perform additional tasks for the user. The output data may also be provided as input data to other components associated with web application 116, such as a native application, a trusted plugin, and/or one or more servers (*e.g.*, server 1 104, server x 106).

[0034] For example, web application 116 may provide audio or video encoding and/or decoding capabilities to the user of computing system 102. First, the user may specify an audio or video file to decode and/or an audio or video stream to encode. Next, web application 116 may obtain a native code module (*e.g.*, native code module 118) containing a codec for the file or stream from a server (*e.g.*, server 1 104, server x 106). The native code module may then execute on a CPU of computing system 102 to provide the codec's encoding and/or decoding capabilities to the user.

[0035] Similarly, web application 116 and native code module 118 may be used by computing system 102 to perform computationally intensive operations for a distributed computing cluster. More specifically, native code module 118 may be obtained by web application 116 and executed by plugin 108 to create a node in the distributed computing cluster on computing system 102. The computationally intensive operations may be performed by native code module 118 during periods of CPU underutilization on computing system 102. Output from native code module 118 may be sent to a server (*e.g.*, server 1 104, server x 106) to aid in the analysis of complex problems such as protein folding, unsolved mathematics problems, detection of gravitational waves, and/or stock forecasting.

[0036] In one or more embodiments, plugin 108 includes a variety of mechanisms to ensure the safe execution of native code module 118. In particular, native code module 118 may be validated by a validator 112 provided by plugin 108 prior to execution. Native code module validation is described in a co-pending non-provisional application by inventors J. Bradley Chen, Matthew T. Harren, Matthew Papakipos, David C. Sehr, and Bennet S. Yee, and filed on the same day as the instant application entitled, "Method for Validating an Untrusted Native Code Module," having serial number TO BE ASSIGNED, and filing date 8 May 2008, which is incorporated herein by reference.

[0037] Once native code module 118 is validated, native code module 118 may be loaded into a secure runtime environment 114 provided by plugin 108. Native code execution in a secure runtime environment is described in a co-pending non-provisional application by inventors J. Bradley Chen, Matthew T. Harren, Matthew Papakipos, David C. Sehr, Bennet S. Yee, and Gregory Dardyk and filed on the same day as the instant application, entitled, "Method for Safely Executing an Untrusted Native Code Module on a Computing Device," having serial number TO BE ASSIGNED, and filing date 8 May 2008, which is incorporated herein by reference.

[0038] Furthermore, because native code module 118 may include binary code that runs directly on hardware, native code module 118 may be platform independent with respect to the operating system of computing system 102, web browser 110, and/or other software components on computing system 102. As described in the above-referenced applications, plugin 108 and/or native code module 118 may also include mechanisms for executing on a variety of instruction set architectures, including the use of "fat binaries" and binary translators.

[0039] FIG. 2 shows the use of inter-module communication (IMC) buffers by a native code module. In particular, FIG. 2 shows the flow of data through native code module 118. As shown in FIG. 2, input data 208 is obtained by native code module 118 from a first inter-module communication (IMC) buffer 204. As described above, input data 208 may be specified by web application 116, a user, and/or a native application. Input data 208 may also be obtained from a variety of sources for placement in IMC buffer 204, including a host on a network, a disk, an input device (e.g., camera, microphone, etc.), and/or a hardware device (e.g., sound card, video card, etc.).

[0040] Native code module 118 may process input data 208 to obtain output data 210, which is placed into a second IMC buffer 206. More specifically, native code module 118 may obtain output data 210 by performing a series of mathematical calculations using input data 208. For example, native code module 202 may perform a series of matrix operations, mathematical transforms and/or inverse transforms, and/or other floating point operations using input data 208. Output data 210 may then be obtained from IMC buffer 206 for use by web application 116 and/or a trusted plugin 202 associated with web application 116. In particular, web application 116 and/or trusted plugin 202 may use output data 210 to perform additional tasks for a user of web application 116 or an entity associated with web application 116. For example, output data 210 may be stored in a file that is provided to the user, plotted

in a chart or graph, uploaded to a database for a distributed computing cluster, and/or used to modify the execution of other applications. Output data 210 may also be used to animate a 3D graphics model, as discussed below with respect to FIG. 3.

[0041] In one or more embodiments, IMC buffers 204-206 correspond to shared memory buffers that are used to facilitate data-intensive collaboration between native code module 118 and external components such as web application 116 and/or trusted plugin 202. Furthermore, sharing of data via IMC buffers 204-204 may be managed by an IMC runtime, as described in the above-referenced applications.

[0042] FIG. 3 shows a system for executing a web-based 3D application 302. 3D application 302 may correspond to a web application that executes in a web browser 300. In addition, 3D application 302 may provide 3D graphics rendering and animation capabilities to a user of 3D application. For example, 3D application 302 may be a 3D computer game, CAD system, and/or a scientific modeling and/or simulation application. Furthermore, 3D application 302 may be written in a web-based scripting language such as Javascript. As a result, 3D application 302 may be unable to meet the demands of real-time 3D graphics rendering and animation. Instead, 3D application 302 may offload graphics rendering to a graphics plugin 306 and animation to a plugin 304.

[0043] As shown in FIG. 3, plugin 304 includes a physics engine 310 that executes on a CPU 318. In one or more embodiments, physics engine 310 corresponds to a native code module that is executed within a secure runtime environment provided by plugin 304. Physics engine 310 may be provided by 3D application 302 (*e.g.*, downloaded over the Internet) and validated prior to execution in plugin 304. Moreover, physics engine 310 may include native code that executes directly on CPU 318 within the constraints set by the secure runtime environment. The execution of physics engine 310 on CPU 318 may thus provide 3D application 302 with real-time animation of objects within 3D application 302.

[0044] Similarly, graphics plugin 304 includes a rendering engine 312 that communicates with a graphics-processing unit (GPU) 320. In one or more embodiments, graphics plugin 304 corresponds to a trusted plugin, such as trusted plugin 202 of FIG. 2. In other words, graphics plugin 306 may be downloaded from a trusted source by web browser 300 and installed by web browser 300 to render graphics for 3D application 302. Furthermore, rendering engine 312 may provide graphics hardware acceleration by performing calculations related to graphics rendering using GPU 320. Graphics plugin 306 and rendering engine 312 are described in a co-pending non-provisional application by

inventors Vangelis Kokkevis, Matthew Papakipos, and Robin Green and filed on the same day as the instant application entitled, "WEB-BASED GRAPHICS RENDERING SYSTEM," having serial number TO BE ASSIGNED, and filing date TO BE ASSIGNED.

[0045] In one or more embodiments, 3D application 302 coordinates the joint
5 execution of physics engine 310 and rendering engine 312. More specifically, 3D application 302 may make method calls to both plugin 304 and graphics plugin 306 for loading a graphics model into rendering engine 312 and a corresponding physics model into physics engine 310. 3D application 302 may also facilitate the passing of data between plugin 304 and graphics plugin 306 using shared memory 308, as described below. On the other hand,
10 plugin 304 and graphics plugin 306 may interface directly with one another to synchronize the animation and rendering of the graphics model.

[0046] In particular, 3D application 302 may provide an input file that describes a graphics model to graphics plugin 306. Alternatively, 3D application 302 may make a set of method calls that describe the graphics model to graphics plugin 306. The graphics model
15 may then be loaded into the internal memory of graphics plugin 306. Next, 3D application 302 may request the allocation of one or more IMC buffers 314-316 in a region of shared memory 308 by an IMC runtime provided by plugin 304 and/or graphics plugin 306. Graphics plugin 306 may then load data relevant to physics simulation into IMC buffers 314-316. For example, graphics plugin 306 may copy vertex positions, normals, triangle indices,
20 and/or transformation matrices into IMC buffers 314-316.

[0047] To animate the graphics model, physics engine 310 may read from IMC buffers 314-316 to create a physics model corresponding to the graphics model in graphics plugin 306. Additional information related to the physics model, such as parameters, may be obtained from 3D application 302 by plugin 304. Next, physics engine 310 may perform a
25 series of physics simulation calculations that update the physics model. For example, physics engine 310 may calculate vertex positions and velocities based on a set of forces acting on objects in the physics model. Plugin 304 may then update IMC buffers 314-316 with new vertex positions, velocities, and/or other data. Finally, the new data is read from IMC buffers 314-316 by graphics plugin 306 and used to update the graphics model. Rendering engine
30 312 may then pass the updated graphics model to GPU 320 for rendering.

[0048] Graphics rendering and animation may continue to be provided by rendering engine 312 and physics engine 310 during execution of 3D application 302. For example, physics engine 310 may continue to update the graphics model as long as forces are felt by

objects in the graphics model. Additional objects and/or forces may also be introduced into the graphics model and/or physics model by 3D application 302. Similarly, rendering engine 312 may render the graphics model at a frame rate specified by 3D application 302 and/or supported by GPU 320. As a result, physics engine 310 and rendering engine 312 may run at
5 different frequencies. For example, physics engine 310 may run four times faster than rendering engine 312. As a result, the graphics model may be rendered once by rendering engine 312 for every four updates to the graphics model made by physics engine 310.

[0049] FIG. 4 shows a flowchart illustrating the process of executing a web application. In one or more embodiments, one or more of the steps may be omitted, repeated,
10 and/or performed in a different order. Accordingly, the specific arrangement of steps shown in FIG. 4 should not be construed as limiting the scope of the technique.

[0050] Initially, a web application is loaded into a web browser (operation 402). The web application may be obtained from a server by the web browser. Furthermore, the web application may be used to provide computationally intensive features, such as financial
15 modeling, computational math or science, and/or AI, to a user. To implement such features in a practical manner, a native code module associated with the web application may be obtained (operation 404). For example, the native code module may be downloaded from a source specified by the web application. The native code module may also be validated (operation 406) prior to executing the native code module. If the native code module is not
20 validated, the native code module is not executed and discarded.

[0051] If the native code module is validated, the native code module is loaded into a secure runtime environment (operation 408). The secure runtime environment may be provided by a plugin associated with the web browser. Next, input data associated with the web application is provided to the native code module (operation 410). The input data may
25 be provided to the native code module using an IMC buffer.

[0052] The input data is then processed by the native code module (operation 412) to obtain output data (operation 414). The output data is then provided to the web application (operation 416) for use by the web application. As with the input data, the output data may be provided to the web application using an IMC buffer. For example, the application may
30 use the output data for simulation, signal processing, artificial intelligence, modeling, and/or analysis. In addition, the output data may optionally be provided to a trusted plugin (operation 418) associated with the web application for use by the trusted plugin. For

example, the output data may be used by the trusted plugin to render graphics for the web application.

[0053] The foregoing descriptions of embodiments of the system have been presented only for purposes of illustration and description. They are not intended to be exhaustive or to
5 limit the system to the forms disclosed. Accordingly, many modifications and variations will be apparent to practitioners skilled in the art. Additionally, the above disclosure is not intended to limit the system. The scope of the present system is defined by the appended claims.

What Is Claimed Is:

1. A method for executing a web application, comprising:
loading the web application in a web browser;
5 loading a native code module associated with the web application into a secure runtime environment;
providing input data associated with the web application to the native code module;
processing the input data using the native code module to obtain output data; and
providing the output data to the web application for use by the web application.
- 10 2. The method of claim 1, further comprising:
validating the native code module prior to loading the native code module into the secure runtime environment.
3. The method of claim 1, wherein the output data is provided to a trusted plugin associated with the web browser for use by the trusted plugin.
- 15 4. The method of claim 3, wherein the output data is used by the trusted plugin to render graphics for the web application.
5. The method of claim 1, wherein the native code module provides at least one of simulation, signal processing, artificial intelligence, modeling, and analysis for the web application.
- 20 6. The method of claim 1, wherein the secure runtime environment is implemented by a plugin associated with the web browser.
7. The method of claim 1, wherein the web application and the native code module are platform independent.
8. The method of claim 1,
25 wherein the input data is provided to the native code module using a first inter-module communication (IMC) buffer, and

wherein the output data is provided to the web application using a second IMC buffer.

9. The method of claim 1, wherein processing the input data involves performing a series of mathematical calculations using the input data.

10. A system for executing a web application, comprising:

a web browser configured to load the web application;

a native code module associated with the web application; and

a secure runtime environment configured to execute the native code module,

wherein the native code module is configured to:

obtain input data associated with the web application;

process the input data to obtain output data; and

provide the output data to the web application for use by the web application.

11. The system of claim 10, further comprising:

a validator configured to validate the native code module prior executing the native code module.

12. The system of claim 11, wherein the secure runtime environment and the validator are implemented using a plugin associated with the web browser.

13. The system of claim 10, further comprising:

a trusted plugin associated with the web browser,

wherein the output data is provided to the trusted plugin for use by the trusted plugin.

14. The system of claim 13, wherein the output data is used by the trusted plugin to render graphics for the web application.

15. The system of claim 10, further comprising:

a first inter-module communication (IMC) buffer configured to provide the input data to the native code module; and

a second IMC buffer configured to provide the output data to the web application.

16. The system of claim 10, wherein the native code module provides at least one of simulation, signal processing, artificial intelligence, modeling, and analysis for the web application.

17. The system of claim 10, wherein the web application and the native code
5 module are platform independent.

18. The system of claim 10, wherein processing the input data involves performing a series of mathematical calculations using the input data.

19. A computer-readable storage medium storing instructions that when executed by a computer cause the computer to perform a method for executing a web application, the
10 method comprising:

loading the web application in a web browser;

loading a native code module associated with the web application into a secure runtime environment;

providing input data associated with the web application to the native code module;

15 processing the input data using the native code module to obtain output data; and
providing the output data to the web application for use by the web application.

20. The computer-readable storage medium of claim 19, the method further comprising:

20 validating the native code module prior to loading the native code module into the secure runtime environment.

21. The computer-readable storage medium of claim 19, wherein the output data is provided to a trusted plugin associated with the web browser for use by the trusted plugin.

22. The computer-readable storage medium of claim 21, wherein the output data is used by the trusted plugin to render graphics for the web application.

23. The computer-readable storage medium of claim 19, wherein the native code module provides at least one of simulation, signal processing, artificial intelligence, modeling, and analysis for the web application.

24. The computer-readable storage medium of claim 19, wherein the secure
5 runtime environment is implemented by a plugin associated with the web browser.

25. The computer-readable storage medium of claim 19, wherein the web application and the native code module are platform independent.

26. The computer-readable storage medium of claim 19,
wherein the input data is provided to the native code module using a first inter-module
10 communication (IMC) buffer, and
wherein the output data is provided to the web application using a second IMC buffer.

27. The computer-readable storage medium of claim 19, wherein processing the input data involves performing a series of mathematical calculations using the input data.

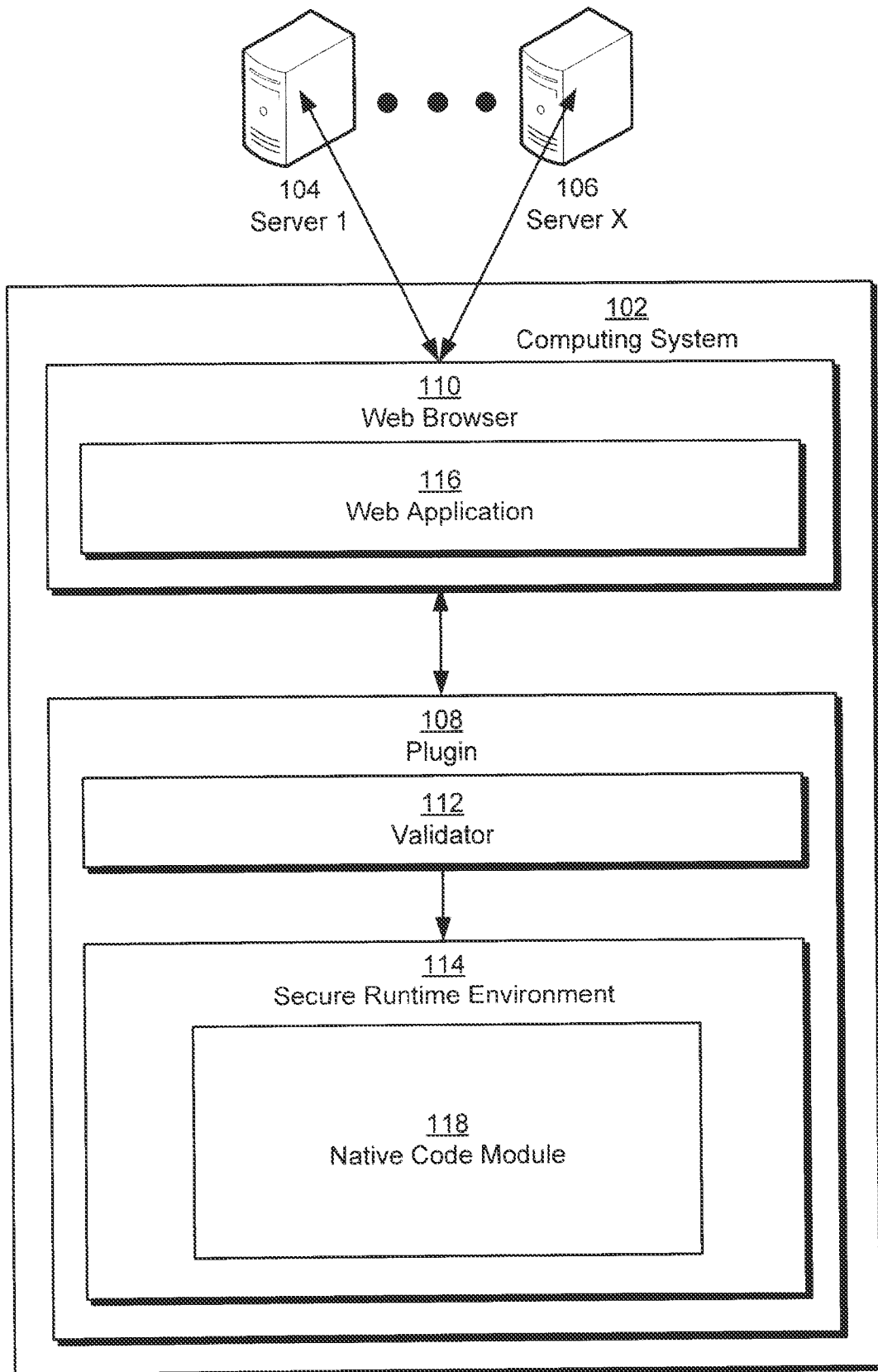
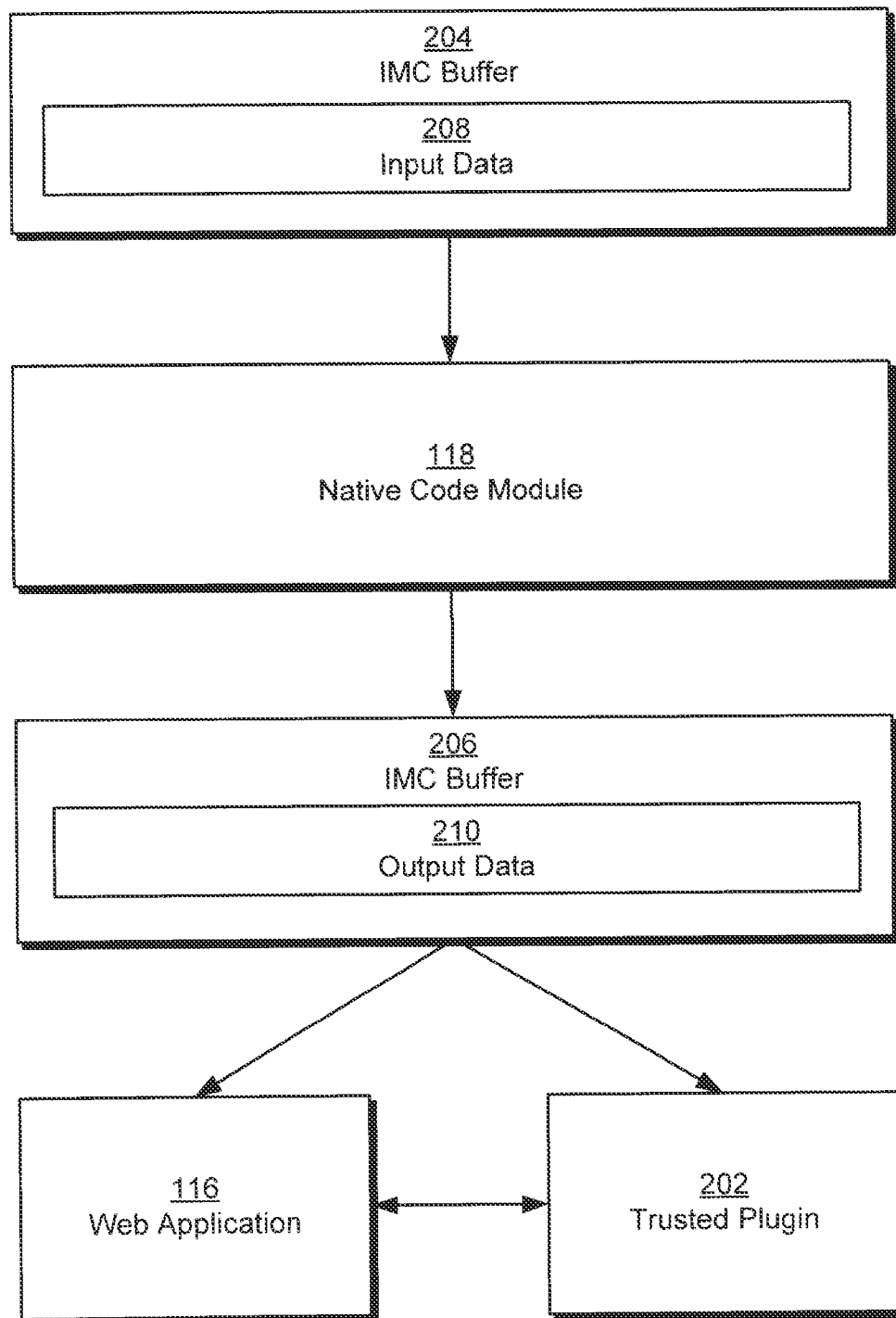
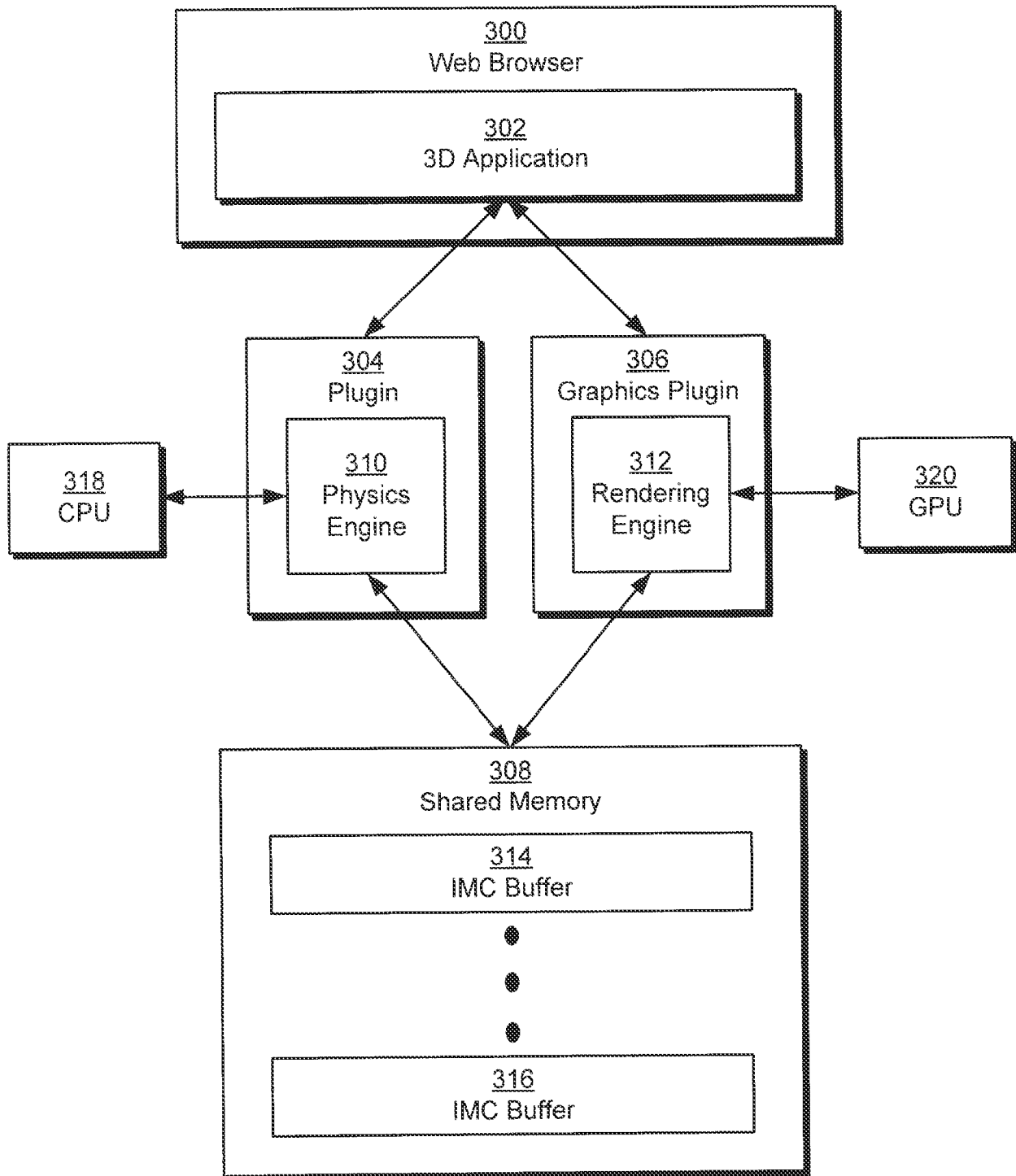


FIG. 1

**FIG. 2**

**FIG. 3**

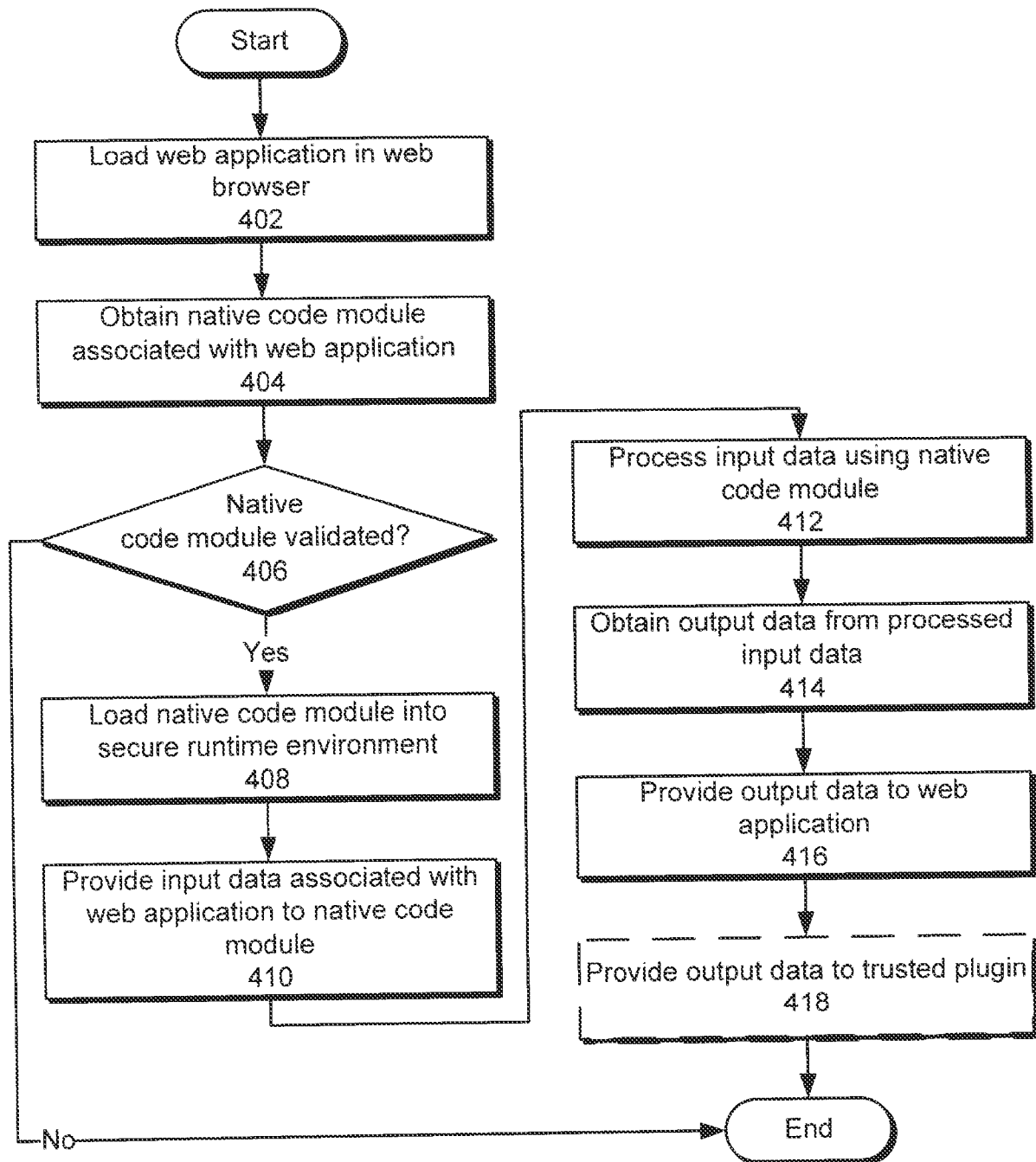


FIG. 4