Innovation, Sciences et Développement économique Canada
Office de la Propriété Intellectuelle du Canada

Innovation, Science and Economic Development Canada
Canadian Intellectual Property Office

CA 2875850 C 2022/07/05

(11)(21) 2 875 850

(12) BREVET CANADIEN
CANADIAN PATENT

(13) C

(22) Date de dépôt/Filing Date: 2014/12/23

(41) Mise à la disp. pub./Open to Public Insp.: 2016/06/23

(45) Date de délivrance/Issue Date: 2022/07/05

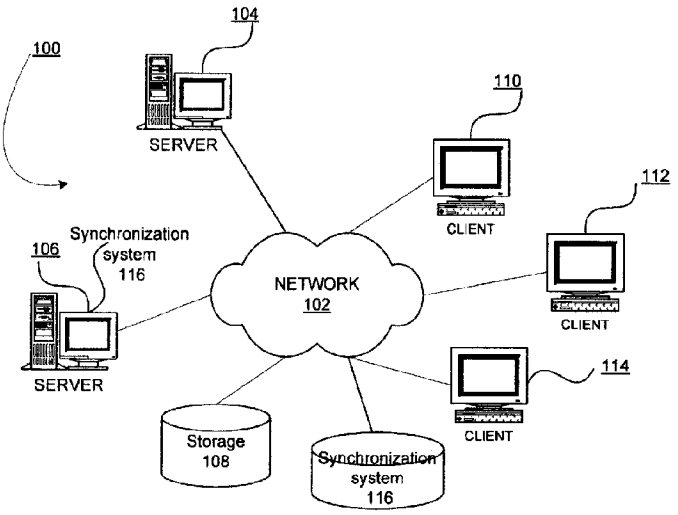
(51) Cl.Int./Int.Cl. *H04L 67/1095* (2022.01)

(72) Inventeurs/Inventors:
COOPER, STEVE, CA;
CHEN, TOM CHING LING, CA;
PETT, ROGER E., CA;
TRUONG, TRONG, CA

(73) Propriétaire/Owner:
IBM CANADA LIMITED - IBM CANADA LIMITEE, CA

(74) Agent: CHAN, BILL W.K.

(54) Titre : SYNCHRONISATION DE VERSION DE COMPOSANTS DEPENDANTS
(54) Title: VERSION SYNCHRONIZATION OF DEPENDENT COMPONENTS



(57) Abrégé/Abstract:
An illustrative embodiment of a process for automatic version synchronization of dependent components running on heterogeneous systems, in response to receiving a communication using a predetermined protocol through a network at a host, determines whether required client code is not present on the client. In response to a determination required client code is not present on the client, a latest version of the client component is requested at the host. One or more loadable modules are located at the host, bound with a host component that represents the client component. An object comprising the one or more loadable modules is sent to the client, wherein the client receives, loads and runs the object as a new component to communicate with the host component at a latest level.

ABSTRACT

An illustrative embodiment of a process for automatic version synchronization of dependent components running on heterogeneous systems, in response to receiving a communication using a predetermined protocol through a network at a host, determines whether required client code is not present on the client. In response to a determination required client code is not present on the client, a latest version of the client component is requested at the host. One or more loadable modules are located at the host, bound with a host component that represents the client component. An object comprising the one or more loadable modules is sent to the client, wherein the client receives, loads and runs the object as a new component to communicate with the host component at a latest level.

VERSION SYNCHRONIZATION OF DEPENDENT COMPONENTS

BACKGROUND

1. Technical Field:

[0001] This disclosure relates generally to version synchronization in a data processing system and more specifically to version synchronization of dependent components running on heterogeneous data processing systems.

2. Description of the Related Art:

[0002] A typical client and host data processing system scenario requires two or more components running on different platform architectures to synchronize to ensure correct operation of a software capability. Synchronized in this scenario defines a relationship in which the host component and the corresponding client component are designed to function together at a proper level of support. Support is typically specified in the form of a version or level of a respective component. For example, on host system *A* there is a component that requires a specific version of a client program, associated with the corresponding component on host system *A*, to be installed on client system *B*.

[0003] A typical approach to maintain synchronization is to package the two corresponding components independently and perform a handshake negotiation to establish compatibility. During performance of the handshake, backward compatibility considerations need to be established and the host component typically needs to handle all levels of function available in older versions of the client component as well as the current version.

[0004] This approach is typically very costly to maintain because the code needs to be developed and tested with the different versions of the client components. Further, deployment is also typically costly and time-consuming. In one example, a user discovers an incompatibility between a client component in use and a corresponding host component, and contacts a particular desktop support department that performed rollout of the current product image. In turn the desktop support department contacts a programmer supporting a respective server. A software vendor may also be contacted for guidance and service.

Once the problem is identified, a new desktop image is built and sent to the user and tested for a resolution of the problem. The process may repeat for every update of the client and server components that require synchronization.

[0005] One approach to resolve the described problem discloses a mechanism for synchronization of a "plugin" which implicitly requires a framework available on the plug-in side with corresponding levels of application programming interfaces (APIs) available within the framework upon which the plugin is dependent.

[0006] However APIs available on systems and subsystems typically vary greatly and therefore may not represent a common mechanism applicable to all systems and subsystems. Further some programming languages have a dependency on type information being available that can be extracted through "reflection" which may limit deployment to systems and subsystems dependent upon having this type of infrastructure available.

[0007] In another example of a resolution, a synchronization system synchronizes applications and data, including applications of differing versions that rely on different underlying data and schemas, and seamlessly map these versions to the underlying synchronized databases. In performing the synchronization the system and associated network typically requires robust resources to maintain, manage and deploy solutions.

SUMMARY

[0008] According to one embodiment, a computer-implemented process for automatic version synchronization of dependent components running on heterogeneous systems in response to receiving a communication using a predetermined protocol through a network at a host, determines whether required client code is not present on the client. In response to a determination required client code is not present on the client, a latest version of the client component is requested at the host. A loadable module is located at the host bound with a host component that represents the client component. The object is sent to the client, wherein the client receives, and loads the object and runs the object as a new component to communicate with the host component at a latest level.

BRIEF DESCRIPTION OF THE DRAWINGS

[0009] For a more complete understanding of this disclosure, reference is now made to the following brief description, taken in conjunction with the accompanying drawings and detailed description, wherein like reference numerals represent like parts.

[0010] **Figure 1** is a block diagram of an exemplary network data processing system operable for various embodiments of the disclosure;

[0011] **Figure 2** is a block diagram of an exemplary data processing system operable for various embodiments of the disclosure;

[0012] **Figure 3** a block diagram of synchronization system operable for various embodiments of the disclosure;

[0013] **Figure 4** is a flowchart of a build process of the synchronization system of **Figure 3** operable for various embodiments of the disclosure; and

[0014] **Figure 5** is a flowchart of a runtime process of the synchronization system of **Figure 3** in accordance with one embodiment of the disclosure.

DETAILED DESCRIPTION

[0015] Although an illustrative implementation of one or more embodiments is provided below, the disclosed systems and/or methods may be implemented using any number of techniques. This disclosure should in no way be limited to the illustrative implementations, drawings, and techniques illustrated below, including the exemplary designs and implementations illustrated and described herein, but may be modified within the scope of the appended claims along with their full scope of equivalents.

[0016] As will be appreciated by one skilled in the art, aspects of the present disclosure may be embodied in which the present invention may be a system, a method, and/or a computer program product. The computer program product may include a computer readable storage medium (or media) having computer readable program instructions thereon for causing a processor to carry out aspects of the present invention.

[0017] The computer readable storage medium can be a tangible device that can retain and store instructions for use by an instruction execution device. The computer readable

storage medium may be, for example, but is not limited to, an electronic storage device, a magnetic storage device, an optical storage device, an electromagnetic storage device, a semiconductor storage device, or any suitable combination of the foregoing. A non-exhaustive list of more specific examples of the computer readable storage medium includes the following: a portable computer diskette, a hard disk, a random access memory (RAM), a read-only memory (ROM), an erasable programmable read-only memory (EPROM or Flash memory), a static random access memory (SRAM), a portable compact disc read-only memory (CD-ROM), a digital versatile disk (DVD), a memory stick, a floppy disk, a mechanically encoded device such as punch-cards or raised structures in a groove having instructions recorded thereon, and any suitable combination of the foregoing. A computer readable storage medium, as used herein, is not to be construed as being transitory signals per se, such as radio waves or other freely propagating electromagnetic waves, electromagnetic waves propagating through a waveguide or other transmission media (e.g., light pulses passing through a fiber-optic cable), or electrical signals transmitted through a wire.

[0018] Computer readable program instructions described herein can be downloaded to respective computing/processing devices from a computer readable storage medium or to an external computer or external storage device via a network, for example, the Internet, a local area network, a wide area network and/or a wireless network. The network may comprise copper transmission cables, optical transmission fibers, wireless transmission, routers, firewalls, switches, gateway computers and/or edge servers. A network adapter card or network interface in each computing/processing device receives computer readable program instructions from the network and forwards the computer readable program instructions for storage in a computer readable storage medium within the respective computing/processing device.

[0019] Computer readable program instructions for carrying out operations of the present invention may be assembler instructions, instruction-set-architecture (ISA) instructions, machine instructions, machine dependent instructions, microcode, firmware instructions, state-setting data, or either source code or object code written in any combination of one or more programming languages, including an object oriented programming language such as Smalltalk, C++ or the like, and conventional procedural programming languages, such

as the "C" programming language or similar programming languages. The computer readable program instructions may execute entirely on the user's computer, partly on the user's computer, as a stand-alone software package, partly on the user's computer and partly on a remote computer or entirely on the remote computer or server. In the latter scenario, the remote computer may be connected to the user's computer through any type of network, including a local area network (LAN) or a wide area network (WAN), or the connection may be made to an external computer (for example, through the Internet using an Internet Service Provider). In some embodiments, electronic circuitry including, for example, programmable logic circuitry, field-programmable gate arrays (FPGA), or programmable logic arrays (PLA) may execute the computer readable program instructions by utilizing state information of the computer readable program instructions to personalize the electronic circuitry, in order to perform aspects of the present invention.

[0020] Aspects of the present invention are described herein with reference to flowchart illustrations and/or block diagrams of methods, apparatus (systems), and computer program products according to embodiments of the invention. It will be understood that each block of the flowchart illustrations and/or block diagrams, and combinations of blocks in the flowchart illustrations and/or block diagrams, can be implemented by computer readable program instructions.

[0021] These computer readable program instructions may be provided to a processor of a general purpose computer, special purpose computer, or other programmable data processing apparatus to produce a machine, such that the instructions, which execute via the processor of the computer or other programmable data processing apparatus, create means for implementing the functions/acts specified in the flowchart and/or block diagram block or blocks. These computer readable program instructions may also be stored in a computer readable storage medium that can direct a computer, a programmable data processing apparatus, and/or other devices to function in a particular manner, such that the computer readable storage medium having instructions stored therein comprises an article of manufacture including instructions which implement aspects of the function/act specified in the flowchart and/or block diagram block or blocks.

[0022] The computer readable program instructions may also be loaded onto a computer, other programmable data processing apparatus, or other device to cause a series of

operational steps to be performed on the computer, other programmable apparatus or other device to produce a computer implemented process, such that the instructions which execute on the computer, other programmable apparatus, or other device implement the functions/acts specified in the flowchart and/or block diagram block or blocks.

[0023] The flowchart and block diagrams in the Figures illustrate the architecture, functionality, and operation of possible implementations of systems, methods, and computer program products according to various embodiments of the present invention. In this regard, each block in the flowchart or block diagrams may represent a module, segment, or portion of instructions, which comprises one or more executable instructions for implementing the specified logical function(s). In some alternative implementations, the functions noted in the block may occur out of the order noted in the figures. For example, two blocks shown in succession may, in fact, be executed substantially concurrently, or the blocks may sometimes be executed in the reverse order, depending upon the functionality involved. It will also be noted that each block of the block diagrams and/or flowchart illustration, and combinations of blocks in the block diagrams and/or flowchart illustration, can be implemented by special purpose hardware-based systems that perform the specified functions or acts or carry out combinations of special purpose hardware and computer instructions.

[0024] With reference now to the figures and in particular with reference to **Figures 1-2**, exemplary diagrams of data processing environments are provided in which illustrative embodiments may be implemented. It should be appreciated that **Figures 1-2** are only exemplary and are not intended to assert or imply any limitation with regard to the environments in which different embodiments may be implemented. Many modifications to the depicted environments may be made.

[0025] **Figure 1** depicts a pictorial representation of a network of data processing systems in which illustrative embodiments may be implemented. Network data processing system **100** is a network of computers in which the illustrative embodiments may be implemented. Network data processing system **100** contains network **102**, which is the medium used to provide communications links between various devices and computers connected together within network data processing system **100**. Network **102** may include connections, such as wire, wireless communication links, or fiber optic cables.

[0026] In the depicted example, server **104** and server **106** connect to network **102** along with storage unit **108**. In addition, clients **110**, **112**, and **114** connect to network **102**. Clients **110**, **112**, and **114** may be, for example, personal computers or network computers. In the depicted example, server **104** provides data, such as boot files, operating system images, synchronization system **116** and applications to clients **110**, **112**, and **114**. Clients **110**, **112**, and **114** are clients to server **104** in this example. Network data processing system **100** may include additional servers, clients, and other devices not shown. Synchronization system **116** is also available for download from another storage instance.

[0027] In the depicted example, network data processing system **100** is the Internet with network **102** representing a worldwide collection of networks and gateways that use the Transmission Control Protocol/Internet Protocol (TCP/IP) suite of protocols to communicate with one another. At the heart of the Internet is a backbone of high-speed data communication lines between major nodes or host computers, consisting of thousands of commercial, governmental, educational and other computer systems that route data and messages. Of course, network data processing system **100** also may be implemented as a number of different types of networks, such as for example, an intranet, a local area network (LAN), or a wide area network (WAN). **Figure 1** is intended as an example, and not as an architectural limitation for the different illustrative embodiments.

[0028] With reference to **Figure 2** a block diagram of an exemplary data processing system operable for various embodiments of the disclosure is presented. In this illustrative example, data processing system **200** includes communications fabric **202**, which provides communications between processor unit **204**, memory **206**, persistent storage **208**, communications unit **210**, input/output (I/O) unit **212**, and display **214**.

[0029] Processor unit **204** serves to execute instructions for software that may be loaded into memory **206**. Processor unit **204** may be a set of one or more processors or may be a multi-processor core, depending on the particular implementation. Further, processor unit **204** may be implemented using one or more heterogeneous processor systems in which a main processor is present with secondary processors on a single chip. As another illustrative example, processor unit **204** may be a symmetric multi-processor system containing multiple processors of the same type.

[0030] Memory **206** and persistent storage **208** are examples of storage devices **216**. A storage device is any piece of hardware that is capable of storing information, such as, for example without limitation, data, program code in functional form, and/or other suitable information either on a temporary basis and/or a permanent basis. Memory **206**, in these examples, may be, for example, a random access memory or any other suitable volatile or non-volatile storage device. Persistent storage **208** may take various forms depending on the particular implementation. For example, persistent storage **208** may contain one or more components or devices. For example, persistent storage **208** may be a hard drive, a flash memory, a rewritable optical disk, a rewritable magnetic tape, or some combination of the above. The media used by persistent storage **208** also may be removable. For example, a removable hard drive may be used for persistent storage **208**.

[0031] Communications unit **210,** in these examples, provides for communications with other data processing systems or devices. In these examples, communications unit **210** is a network interface card. Communications unit **210** may provide communications through the use of either or both physical and wireless communications links.

[0032] Input/output unit **212** allows for input and output of data with other devices that may be connected to data processing system **200**. For example, input/output unit **212** may provide a connection for user input through a keyboard, a mouse, and/or some other suitable input device. Further, input/output unit **212** may send output to a printer. Display **214** provides a mechanism to display information to a user.

[0033] Instructions for the operating system, applications and/or programs including synchronization system **224** in may be located in storage devices **216,** which are in communication with processor unit **204** through communications fabric **202.** In these illustrative examples the instructions are in a functional form on persistent storage **208**. These instructions may be loaded into memory **206** for execution by processor unit **204**. The processes of the different embodiments may be performed by processor unit **204** using computer-implemented instructions, which may be located in a memory, such as memory **206**.

[0034] These instructions are referred to as program code, computer usable program code, or computer readable program code that may be read and executed by a processor in processor unit **204**. The program code in the different embodiments may be embodied on

different physical or tangible computer readable storage media, such as memory **206** or persistent storage **208**.

[0035] Program code **218** is located in a functional form on computer readable storage media **220** that is selectively removable and may be loaded onto or transferred to data processing system **200** for execution by processor unit **204**. Program code **218** and computer readable storage media **220** form computer program product **222** in these examples. In one example, computer readable storage media **220** may be in a tangible form, such as, for example, an optical or magnetic disc that is inserted or placed into a drive or other device that is part of persistent storage **208** for transfer onto a storage device, such as a hard drive that is part of persistent storage **208**. In a tangible form, computer readable storage media **220** also may take the form of a persistent storage, such as a hard drive, a thumb drive, or a flash memory that is connected to data processing system **200**. The tangible form of computer readable storage media **220** is also referred to as computer recordable storage media or a computer readable data storage device. In some instances, computer readable storage media **220** may not be removable.

[0036] Alternatively, program code **218** may be transferred to data processing system **200** from computer readable storage media **220** through a communications link to communications unit **210** and/or through a connection to input/output unit **212**. The communications link and/or the connection may be physical or wireless in the illustrative examples. Synchronization system **224** in another embodiment is also available for download in the form of program code **218**.

[0037] In some illustrative embodiments, program code **218** may be downloaded over a network to persistent storage **208** from another device or data processing system for use within data processing system **200**. For instance, program code stored in a computer readable data storage device in a server data processing system may be downloaded over a network from the server to data processing system **200**. The data processing system providing program code **218** may be a server computer, a client computer, or some other device capable of storing and transmitting program code **218.**

[0038] Using data processing system **200** of **Figure 2** as an example, a computer-implemented process for automatic version synchronization of dependent components running on heterogeneous systems is presented. Processor unit **204** in response to receiving

a communication using a predetermined protocol through a network at a host determines whether required client code is not present on the client. In response to a determination required client code is not present on the client, a latest version of the client component is requested at the host. A loadable module is located at the host bound with a host component that represents the client component by processor unit **204**. The object is sent to the client, by processor unit **204** wherein the client receives, and loads the object and runs the object as a new component to communicate with the host component at a latest level.

[0039] Embodiments of the disclosure provide a capability for tightly binding the host component and the synchronized client component into a logical unit, dynamically un-packaging the pieces at runtime, sending the corresponding correct client components over a network to the desktop, and automatically installing the correct client components.

[0040] Embodiments of the disclosure ensure the host components and the client components are correct and current. Embodiments of the disclosure can further dynamically replace the client component without any user or administrator intervention. As a result, there is significant advantage in speed and efficiency of the rollout of new updates to functionality that requires synchronized client and server components.

[0041] Embodiments of the disclosure enable the integration of component synchronization across a set of subsystems and system comprising MVS™, IMS®, CICS®, and DB2® using a unified method including a distributed client component as a *load module* of the host component. An embodiment further provides a capability to enable a distributed program, containing a client component and a host component, to synchronize during program initialization, before user interactions (MVS®, IMS®, CICS™, and DB2® are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both).

[0042] Embodiments of the disclosure use a tightly coupled approach, which does not use plug-ins and explicitly updates client code. Components of an instance of an embodiment of the disclosure are not bound at run time; rather the host, or server, components and the client components are bound at compile time. The embodiments do not synchronize applications of differing versions; rather the embodiments provide a capability for distributing a matching client component with a particular active server component. Embodiments of the disclosure identify a required level of a client component

not present in the client currently and in response to requesting the host component for the required client component, the host component sends the required client component to the client to synchronize the client with the host. The version matching of client and server components does not require an outside deployment product, and works regardless of a supporting virtual machine to which one is connected. Specifically, the client component is kept as a pseudo-loadable code module of the server, which is not actually loaded on the server but uploaded to replace the client providing synchronization between the server of the system and the respective client.

[0043] With reference to **Figure 3** a block diagram of synchronization system operable for various embodiments of the disclosure is presented. Synchronization system **300** is an example of an embodiment of the disclosed method for version synchronization of dependent components running on heterogeneous data processing systems.

[0044] Synchronization system **300** provides a capability of creating maintaining and deploying a specialized data structure specifically designed to provide self-evident synchronization between a host component and a corresponding interdependent client component. Component as used here may be one or more components as in a set of components without limitation. Synchronization defines a requirement for a host component in one particular instance to have a corresponding client component instance so that each may communicate with the other and perform a predefined set of functions to produce an expected result.

[0045] Synchronization system **300** comprises a number of function parts including host component **302**, client component **304**, compiler **306**, client object **308**, linker **310**, host module **312**, module repository **314** and transceiver **316**.

[0046] Host component **302** provides a capability of a hosted set of function typically assigned as a set of services offered on a server in a client server relationship. Host component **302** is a binary built to function on a particular hardware platform, such as that of a server. However in an embodiment of the disclosure, host component **302** has extra components bound to it that are not typically found in a host environment. In this instance host component **302** has extra components that are not native to the host, but instead represent the binary parts of client component **304**.

[0047] Client component **304** provides a set of functions as parts needed on a client hardware platform to deliver a set of particular capabilities while matching a corresponding instance of host component **302**.

[0048] Compiler **306** provides a set of compilation capabilities as an enhanced compiler. Compiler **306** transforms client component **304**, comprising a set of parts, into object **308**. Object **308** is an intermediate representation of client component **304** in a form of an array. The array of object **308** is populated with the data of the set of parts defined in client component **304**.

[0049] Linker **310** provides a set of capabilities to link the output of compiler **306**, which is object **308**, into one or more host loadable module **312** in accordance with the specifications of the particular hardware platform of the host environment on which linker **310** executes. During link time, object **308** is built into a loadable module associated with the binary of host component **302**. Depending on the host operating system conventions, loadable module **312** could either be a separate module, or directly linked into the binary of host component **302**. Loadable module **312** may also be referred to as a distributed client component. However loadable module **312** is not designed for loading and execution on host environment. Loadable module **312** is particularly designed as a pseudo load module to exploit the built in storage and location services of the host environment. For example, host component **302** is easily able to locate loadable module **312**, after successful linking, through normal lookup facilities of the particular hardware platform. Therefore a risk of not being able to locate an associated loadable module is minimized. More information on format and reading of load modules, as in the example, may be found in the publication *MVS Diagnosis: Tools and Service Aids, Version 2 Release 1* (IBM Corporation 1988, 2014).

[0050] Host component **302** and loadable module **312** are referred to as an "early bind" because the binding or association between the two is set at compile time rather than at run time when a term of "late bind" or "dynamic bind" is used. The early bind specifically addresses a requirement to maintain proper synchronization between host component **302** and client component **304** through use of loadable module **312**.

[0051] Host component **302** and host loadable module **312** may then be stored in module repository **314** for subsequent retrieval and use. Module repository **314** provides a

Date Reçue/Date Received 2021-07-08

capability on the particular hardware platform, such as that of a server in the host environment to store, maintain and retrieve on request host component **302** and host loadable module **312**.

[0052] Transceiver **316** provides a capability in the host environment to send to and receive from one or more client communications in the form of structured and unstructured data. Structured data includes a handshaking protocol used in an exchange between an executing host component **302** and a corresponding executing client component. The handshaking protocol provides a capability to obtain attributes of a client, comprising a particular version of the client as well as an operating environment including a hardware platform and operating system in use by the client.

[0053] In examples of the disclosure, a set of subsystems and system are used comprising MVS™, IMS®, CICS®, and DB2®, as an example of the host environment using a unified method including a distributed client component as a loadable module of the host component. However the examples provided are for illustrative purpose only and do not imply an unnecessary limitation solely to the instances described in the examples.

[0054] System z® is an enterprise class system typically viewed as a market leader for systems of records. (System z® is a registered trademark of International Business Machines Corporation in the United States, other countries, or both) Effective problem determination is an important aspect of ensuring high quality on these systems. Embodiments of the disclosure typically provide a capability to accelerate innovation in delivery of new debugging capability, to significantly lower overhead of testing the innovation, and to remove a burden of maintaining *n-2* compatibility between a client and a corresponding host.

[0055] Embodiments of the disclosure are applicable for all major System z sub-systems including MVS, CICS, IMS and DB2. During a debug session, a debugger becomes an integral part of a subsystem and accordingly is subject to a uniqueness of services of that subsystem including application programming interfaces available to locate a client binary. For example, within MVS, C programming language runtime routines are available, but different APIs exist for other subsystems. Using the proposed binding of the disclosure of the client binary as a loadable module of the host component, an embodiment is able to use a common technique to find a client binary across all the subsystems. In addition, by having

the client binary tightly bound to the host component, problems in locating the client binary when installed in a separate location (dataset or hierarchical file system (HFS) path) and requiring a system programmer to properly install the client binary including setting appropriate security access are avoided.

[0056] In addition using embodiments of the disclosure provides a capability of sending full debug information to the client. Sending full debug information to the client delivers a set of new capabilities and debugging innovation without concern for dependent client frameworks or wrappers that need synchronization. Accordingly new features can be delivered in entirety without a burden of lowest common denominator considerations for backward compatibility. The host component is specifically built with the corresponding client component required, therefore by design there can be no mismatch between the host component and the client component.

[0057] Further, embodiments of the disclosure typically lower testing cost because as in the current example, there is an engine corresponding to an installed host component across all System z subsystems. In absence of an embodiment, all possible permutations of client versions and host versions in the field and across all subsystems would have to be tested representing a relatively large investment in time and resources.

[0058] With reference to **Figure 4** a flowchart of a build process operable for various embodiments of the disclosure is presented. Process **400** is an example of a build process used to create a loadable module used for version synchronization of dependent components running on heterogeneous data processing systems.

[0059] Process **400** begins (step **402**) and receives a host component binary (step **404**). This instance of a host component is particular in that this host component is a binary built with a binding of a set of extra components. The extra components however are not native to the host environment of the host component binary. Rather the extra components represent binary parts of a client component, which is defined to correspond to the particular instance of the host component binary. Therefore the particular pairing of the host component binary and the client component are explicitly synchronized.

[0060] Process **400** compiles the set of extra components (step **406**). During the compilation, an encoding is performed that transforms the client component parts into an object **408** with an array that is populated with the data of the set of extra components.

[0061] Process **400** links object **408** produced as output of the compilation into a loadable module associated with the host component binary (step **410**). Depending on the host operating system, the loadable module is one or more of separate module **412**, or directly linked into the host component binary **414**.

[0062] Process **400** stores the host component binary and the one or more loadable modules (when a separate module) in a module repository on the host environment (step **416**) and terminates thereafter (step **418**).

[0063] Process **400** describes a method for tightly binding both the host component and the synchronized client component into one logical unit. The logical unit is dynamically unpackaged at runtime into respective parts to enable sending correct client components over a network to a desktop, and automatically installing the correct client components.

[0064] This method always ensures that the host and client components are up-to-date and can dynamically replace the client component without any user or administrator intervention. As a result, there are significant advantages in the speed and efficiency of rollout new updates to functionality that requires a synchronized client and server components.

[0065] With reference to **Figure 5** a flowchart of a runtime process operable for various embodiments of the disclosure is presented. Process **500** is an example of a communication exchange used in the method for version synchronization of dependent components running on heterogeneous data processing systems.

[0066] Process **500** begins (step (**502**) and performs an initial handshake between a client and a host component to identify a required level of the client component (step **504**). Process **500** determines whether the required client code is not present on the client (step **506**). In response to a determination the required client code is not present on the client, a request is made to the host asking for the latest version of the client component code (step **508**). When process **500** determines the required client code is present on the client, process **500** terminates (step **518**).

[0067] Process **500** using the host component locates (and retrieves) the object that represents the client component code (on the host environment) (step **510**). The host component is able to locate the client component code because the host component has location information for the client component code created during the link process. The

client component is part of the host component binary itself, which makes locating and retrieval of the required objects of the client component code for the host component relatively straightforward.

[0068] In response to locating (and retrieving) the required objects of the client component code, process **500** sends the new client component to the client (step **512**). The client component is in a compressed form, which reduces the time to send as well as network resources used in the transmitting and receiving of the new client components.

[0069] In response to receiving the new client component, the client decompresses and loads then runs the new component (step **514**). The decompression, loading and execution on the client is part of a self-install of the new client component, which reduces the effort on the client to implement the new client component.

[0070] A logical extension of process **500** occurs when there are several incompatible clients to be served. For example, when a first client operates in a first client environment and a second client operates in a second client environment in which the client environment differs from the first client environment. In this case, the initial handshake includes both a version of the client and a type of the client. The host component now includes required objects for each respective supported client.

[0071] Once the new component is installed on the client, the client communicates with the host component at the latest level (step **516**) and terminates thereafter (step **518**).

The operation of process **500** relies on the new client component created using process **400** of **Figure 4** to provide the host component and loadable module using a tightly coupled approach, which does not use plug-ins and explicitly updates client code. Process **500** accordingly provides a capability for distributing a matching client component with a particular active server component. Process **500** ensures a response to requesting the host component for the required client component, is fulfilled by the host component sending the required client component to the client to synchronize the client with the host. The version matching of client and server components is self-contained because of the fixed association between the host component and the loadable module of the client component and therefore does not require an outside deployment product. Process **500** works regardless of a supporting virtual machine to which a client is connected because the host component is built with knowledge of the specific client component and a respective

location. Specifically, the client component is kept as a pseudo-loadable code module of the server, (host component) but is not loaded on the server. Rather the pseudo-loadable code module is uploaded to the client as a new installation, which does not replace a previous version of the client component, thereby providing a correct combination of the server and respective client ensuring proper synchronization programmatically. The existing client version is retained because the existing client version may be required to communicate with another down-level host component. Using this technique enables a single client to participate in respective sessions with the required set of current host versions that are deployed.

[0072] Thus is presented in an illustrative embodiment a computer-implemented process for automatic version synchronization of dependent components running on heterogeneous systems, in response to receiving a communication using a predetermined protocol through a network at a host, determines whether required client code is not present on the client. In response to a determination required client code is not present on the client, a latest version of the client component is requested at the host. A loadable module is located at the host bound with a host component that represents the client component. The object is sent to the client, wherein the client receives, and loads the object and runs the object as a new component to communicate with the host component at a latest level.

[0073] The flowchart and block diagrams in the figures illustrate the architecture, functionality, and operation of possible implementations of systems, methods, and computer program products according to various embodiments of the present invention. In this regard, each block in the flowchart or block diagrams may represent a module, segment, or portion of code, which comprises one or more executable instructions for implementing a specified logical function. It should also be noted that, in some alternative implementations, the functions noted in the block might occur out of the order noted in the figures. For example, two blocks shown in succession may, in fact, be executed substantially concurrently, or the blocks may sometimes be executed in the reverse order, depending upon the functionality involved. It will also be noted that each block of the block diagrams and/or flowchart illustration, and combinations of blocks in the block diagrams and/or flowchart illustration, can be implemented by special purpose hardware-

based systems that perform the specified functions or acts, or combinations of special purpose hardware and computer instructions.

[0074] The corresponding structures, materials, acts, and equivalents of all means or step plus function elements in the claims below are intended to include any structure, material, or act for performing the function in combination with other claimed elements as specifically claimed. The description of the present invention has been presented for purposes of illustration and description, but is not intended to be exhaustive or limited to the invention in the form disclosed. Many modifications and variations will be apparent to those of ordinary skill in the art without departing from the scope and spirit of the invention. The embodiment was chosen and described in order to best explain the principles of the invention and the practical application, and to enable others of ordinary skill in the art to understand the invention for various embodiments with various modifications as are suited to the particular use contemplated.

[0075] The invention can take the form of an entirely hardware embodiment, an entirely software embodiment or an embodiment containing both hardware and software elements. In a preferred embodiment, the invention is implemented in software, which includes but is not limited to firmware, resident software, microcode, and other software media that may be recognized by one skilled in the art.

[0076] It is important to note that while the present invention has been described in the context of a fully functioning data processing system, those of ordinary skill in the art will appreciate that the processes of the present invention are capable of being distributed in the form of a computer readable data storage device having computer executable instructions stored thereon in a variety of forms. Examples of computer readable data storage devices include recordable-type media, such as a floppy disk, a hard disk drive, a RAM, CD-ROMs, DVD-ROMs. The computer executable instructions may take the form of coded formats that are decoded for actual use in a particular data processing system.

[0077] A data processing system suitable for storing and/or executing computer executable instructions comprising program code will include one or more processors coupled directly or indirectly to memory elements through a system bus. The memory elements can include local memory employed during actual execution of the program code, bulk storage, and cache memories which provide temporary storage of at least some

program code in order to reduce the number of times code must be retrieved from bulk storage during execution.

[0078] Input/output or I/O devices (including but not limited to keyboards, displays, pointing devices, etc.) can be coupled to the system either directly or through intervening I/O controllers.

[0079] Network adapters may also be coupled to the system to enable the data processing system to become coupled to other data processing systems or remote printers or storage devices through intervening private or public networks. Modems, cable modems, and Ethernet cards are just a few of the currently available types of network adapters.

CLAIMS:

What is claimed is:

1. A computer-implemented method for synchronizing components of heterogeneous systems, comprising:

retrieving, from one or more loadable modules at a host computer, an object representing a latest version of a client component pertaining to client code, the host computer comprising a host component, the host component being host software configured to be executed in the host computer;

sending the object to a client computer, wherein the client computer is configured to receive, load and run the object as a new client component to communicate with the host component at a latest level of the client component;

building a host component binary with a set of additional components bound to host component binary, wherein the additional components are not native to the host computer, and represent binary parts of a corresponding client component;

generating an encoding during compilation of the additional components, wherein the encoding transforms the client component into the object with an array populated with data of the client component; and

building, during link time, the object into the one or loadable modules tightly bound with the host component binary, wherein the one or more loadable modules is one of one or more separate modules, or directly linked into the host component binary in accordance with a host operating system.

2. The computer-implemented method of claim 1, further comprising locating the one or more loadable modules at the host computer, the one or more loadable modules bound with the host component, prior to retrieving the object from the one or more loadable modules.

3. The computer-implemented method of claim 2, wherein the locating is carried out in response to a request for the latest version of the client component pertaining to the client code.

Date Reçue/Date Received 2021-07-08

4. The computer-implemented method of claim 3, wherein the request is made in response to a determination that the client code is not present on the client computer.

5. The computer-implemented method of claim 4, wherein the determination is made in response to a communication using a predetermined protocol identifying the client code.

6. The computer-implemented method of claim 5 wherein the communication is an initial handshake between the client component and the host component identifying a required level of the client component.

7. The computer-implemented method of claim 6 wherein the initial handshake further identifies at least one of a hardware platform and an operating system variant.

8. The computer-implemented method of either claim 6 or 7 wherein there are several incompatible clients, the initial handshake includes a version and a client type, and the host component includes objects for all clients.

9. The computer-implemented method of any one of claims 1 to 8 wherein retrieving the object comprises:

the host component locating the object that represents the client component, wherein a single variant of the client component exists using a required level of the client component wherein the host component comprises a set of objects for other client computers, and wherein each object of the set of objects is specific to a respective other client computer.

10. The computer-implemented method of any one of claims 1 to 9 wherein the one or more loadable modules are pseudo load modules of a same predetermined type having a same set of predetermined characteristics as a normal load module, but the one or more loadable modules tightly bound with the host component binary are non-executable on the host operating system.

11. A computer-readable medium storing code which, when executed by one or more processors of one or more computer systems, causes the one or more computer systems to implement the method of any one of claims 1 to 10.

Date Reçue/Date Received 2021-07-08

12. A computer system, comprising:

a communications fabric;

a memory connected to the communications fabric, wherein the memory contains computer executable program code;

a communications unit connected to the communications fabric;

an input/output unit connected to the communications fabric; and

a processor unit connected to the communications fabric, wherein the processor unit executes the computer executable program code to direct the computer system to implement the method of any one of claims 1 to 10.

Date Reçue/Date Received 2021-07-08

FIG. 1

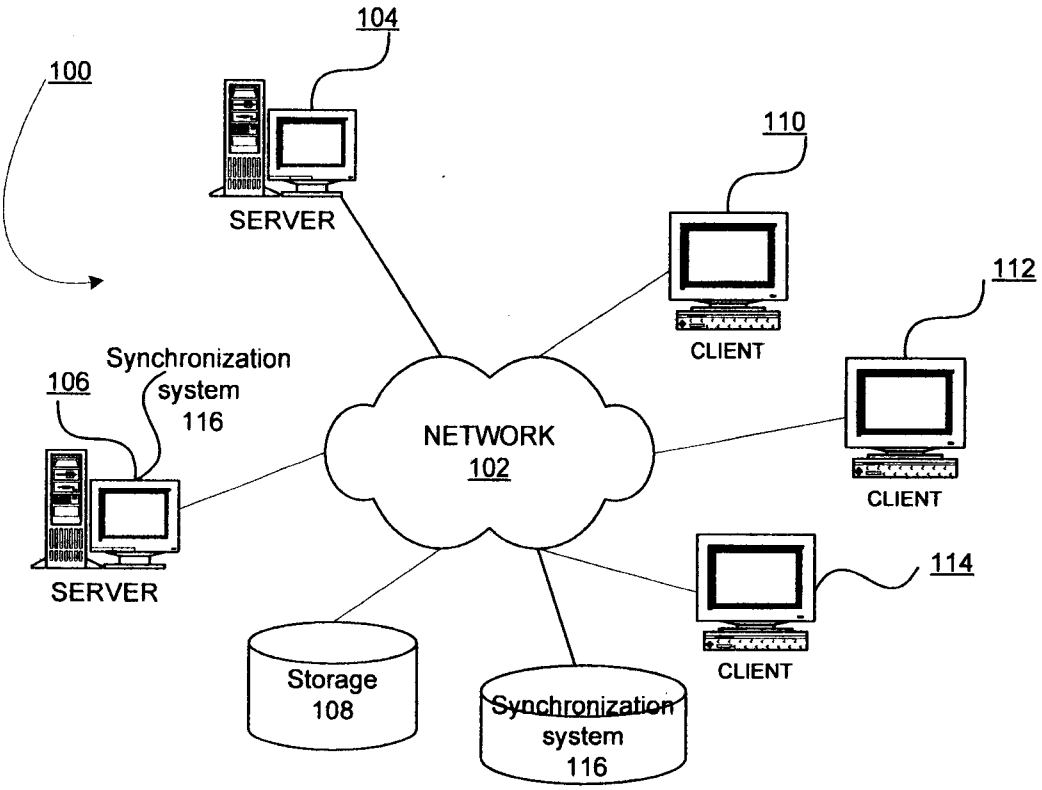

100
104
SERVER
110
CLIENT
112
CLIENT
106
Synchronization system 116
SERVER
NETWORK 102
114
CLIENT
Storage 108
Synchronization system 116

FIG. 2

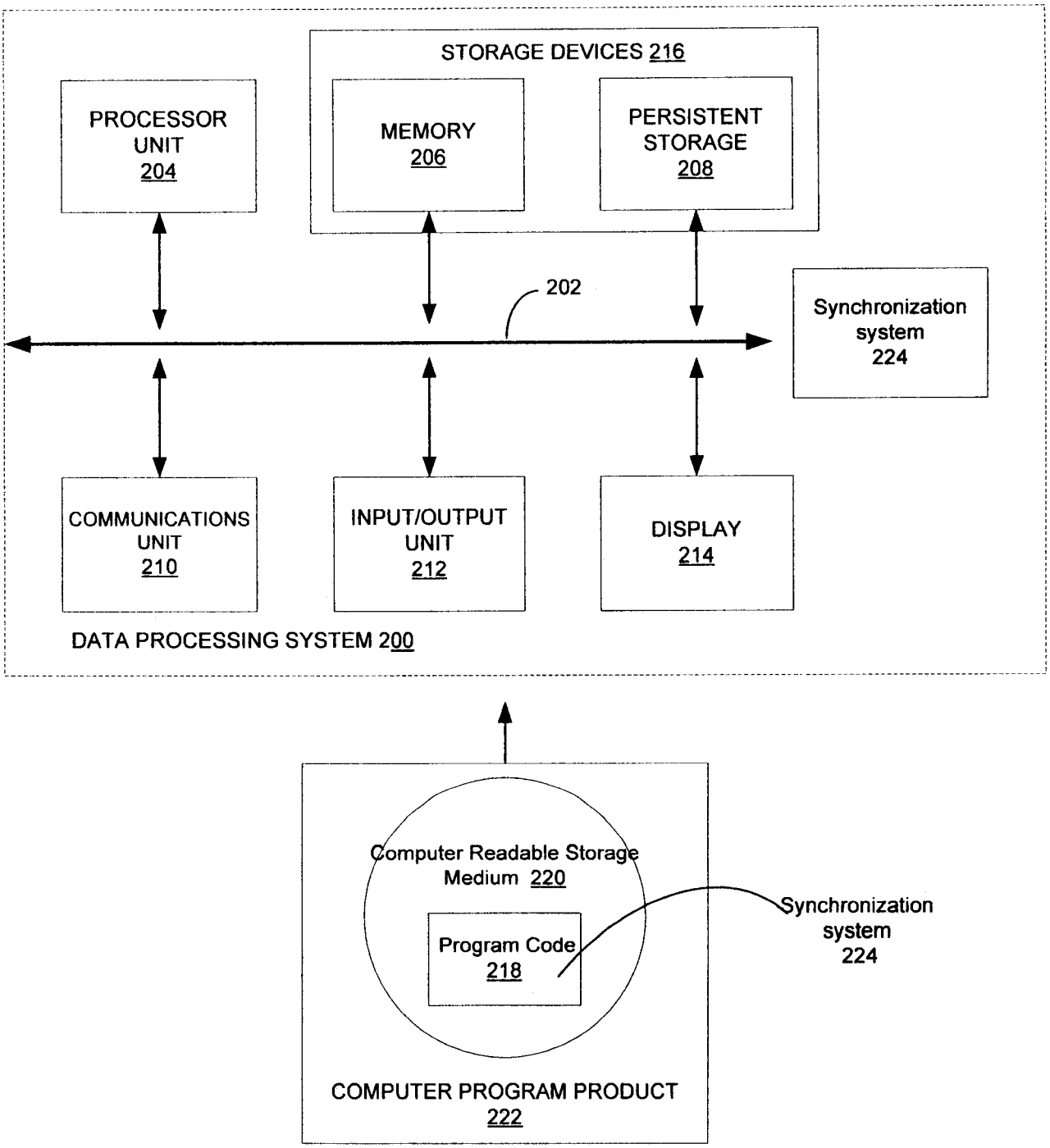

STORAGE DEVICES 216
PROCESSOR UNIT 204
MEMORY 206
PERSISTENT STORAGE 208
202
Synchronization system 224
COMMUNICATIONS UNIT 210
INPUT/OUTPUT UNIT 212
DISPLAY 214
DATA PROCESSING SYSTEM 200
Computer Readable Storage Medium 220
Program Code 218
Synchronization system 224
COMPUTER PROGRAM PRODUCT 222

FIG. 3

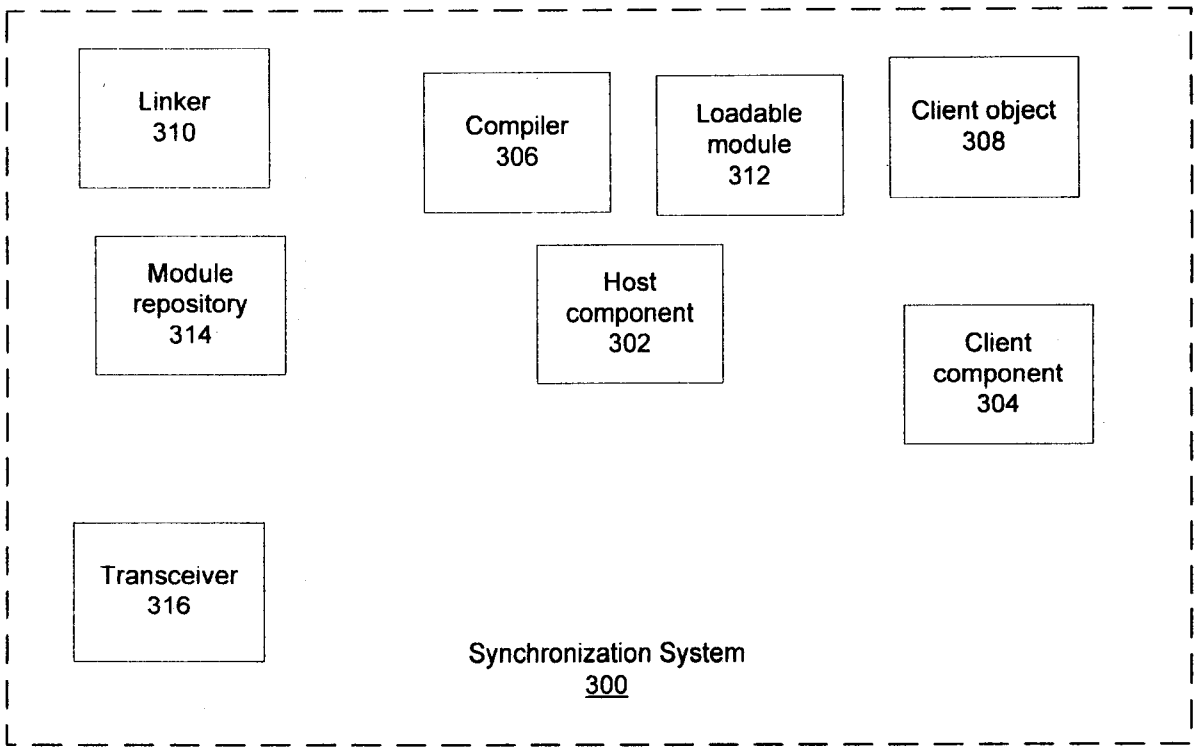

Linker
310
Compiler
306
Loadable
module
312
Client object
308
Module
repository
314
Host
component
302
Client
component
304
Transceiver
316
Synchronization System
300

FIG. 4

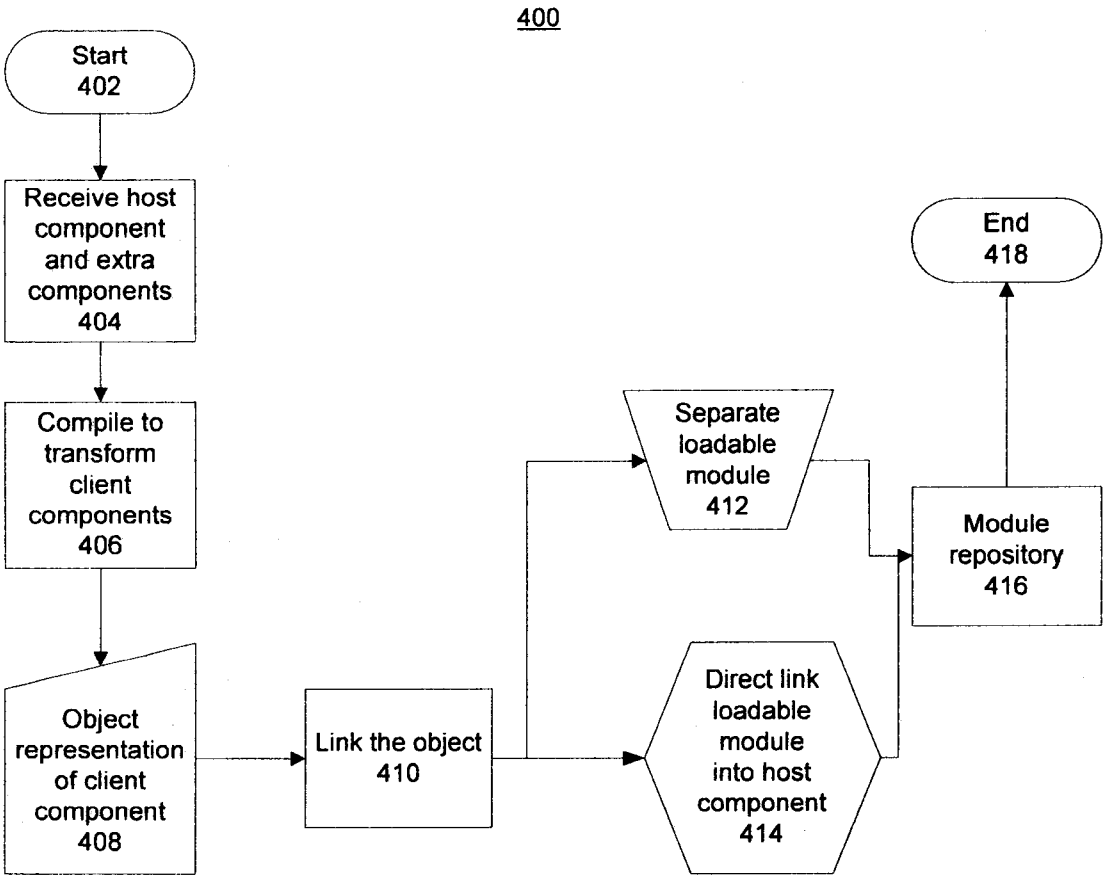
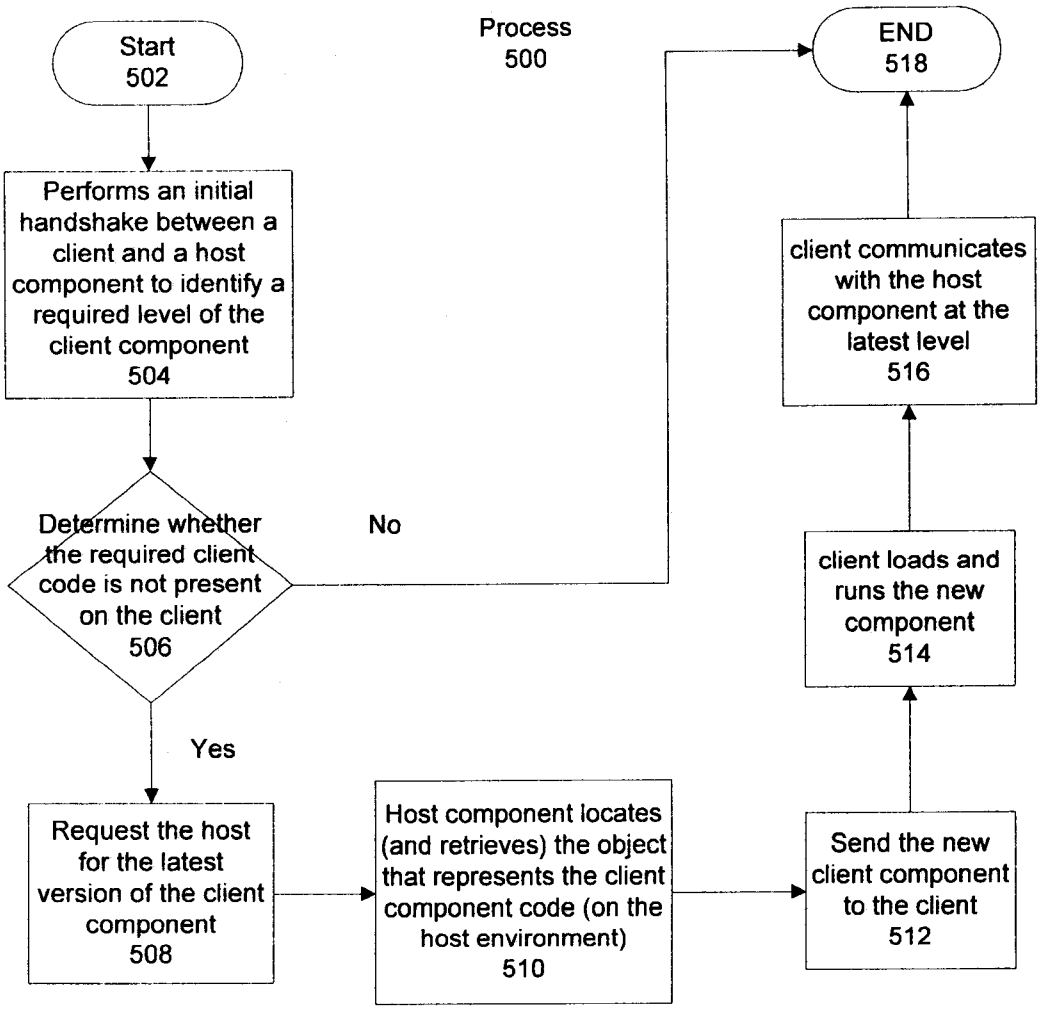
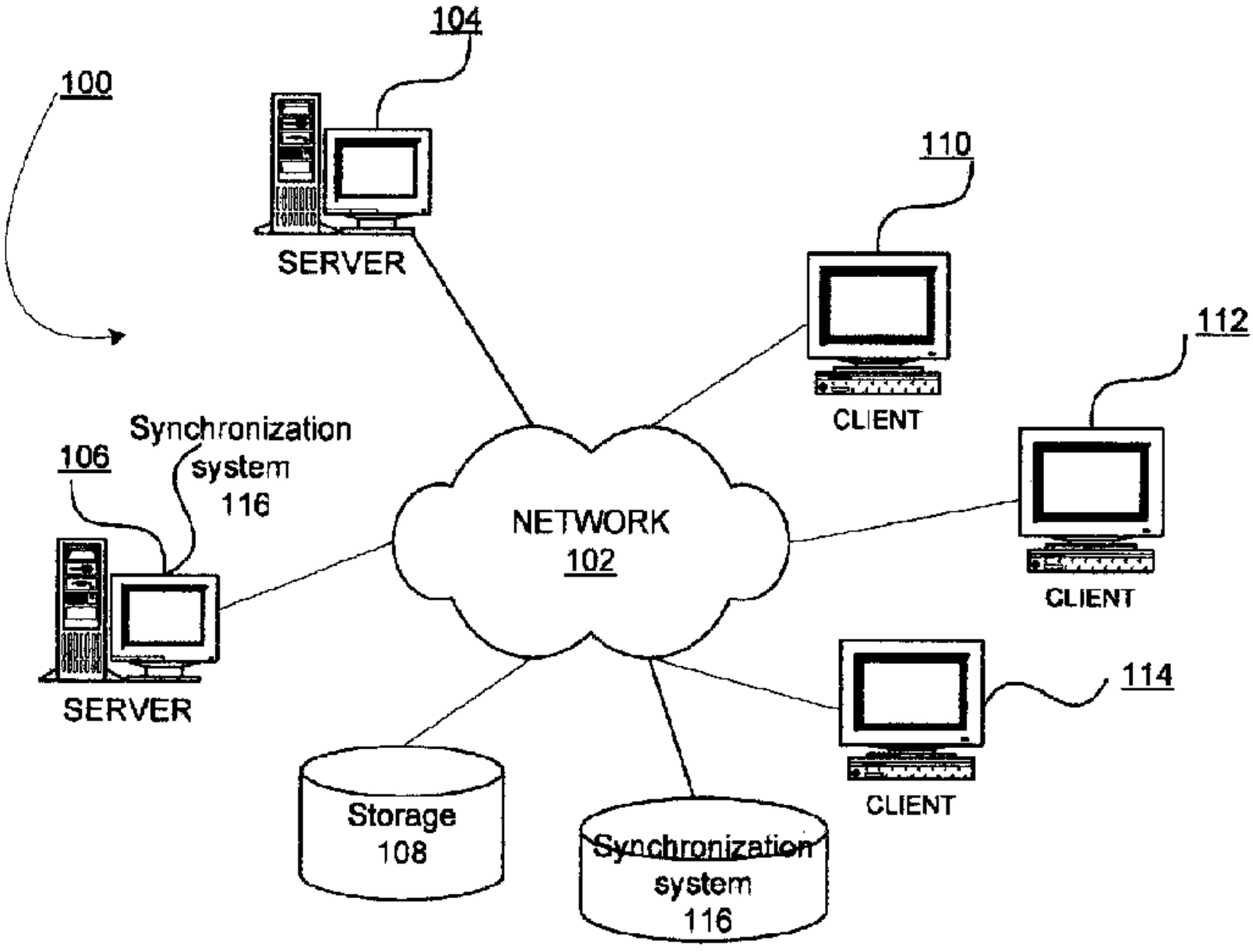

400
Start
402
Receive host component and extra components
404
Compile to transform client components
406
Object representation of client component
408
Link the object
410
Separate loadable module
412
Direct link loadable module into host component
414
Module repository
416
End
418

FIG. 5


Process
500
Start
502
Performs an initial handshake between a client and a host component to identify a required level of the client component
504
Determine whether the required client code is not present on the client
506
No
Yes
Request the host for the latest version of the client component
508
Host component locates (and retrieves) the object that represents the client component code (on the host environment)
510
Send the new client component to the client
512
client loads and runs the new component
514
client communicates with the host component at the latest level
516
END
518


100
104
SERVER
110
CLIENT
112
CLIENT
Synchronization
system
116
106
NETWORK
102
SERVER
114
CLIENT
Storage
108
Synchronization
system
116