US 20020054051A1

(19) **United States**

(12) **Patent Application Publication**

(10) **Pub. No.: US 2002/0054051 A1**

(43) **Pub. Date: May 9, 2002**

LADD

(54) **PARALLEL PROGRAMMING DEVELOPMENT ENVIRONMENT**

(76) Inventor: **PATRICK G. LADD**, SAN MARCOS, CA (US)

Correspondence Address:
JAMES M. STOVER
NCR CORPORATION
1700 SOUTH PATTERSON BLVD, WHQ4
DAYTON, OH 45479 (US)

(*) Notice: This is a publication of a continued prosecution application (CPA) filed under 37 CFR 1.53(d).

(21) Appl. No.: **09/222,482**

(22) Filed: **Dec. 29, 1998**

Publication Classification

(51) **Int. Cl.**[7] **G06F 13/00**

(52) **U.S. Cl.** **345/700**

(57) **ABSTRACT**

A method, apparatus, and article of manufacture for creating a parallel programming development environment. The environment comprises a graphical user interface, that contains a system screen, an application screen, a code generator, a process distributor, and an applications monitor. The system and application screens are displayed on the monitor and are used to display the topology of the computer system and for selecting portions of the system for use in a parallel application. The code generator receives a user application file from the application screen and generates programming code based on the contents of the user application file. The process distributor, distributes the executable code within the topology of the computer system as allocated by the user in the application file. The application monitor monitors the user application file and maintains statistics on the user application file.

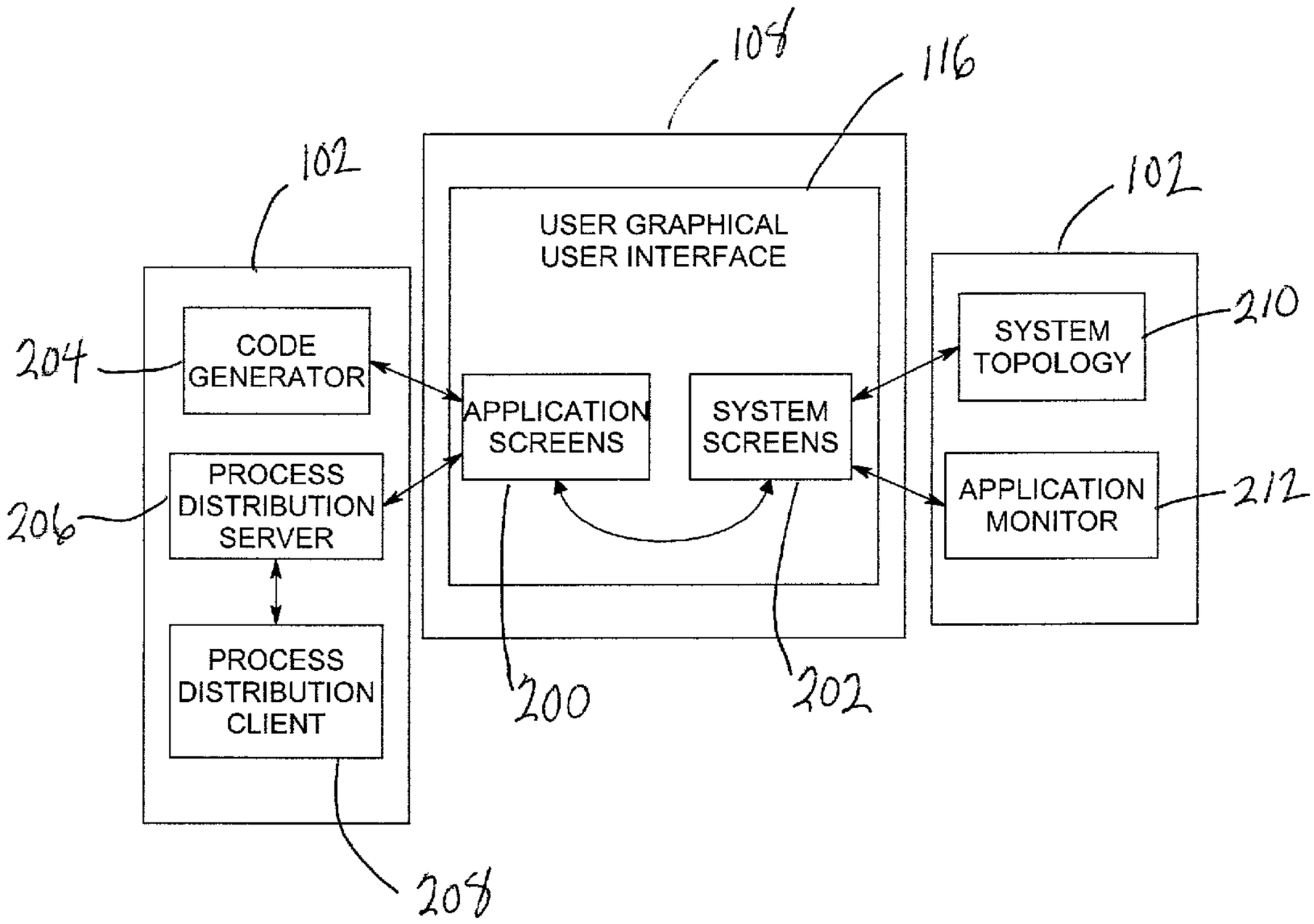
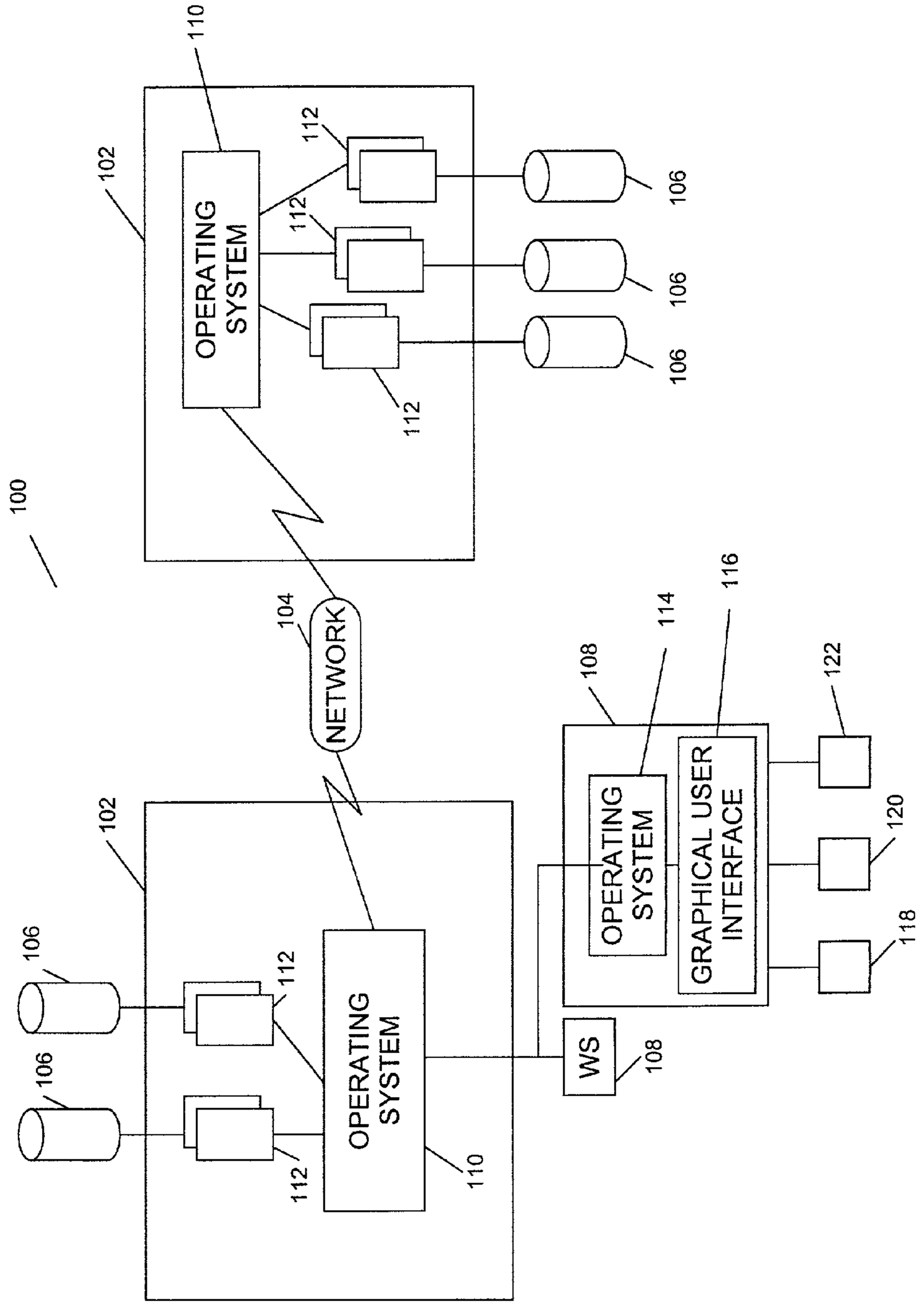


FIG. 1


100
102
110
OPERATING SYSTEM
112
106
104
NETWORK
108
114
116
122
GRAPHICAL USER INTERFACE
120
118
WS

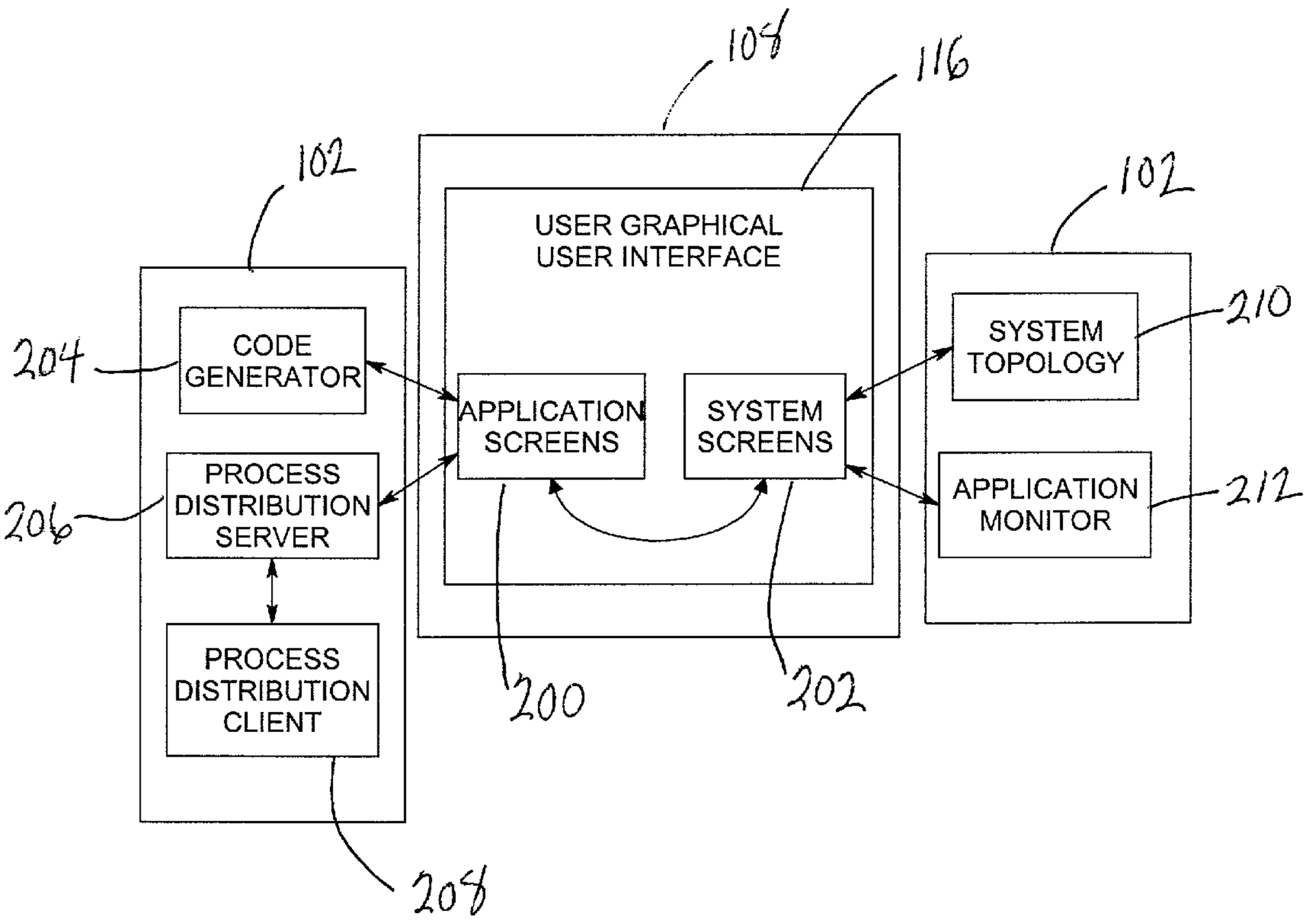
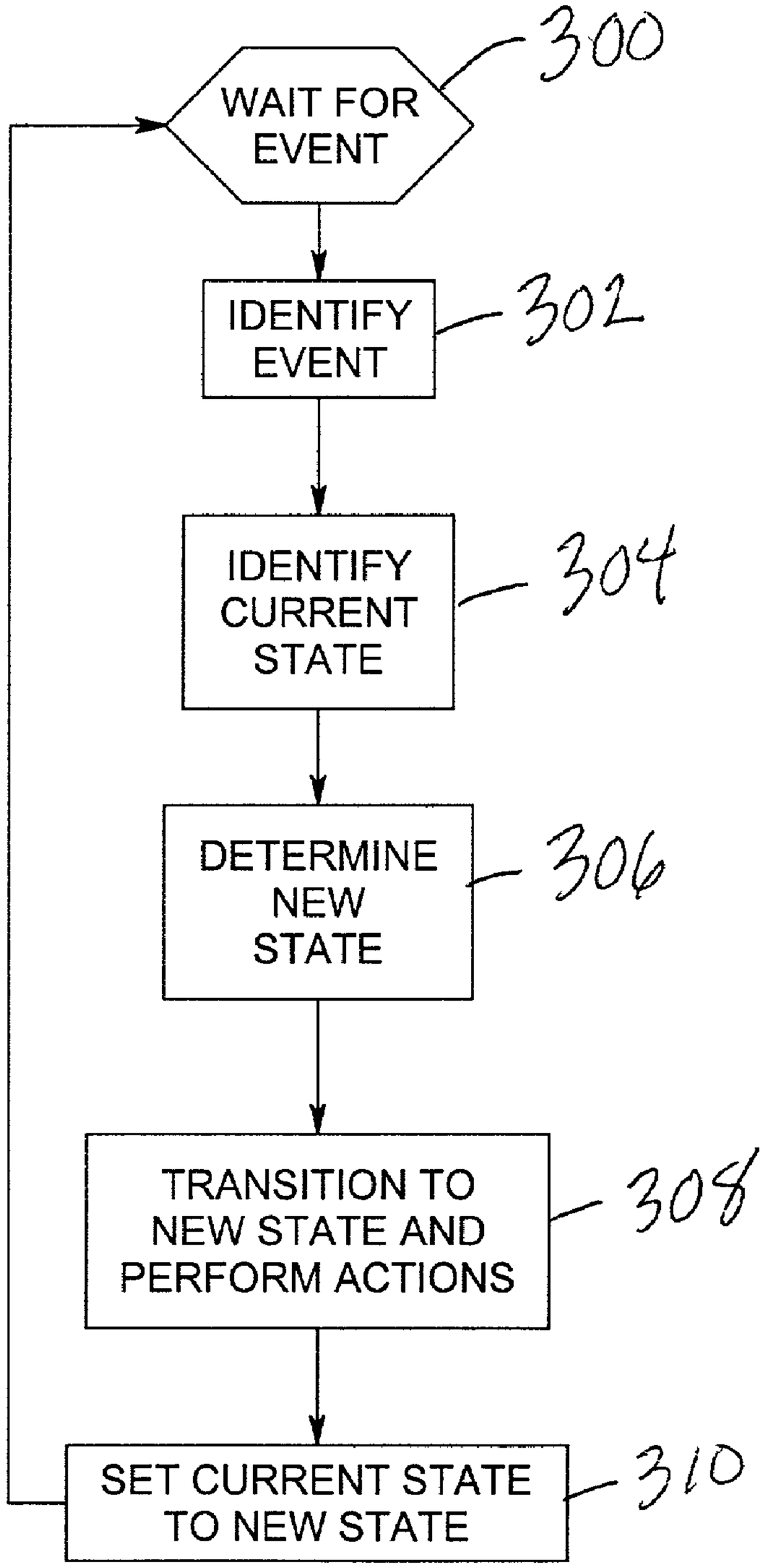

104
116
102
USER GRAPHICAL USER INTERFACE
102
CODE GENERATOR
204
PROCESS DISTRIBUTION SERVER
206
PROCESS DISTRIBUTION CLIENT
208
APPLICATION SCREENS
SYSTEM SCREENS
200
202
SYSTEM TOPOLOGY
210
APPLICATION MONITOR
212

FIG. 2

FIG. 3


WAIT FOR EVENT
300
IDENTIFY EVENT
302
IDENTIFY CURRENT STATE
304
DETERMINE NEW STATE
306
TRANSITION TO NEW STATE AND PERFORM ACTIONS
308
SET CURRENT STATE TO NEW STATE
310

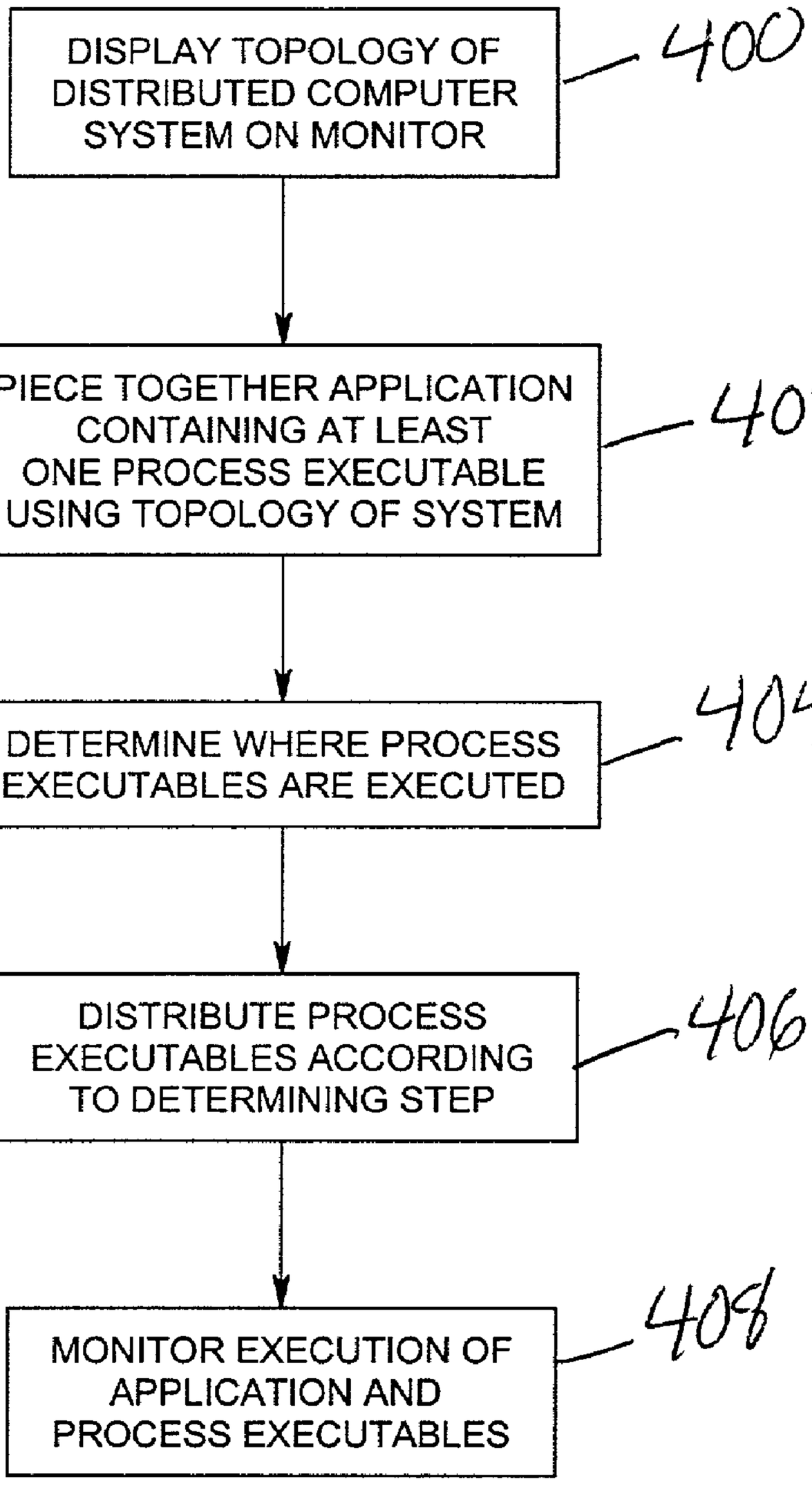

DISPLAY TOPOLOGY OF DISTRIBUTED COMPUTER SYSTEM ON MONITOR
400
PIECE TOGETHER APPLICATION CONTAINING AT LEAST ONE PROCESS EXECUTABLE USING TOPOLOGY OF SYSTEM
402
DETERMINE WHERE PROCESS EXECUTABLES ARE EXECUTED
404
DISTRIBUTE PROCESS EXECUTABLES ACCORDING TO DETERMINING STEP
406
MONITOR EXECUTION OF APPLICATION AND PROCESS EXECUTABLES
408

FIG. 4

PARALLEL PROGRAMMING DEVELOPMENT ENVIRONMENT

BACKGROUND OF THE INVENTION

[0001] 1. Field of the Invention

[0002] This invention relates in general to parallel processor computer systems, and in particular, to a parallel programming development environment used to program parallel processor computer systems.

[0003] 2. Description of Related Art

[0004] Parallel processor computer systems are frequently comprised of an operating system and arrays of individual computers (i.e., processor nodes), each with their own central processing unit (CPU), memory, and data storage unit. Tasks are executed in parallel by utilizing each processor node.

[0005] During the execution of a task, a body of work is divided into multiple threads. A thread is a stream of instructions executed by the computer on behalf of a task. As an analogy, a task such as an orchestra performing a symphony can be decomposed into many threads which would be the individual musicians, each playing their part.

[0006] Typically, in a parallel processor computer system, each thread is allocated to a different processor node. Each of these threads is then executed in parallel at their respective separate nodes. For instance, three threads can occupy and execute simultaneously on three different nodes at the same time.

[0007] Although parallel processing has merits, there are shortcomings. Conventional processing techniques may result in an inefficient use of the available hardware. Industry standard libraries, such as the Message Passing Interface (MPI) have made parallel programming even more difficult because the MPI standard is programming language dependent. This dependency creates problems for computer systems that perform some of their programming tasks in one language and other tasks in another language, because present day parallel programming efforts will then be unable to interact with programming that is written in two different programming languages. This makes parallel programming efforts more costly and more time consuming.

[0008] It can be seen, then, that there is a need in the art for a method to develop parallel programming that can be used with multiple computer programming languages. Further, there is a need for a parallel programming development environment that will be lest costly. There is also a need for a parallel programming development environment that is less time consuming. There is also a need in the art for modifications to conventional techniques that exploit the hardware available in parallel processor computer systems.

SUMMARY OF THE INVENTION

[0009] To overcome the limitations in the prior art described above, and to overcome other limitations that will become apparent upon reading and understanding the present specification, the present invention discloses a method, apparatus, and article of manufacture for creating a parallel programming development environment. The environment comprises a graphical user interface, that contains a system screen, an application screen, a code generator, a process distributor, and an applications monitor. The system and application screens are displayed on the monitor and are used to display the topology of the computer system and for selecting portions of the system for use in a parallel application. The code generator receives a user application file from the application screen and generates programming code based on the contents of the user application file. The process distributor, distributes the executable code within the topology of the computer system as allocated by the user in the application file. The application monitor monitors the user application file and maintains statistics on the user application file.

[0010] An object of the present invention is to provide more efficient usage of parallel processor computer systems. Another benefit is higher system availability without undue programing overhead in the application. Still another benefit of the present invention is its ability to provide faster and more cost effective parallel programming development.

[0011] These and various other advantages and features of novelty which characterize the invention are pointed out with particularity in the claims annexed hereto and form a part hereof. However, for a better understanding of the invention, its advantages, and the objects obtained by its use, reference should be made to the drawings which form a further part hereof, and to the accompanying detailed description, in which there is illustrated and described specific examples of a method, apparatus, and article of manufacture in accordance with the invention.

BRIEF DESCRIPTION OF THE DRAWINGS

[0012] Referring now to the drawings in which like reference numbers represent corresponding parts throughout:

[0013] **FIG. 1** is a block diagram that illustrates an exemplary hardware environment that could be used with the present invention;

[0014] **FIG. 2** illustrates the components and interrelationship between the components of the development environment of the present invention;

[0015] **FIG. 3** is a flowchart that illustrates the general logic of a message or event-driven node performing the steps of the present invention;

[0016] **FIG. 4** is a flowchart that illustrates exemplary logic performed by the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

[0017] In the following description of the preferred embodiment, reference is made to the accompanying drawings which form a part hereof, and in which is shown by way of illustration a specific embodiment in which the invention may be practiced. It is to be understood that other embodiments may be utilized and structural changes may be made without departing from the scope of the present invention.

[0018] Overview

[0019] The present invention discloses a method, apparatus, and article of manufacture for creating a parallel programing development environment. The environment comprises a graphical user interface, that contains a system screen, an application screen, a code generator, a process

distributor, and an applications monitor. The system and application screens are displayed on the monitor and are used to display the topology of the computer system and for selecting portions of the system for use in a parallel application. The code generator receives a user application file from the application screen and generates programming code based on the contents of the user application file. The process distributor distributes the executable code within the topology of the computer system as allocated by the user in the application file. The application monitor monitors the user application file and maintains statistics on the user application file.

[0020] The present invention provides a Graphical User Interface (GUI) that is used to display and create a graphical representation of the desired parallel programming application. The present invention also displays a diagram of the nodes of the system and how they are interconnected, so that programmers can take advantage of the architecture of the system that is being programmed. The present invention allows users to route processes and/or threads to specific nodes, determine the use of shared memory devices, and customize an application prior to generation of computer programming code.

[0021] The present invention provides a graphical interface to users and offers objects such as Groups, Links, and Processes, e.g., root and children, that can be dragged and dropped, connected, and customized in order to form a graphical representation of the desired parallel application. Additionally, on a system screen, the present invention can show a diagram of the nodes in the system and how they are interconnected e.g., distributed memory, networked, etc. Users can describe how the application should behave in the system in order to take advantage of the architecture shown on the system screen. This activity can entail the determination of process distribution to specific nodes, or how shared memory is to be used. The present invention saves this information and uses the information for code generation and for process distribution when the application is started.

[0022] When the overall application is described, individual processes are designed using object-oriented software design techniques with the addition of paradigm specific, e.g., MPI objects. A tool incorporating a modeling language can be used, but in order to maintain ease of parallel application programmability a library of paradigm specific objects must be made available.

[0023] With the application completely described, the user can generate language specific codes for a specific standardized parallel programming paradigm, such as MPI. The present invention can be used to start the application by distributing the processes and signal the start of their execution. Finally, the present invention can be used to monitor an application in order to gather statistics, displayed graphically in real-time, for correct operation and performance evaluation.

[0024] Hardware Environment

[0025] **FIG. 1** illustrates an exemplary computer hardware environment that could be used with the present invention. In the exemplary computer hardware environment, a computer system **100** is comprised of one or more processors or nodes **102** interconnected by a network **104**. Each of the nodes **102** is typically a symmetric multi-processor (SMP) architecture and is comprised of a plurality of microprocessors, random access memory (RAM), read-only memory (ROM), and other components. It is envisioned that attached to the nodes **102** may be one or more fixed and/or removable data storage units (DSUs) **106** and data communications units (DCUs).

[0026] Each of the nodes **102** operates under the control of an operating system **110**, such as the UNIX™ operating system. Further, each of the nodes **102** executes one or more computer programs **112** under the control of the operating system **110**. Generally, the operating system **110** and the computer programs **112** are tangibly embodied in and/or retrieved from RAM, ROM, and/or one or more other DSUs **106** or DCUs. Further, both the operating system **110** and the computer programs **112** are loaded into RAM for execution by the node **102**. In any embodiment, both the operating system **110** and the computer programs **112** comprise instructions which, when read and executed by the node **102**, causes the node **102** to perform the steps necessary to execute the steps or elements of the present invention.

[0027] In the exemplary environment of **FIG. 1, a** client-server architecture is disclosed. At least one of the nodes **102** provide the connection to client systems operating on workstations **108**. Operators of the system **100** use a workstation **108** or terminal to transmit electrical signals to and from server systems operating on the node **102** in the system **100**, wherein the electrical signals represent commands for performing various functions in the system **100**, such as search and retrieval functions against the databases. The present invention has application to any function or software that can be performed by a system **100**.

[0028] The workstation **108** usually operates under the control of an operating system **114**. The present invention is usually implemented in one or more Graphical User Interfaces (GUIs) **116** that operate under the control of and in conjunction with the operating system **114**.

[0029] For human interface with the workstation **108**, attached to the workstation **108** is a keyboard **118**, a mouse or other pointing device **120**, and a monitor **122**. The GUIs **116** are displayed on monitor **122** and the user can interact with the GUIs **116** by using the keyboard **118** and/or the pointing device **120** to command the workstation **108** to perform certain tasks.

[0030] The present invention uses the GUI **116** to help resolve the problem associated with tasks that do not easily lend themselves to being divided into parallel processing sub-tasks. Methods which require knowledge of the physical configuration of the system **100** typically present undesirable levels of software complexity and platform dependencies. The present invention minimizes the levels of software complexity by presenting the user with a graphical representation of the system **100** and allowing the user to program the system **100** in a graphical manner. Supporting this global knowledge requires heavy use of the network **104** between the nodes **102** of the system **100**.

[0031] Generally, the GUI **116** comprises instructions and/or data that are embodied in or retrievable from a computer-readable device, medium, or carrier, e.g., the data storage device **106**, a remote device coupled to the workstation **108** via the node **102**, etc. Moreover, these instructions and/or

data, when read, executed, and/or interpreted by the workstation **108** cause the workstation **108** to perform the steps necessary to implement and/or use the present invention.

[**0032**] Thus, the present invention may be implemented as a method, apparatus, or article of manufacture using standard programming and/or engineering techniques to produce software, firmware, hardware, or any combination thereof. The term "article of manufacture" (or alternatively, "computer program product") as used herein is intended to encompass a computer program accessible from any computer-readable device, carrier, or media. Many modifications may be made to this configuration without departing from the scope of the present invention.

[**0033**] Any combination of the above components, or any number of different components, including computer programs, peripherals, and other devices, may be used to implement the present invention, so long as similar functions are performed thereby.

[**0034**] Relationships and Operation

[**0035**] **FIG. 2** illustrates the components and interrelationship between the components of the development environment of the present invention.

[**0036**] The present invention comprises a user GUI **116**, which includes application screens **200** and system screens **202**. The user GUI **116** provides an interface for the user to create parallel applications **112**, generate code for the parallel applications **112**, distribute the parallel applications **112**, run the parallel applications **112**, and monitor the progress of the parallel applications **112**.

[**0037**] To perform all of the monitoring, running, distribution, creation, and generation tasks, the User GUI **116** is divided into at least two parts. The first part is the application screens **200**. The application screens **200** provide an environment for users to piece together a parallel application using objects that correspond to pieces of the application. Once a user has pieced together the objects using the application screens **200**, the application screens **200**, through the user GUI **116**, pass the user applications **112** to the code generator **204** and process distribution server **206**.

[**0038**] The system screens **202** comprise another part of the user GUI **116**, and display current system **100** topology in terms of nodes **102**, network **104** connections (e.g., clustered, MPP, SMP, etc.) and shared memory architectures. The system screens **202**, through the user GUI **116**, pass the system **100** topology information to the applications screens **200** for use in parallel application **112** creation.

[**0039**] The code generator **204** accepts application **112** information files for application **112** behavior from the application screens **200** and generates language specific code for a particular parallel programming paradigm, such as MPI. Many different computer languages, e.g., C, C++, Fortran, Cobol, etc., along with many different parallel programming paradigms, are supportable by the code generator **204**.

[**0040**] The process distribution server **206** distributes process executables to nodes **102** as defined by the system **100** architecture described by the user. The process distribution server **206** also sends processes and applications **112** to process distribution clients **208** on nodes **112** identified in application **112** groups. The process distribution server **206** also signals the process distribution client **208** to execute appropriate processes and/or applications **112**.

[**0041**] The system topology **210** updates the graphical representation of the system **100** interconnections and distributed memory layout upon user request. The system **100** information is used to determine the architecture model of the application **112** and allows for determination of the number of parallel processes to use and where to execute them (e.g., which node **102** or workstation **108**).

[**0042**] The application monitor **212** monitors the start, progress, and completion of a parallel application **112**, maintains statistics during the execution of the parallel application **112**, and calculates system **100** performance during and after the execution of application **112**. The application monitor **212** also has a screen within system screens **202** that shows real time activity and statistical updates of an application **112** during execution.

[**0043**] Logic of the Management Interface

[**0044**] **FIG. 3** is a flowchart that illustrates the general logic of a message or event-driven node **102** performing the steps of the present invention. In such a computer **102**, operations are performed when transitions are made, based upon the receipt of messages or events, from present or current states to new states.

[**0045**] Generally, the flowchart begins by waiting at block **300** for an event (e.g., a mouse button click). It should be appreciated that during this time, other operating system tasks, e.g., file, memory, and video tasks, etc., may also be carried out. When an event occurs, control passes to block **302** to identify the event. Based upon the event, as well as the current state of the system determined in block **304**, a new state is determined in block **306**. In block **308**, the logic transitions to the new state and performs any actions required for the transition. In block **310**, the current state is set to the previously determined new state, and control returns to block **300** to wait for more input events.

[**0046**] The specific operations that are performed by block **308** when transitioning between states will vary depending upon the current state and the event. The various operations required to implement and maintain the workstation **108** and GUI **116** of the present invention represent particular events handled by the logic. However, it should be appreciated that these operations represent merely a subset of all of the events handled by the node **102**.

[**0047**] **FIG. 4** is a flowchart that illustrates exemplary logic performed by the present invention.

[**0048**] Block **400** represents the workstation **108** displaying a topology of the distributed computer system **100** on a monitor **122**.

[**0049**] Block **402** represents piecing together an application **112** containing at least one process executable using the topology of the distributed computer system **100**.

[**0050**] Block **404** represents determining where, within the topology of the distributed computer system **100**, the process executables are executed.

[**0051**] Block **406** represents the node **102** distributing the process executables according to the determining step.

[0052] Block **408** represents the node **102** monitoring execution of the pieced together application **112** and the process executables.

[0053] Conclusion

[0054] This concludes the description of the preferred embodiment of the invention. The following describes some alternative embodiments for accomplishing the present invention. For example, any type of computer, such as a mainframe, minicomputer, or personal computer, could be used with the present invention. In addition, any software program utilizing (either partially or entirely) object-oriented programming or a parallel processing schema could benefit from the present invention.

[0055] In summary, the present invention discloses a method, apparatus, and article of manufacture for creating a parallel programming development environment. The environment comprises a graphical user interface that contains a system screen, an application screen, a code generator, a process distributor, and an applications monitor. The system and application screens are displayed on the monitor and are used to display the topology of the computer system and for selecting portions of the system for use in a parallel application. The code generator receives a user application file from the application screen and generates programming code based on the contents of the user application file. The process distributor, distributes the executable code within the topology of the computer system as allocated by the user in the application file. The application monitor monitors the user application file and maintains statistics on the user application file.

[0056] The foregoing description of the preferred embodiment of the invention has been presented for the purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise form disclosed. Many modifications and variations are possible in light of the above teaching. It is intended that the scope of the invention be limited not by this detailed description, but rather by the claims appended hereto.

What is claimed is:

1. A graphical user interface for a parallel programming development environment on a computer system, comprising:

a system screen, displayed on a monitor attached to the computer system, for displaying a topology of the computer system;

an application screen, displayed on the monitor attached to the computer system, for receiving the topology of the computer system and for piecing together a parallel application, the parallel application including at least one process executable;

a code generator, for receiving the pieced together parallel application from the application screen and for generating programming code;

a process distributor, for distributing at least one process executable within the topology of the computer system; and

an application monitor, for monitoring the pieced together application.

2. The graphical user interface of claim 1, wherein the application monitor maintains statistics on an execution of the pieced together application.

3. The graphical user interface of claim 1, wherein the application monitor calculates performance statistics on an execution of the user application file.

4. The method of claim 1, wherein the application screen pieces together the parallel application using at least one object, wherein each object corresponds to a piece of the application.

5. The method of claim 1, wherein the code generator generates code for a particular parallel programming paradigm, such as MPI.

6. The method of claim 1, wherein the code generator generates language specific code.

7. The method of claim 1, wherein the process distributor distributes the process executable to a node of the computer system.

8. A method for developing parallel programming on a distributed computer system, comprising the steps of:

displaying a topology of the distributed computer system on a monitor;

piecing together an application containing at least one process executable using the topology of the distributed computer system;

determining where, within the topology of the distributed computer system, the process executables are executed;

distributing the process executables according to the determining step; and

monitoring execution of the pieced together application and the process executables.

9. A program storage device, readable by a computer, tangibly embodying one or more programs of instructions executable by the computer to perform method steps of developing parallel programming on a distributed computer system, comprising the steps of:

displaying a topology of the distributed computer system on a monitor;

piecing together an application containing at least one process executable using the topology of the distributed computer system;

determining where, within the topology of the distributed computer system, the process executables are executed;

distributing the process executables according to the determining step; and

monitoring execution of the pieced together application and the process executables.

* * * * *