US 20030051029A1

(19) United States

(12) Patent Application Publication (10) Pub. No.: US 2003/0051029 A1

Reedy et al. (43) Pub. Date: Mar. 13, 2003

(54) DYNAMIC PROVISIONING OF SEVICE COMPONENTS IN A DISTRIBUTED SYSTEM

(76) Inventors: **Dennis G. Reedy**, Manassas, VA (US); **James B. Clarke**, Sanford, FL (US)

Correspondence Address:
FINNEGAN, HENDERSON, FARABOW, GARRETT & BUNNER L.L.P.
1300 I Street, N.W.
Washington, DC 20005 (US)

(21) Appl. No.: **09/947,528**

(22) Filed: **Sep. 7, 2001**

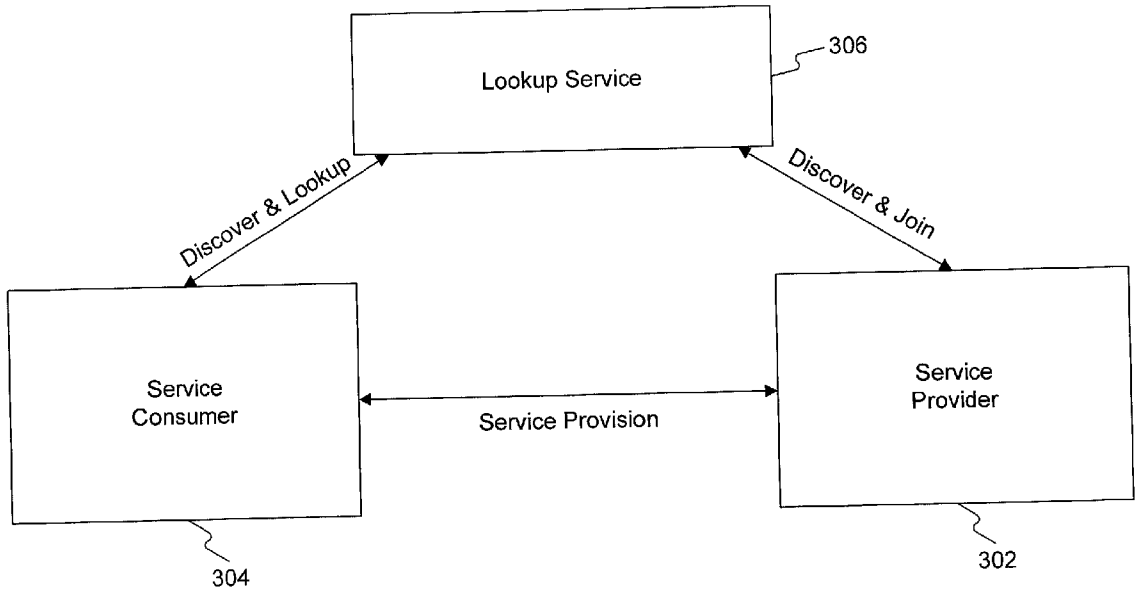
Publication Classification

(51) **Int. Cl.**[7] **G06F 15/173**

(52) **U.S. Cl.** **709/224**

(57) **ABSTRACT**

Systems and methods simplify the provision of complex services over a distributed network by breaking a complex service into a collection of simpler services. Systems and methods provide the tools to deconstruct a complex, provision service elements that are needed to make up the complex service, and monitor the service elements to ensure that the complex service is supported. Quality of service is provided by matching service requirements to compute resource capabilities.



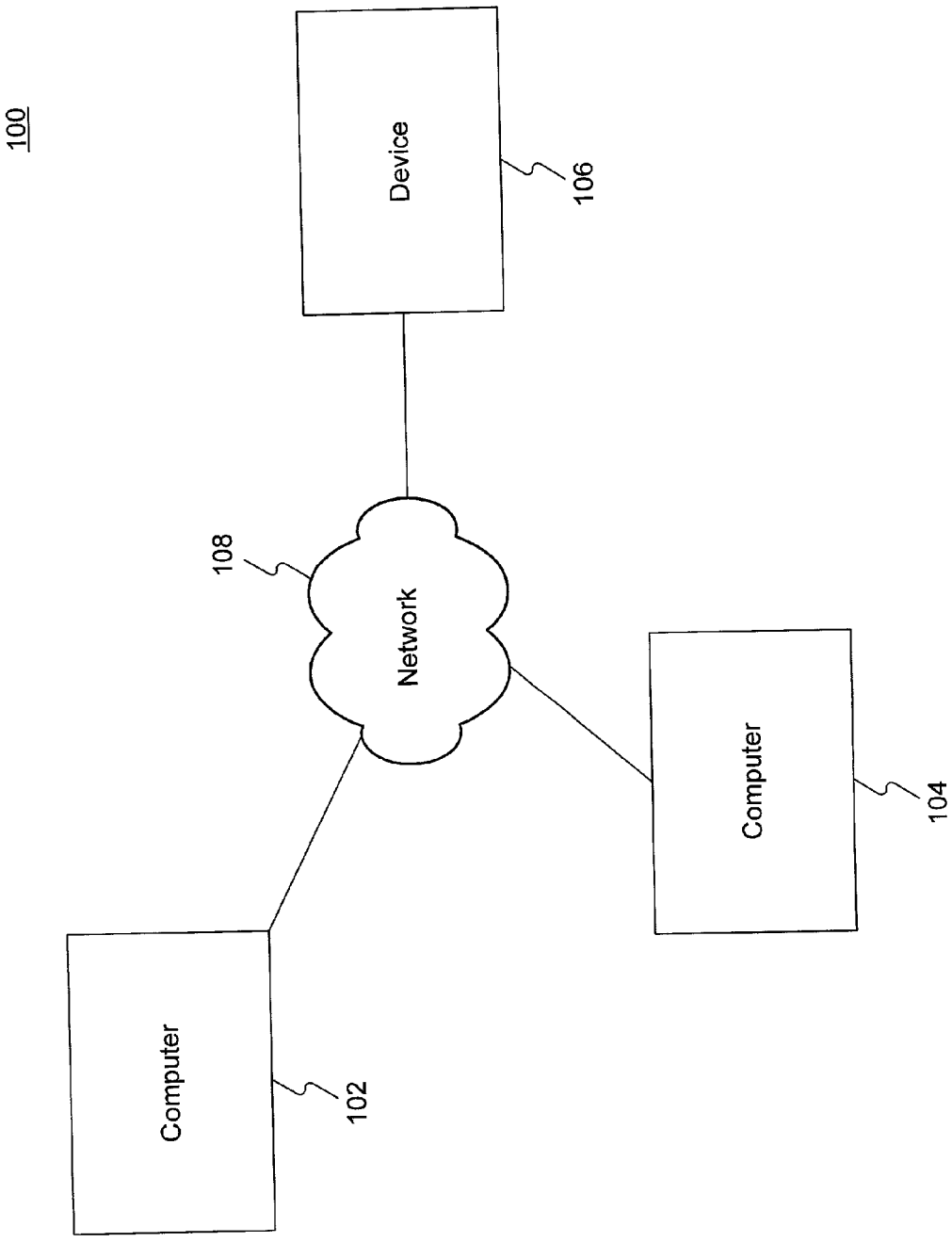


Fig. 1

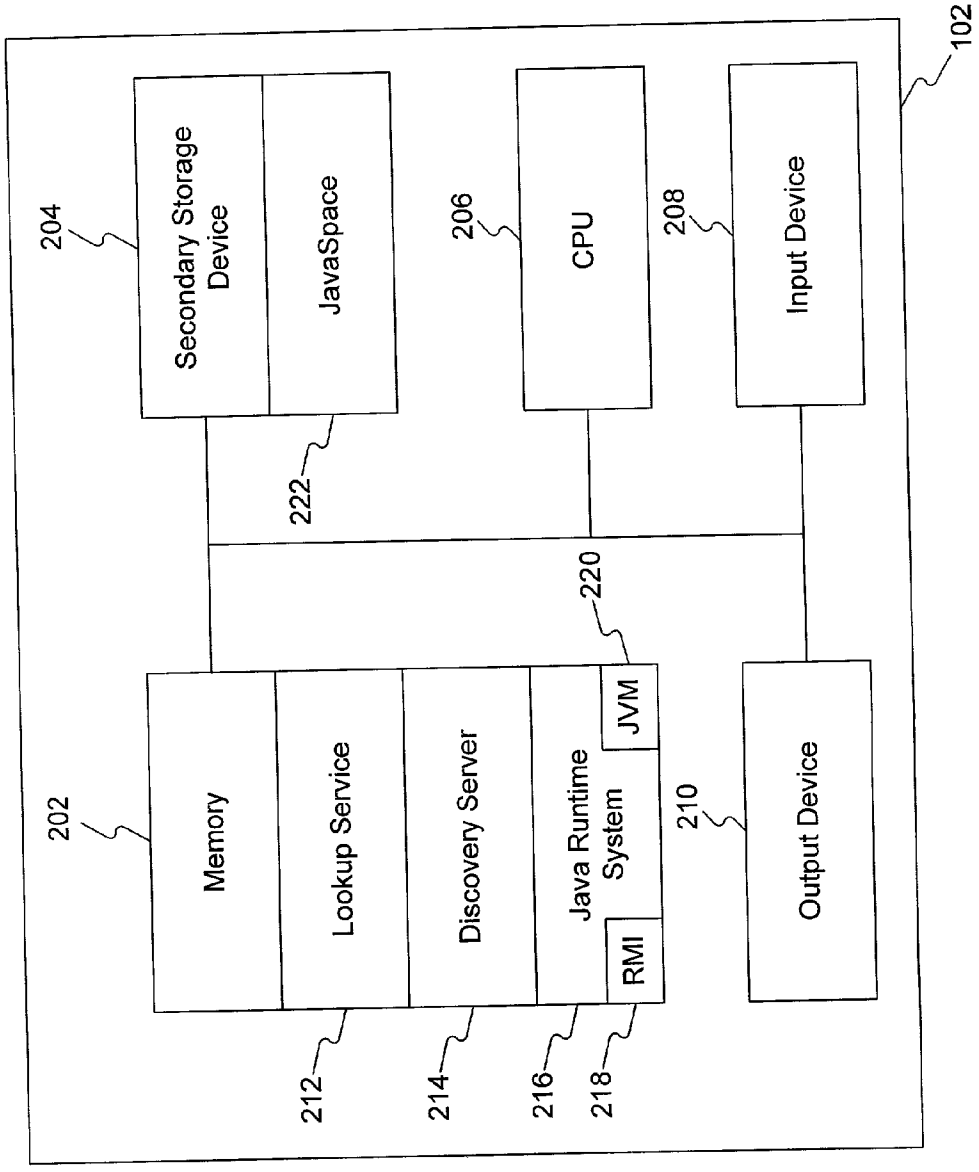

102
204
Secondary Storage Device
JavaSpace
206
CPU
208
Input Device
222
220
202
Memory
Lookup Service
Discovery Server
Java Runtime System
JVM
RMI
210
Output Device
212
214
216
218

Fig. 2

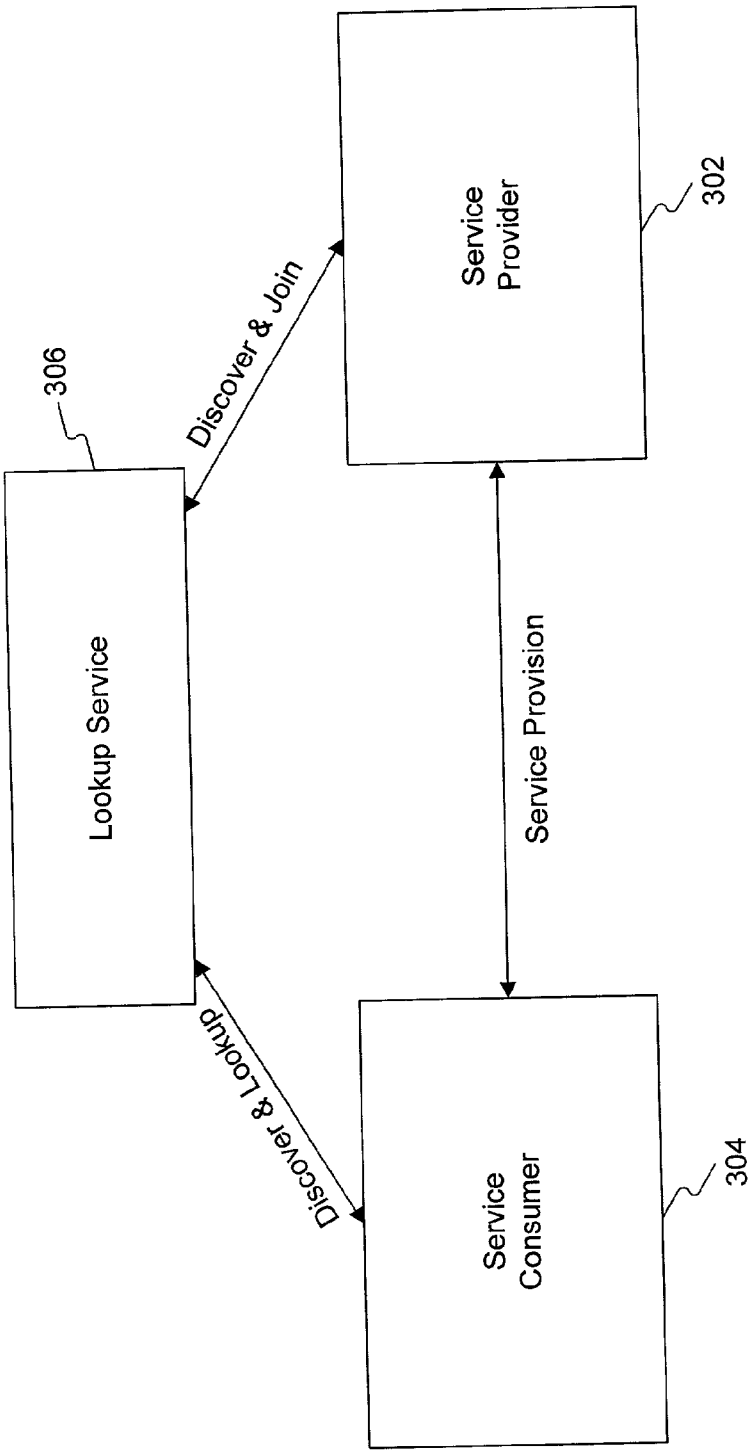

Service Provider
302
Discover & Join
306
Lookup Service
Service Provision
Discover & Lookup
Service Consumer
304

Fig. 3

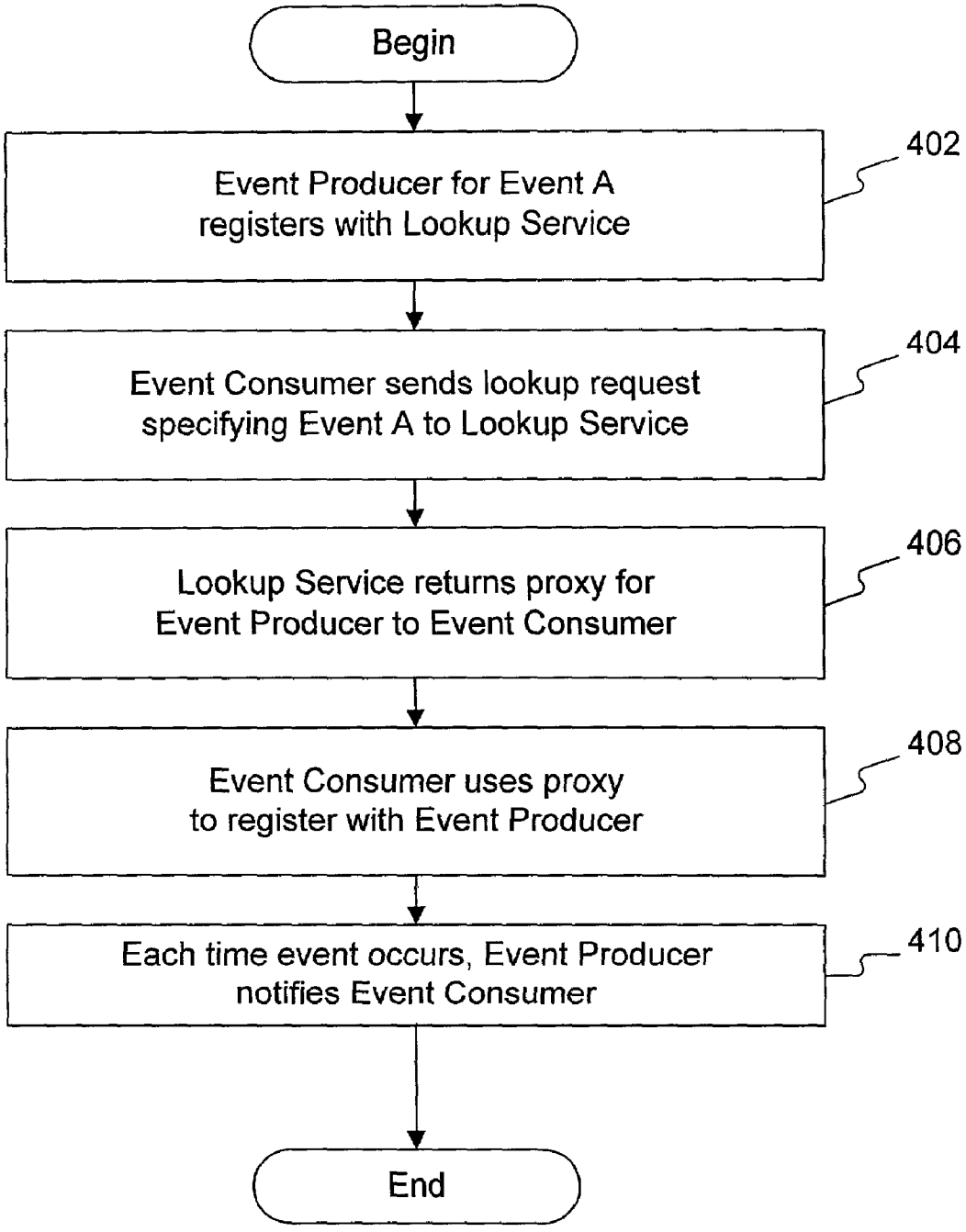

Begin
Event Producer for Event A registers with Lookup Service
402
Event Consumer sends lookup request specifying Event A to Lookup Service
404
Lookup Service returns proxy for Event Producer to Event Consumer
406
Event Consumer uses proxy to register with Event Producer
408
Each time event occurs, Event Producer notifies Event Consumer
410
End

Fig. 4

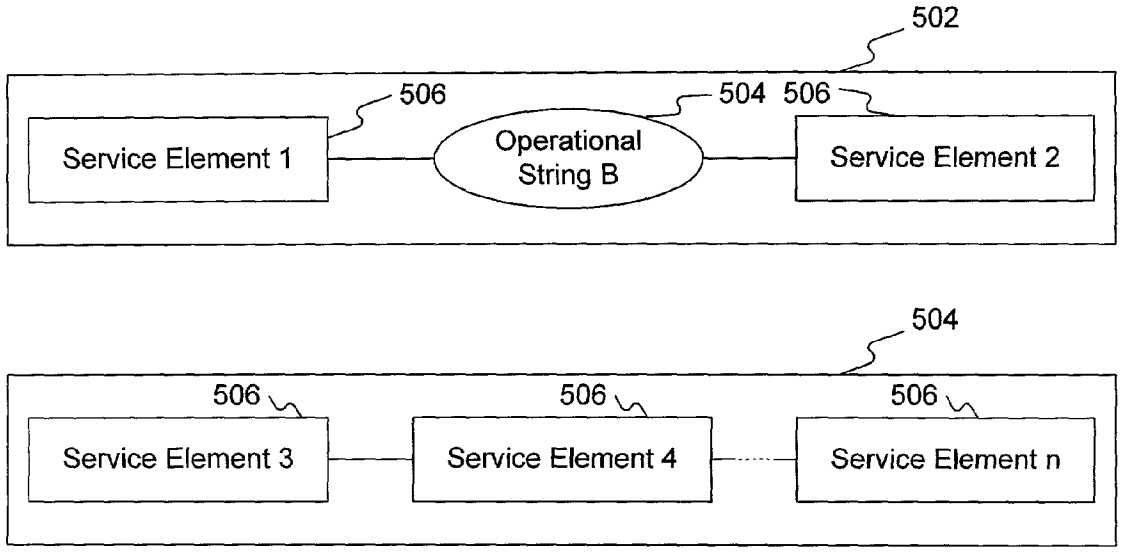

502
506
504
506
Service Element 1
Operational String B
Service Element 2
504
506
506
506
Service Element 3
Service Element 4
Service Element n

Fig. 5

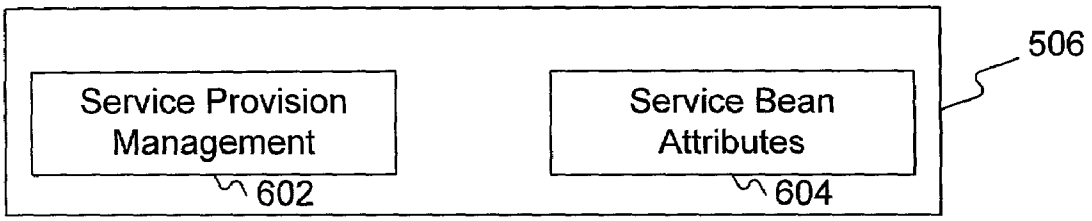

Service Provision Management
602
Service Bean Attributes
604
506

Fig.6

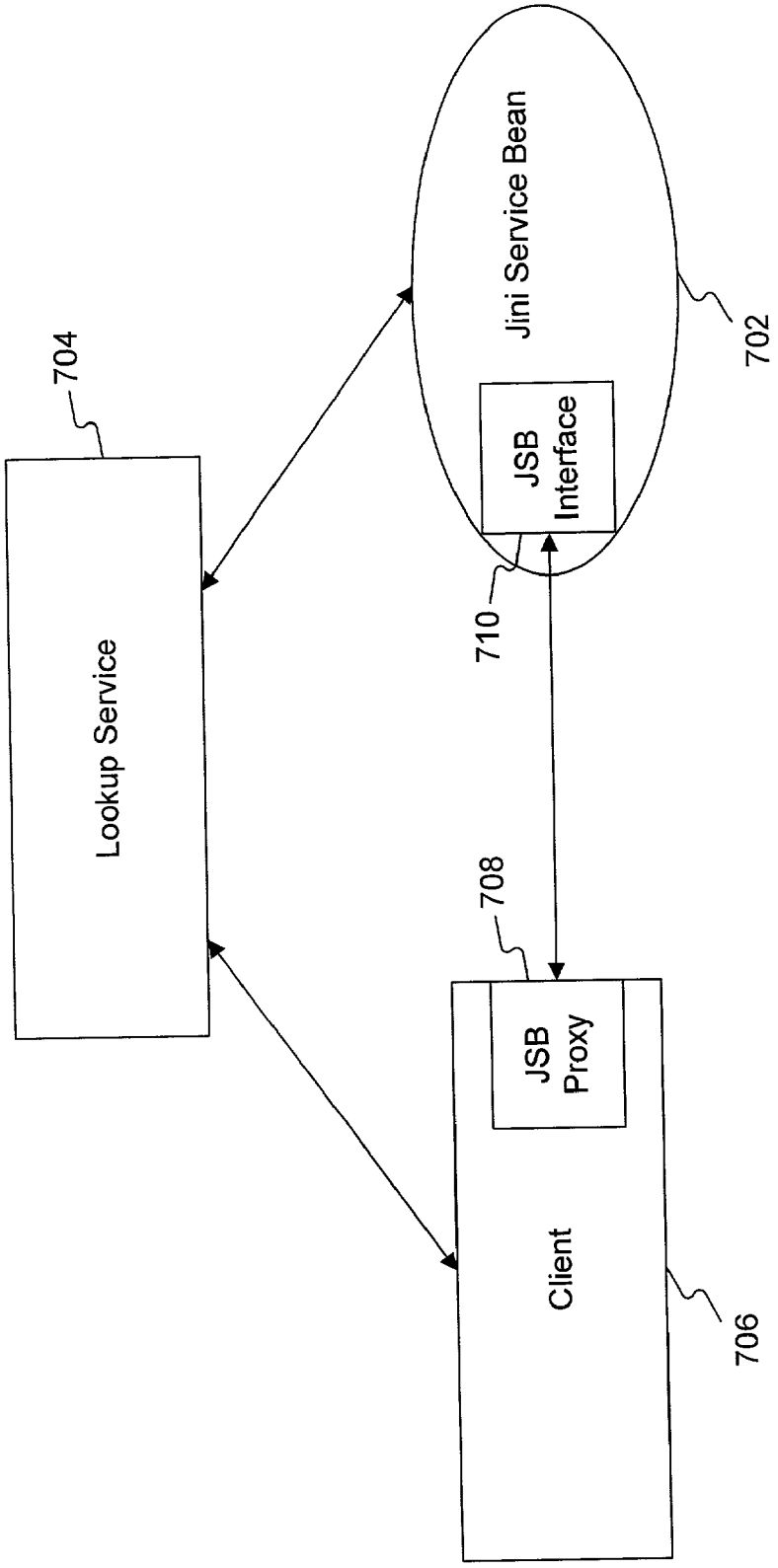

Jini Service Bean
702
JSB Interface
710
704
Lookup Service
708
JSB Proxy
Client
706

Fig. 7

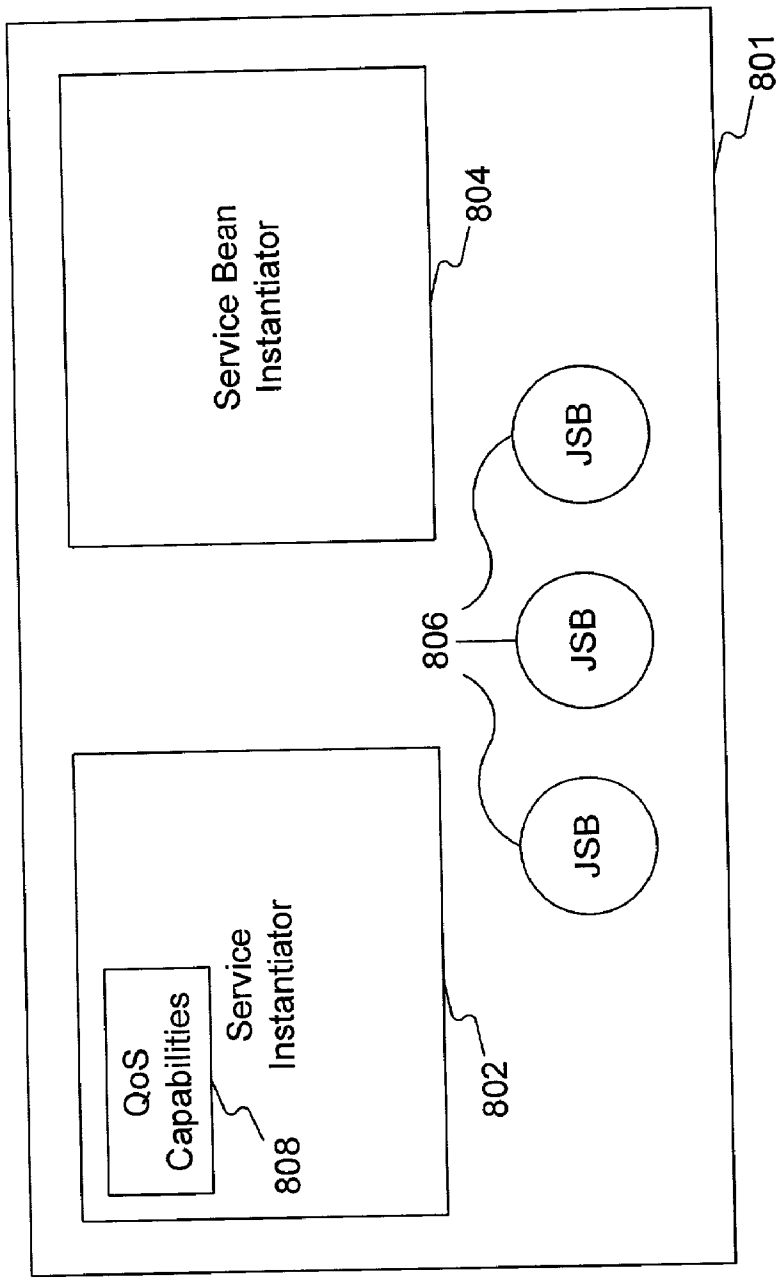

801
Service Bean Instantiator
804
806
JSB
JSB
JSB
QoS Capabilities
Service Instantiator
808
802

Fig. 8

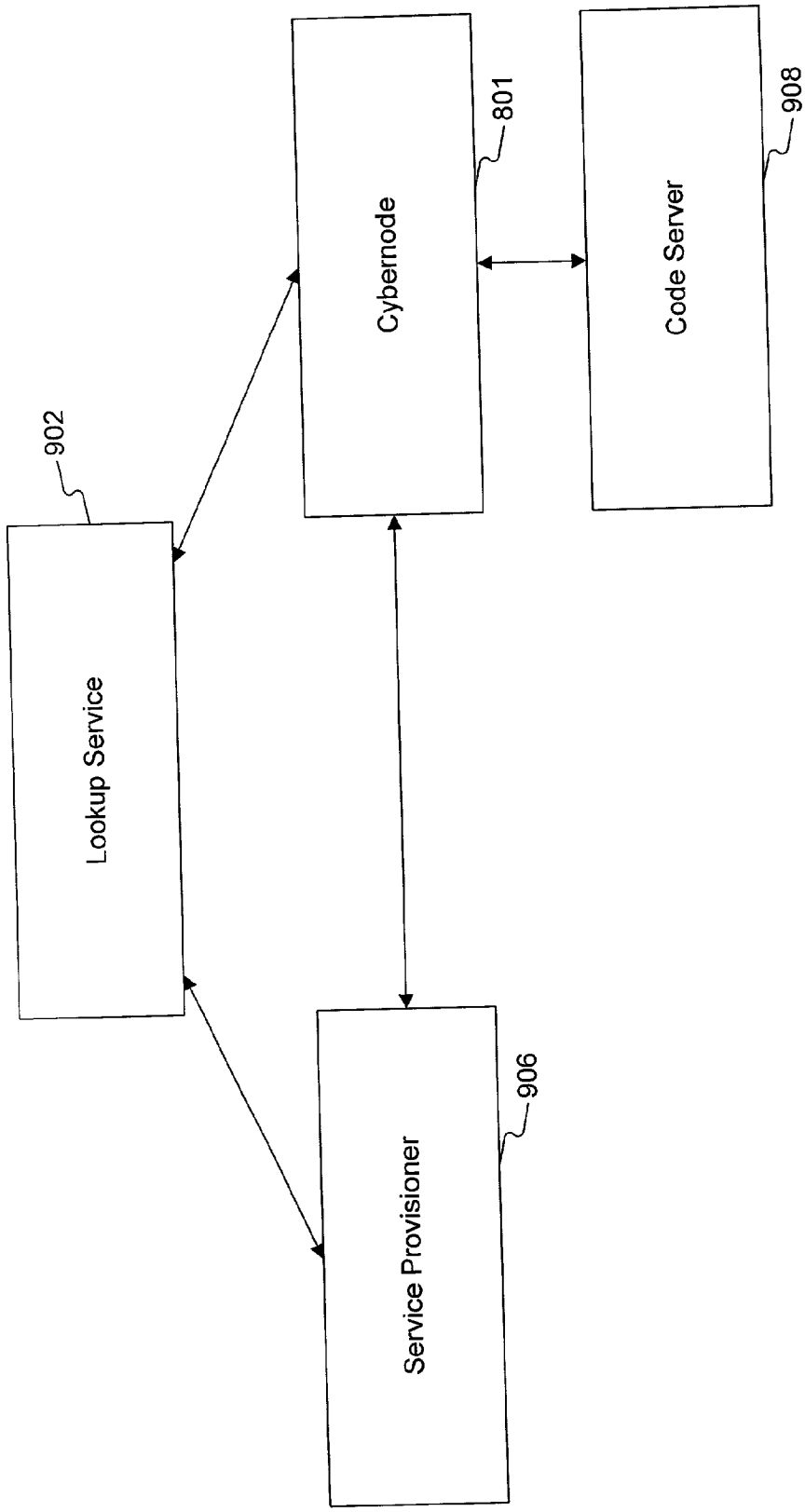

Cybernode
801
Code Server
908
902
Lookup Service
Service Provisioner
906

Fig. 9

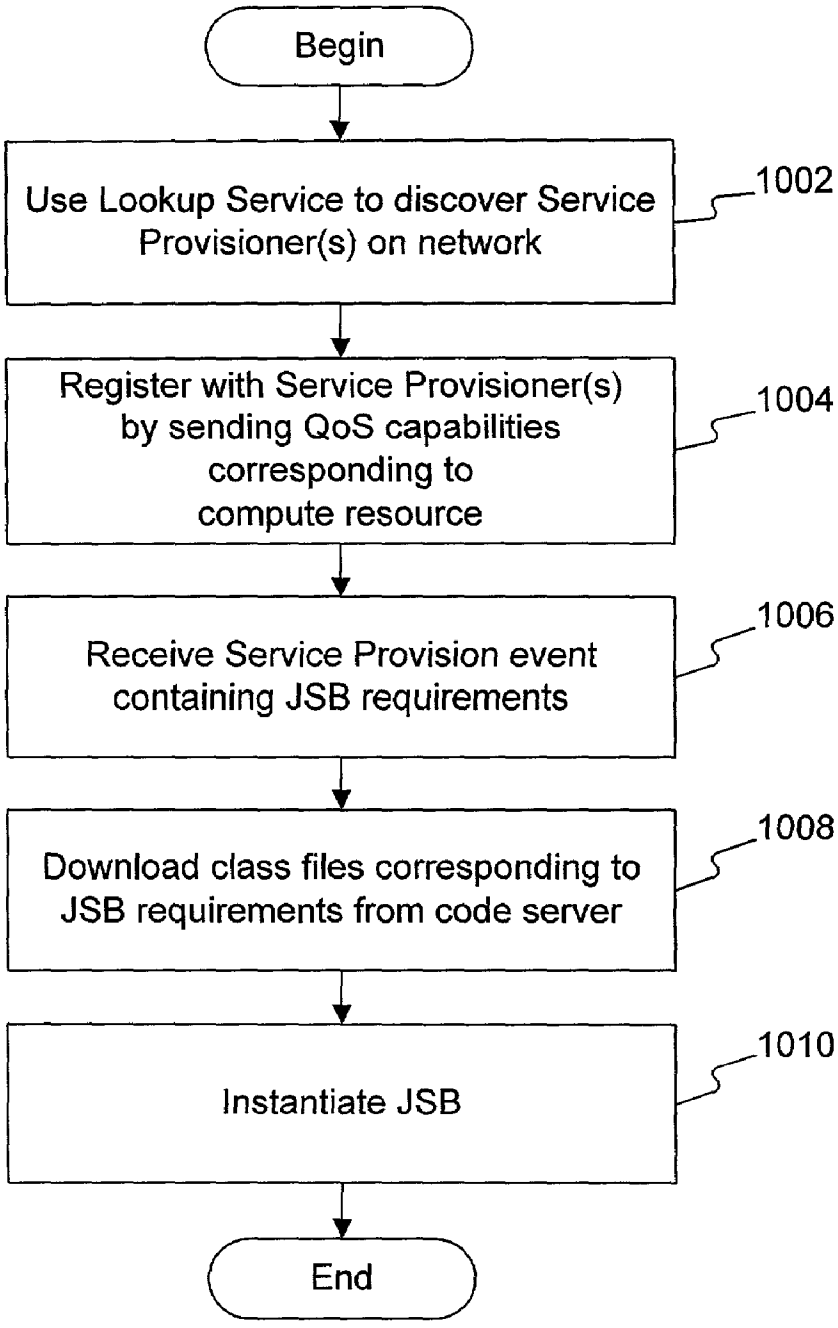

Begin
Use Lookup Service to discover Service Provisioner(s) on network
1002
Register with Service Provisioner(s) by sending QoS capabilities corresponding to compute resource
1004
Receive Service Provision event containing JSB requirements
1006
Download class files corresponding to JSB requirements from code server
1008
Instantiate JSB
1010
End

Fig. 10

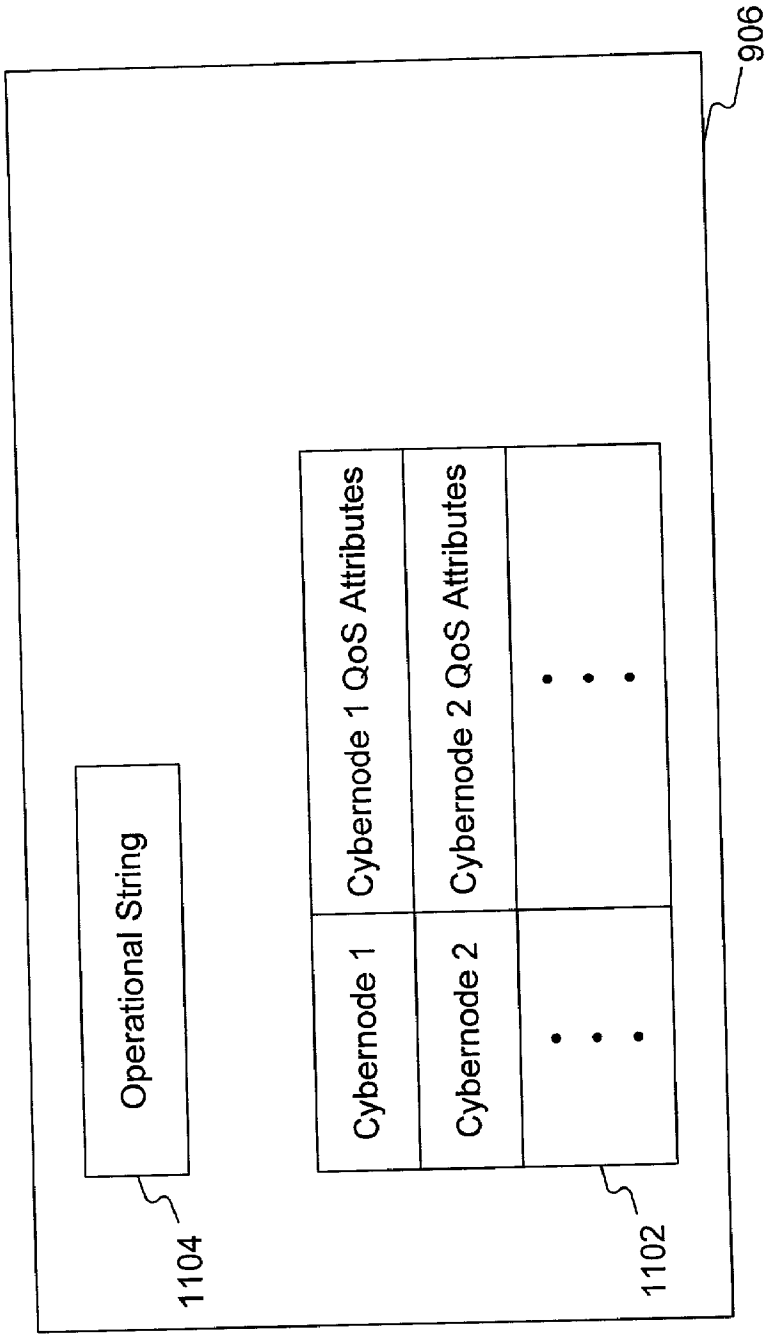

906
Operational String
1104
Cybernode 1
Cybernode 1 QoS Attributes
Cybernode 2
Cybernode 2 QoS Attributes
1102

Fig. 11

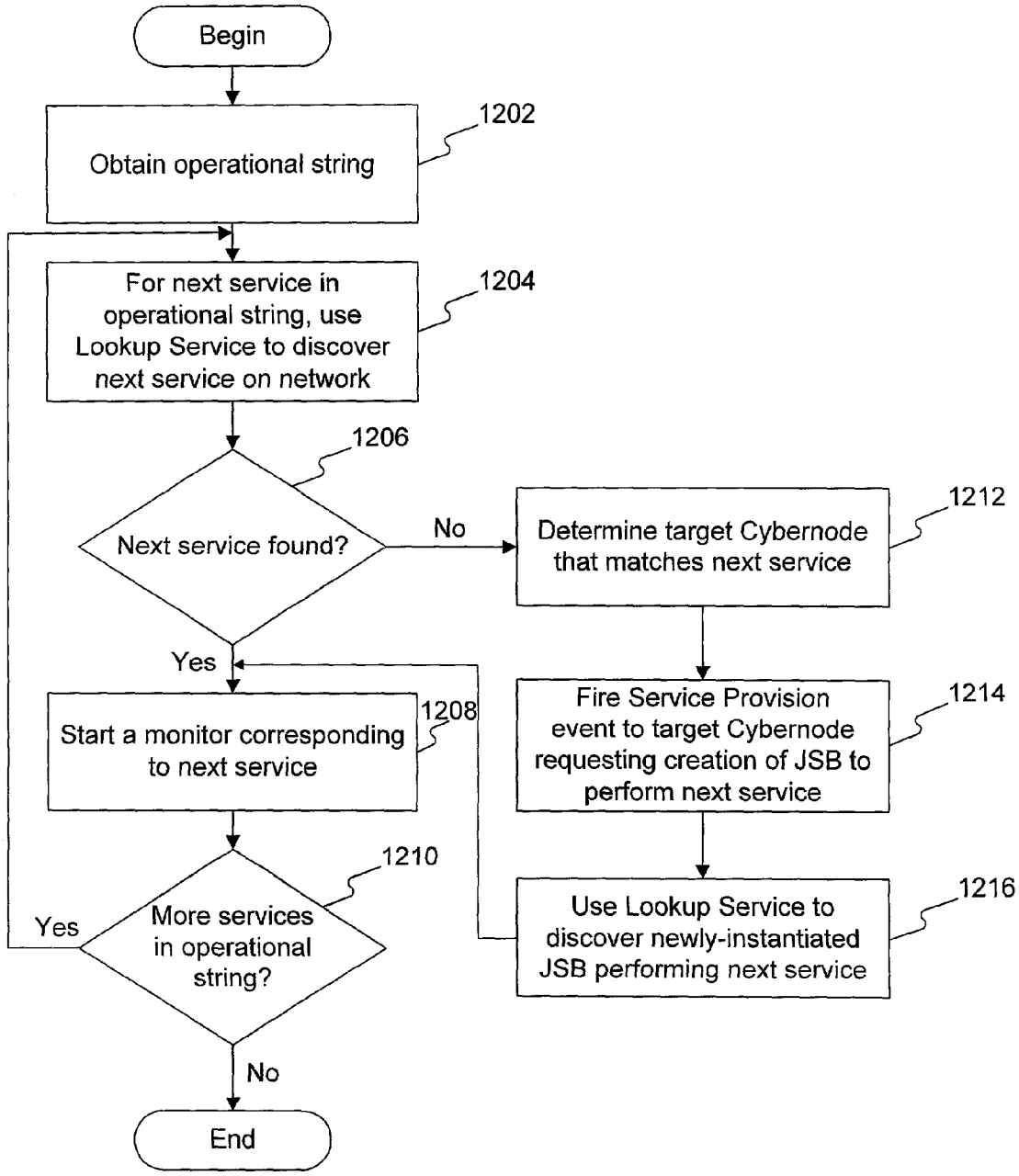

Begin
Obtain operational string
1202
For next service in operational string, use Lookup Service to discover next service on network
1204
1206
Next service found?
No
Determine target Cybernode that matches next service
1212
Yes
Start a monitor corresponding to next service
1208
Fire Service Provision event to target Cybernode requesting creation of JSB to perform next service
1214
1210
More services in operational string?
Yes
Use Lookup Service to discover newly-instantiated JSB performing next service
1216
No
End

Fig. 12

DYNAMIC PROVISIONING OF SEVICE COMPONENTS IN A DISTRIBUTED SYSTEM

RELATED APPLICATIONS

[0001] This application is related to an application for Distributed Metric Discovery and Collection in a Distributed System, attorney docket no. 06502.0384, filed Sep. 7, 2001, which is relied upon and incorporated by reference.

FIELD OF THE INVENTION

[0002] This invention relates to providing services in a distributed system and, more particularly, to methods and systems for dynamically provisioning services according to software requirements and compute resource capabilities.

BACKGROUND OF THE INVENTION

[0003] Distributed systems today enable a device connected to a communications network to take advantage of services available on other devices located throughout the network. Each device in a distributed system may have its own internal data types, its own address alignment rules, and its own operating system. To enable such heterogeneous devices to communicate and interact successfully, developers of distributed systems can employ a remote procedure call (RPC) communication mechanism.

[0004] RPC mechanisms provide communication between processes (e.g., programs, applets, etc.) running on the same device or different devices. In a simple case, one process, i.e., a client, sends a message to another process, i.e., a server. The server processes the message and, in some cases, returns a response to the client. In many systems, the client and server do not have to be synchronized. That is, the client may transmit the message and then begin a new activity, or the server may buffer the incoming message until the server is ready to process the message.

[0005] The Java™ programming language is an object-oriented programming language that may be used to implement such a distributed system. The Java™ language is compiled into a platform-independent format, using a bytecode instruction set, which can be executed on any platform supporting the Java™ virtual machine (JVM). The JVM may be implemented on any type of platform, greatly increasing the ease with which heterogeneous machines can be federated into a distributed system.

[0006] The Jini™ architecture has been developed using the Java™ programming language to enable devices in a distributed to share services using remote method invocation (RMI). Traditional Jini™ systems use RMI to enable a client device to request and receive a service provided by a server device on a remote machine. While conventional Jini™ systems provide a basic architecture for providing services in a distributed system, they do not provide tools specifically directed to providing complex services. Current systems do not address provisioning a service, such as application software, to make it available to the distributed system in the first place. Furthermore, conventional systems do not consider the requirements of a specific service before provisioning the service to make it available in the distributed system.

SUMMARY OF THE INVENTION

[0007] Methods and systems consistent with the present invention provide a service in a distributed system, the service consisting of a collection of service elements. It is determined whether an instance of each service element in the collection is running in the distributed system. For each service element in the collection that does not have an instance running in the distributed system, a new service element instance is created.

[0008] In accordance with an aspect of the invention, a system facilitates providing a service in a distributed system. A list of service elements that together constitute the service is received. For each service element in the list, it is discovered whether an application corresponding to the service element is running in the distributed system. The application corresponding to the service element is created, if the application corresponding to the service element is not running in the distributed system. The application is monitored to detect whether the application fails.

[0009] According to the present invention, a method handles events in a distributed system. A capability to notify members of the distributed system when an event of a predetermined type occurs is advertised. Registration requests are received from a plurality of event consumers, the registration requests specifying the predetermined type. When an event of the predetermined type occurs, an order is determined in which to notify each of the plurality of event consumers, and an event notification is sent to each of the plurality of event consumers in the order determined.

[0010] Additional features of the invention will be set forth in part in the description which follows, and in part will be obvious from the description, or may be learned by practice of the invention.

BRIEF DESCRIPTION OF THE DRAWINGS

[0011] It is to be understood that both the foregoing general description and the following detailed description are exemplary and explanatory only and are not restrictive of the invention, as claimed. The accompanying drawings, which are incorporated in and constitute a part of this specification, illustrate several embodiments of the invention and together with the description, serve to explain the principles of the invention. In the drawings:

[0012] **FIG. 1** is a high level block diagram of an exemplary system for practicing systems and methods consistent with the present invention;

[0013] **FIG. 2** depicts a computer in greater detail to show a number of the software components of an exemplary distributed system consistent with the present invention;

[0014] **FIG. 3** depicts an embodiment of the discovery process in more detail, in accordance with the present invention;

[0015] **FIG. 4** is a flow chart of an embodiment of the event handling process, in accordance with the present invention;

[0016] **FIG. 5** is a block diagram of an exemplary operational string, in accordance with the present invention;

[0017] **FIG. 6** is a block diagram of an exemplary service element, in accordance with the present invention;

[0018] **FIG. 7** depicts a block diagram of a system in which a Jini Service Bean (JSB) provides its service to a client, in accordance with the present invention;

[0019] **FIG. 8** depicts a block diagram of a cybernode in accordance with the present invention;

[0020] **FIG. 9** depicts a block diagram of a system in which a cybernode interacts with a service provisioner, in accordance with the present invention;

[0021] **FIG. 10** is a flow chart of Jini Service Bean (JSB) creation performed by a cybernode, in accordance with the present invention;

[0022] **FIG. 11** is a block diagram of a service provisioner in greater detail, in accordance with the present invention; and

[0023] **FIG. 12** is a flow chart of dynamic provisioning performed by a service provisioner, in accordance with the present invention.

DETAILED DESCRIPTION

[0024] The following description of embodiments of this invention refers to the accompanying drawings. Where appropriate, the same reference numbers in different drawings refer to the same or similar elements.

[0025] A. Introduction

[0026] Systems consistent with the present invention simplify the provision of complex services over a distributed network by breaking a complex service into a collection of simpler services. For example, automobiles today incorporate complex computer systems to provide in-vehicle navigation, entertainment, and diagnostics. These systems are usually federated into a distributed system that may include wireless connections to a satellite, the Internet, etc. Any one of an automobile's systems can be viewed as a complex service that can in turn be viewed as a collection of simpler services.

[0027] A car's overall diagnostic system, for example, may be broken down into diagnostic monitoring of fluids, such as oil pressure and brake fluid, and diagnostic monitoring of the electrical system, such as lights and fuses. The diagnostic monitoring of fluids could then be further divided into a process that monitors oil pressure, another process that monitors brake fluid, etc. Furthermore, additional diagnostic areas, such as drive train or engine, may be added over the life of the car.

[0028] Systems consistent with the present invention provide the tools to deconstruct a complex service into service elements, provision service elements that are needed to make up the complex service, and monitor the service elements to ensure that the complex service is supported. One embodiment of the present invention can be implemented using the Rio architecture created by Sun Microsystems and described in greater detail below. Rio uses tools provided by the Jini™ architecture, such as discovery and event handling, to provision and monitor complex services in a distributed system.

[0029] **FIG. 1** is a high level block diagram of an exemplary distributed system consistent with the present invention. **FIG. 1** depicts a distributed system **100** that includes computers **102** and **104** and a device **106** communicating via a network **108**. Computers **102** and **104** can use any type of computing platform. Device **106** may be any of a number of devices, such as a printer, fax machine, storage device, or computer. Network **108** may be, for example, a local area network, wide area network, or the Internet. Although only two computers and one device are depicted in distributed system **100**, one skilled in the art will appreciate that distributed system **100** may include additional computers and/or devices.

[0030] The computers and devices of distributed system **100** provide services to one another. A "service" is a resource, data, or functionality that can be accessed by a user, program, device, or another service. Typical services include devices, such as printers, displays, and disks; software, such as programs or utilities; and information managers, such as databases and file systems. These services may appear programmatically as objects of the Java™ programming environment and may include other objects, software components written in different programming languages, or hardware devices. As such, a service typically has an interface defining the operations that can be requested of that service.

[0031] **FIG. 2** depicts computer **102** in greater detail to show a number of the software components of distributed system **100**. One skilled in the art will recognize that computer **104** and device **106** could be similarly configured. Computer **102** contains a memory **202**, a secondary storage device **204**, a central processing unit (CPU) **206**, an input device **208**, and output device **210**. Memory **202** includes a lookup service **212**, a discovery server **214**, and a Java™ runtime system **216**. Java™ runtime system **216** includes Remote Method Invocation (RMI) process **218** and Java™ virtual machine (JVM) **220**. Secondary storage device **204** includes a Java™ space **222**.

[0032] Memory **202** can be, for example, a random access memory. Secondary storage device **204** can be, for example, a CD-ROM. CPU **206** can support any platform compatible with JVM **220**. Input device **208** can be, for example, a keyboard or mouse. Output device **210** can be, for example, a printer.

[0033] JVM **220** acts like an abstract computing machine, receiving instructions from programs in the form of bytecodes and interpreting these bytecodes by dynamically converting them into a form for execution, such as object code, and executing them. RMI **218** facilitates remote method invocation by allowing objects executing on one computer or device to invoke methods of an object on another computer or device. Lookup Service **212** and Discovery Server **214** are described in great detail below. Java™ space **222** is an object repository used by programs within distributed system **100** to store objects. Programs use Java space **222** to store objects persistently as well as to make them accessible to other devices within distributed system **100**.

[0034] A. The Jini™ Environment

[0035] The Jini™ environment enables users to build and maintain a network of services running on computers and devices. Jini™ is an architectural framework provided by Sun Microsystems that provides an infrastructure for creating a flexible distributed system. In particular, the Jini™ architecture enables users to build and maintain a network of services on computers and/or devices. The Jini™ architecture includes Lookup Service **212** and Discovery Server **214** that enable services on the network to find other services and establish communications directly with those services.

[0036] Lookup Service **212** defines the services that are available in distributed system **100**. Lookup Service **212** contains one object for each service within the system, and each object contains various methods that facilitate access to the corresponding service. Discovery Server **214** detects when a new device is added to distributed system **100** during a process known as boot and join, or discovery. When a new device is detected, Discovery Server **214** passes a reference to the new device to Lookup Service **212**. The new device may then register its services with Lookup Service **212**, making the device's services available to others in distributed system **100**. One skilled in the art will appreciate that exemplary distributed system **100** may contain many Lookup Services and Discovery Servers.

[0037] **FIG. 3** depicts an embodiment of the discovery process in more detail. This process involves a service provider **302**, a service consumer **304**, and a lookup service **306**. One skilled in the art will recognize that service provider **302**, service consumer **304**, and lookup service **306** may be objects running on computer **102**, computer **104**, or device **106**.

[0038] As described above, service provider **302** discovers and joins lookup service **306**, making the services provided by service provider **302** available to other computers and devices in the distributed system. When service consumer **304** requires a service, it discovers lookup service **306** and sends a lookup request specifying the needed service to lookup service **306**. In response, lookup service **306** returns a proxy that corresponds to service provider **302** to service consumer **304**. The proxy enables service consumer **304** to establish contact directly with service provider **302**. Service provider **302** is then able to provide the service to service consumer **304** as needed. An implementation of the lookup service is explained in "The Jini™ Lookup Service Specification," contained in Arnold et al., *The Jini™ Specification,* Addison-Wesley, 1999, pp. 217-231.

[0039] Distributed systems that use the Jini™ architecture often communicate via an event handling process that allows an object running on one Java™ virtual machine (i.e., an event consumer or event listener) to register interest in an event that occurs in an object running on another Java™ virtual machine (i.e., an event generator or event producer). An event can be, for example, a change in the state of the event producer. When the event occurs, the event consumer is notified. This notification can be provided by, for example, the event producer.

[0040] **FIG. 4** is a flow chart of one embodiment of the event handling process. An event producer that produces event A registers with a lookup service (step **402**). When an event consumer sends a lookup request specifying event A to the lookup service (step **404**), the lookup service returns a proxy for the event producer for event A to the event consumer (step **406**). The event consumer uses the proxy to register with the event producer (step **408**). Each time the event occurs thereafter, the event producer notifies the event consumer (step **410**). An implementation of Jini™ event handling is explained in "The Jini™ Distributed Event Specification," contained in Arnold et al., *The Jini™ Specification,* Addison-Wesley, 1999, pp. 155-182.

[0041] B. Overview of Rio Architecture

[0042] The Rio architecture enhances the basic Jini™ architecture to provision and monitor complex services by considering a complex service as a collection of service elements. To provide the complex service, the Rio architecture instantiates and monitors a service instance corresponding to each service element. A service element might correspond to, for example, an application service or an infrastructure service. In general, an application service is developed to solve a specific application problem, such as word processing or spreadsheet management. An infrastructure service, such as the Jini™ lookup service, provides the building blocks on which application services can be used. One implementation of the Jini lookup service is described in U.S. Pat. No. 6,185,611, for "Dynamic Lookup Service in a Distributed System."

[0043] Consistent with the present invention, a complex service can be represented by an operational string. **FIG. 5** depicts a exemplary operational string **502** that includes one or more service elements **506** and another operational string **504**. Operational string **504** in turn includes additional service elements **506**. For example, operational string **502** might represent the diagnostic monitoring of an automobile. Service element **1** might be diagnostic monitoring of the car's electrical system and service element **2** might be diagnostic monitoring of the car's fluids. Operational string B might be a process to coordinate alerts when one of the monitored systems has a problem. Service element **3** might then be a user interface available to the driver, service element **4** might be a database storing thresholds at which alerts are issued, etc. In an embodiment of the present invention, an operation string can be expressed as an XML document. It will be clear to one of skill in the art that an operational string can contain any number of service elements and operational strings.

[0044] **FIG. 6** is a block diagram of a service element in greater detail. A service element contains instructions for creating a corresponding service instance. In one implementation consistent with the present invention, service element **506** includes a service provision management object **602** and a service bean attributes object **604**. Service provision management object **602** contains instructions for provisioning and monitoring the service that corresponds to service element **506**. For example, if the service is a software application, these instructions may include the requirements of the software application, such as hardware requirements, response time, throughput, etc. Service bean attributes object **604** contains instructions for creating an instance of the service corresponding to service element **506**. In one implementation consistent with the present invention, a service instance is referred to as a Jini™ Service Bean (JSB).

[0045] C. Jini™ Service Beans

[0046] A Jini™ Service Bean (JSB) is a Java™ object that provides a service in a distributed system. As such, a JSB implements one or more remote methods that together constitute the service provided by the JSB. A JSB is defined by an interface that declares each of the JSB's remote methods using Jini™ Remote Method Invocation (RMI) conventions. In addition to its remote methods, a JSB may include a proxy and a user interface consistent with the Jini™ architecture.

[0047] **FIG. 7** depicts a block diagram of a system in which a JSB provides its service to a client. This system includes a JSB **702**, a lookup service **704**, and a client **706**. When JSB **702** is created, it registers with lookup service

704 to make its service available to others in the distributed system. When a client **706** needs the service provided by JSB **702**, client **706** sends a lookup request to lookup service **702** and receives in response a proxy **708** corresponding to JSB **706**. Consistent with an implementation of the present invention, a proxy is a Java™ object, and its types (i.e., its interfaces and superclasses) represent its corresponding service. For example, a proxy object for a printer would implement a printer interface. Client **706** then uses JSB proxy **708** to communicate directly with JSB **702** via a JSB interface **710**. This communication enables client **706** to obtain the service provided by JSB **702**. Client **706** may be, for example, a process running on computer **102**, and JSB **702** may be, for example, a process running on device **106**.

[**0048**] D. Cybernode Processing

[**0049**] A JSB is created and receives fundamental life-cycle support from an infrastructure service called a "cybernode." A cybernode runs on a compute resource, such as a computer or device. In one embodiment of the present invention, a cybernode runs as a Java™ virtual machine, such as JVM **220**, on a computer, such as computer **102**. Consistent with the present invention, a compute resource may run any number of cybernodes at a time and a cybernode may support any number of JSBs.

[**0050**] **FIG. 8** depicts a block diagram of a cybernode. Cybernode **801** includes service instantiator **802** and service bean instantiator **804**. Cybernode **801** may also include one or more JSBs **806** and one or more quality of service (QoS) capabilities **808**. QoS capabilities **808** represent the capabilities, such as CPU speed, disk space, connectivity capability, bandwidth, etc., of the compute resource on which cybernode **801** runs.

[**0051**] Service instantiator object **802** is used by cybernode **801** to register its availability to support JSBs and to receive requests to instantiate JSBs. For example, using the Jini™ event handling process, service instantiator object **802** can register interest in receiving service provision events from a service provisioner, discussed below. A service provision event is typically a request to create a JSB. The registration process might include declaring QoS capabilities **808** to the service provisioner. These capabilities can be used by the service provisioner to determine what compute resource, and therefore what cybernode, should instantiate a particular JSB, as described in greater detail below. In some instances, when a compute resource is initiated, its capabilities are declared to the cybernode **801** running on the compute resource and stored as QoS capabilities **808**.

[**0052**] Service bean instantiator object **804** is used by cybernode **801** to create JSBs **806** when service instantiator object **804** receives a service provision event. Using JSB attributes contained in the service provision event, cybernode **801** instantiates the JSB, and ensures that the JSB and its corresponding service remain available over the network. Service bean instantiator object **804** can be used by cybernode **801** to download JSB class files from a code server as needed.

[**0053**] **FIG. 9** depicts a block diagram of a system in which a cybernode interacts with a service provisioner. This system includes a lookup service **902**, a cybernode **801**, a service provisioner **906**, and a code server **908**. As described above, cybernode **801** is an infrastructure service that supports one or more JSBs. Cybernode **801** uses lookup service **902** to make its services (i.e., the instantiation and support of JSBs) available over the distributed system. When a member of the distributed system, such as service provisioner **906**, needs to have a JSB created, it discovers cybernode **801** via lookup service **902**. In its lookup request, service provisioner **906** may specify a certain capability that the cybernode should have. In response to its lookup request, service provisioner **906** receives a proxy from lookup service **902** that enables direct communication with cybernode **801**.

[**0054**] **FIG. 10** is a flow chart of JSB creation performed by a cybernode. A cybernode, such as cybernode **801**, uses lookup service **902** to discover one or more service provisioners **906** on the network (step **1002**). Cybernode **801** then registers with service provisioners **906** by declaring the QoS capabilities corresponding to the underlying compute resource of cybernode **801** (step **1004**). When cybernode **801** receives a service provision event containing JSB requirements from service provisioner **906** (step **1006**), cybernode **801** may download class files corresponding to the JSB requirements from code server **908** (step **1008**). Code server **908** may be, for example, an HTTP server. Cybernode **801** then instantiates the JSB (step **1010**).

[**0055**] As described above, JSBs and cybernodes comprise the basic tools to provide a service corresponding to a service element in an operational string consistent with the present invention. A service provisioner for managing the operational string itself will now be described.

[**0056**] E. Dynamic Service Provisioning

[**0057**] A service provisioner is an infrastructure service that provides the capability to deploy and monitor operational strings. As described above, an operational string is a collection of service elements that together constitute a complex service in a distributed system. To manage an operational string, a service provisioner determines whether a service instance corresponding to each service element in the operational string is running on the network. The service provisioner dynamically provisions an instance of any service element not represented on the network. The service provisioner also monitors the service instance corresponding to each service element in the operational string to ensure that the complex service represented by the operational string is provided correctly.

[**0058**] **FIG. 11** is a block diagram of a service provisioner in greater detail. Service provisioner **906** includes a list **1102** of available cybernodes running in the distributed system. For each available cybernode, the QoS attributes of its underlying compute resource are stored in list **1102**. For example, if an available cybernode runs on a computer, then the QoS attributes stored in list **1102** might include the computer's CPU speed or storage capacity. Service provisioner **406** also includes one or more operational strings **1104**.

[**0059**] **FIG. 12** is a flow chart of dynamic provisioning performed by a service provisioner. Service provisioner **906** obtains an operational string consisting of any number of service elements (step **1202**). The operational string may be, for example, operational string **502** or **504**. Service provisioner **906** may obtain the operational string from, for example, a programmer wishing to establish a new service

in a distributed system. For the first service in the operational string, service provisioner **906** uses a lookup service, such as lookup service **902**, to discover whether an instance of the first service is running on the network (step **1204**). If an instance of the first service is running on the network (step **1206**), then service provisioner **906** starts a monitor corresponding to that service element (step **1208**). The monitor detects, for example, when a service instance fails. If there are more services in the operational string (step **1210**), then the process is repeated for the next service in the operational string.

[**0060**] If an instance of the next service is not running on the network (step **1206**), then service provisioner **906** determines a target cybernode that matches the next service (step **1212**). The process of matching a service instance to a cybernode is discussed below. Service provisioner **906** fires a service provision event to the target cybernode requesting creation of a JSB to perform the next service (step **1214**). In one embodiment, the service provision event includes service bean attributes object **604** from service element **506**. Service provisioner **906** then uses a lookup service to discover the newly instantiated JSB (step **1216**) and starts a monitor corresponding to that JSB (step **1208**).

[**0061**] As described above, once a service instance is running, service provisioner **906** monitors it and directs its recovery if the service instance fails for any reason. For example, if a monitor detects that a service instance has failed, service provisioner **906** may issue a new service provision event to create a new JSB to provide the corresponding service. In one embodiment of the present invention, service provisioner **906** can monitor services that are provided by objects other than JSBs. The service provisioner therefore provides the ability to deal with damaged or failed resources while supporting a complex service.

[**0062**] Service provisioner **906** also ensures quality of service by distributing a service provision request to the compute resource best matched to the requirements of the service element. A service, such as a software component, has requirements, such as hardware requirements, response time, throughput, etc. In one embodiment of the present invention, a software component provides a specification of its requirements as part of its configuration. These requirements are embodied in service provision management object **602** of the corresponding service element. A compute resource may be, for example, a computer or a device, with capabilities such as CPU speed, disk space, connectivity capability, bandwidth, etc.

[**0063**] In one implementation consistent with the present invention, the matching of software component to compute resource follows the semantics of the Class.isAssignable( ) method, a known method in the Java™ programming language. If the class or interface represented by QoS class object of the software component is either the same as, or is a superclass or superinterface of, the class or interface represented by the class parameter of the QoS class object of the compute resource, then a cybernode resident on the compute resource is invoked to instantiate a JSB for the software component. Consistent with the present invention, additional analysis of the compute resource may be performed before the "match" is complete. For example, further analysis may be conducted to determine the compute resource's capability to process an increased load or adhere to service level agreements required by the software component.

[**0064**] F. Enhanced Event Handling

[**0065**] Systems consistent with the present invention may expand upon traditional Jini™ event handling by employing flexible dispatch mechanisms selected by an event producer. When more than one event consumer has registered interest in an event, the event producer can use any policy it chooses for determining the order in which it notifies the event consumers. The notification policy can be, for example, round robin notification, in which the event consumers are notified in the order in which they registered interest in an event, beginning with the first event consumer that registered interest. For the next event notification, the round robin notification will begin with the second event consumer in the list and proceed in the same manner. Alternatively, an event producer could select a random order for notification, or it could reverse the order of notification with each event.

[**0066**] As described above, in an embodiment of the present invention, a service provisioner is an event producer and cybernodes register with it as event consumers. When the service provisioner needs to have a JSB instantiated to complete an operational string, the service provisioner fires a service provision event to all of the cybernodes that have registered, using an event notification scheme of its choosing.

[**0067**] The foregoing description of an implementation of the invention has been presented for purposes of illustration and description. It is not exhaustive and does not limit the invention to the precise form disclosed. Modifications and variations are possible in light of the above teachings or may be acquired from practicing of the invention. Additional modifications and variations of the invention may be, for example, the described implementation includes software but the present invention may be implemented as a combination of hardware and software or in hardware alone. The invention may be implemented with both object-oriented and non-object-oriented programming systems.

[**0068**] Furthermore, one skilled in the art would recognize the ability to implement the present invention in many different situations. For example, the present invention can be applied to the telecommunications industry. A complex service, such as a telecommunications customer support system, may be represented as a collection of service elements such as customer service phone lines, routers to route calls to the appropriate customer service entity, and billing for customer services provided. The present invention could also be applied to the defense industry. A complex system, such as a battleship's communications system when planning an attack, may be represented as a collection of service elements including external communications, weapons control, and vessel control.

[**0069**] Additionally, although aspects of the present invention are described as being stored in memory, one skilled in the art will appreciate that these aspects can also be stored on other types of computer-readable media, such as secondary storage devices, like hard disks, floppy disks, or CD-ROM; a carrier wave from the Internet or other propagation medium; or other forms of RAM or ROM. The scope of the invention is defined by the claims and their equivalents.

What is claimed is:

1. A method for providing a service in a distributed system, comprising:

receiving a list of service elements, wherein at least several of the service elements constitute the service; and

for each service element in the list,

determining whether an application corresponding to the service element is available in the distributed system, and

monitoring the application to detect whether the application fails.

2. The method of claim 1, further comprising

creating the application corresponding to the service element, if the application corresponding to the service element is not available in the distributed system

3. The method of claim 1, further comprising

recreating the application, if it is detected that the application fails.

4. The method of claim 1, wherein the creating further comprises:

determining requirements of the service element;

matching the requirements of the service element to a resource on the distributed system; and

sending the requirements of the service to the resource to initiate creation of the application.

5. A method in a distributed system for providing a service consisting of a collection of service elements, comprising:

determining whether an instance of each service element in the collection is available in the distributed system; and

for each service element in the collection that does not have an instance available in the distributed system, creating a new service element instance.

6. The method of claim 5, further comprising:

monitoring a running service element instance corresponding to each service element in the collection to determine if the running service element instance fails; and

creating a new service element instance, if the running service element instance fails.

7. The method of claim 5, wherein the determining further comprises:

using a lookup service to determine whether an instance of each service element in the collection is running in the distributed system.

8. A method for providing services in a distributed system, comprising:

receiving capabilities of a compute resource from an executable application on the compute resource;

receiving a request to add a new service to the distributed system, including information reflecting requirements of the new service;

matching the requirements of the new service to the capabilities of the compute resource; and

directing the executable application running on the compute resource to instantiate the new service.

9. The method of claim 8, wherein the request further includes attributes of the new service and the directing further comprises:

sending the attributes of the new service to the compute resource.

10. The method of claim 8, wherein the object is a Java™ virtual machine.

11. A method for handling events in a distributed system, comprising:

advertising a capability to notify members of the distributed system when an event of a predetermined type occurs;

receiving registration requests from a plurality of event consumers, the registration requests specifying the predetermined type; and

when an event of the predetermined type occurs,

determining an order in which to notify each of the plurality of event consumers, and

sending an event notification to each of the plurality of event consumers in the order determined.

12. The method of claim 11, wherein the predetermined type of event is detection of a need to create a service on the distributed system.

13. The method of claim 11, wherein the event notification is a request to create a service on the distributed system.

14. The method of claim 11, further comprising:

sending the event notification to each of the plurality of event consumers in a random order.

15. The method of claim 11, further comprising:

sending the event notification to each of the plurality of event consumers in a round robin order.

16. A method for providing a service in a distributed system, comprising:

receiving a list of service elements, wherein at least several of the service elements constitute the service; and

for each service element in the list,

determining whether an application corresponding to the service element is available in the distributed system,

creating the application corresponding to the service element, if the application corresponding to the service element is not available in the distributed system, and

monitoring the application to detect whether the application fails.

17. A system for providing a service in a distributed system, comprising:

a receiving component configured to receive a list of service elements, wherein at least several of the service elements constitute the service;

a service determining component configured to determine, for each service element in the list, whether an application corresponding to the service element is available in the distributed system, and

a monitoring component configured to monitor the application to detect whether the application fails.

18. The system of claim 18, further comprising

a creating component configured to create the application corresponding to the service element, if the application corresponding to the service element is not available in the distributed system

19. The system of claim 18, further comprising

a recreating component configured to recreate the application, if it is detected that the application fails.

20. The system of claim 18, wherein the creating further comprises:

a requirements determining component configured to determine requirements of the service element;

a matching component configured to match the requirements of the service element to a resource on the distributed system; and

a sending component configured to send the requirements of the service to the resource to initiate creation of the application.

21. A system in a distributed system for providing a service consisting of a collection of service elements, comprising:

a determining component configured to determine whether an instance of each service element in the collection is available in the distributed system; and

a creating component configured to create a new service element instance for each service element in the collection that does not have an instance available in the distributed system.

22. The system of claim 21, further comprising:

a monitoring component configured to monitor a running service element instance corresponding to each service element in the collection to determine if the running service element instance fails; and

a failure creating component configured to create a new service element instance, if the running service element instance fails.

23. The system of claim 21, wherein the determining component further comprises:

a using component configured to use a lookup service to determine whether an instance of each service element in the collection is running in the distributed system.

24. A system for providing services in a distributed system, comprising:

a capabilities receiving component configured to receive capabilities of a compute resource from an executable application on the compute resource;

a request receiving component configured to receive a request to add a new service to the distributed system, including information reflecting requirements of the new service;

a matching component configured to match the requirements of the new service to the capabilities of the compute resource; and

a directing component configured to direct the executable application running on the compute resource to instantiate the new service.

25. The system of claim 24, wherein the request further includes attributes of the new service and the directing component further comprises:

a sending component configured to send the attributes of the new service to the compute resource.

26. The system of claim 24, wherein the object is a Java™ virtual machine.

27. A system for handling events in a distributed system, comprising:

an advertising component configured to advertise a capability to notify members of the distributed system when an event of a predetermined type occurs;

a receiving component configured to receive registration requests from a plurality of event consumers, the registration requests specifying the predetermined type;

a determining component configured to determine, when an event of the predetermined type occurs, an order in which to notify each of the plurality of event consumers; and

a sending component configured to send an event notification to each of the plurality of event consumers in the determined order.

28. The system of claim 27, wherein the predetermined type of event is detection of a need to create a service on the distributed system.

29. The system of claim 27, wherein the event notification is a request to create a service on the distributed system.

30. The system of claim 27, further comprising:

a random sending component configured to send the event notification to each of the plurality of event consumers in a random order.

31. The system of claim 27, further comprising:

a round robin sending component configured to send the event notification to each of the plurality of event consumers in a round robin order.

32. A system for providing a service in a distributed system, comprising:

a receiving component configured to receive a list of service elements, wherein at least several of the service elements constitute the service;

a determining component configured to determine, for each service element in the list, whether an application corresponding to the service element is available in the distributed system,

a creating component configured to create the application corresponding to the service element, if the application corresponding to the service element is not available in the distributed system; and

a monitoring component configured to monitor the application to detect whether the application fails.

* * * * *