Title: AN APPARATUS IMPLEMENTING A METHOD FOR DISCOVERING NETWORK BASED DISTRIBUTED APPLICATIONS

Abstract: An apparatus implementing a method for discovering applications having components distributed over a plurality of nodes in a network is disclosed. The method comprises the steps of obtaining information associated with at least one application, said information providing first and second instructions, executing processing associated with the first instructions, determining a tentative identification of at least one application based on responses associated with the processing of the first instructions, executing processing associated with the second instructions based on each of the tentatively identified applications, and confirming an identification of at least one application based on responses associated with the processing of the second instructions. In one aspect of the invention, the method further comprises the step of storing the application identification and characteristics in a representation of the application.
An Apparatus Implementing a Method for
Discovering Network Based Distributed Applications

Field of Invention

[0001] This application is related to the field of distributed systems, and more specifically, discovery of distributed application components and identification of the related application topology.

Background of the Invention

[0002] The use of computer networks has become an integral part of the way businesses provide goods and services to their customers. One advantage the use of networks provides is to enable the distribution of applications and the underlying business logic closer to the actual user or customer. This enables these businesses to offer higher levels of service to disparate groups of customers in a wider geographic area than ever before. This has also enabled businesses to allow customers access to the business network, albeit limited, for example, to directly track their purchases. In this case, each customer may have access to standardized or “tailored” application software packages or to custom developed software packages to perform desired operations.

[0003] Initially, networks were of a client/server type where the client represented a requestor of services and a server was the provider of the requested servers. However, this network configuration proved to be limiting and multi-tier networks were next developed. The multi-tier network configuration provides improved flexibility and scalability over the client/server network.

[0004] In the multi-tier network, a middle tier, between a client requesting information and server including a data base, developed that provided services such as transaction monitoring, message servicing and applications services. The middle tier layer thus provided queuing of client requests, application execution and data base staging. The middle tier layer may be further divided into units of different functions to further improve flexibility and scalability. In this case, the middle tier may include applications written in HTML (Hyper-Link Textual Markup Language), which is well-known in the art, for communication with the client and application servers written in C++ or Java programming languages, which are also well-known in the art. To fill the gap between the HTML and C++ applications, an intermediate web server layer may be incorporated to translate messages between the two application layers.
In a further network expansion, a distributed/collaborative enterprise architecture based on an Object Request Broker and/or a Common Object Request Broker Architecture was developed. This enterprise architecture allows for the use, and reuse, of business models on an enterprise-wide scale; an enterprise, in this case, represents a system comprised on multiple business systems or multiple subsystems.

However, as businesses take advantage of their networks and their networks expand, either in a planned manner or by the acquisition of other networks, the number of application packages may increase significantly. In some cases, the state of all the application packages, e.g., “running,” “installed but non-running,” and their locations may not be known or appreciated, particularly for those application packages that may be tailored or those that have narrow usage. In addition, enterprise applications, telecom services and other such services, need not be isolated entities existing on a single host, but rather may be distributed with dependent components present on multiple hosts within their enterprise and sometimes even spanning enterprises. In addition, application components may be updated on some servers and not in others. Hence, while application is composed of software-compatible components, a single definition of the application is not necessarily determinable.

In order to determine the existence of the application and their operating state, it is often required to discover many of the distributed pieces or components and the relationships between them, i.e., the application’s “topology,” and further to make a determination that the application has indeed been found. This is not a straightforward task as the variability in configuration and deployment options for these applications is high. For example, to discover simple processes that are running in a UNIX-based system, a user may use a command line tool, e.g., an instruction, such as UNIX command “ps” to “dump the process table,” for example. This command line tool creates a list of processes executing on a specific host on the network. The list may then be filtered using the UNIX “grep” command line with known search criteria. This specific methodology is, of course, of limited value as it is unable to discover non-running applications and does not discover the applications topology (i.e., the relationships among distributed components). More sophisticated tools, referred to as agents, may be built or created to probe still deeper into the components and their relationships. However, as in the prior example, there is no knowledge of what the relationships among multiple processes are, and only currently running processes may be discovered.
Thus, as the network expands it can become bloated with forgotten application packages that may have little or no usage, but are left in place as the consequence of their removal is unknown. On the other hand, leaving unused applications where they are installed may cause harm by consuming valuable disk space and/or if running, also consuming valuable CPU cycles. Most importantly, there are critical applications that must be running with optimal performance for a business to service their customers and effectively run their operation.

To manage the applications, whether active, active and forgotten, or not running, it is important to understand or have knowledge of the configuration of the application components. Application configuration information includes the description of the application, its components, the relationship between applications, the relationship between the components, and how the components are related with the underlying system and environment on which they are running. Examples of aspects of an application component include its structure at a device, its structure across devices, its performance characteristics, its dependencies with other applications in the device, and its dependencies with other applications in other devices. However, no systematic method exists to interrogate the network and determine applications residing on the network, and their status based on the discovered components.

Hence, there is a need in the industry for a systematic method and apparatus for discovering distributed application components and identifying the associated application topology.

**Summary of the Invention**

An apparatus implementing a method for discovering applications having components distributed over a plurality of nodes in a network is disclosed. The method comprises the steps of obtaining information associated with at least one application, the information providing first and second instructions, executing processing associated with the first instructions, determining a tentative identification of at least one application based on responses associated with the processing of the first instructions, executing processing associated with the second instructions based on each of the tentatively identified applications, and confirming an identification of at least one application based on responses associated with the processing of the second instructions. The method further comprises the step of storing the application identification and characteristics in a representation of the application. In a further aspect of the invention, the method comprises the steps of creating a first section
containing at least one directive for specifying information or initiating at least one external process for discovering characteristics of the application, creating a second section containing at least one directive for specifying information or initiating at least one external process for storing the discovered characteristics, and creating a third section containing at least one directive for specifying information or initiating at least one external process for testing the discovered application characteristics.

**Brief Description of the Figures**

[0012] Figure 1A illustrates a conventional multi-tiered network configuration;

[0013] Figure 1B illustrates abstract relationship among components in a multi-tier network configuration;

[0014] Figure 2 illustrates a conventional J2EE distributed system;

[0015] Figures 3A-3C illustrate flow charts of an exemplary process for discovering application processes in accordance with the principles of the invention;

[0016] Figure 4A illustrates an exemplary model in accordance with the principles of the invention;

[0017] Figure 4B illustrates an exemplary model of a J2EE application in accordance with the principles of the invention;

[0018] Figure 5 illustrates a flow chart of an exemplary process for discovering applications in accordance with the principles of the invention;

[0019] Figures 6A and 6B illustrate exemplary correlation operations for identifying applications in accordance with the principles of the invention;

[0020] Figures 7A-7C illustrate exemplary methods for determining selected executable processes in accordance with the principles of the invention;

[0021] Figures 8A and 8B illustrate exemplary application information in accordance with the principles of the invention;

[0022] Figure 9 illustrates an exemplary application discovery correlation matrix based on the application information shown in Figures 8A and 8B;

[0023] Figures 10A and 10B illustrate exemplary application topology of the applications shown in Figures 8A and 8B, respectively;

[0024] Figure 11 illustrates a flow chart of an exemplary process for determining and deploying application information shown in Figures 8A and 8B;
[0025] Figure 12 illustrates a flow chart of an exemplary process for specifying the Discovery section of an application signature in accordance with the principles of the invention;

[0026] Figure 13 illustrates a flow chart of an exemplary process for classifying a relationship among processes associated with an application signature in accordance with the principles of the invention;

[0027] Figure 14 illustrates a flow chart of an exemplary process for specifying how testing the relationships among processes associated with an application signature would function in accordance with the principles of the invention;

[0028] Figure 15 illustrates a user-interface for creating a specific instance of an application signature;

[0029] Figure 16 illustrates an exemplary application signature schema associated with the user-interface information shown in Figure 15; and

[0030] Figure 17 illustrates an exemplary system for executing the processing shown herein.

[0031] It is to be understood that these drawings are solely for purposes of illustrating the concepts of the invention and are not intended as a definition of the limits of the invention. The embodiments shown in the figures herein and described in the accompanying detailed description are to be used as illustrative embodiments and should not be construed as the only manner of practicing the invention. Also, the same reference numerals, possibly supplemented with reference characters where appropriate, have been used to identify similar elements

**Detailed Description**

[0032] Figure 1A illustrates a conventional multi-tiered network configuration 100 containing middle tier servers 120 -160, which are selectively in communication with data base servers 170 and 180. Further illustrated are client servers in communication with associated middle tier servers, e.g., client servers 120.1, 120.2 in communication with middle tier server 120, and client servers 150.1-150.n in communication with middle tier server 150. Network 100 may represent a conventional distributed system suitable for e-business transactions such purchasing over public or private networks, e.g., Internet. In the illustrated case, users (not shown) in communication with client servers 120.2, for example, may request
information stored on data bases 170 and/or 180 through corresponding middle layer
server 120.

[0033] Figure 1B illustrates an exemplary abstract relationship among
components in the multi-tier network configuration shown in Figure 1A. In this
abstract representation an application 190 associate with a web browser is hosted on
client server 120.1. Application 190 contains at least the two illustrated components
190.1 and 190.2. Also illustrated is middle tier server 120, which may be shared, as
shown, between or among servers. For example, web server 122 may be hosted on the
same server 120.1 as the web browser or may be hosted on the same server that hosts
the application server 124. In either case, application 192, containing components
192.1, 192.2, is hosted on web server 122 and application 194 is hosted on application
server 124. Further illustrated is data base 170 hosting application 196 having
components 196.1, 196.2. As one skilled in the art would recognize, the applications
190, 192, 194 and 196, together, represent an application that enables the completion
of a desired function or task. For example, the components of each application 190-196,
may be written specifically for the application or may be a commercially-
available software package that performs desired operations or functions. For
example, application 190 may be a commercially-available data base management
software, such as ORACLE, and component 190.2 may represent a “glue-ware” that
enables operations from custom applications to access available data using an
ORACLE-type commands. ORACLE is a registered trademark of the Oracle
Corporation.

[0034] Figure 2 illustrates the distribution of a conventional J2EE software
application using the multi-tier method described in Figures 1A and 1B. In this
illustrated application, information is received over the Internet 110 and, when
acceptable, passes through firewall 205. The information is provided to load balancer
nodes 210, which are used to distribute, substantially evenly, information among
connected or associated meddle tier webservers 215. The webservers 215, receiving
the data via an HTTP server 217, directs the information, using an appropriate plugin
219, to the desired application server node 230 and associated application server 235.
The information may request additional information be retrieved from database server
240. In this aspect, database server 240 hosting application software 245, executes
appropriate operations, e.g., read/write, in response to the requests provided.
Figure 3A illustrates a flow chart of an exemplary process 300 for discovering, or identifying and classifying, applications in accordance with the principles of the invention. In this exemplary process, an abstract model of the application is developed at block 310. At block 312, types of relationships among the components of the abstract model are determined and stored within the model. At block 314, a model of the relationships among the components is defined. At block 316, instrumentation methods associated with the type of application are defined. The instrumentation methods, referred to as detectors and/or probes, as will be more fully discussed, are developed for each different computer operating system for which the application is running on. Well-known operating systems are Windows, Unix, Linux, MacOS. Further, the same detectors and probes may be suitable for each operating system or may be tailored dependent upon the specific characteristics of each operating system. At block 318, the probes or detectors are further defined for each type of computing system, i.e., platform, for which the application is running on. Well-known types of platforms are Wintel (i.e., Windows/INTEL) Personal Computers (PC), Sun Microsystems Corporation Solaris-based servers, etc.

At block 320, probes and/or detectors to discover specific applications are further defined and at block 322 probes and/or detectors to discover the relationships of the specific applications are defined. At block 324, the information from the detectors and probes are consolidated and stored into the abstract model.

Figure 3B illustrates a flow chart of an exemplary process 330 for discovering applications and processes on a local application server (i.e., platform or host). In this illustrative process, discovery of physical and operating system resources are discovered at block 332 using defined detectors and probes. At block 334, running processes are discovered. At block 336, ports associated with the running processes are discovered. At block 338, the installations of the discovered applications are discovered and, at block 340, the resources of the application server are discovered. At block 342, the configuration files are then reviewed and stored.

Figure 3C illustrates a flow chart of an exemplary process 350 for discovering processes on remote application servers. In this illustrative process, discovery of local resources are performed at each of the remote sites in a manner similar to that shown in Figure 3B, at block 352. At block 354, vendor information of the remote site is obtained to identify specific components. At block 356, the information regarding the discovered application is consolidated and, at block 458,
information regarding the communication protocol is consolidated and provided to the
requesting platform or host.

[0039] Figure 4A illustrates an abstract model or representation 400 of an
application, in this case, an application server, in accordance with the principles of the
invention. The model shown is an extension of a known network models, such as the
SMARTS® InCharge™ Common Information Model (ICIM), 410 or similarly
defined or pre-existing CIM-based model and adapted for the network. SMARTS and
Incharge are trademarks of System Management ARTs, Inc., White Plains, NY, USA.
This model representation shown is an extension of the DMTF/SMI model. Model-
based system representation is discussed in commonly-owned USPA Serial No.
10/400,718, now US Patent No. _____ and US Patent Nos. 5,528,516, 5,661,668 and
6,249,755, the contents of all of which are incorporated by reference herein.

[0040] Abstract model 410 includes network elements or components that are
selected for representation and referred to as managed components. The representation
of the managed components includes aspects or properties of the component
represented. In this case, an application server is represented as a managed object,
referred to as ApplicationService 412. ApplicationService 412 is a generalization of
objects of class Application 414 and MgmtAgent 416. Application 414 is a
generalization of objects of classes ApplicationServer 418, ApplicationCluster 420 and
WebServer 422. LoadBalancer 424 is an object within the ApplicationCluster 420
object. Also shown is that object class MgmtAgent 416 contains objects NodeAgent
426 and DeploymentManager 428.

[0041] Figure 4B illustrates an exemplary model of a J2EE WebServer
application, shown in Figure 2, with regard to the model shown in Figure 4A, and the
associated relationships in accordance with the principles of the invention. In this
illustrative model, classes ApplicationServer 412 contains object classes
WebContainer 435, EnterpriseBeanContainer 440, JCAContainer 445,
TransactionManager 450, Servlet 455, ConnectionPool 460 and EnterpriseBean 465.
Also shown is object class Transaction 470 which represents, as the name implies, a
transaction conducted between two parties. Transaction 470 contains object classes
ServletTransaction 475 and DataTransaction 480.

[0042] The objects and relationships may be more fully described as:

[0043] 1. Object NodeAgent is a subclass of object MgmtAgent, which
represents the process that runs in a host that serves as a management contact point
with all ApplicationServers hosted by the host. When object NodeAgent has problems, administration ApplicationServers is impacted. As shown, object NodeAgent is a member of a cell, produces a session with object DeploymentManager, consumes sessions with ApplicationServer on the same host and has a HostedBy relationship with the host server;

[0044] 2. Object DeploymentManager is a subclass of the object MgmtAgent, which is the focal point for administration of a cell in the object WebSphere. It has sessions with all of the NodeAgent objects that manage application servers in the cell that the object DeploymentManager manages. Object DeploymentManager is a member of a cell, consumes Session with the object NodeAgent and has a HostedBy relationship with the host;

[0045] 3. LoadBalancer object is a specialization of the ApplicationCluster object intended to model the edge load balancer, which distributes the client requests to the web server. The main purpose of this specialization class is to provide a “placeholder” for the contact information for the SNMP instrumentation of the transactions to the individual WebServers that compose the cluster. Attributes of the object class LoadBalancer are the AgentAddress, which is a string like “address:port” which describes the port where the SNMP agent that instruments the load balancer can be found;

[0046] 4. Object WebServer is an application that represents an instance of a WebServer process that is used to handle high volume interaction with the web browser, while forwarding dynamic contents to the WebSphere Application server. This component can be discovered through the configuration of the edge LoadBalancer object. The WebServer has a PartOf relationship to LoadBalancer and ApplicationCluster objects, produces transactions with the WebBrowsers, WebClients and Sessions with the LoadBalancer. It further consumes transactions with the WebContainer object of the ApplicationServers objects and has a HostedBy relationship with the host. Attributes of the WebServer object class are the port, which has a default value of “80” and a secure port, which has a default value of “443;”

[0047] 5. Object ApplicationServer represents the JVM (Java Virtual Machine) that runs the WebSphere. The actual web applications that run are layered over this application;
6. Object WebContainer is a subclass of ApplicationService, which manages the web components of the application;

7. Object EnterpriseBeanContainer is a subclass of the object ApplicationService, which manages the Enterprise Java Bean (EJB) components of the application;

8. Object JCAContainer is a specialization of the ApplicationsService, which manages the J2C and JDBC connections of the application in the WebSphere environment;

9. Object TransactionManager is an object class that manages the transaction;

10. Object Servlet is an ApplicationService subclass, which can be monitored with regard to performance;

11. Object ConnectionPool is a subclass of ApplicationService, which serves as an anchor for database and legacy external data links. Applications communicate with Connection pools through object ConnectionPools;

12. Object Cell is a manageable element and a WebSphere is a goup of manageable servers, and its general health may be determined based on the health of the components;

13. JMSServer is a subclass of the object MessageQueueService. More specifically, the J2EE specification mandates a JMS provider be always available.

The model shown in Figs. 4A and 4B are associated with the processing represented by blocks 310, 312 and 314 shown in Fig. 3A.

Figure 5 illustrates a flow chart of an exemplary process 500 for discovering application components in accordance with the principles of the invention. In this exemplary process information, e.g., a definition or signature, of the application is obtained at block 510. The application signature defines, in part, instructions or directives, that are represented, herein, as detector(s) and probe(s), used for discovering the applications. The processing referred to at block 510 is similar to that described with regard to blocks 320 and 322 in Fig. 3A and is more fully described with regard to Figures 11-15.

At block 520, processing associated with the detector, i.e., directives or instructions, obtained from the application signature is performed. In this case, the detectors provide a basic capability to find or determine clues with regard to the presence and identity of distributed applications. Detectors may represent elements,
such as code or devices, that are responsible for interacting with the environment, i.e., network elements, and provide broad knowledge of the distributed application. The code or devices may, for example, initiate commands such as "presence requests" or an HTTP request of a port and may further operate on received responses to the initiated directives. Detectors provide an initial view, which is relatively broad, of the application and are executed with a low frequency. Detectors, in addition to information contained in the application signature, further provide instruction for more detailed analysis, as will more fully explained.

[0059] At block 530, the results of the detector responses are used to determine an initial or tentative classification and/or identification of an application. In one aspect the tentative identification may be made using a correlation function derived from the information contained in the application signatures, as will be more fully discussed.

[0060] After the tentative application identification, additional information regarding the application is obtained by selecting one or more of probes 540.1-540.n, which may include instructions or directives and executing the processing associated with the selected information. In this illustrate case, probes represent code responsible for interacting with the environment (network elements) and are provided with knowledge of the application in order to obtain characteristics and properties of the tentatively identified application. Probes 540.1-540.n, e.g., status requests, obtain more detailed information regarding the tentatively identified application to more fully identify the application.

[0061] At block 550, the characteristics, attributes, and/or properties obtained by the executed probes are collected and combined to confirm the identification of the tentatively identified application. In one aspect of the invention, the characteristics, attributes and properties obtained by the selected probes are sufficient to confirm the identification of the tentatively identified application and no further processing is necessary. In another aspect of the invention, the characteristics and properties obtained by the selected probes are not sufficient to confirm the identification of the tentatively identified application and additional processing is required. In this aspect of the invention, information from other sources may be utilized to complete the process of determining the identification of the distributed applications. For example, patterns of known relationships among application elements or components may be used to complete the identification and classification process when the information
from selected probes is not sufficient to provide the information. This is patterns may be used to substantiate incomplete or known incorrect relationships. In another aspect, knowledge of known relationships, referred to as application endpoints, which are gateways between applications and/or application components, may be used to complete the identification and classification process. In still another aspect of the invention, responses from probes not associated with the tentatively identified application may be used to confirm or deny the identification.

[0062] Figures 6A and 6B illustrate an exemplary method for providing an initial application identification in accordance with one aspect of the invention. Fig. 6A, illustrates an exemplary method for correlating responses from selected detectors to applications in accordance with the principles of the invention. In this illustrates case, information regarding detectors obtained from the information associated with each application to be identified is organized such that the relationship between the responses from each of the detectors $\text{DET}_1 - \text{DET}_m$ are associated with each of the applications $\text{APP}_1 - \text{APP}_n$. The correlation or association values are in the form of known value, (e.g., 0 and 1) or a probabilistic relationship (in the range of 0.0-1.0) which may be predetermined or, in one aspect of the invention, adapted based on known information.

[0063] Figure 6A illustrates the presence of application $\text{App}_1$ may be determined from responses from at least detector 1 ($\text{DET}_1$) and detector 2 ($\text{DET}_2$), as responses from $\text{DET}_3$ and $\text{DET}_4$ are only expected with a probability of 0.5 and 0.75, respectively. $\text{DET}_3$ and $\text{DET}_4$ may be associated with optional application components, for example, and need not be present to tentatively identify the application. Similarly, the presence of application $\text{App}_2$ requires a response from $\text{DET}_2$ and responses from $\text{DET}_1$, $\text{DET}_3$ and $\text{DET}_m$ are only expected with a probability of 0.5, 0.7 and 0.2, respectively.

[0064] In one aspect of the invention, an application may be initially or tentatively identified as that application having a high correlation determined as the largest accumulated value in response to the detectors activated. In a second aspect of the invention, an application may initially or tentatively be identified as that application having a high correlation determined as the number of responses and/or the probability that a response is required. For example, failure to receive a response from $\text{DET}_1$ indicates that the application is not $\text{APP}_1$ as a response to the specific detector is required. However, failure to receive a response from $\text{DET}_3$, indicates that
the application is not APP_n. In this case, the application may be tentatively identified as either APP_1 or APP_n, dependent upon response from the other detectors.

[0065] Figure 6B illustrates a second example of a correlation operation for tentatively identifying applications based on the responses to the detectors. In this illustrative example, responses from detectors 1, 2, 3, i and j, indicate that applications APP_1 and APP_n are both present and a tentative identification of both APP_1 and APP_n may be made.

[0066] Figures 7A-7C illustrate different for selecting probes based on a tentative application identification. Figure 7A illustrates that probes may be associated with each tentatively identified application and based on the tentative application identification, the associated probes are executed. As would be recognized, the associated probes may be executed sequentially or in parallel and the results collected as discussed with regard to block 550 (Fig. 5). Figure 7B illustrates a second method for selecting probes based on a tentative application identification, wherein each application is associated with one or more probes selected from a plurality of probes. Figure 7C illustrates a third, and preferred, method for selecting probes based on a tentative application identification, wherein each application is associated with one or more probes selected from a plurality of probes and the selected probes may further select additional probes based on the responses received. Hence, the information obtained from a selected probe may be supplemented or enhanced by the subsequent selection of a secondary probe. As would be recognized, by utilizing the processing shown in Figure 7C for example, the probe selected based on the tentative application identification may initiate subsequent probes to identify additional components of one or more applications. Although only selection of a secondary probe is illustrated, it would be recognized by those skilled in the art, additional layers of subsequent probe selection may be included and is considered within the scope of the invention.

[0067] Figures 8A and 8B illustrate examples of application signatures in accordance with the processing disclosed herein. Figure 8A illustrates an exemplary signature or information regarding the application, WebSphere AppServer and Figure 8B illustrates an exemplary signature of information regarding the application WebSphere DeploymentManager. WebSphere Application Servers and WebSphere Deployment Managers are components of the IBM WebSphere architecture. WebSphere Application Server interacts with clients served over the world wide web.
(Internet), while WebSphere Deployment Manager Application allow for the administration of a set of WebSphere Application Servers.

[0068] With regard to Figure 8A, exemplary information regarding the detectors, i.e., first instruction processing, is represented as Port:9091, 810, Request: Get/Http/1.0, 812 and ProcessName /WebSphere/AppServer/ /Java, 816. An expected response to the Request: Get, 812, is represented as Response: “Server: WebSphere” 814. More specifically, instructions are provided to interrogate the network using a specific command on port 9091. An expected response to the instruction is “Server: WebSphere, if the server is hosting a WebSphere application. When the WebSphere application is tentatively identified, instructions, referred to as “WAS Probe,” 818, may be executed to determine more detailed characteristics of the tentatively identified application.

[0069] Figure 8B illustrates a similar signature for a WebSphere Deployment Manager Application. In this case, the detector information, is similar to that shown in Figure 8A, i.e., Port:9091, 820, Request: Get/Http/1.0, 822, however, the ProcessName /WebSphere/Deployment/ /Java, 826 is different. The expected response to the Request: Get 822 is Response: “Server: WebSphere” 824. In this case, when a WebSphere Deployment Manager Application is tentatively identified, the probe “WAS Domain” 828 is executed to determine additional characteristics of the WebSphere Deployment Manager Application.

[0070] There is also shown a section that may be used to provide instruction for testing the identified application. The testing may include, for example, processing that is performed at a known rate. Details regarding the contents of the detector, probe and test section are more fully disclosed with regard to Figures 11-16.

[0071] Figure 9 illustrates an exemplary correlation relationship derived from the information associated with the detectors and applications for the examples shown in Figures 8A and 8B. In this case, Websphere application server requires, i.e, 1, an appropriate response from the Port detector and a first ProcessName detector whereas the Websphere DeploymentManager application requires, i.e., 1, an appropriate response from the Port detector and a second ProcessName detector.

[0072] Dependent upon the responses received from the detectors, a tentative identification of the application may be made and the appropriate probe (second instructions processing) may be initiated. The probes are, in this illustrative example,
contained in corresponding application information signatures and represented as Probe: WAS Probe and Probe: WAS Domain, respectively.

Figures 10A and 10B illustrate an exemplary WebSphere topology in view of the execution of the exemplary application signatures shown in Figures 8A and 8B. More specifically, the probe referred to as WAS Probe shown in Figure 8A, may inspect the environment and instantiate the model, as described in Figure 4b, as it becomes relevant to the current environment, for which an example is shown in Figure 10A. Similarly, the probe WAS Domain, shown in Figure 8B, may inspect the environment and instantiate the model, as described in Figure 4b, as it becomes relevant to the current environment, for which an example is shown in Figure 10B.

As would be further recognized, by inspecting and probing the WebSphere Deployment Manager (Figure 10B) it is possible to discover a list of WebSphere Application Servers (WAS) and instantiate their representation by the same WebSphere Application Servers discovered and represented as in Figure 10A. Hence, the topology of the Deployment Manager may be obtained by collecting topology information of previously determined WebSphere applications and utilizing this information to determine the topology of the Deployment Manager. This utilization information of other applications is advantageous as it reduces the number of probes required and further allows for the determination of relationships for which no information or probes are available.

Figure 11 illustrates a flow chart 1100 specifying an exemplary process for creating and then deploying an Application signature. Further details regarding particular steps shown in Figure 11 are presented with regard to Figures 12-15. Figure 11 illustrates the steps for discovering, block 1115, classifying, block1 120, and testing, block 1125, an application on a network.

In addition to the specification of the details for the three sections of discover, classify and test, the exemplary flow chart 1100 further illustrates the usage of the created Application Signature object as a component of an application management solution. In this illustrative process, the Application Signature, once created as an object, block 1130, is distributed, block 1135, to a collection of programs referred to as Beacons, which are similar to the well-known concept of software Agents. However, Beacons are used primarily for the actualization or execution of the processes referred to in the Application Signature. Beacons (Agents) manage and control the management processes defined in one or more Application
Signatures and communicate the results of these activities with a management server where the data is “safe-stored” in a network-available repository. The Beacon executes the processes defined in the Application Signature at a predetermined, or pre-established interval. Or the processes may be executed when commanded by an operator. The management server utilizing the results obtained from the Beacon’s execution of the Application Signature to create visual representations of its discovery and classification in the form of application maps (topology). The results pertaining to the test section are used as input to the management servers’ analysis process which will then be reflected in the form of the state of the application on the aforementioned application maps.

[0077] Figure 12 illustrates in more detail exemplary steps in creating the Discover Section of the Application Signature. Using the Microsoft Internet Information Servers (IIS) application as an example, the discovery section instructions or directives (block 1115, Figure 11) of the Application Signature may include

1) specify the application name, e.g. Windows 2000 Service (w3svc) to be matched, block 1210;

2) specify that processes to be matched; e.g., inetinfo.exe and dllhost.exe, block 1220;

3) specify the port to be matched, e.g., port 80, block 1230;

4) specify the Logfile, e.g., MyLogFile, block 1 240; and

5) specify the name of the Virtual directory (DocRoot) file; block 1250.

[0078] Figure 13 illustrates, in more detail, exemplary steps in creating the Classify Section (block 1120, Figure 11) instructions or directives of an Application Signature;

1) specify that the class of the application may be an instance of the class, e.g., IIS-WebServer, block 310;

2) The IIS-WebServer may have a RealizedBy relationship with the operating system processes, e.g., inetinfo.exe and dllhost.exe, block 1320;

3) The IIS-WebServer has an AccessedBy relationship with port, e.g., 80, block 1330;
4) An instance of the W2K Service class may be created and it may have a LayeredOver relationship with the processes, e.g., inetinfo.exe and dllhost.exe, block 1340;

5) Instances of the web sites discovered within the Web Server may be created with a LayeredOver relationship to the Web Server (a representation of which may also be created), block 1350;

6) The discovered Virtual directories for each Web Site may be created with a MountedOn relationship to the operating systems disk file system (a representation of which may also be created), block 1360; and

7) The operating system’s file system should have a ResidesOn relationship to the operating system’s logical disk (a representation of which may also be created); block 1370.

Figure 14 illustrates in more detail exemplary steps for creating a test section (block 1125, Figure 11) instructions or directions of the Application Signature:  
1) URL for a web applications may be entered. For example, in a retail situation wherein bluejeans are sold, a URL may be expressed as http://www.bluejeans.com/BrowseCat.asp in order to be certain that it returned the correct content, e.g., the string “BlueJeans”, block 1410;

2) This URL may then form the core of a synthetic transaction that is issued according to the Test statement in Application Signature to test if the topology (IIS and its many components) are still there and how rapidly each IIS Web Site responds to a query; (the URL may be issued periodically); and

3) When the synthetic transactions are issued, success criteria are deemed to be “true” when the transaction completes and it returns content containing the phrase, “BlueJeans”.

Figure 15 illustrates an exemplary application signature user interface for the application package referred to as ORACLE. ORACLE is a registered trademark of the ORACLE International Corporation having a principle place of business at 500 Oracle Parkway, Redwood City, California, 94065, USA. In this exemplary interface, the application name is referred to a Oracle and it is of the Class
database server. The Instance name is ACCT-Receiver-DB and the product name is Oracle 9i RDBMS (relational database management system). The application is described as an RDBMS and the Major and Minor versions are 9 and i, respectively. The ports are used to test whether a TCP/IP connection can be made to the remote application in order to test its current availability. In addition, processes: ORASRV, PMON, listener, DBWR and LGWR are specified in order to further identify the application Oracle and to signify that once discovered they should be created in the resultant topology as instances of the class process and having a relationship of ComposedOf with the Oracle application instance. In addition, files config.ora, tnsnames.ora, listener.ora and SQLnet.ora are specified so they will need to be “found” or discovered in order for the discover function to complete successfully.

Figure 16 illustrates elements of an exemplary schema of an application signature, referred to as object ApplicationSignature, associated with the user-interface shown in Figure 15. In this illustrated case, the information is entered in the user-interface is persisted as an object; specifically as an instance of the object class ApplicationSignature. In the example illustrated, an instance of the class ApplicationSignature is created containing the information entered by the user to discover, classify and test (or monitor) the software application Oracle. The object class ApplicationSignature contains elements describing detector type (for discovery), topology description (for Classification) and Monitoring Action (for testing). The label column has been added to make it easier for the reader to relate this back to the user-interface screen shown in Figure 8A:

1. Detector Type – The Detector type describes to a discovery probe what to look for when attempting to discover this application. The description may include or provide instruction or directives for the operation of the discovery probe(s);

2. Topology Description – The Topology Description is used to describe what topology would be created in a repository once discovery completes successfully. The description may include or provide instruction or directives for how the topology may be formulated or organized; and

3. Monitoring Action - the Monitoring Action is used to describe how to test that the application topology discovered is still available and what a reasonable response time would be for this application to respond to a test request. The description may include or provide instruction or directives for the operation of the specified action.
[0082] In one aspect of the invention, the files may be utilized to identify the Oracle application when it is not currently running. In this case, an Application Signature can specify both processes and files. If the application is not currently running, then the processes will not be active and, thus, not found. However, if a sufficient number of the files are located, then the application can be deemed to be “discovered, but not running.”

[0083] The Test procedure employs a monitoring action, in this case referred to as OraclePing, which executes a test that the identified application package is operating correctly. In this case, the test is implemented by utilizing a previously created DB stored procedure.

[0084] Figure 17 illustrates an exemplary embodiment of a system 1700 that may be used for implementing the principles of the present invention. System 1700 includes processing unit 1710 that may contain one or more input/output devices 1702, processors 1703 and memories 1704. I/O devices 1702 may access or receive information from one or more sources or devices 1701. Sources or devices 1701 may be devices such as routers, servers, computers, notebook computer, PDAs, cell phones or other devices suitable for transmitting and receiving information responsive to the processes shown herein. Devices 1701 may have access over one or more network connections 1750 via, for example, a wireless wide area network, a wireless metropolitan area network, a wireless local area network, a terrestrial broadcast system (Radio, TV), a satellite network, a cell phone or a wireless telephone network, or similar wired public networks, such as POTS, INTERNET, LAN, WAN and/or private networks, e.g., INTRANET, as well as portions or combinations of these and other types of networks. Network 1750 may similarly represent a communication bus, such as PCI, USB, Firewire, etc.

[0085] Input/output devices 1702, processors 1703 and memories 1704 may communicate over a communication medium 1725. Communication medium 1725 may represent, for example, a bus, a communication network, one or more internal connections of a circuit, circuit card or other apparatus, as well as portions and combinations of these and other communication media. Input data from the devices 1701 is processed in accordance with one or more programs that may be stored in memories 1704 and executed by processors 1703. Processors 1703 may be any means, such as general purpose or special purpose computing system, such as a laptop computer, desktop computer, a server, handheld computer, or may be a hardware
configuration, such as dedicated logic circuit, or integrated circuit. Processors 1703 may also be Programmable Array Logic (PAL), or Application Specific Integrated Circuit (ASIC), etc., which may be "programmed" to include software instructions or code that provides a known output in response to known inputs. In one aspect, hardware circuitry may be used in place of, or in combination with, software instructions to implement the invention. The elements illustrated herein may also be implemented as discrete hardware elements that are operable to perform the operations shown using coded logical operations or by executing hardware executable code.

In a one aspect, the processes shown herein may be represented by computer readable code stored on a computer readable medium. The code may also be stored in the memory 1704. The code may be read/downloaded from a memory medium 1783, an I/O device 1785 or magnetic or optical media, such as a floppy disk, a CD-ROM or a DVD, 1787. Although not shown, it would be recognized that the code may be stored on a device and downloaded, i.e., electronically transferred, via a network to processor 1703. The downloaded computer readable code may be stored in memory 1704 or executed directly by processor 1703. Further it would be understood that the code may be processor specific or processor non-specific. Code written in the Java programming language is an example of processor non-specific code. Java is a trademark of the Sun Microsystems Corporation.

Information from device 1701 received by I/O device 1702, after processing in accordance with one or more software programs operable to perform the functions illustrated herein, may also be transmitted over network 1780 to one or more output devices represented as display 1792, reporting device 1790, e.g., printer, or second processing system 1195. As one would recognize, networks 1725, 1750 and 1780 may be physically be the same network or may be different networks that operate on the same or different communication principles.

As one skilled in the art would recognize, the term computer or computer system may represent one or more processing units in communication with one or more memory units and other devices, e.g., peripherals, connected electronically and communicating with the at least one processing unit. Furthermore, the devices may be electronically connected to the one or more processing units via internal busses, e.g., ISA bus, microchannel bus, PCl bus, PCMCIA bus, USB, etc., or one or more internal connections of a circuit, circuit card
or other device, as well as portions and combinations of these and other communication media or external networks, e.g., the Internet and Intranet.

[0089] While there has been shown, described, and pointed out fundamental novel features of the present invention as applied to embodiments thereof, it will be understood that various omissions and substitutions and changes in the apparatus described, in the form and details of the devices disclosed, and in their operation, may be made by those skilled in the art without departing from the spirit of the present invention. For example, while the flow charts depict a sequence of operating steps, this is shown for illustrative purposes only as the steps may be executed or performed in another sequence or order. It is expressly intended that all combinations of those elements that perform substantially the same function in substantially the same way to achieve the same results are within the scope of the invention. Substitutions of elements from one described embodiment to another are also fully intended and contemplated.
Claims

What is claimed is:

1. An apparatus for discovering applications having components distributed over a plurality of nodes in a network, the apparatus comprising:
   a processor in communication with a memory, the processor executing code for:
   obtaining information associated with at least one application, said information providing first and second instructions;
   executing processing associated with the first instructions;
   determining a tentative identification of at least one application based on responses associated with the processing of the first instructions;
   executing processing associated with the second instructions based on each of the tentatively identified applications; and
   confirming an identification of at least one application based on responses associated with the processing of the second instructions.
2. The apparatus as recited in claim 1, the processor further executing code for:
   storing said application identification and characteristics of the identified application in a representation of the application.
3. The apparatus as recited in claim 1, wherein the first instructions processing determining broad characteristics of the at least one application.
4. The apparatus as recited in claim 1, wherein the second instructions processing further determining detailed characteristics of the at least one application.
5. The apparatus as recited in claim 1, the processor further executing code for:
   obtaining correlating values from the information associated with each of the at least one applications, the correlating values representing first instructions processing responses and the at least one application; and
   generating a correlating relationship between each of the at least one applications and responses associated with the first instructions processing.
6. The apparatus as recited in claim 5, the processor further executing code for:
   altering the correlating values by a known factor.
7. The apparatus as recited in claim 1, the processor further executing code for determining a tentative identification by:
   identifying an application having a high correlation based on the first
instructions processing responses.

8. The apparatus as recited in claim 1, the processor further executing code for:
   identifying known additional application characteristics, not responsive to the
   first or second instructions processing, based on information associated with the
   confirmed application identification.

9. The apparatus as recited in claim 1, the processor further executing code for
   confirming the application identification by:
   utilizing responses associated with the second instructions processing not
   associated with the tentatively identified application.

10. The apparatus as recited in claim 1, wherein the information associated with each
    of the at least one applications is contained in an associated application signature.

11. The apparatus as recited in claim 2, wherein the representation is a model
    composed of object classes and relationships between the object classes.

12. The apparatus as recited in claim 11, wherein the object classes are selected from
    the group consisting of: software element, software service, application service,
    database service, software request, software server, and software component.

13. The apparatus as recited in claim 11, wherein the relationships between the object
    classes are selected from the group consisting of: Hosted/HostedBy,
    Composed/ComposedOf, Initiated/InitiatedBy, Executed/ExecutedBy, and
    ConnectedTo/ConnectedVia.

14. The apparatus as recited in claim 1, the processor further executing code for:
    executing processing associated with a third instruction, said third instructions
    based on the at least one identified application.

15. The apparatus as recited in claim 14, the processor further executing code for
    performing the third instructions at a predetermined time.

16. The apparatus as recited in claim 1, further comprising:
    an Input/Output device in communication with the processor.

17. The apparatus as recited in claim 1, wherein the code is stored in the memory.

18. The apparatus as recited in claim 1, wherein the processor further executing code
    for:
    constructing a signature containing information associated with each of the at
    least applications, the signature comprising:
    a first section containing at least one directive for specifying information or
initiating at least one external process for discovering characteristics of said application;

a second section containing at least one directive for specifying information or initiating at least one external process for storing the discovered characteristics; and

a third section containing at least one directive for specifying information or initiating at least one external process for testing the discovered characteristics.

19. The apparatus as recited in claim 18, wherein the at least one directive in the first section is selected from the group consisting of: application name, instance name, processes to match, ports to match.

20. The apparatus as recited in claim 18, wherein the at least one directive in the second section is selected from the group consisting of: class, vendor name, description, major version designation, and minor version designation.

21. The apparatus as recited in claim 18, wherein the at least one directive in the third section is selected from the group consisting of: a synthetic transaction, a URL string, an SQL query, an API call, and a port test.

22. A computer-readable medium containing code thereon, the code suitable for discovering applications having components distributed over a plurality of nodes in a network, by providing instructions to a computing system for executing the steps of:

obtaining information associated with at least one application, said information providing first and second instructions;

executing processing associated with the first instructions;

determining a tentative identification of at least one application based on responses associated with the processing of the first instructions;

executing processing associated with the second instructions based on each of the tentatively identified applications; and

confirming an identification of at least one application based on responses associated with the processing of the second instructions.

23. The computer-readable medium as recited in claim 22, the code further instruction for executing the step of:

storing said application identification and characteristics of the identified application in a representation of the application.
24. The computer-readable medium as recited in claim 22, wherein the first instructions processing determining broad characteristics of the at least one application.

25. The computer-readable medium as recited in claim 22, wherein the second instructions processing further determining detailed characteristics of the at least one application.

26. The computer-readable medium as recited in claim 22, the code further providing instruction for executing the steps of:
   - obtaining correlating values from the information associated with each of the at least one applications, the correlating values representing first instructions processing responses and the at least one application; and
   - generating a correlating relationship between each of the at least one applications and responses associated with the first instructions processing.

27. The computer-readable medium as recited in claim 26, the code further providing instruction for executing the step of:
   - altering the correlating values by a known factor.

28. The computer-readable medium as recited in claim 22, the code further providing instruction for determining a tentative identification by:
   - identifying an application having a high correlation based on the first instructions processing responses.

29. The computer-readable medium as recited in claim 22, the code further providing instruction for executing the step of:
   - identifying known additional application characteristics, not responsive to the first or second instructions processing, based on information associated with the confirmed application identification.

30. The computer-readable medium as recited in claim 22, the code further providing instruction for executing the step of:
   - utilizing responses associated with the second instructions processing not associated with the tentatively identified application.

31. The computer-readable medium as recited in claim 22, wherein the information associated with each of the at least one applications is contained in an associated application signature.
32. The computer-readable medium as recited in claim 31, wherein the representation is a model composed of object classes and relationships between the object classes.

33. The computer-readable medium as recited in claim 31, the code further providing instruction for executing the step of:

   constructing the application signature containing information associated with each of the at least applications, the signature comprising:
   
   a first section containing at least one directive for specifying information or initiating at least one external process for discovering characteristics of said application;
   
   a second section containing at least one directive for specifying information or initiating at least one external process for storing the discovered characteristics; and
   
   a third section containing at least one directive for specifying information or initiating at least one external process for testing the discovered characteristics.

34. The computer-readable medium as recited in claim 33, wherein the at least one directive in the first section is selected from the group consisting of: application name, instance name, processes to match, ports to match.

35. The computer-readable medium as recited in claim 33, wherein the at least one directive in the second section is selected from the group consisting of: class, vendor name, description, major version designation, and minor version designation.

36. The computer-readable medium as recited in claim 33, wherein the at least one directive in the third section is selected from the group consisting of: a synthetic transaction, a URL string, an SQL query, an API call, and a port test.

37. The computer-readable medium as recited in claim 22, wherein the medium is selected from the group consisting of: semiconductor, magnetic, optical, and electronically transferred.
ON ENTRY

DEFINE ABSTRACT MODEL

TYPE RELATIONSHIP AMONG COMMON

DEFINE A MODEL OF RELATIONSHIP

DEFINE INSTRUMENTATION METHODS PER EDA PPM

DEFINE INSTRUMENTATION TO DISCOVER APPLICATIONS

DEFINE INSTRUMENTATION TO DISCOVER RELATIONSHIPS

CONSOLIDATE DISCOVERED INFORMATION

END IT
ENTRY

DISCOVER PHYSICAL/OS RESOURCES

FIND PROCESSES RUNNING

FIND TELNETS ASSOCIATED WITH PROCESSES

LOCATE INSTALLATION OF FOUND APPLICATIONS

DISCOVER RESOURCES OF MS. SERVER

ANALYZE CONFIGURATION FILES

EXIT

FIGURE 3B
350

ENTRY

APPLY LOCAL DISCOVERY AT EACH DEVICE

USE VENDOR INFORMATION, TO ID COMPONENTS

CONSOLIDATE DISTRIBUTED INFORMATION

CONSOLIDATE COMMUNICATION INFORMATION

EXIT

FIGURE 3C
ENTRY

SIGNATURE DEFINITION

DETECTOR

SIGNATURE CORRELATION

PROBES

TOPOLOGY STITCHING

EXIT

FIGURE 5
<table>
<thead>
<tr>
<th>Det.</th>
<th>App.1</th>
<th>App.2</th>
<th>...</th>
<th>App.6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Det1</td>
<td>1.0</td>
<td>.5</td>
<td></td>
<td>.3</td>
</tr>
<tr>
<td>Det2</td>
<td>1.0</td>
<td>1.0</td>
<td></td>
<td>.4</td>
</tr>
<tr>
<td>Det3</td>
<td>.5</td>
<td>.7</td>
<td></td>
<td>1.0</td>
</tr>
<tr>
<td>Det4</td>
<td>.75</td>
<td>0</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Det5</td>
<td>0</td>
<td>.2</td>
<td></td>
<td>.5</td>
</tr>
</tbody>
</table>

**Figure GA**
<table>
<thead>
<tr>
<th></th>
<th>App1</th>
<th>App2</th>
<th>...</th>
<th>Appm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Det1</td>
<td>1.0</td>
<td>0.5</td>
<td></td>
<td>1.3</td>
</tr>
<tr>
<td>Det2</td>
<td>1.0</td>
<td>1.0</td>
<td></td>
<td>1.4</td>
</tr>
<tr>
<td>Det3</td>
<td>0.5</td>
<td>0.7</td>
<td></td>
<td>1.0</td>
</tr>
<tr>
<td>Det4</td>
<td>0.75</td>
<td>0</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>e</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Det2</td>
<td>0</td>
<td>0.5</td>
<td></td>
<td>1.0</td>
</tr>
<tr>
<td>Det1</td>
<td>0</td>
<td>0.5</td>
<td></td>
<td>1.0</td>
</tr>
<tr>
<td>e</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Detm</td>
<td>0</td>
<td>1.2</td>
<td></td>
<td>1.5</td>
</tr>
</tbody>
</table>

**Figure 6b**
| Port: 9091  |  810 |
| Request: GET/HTTP/1.0 |  812 |
| Response: "Server: Websphere" |  814 |
| Process Name: /Websphere/Deployment/JAVA |  816 |
| Probe: WAS Domain |  828 |

**Figure 8A**

| Port: 9091  |  920 |
| Request: GET/HTTP/1.0 |  922 |
| Response: "Server: Websphere" |  924 |
| Process Name: /Websphere/Deployment/JAVA |  926 |
| Probe: WAS Domain |  828 |

**Figure 8B**
<table>
<thead>
<tr>
<th></th>
<th>APP SERVER</th>
<th>DEPLOYMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Port</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Proc 1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Proc 2</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 9.
WEB APP

CLUSTER

WEB APPLICATION
Server
(WAS)

HOST

SUM

ORBS

WEB APP

JDBC

JCA

WC

SCA

FIGURE 10A
Figure 10B
Figure 13
<table>
<thead>
<tr>
<th>appSignatureDefinition</th>
<th>Schema</th>
<th>Parameter</th>
<th>Label</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Detector Type</td>
<td>Process</td>
<td>command</td>
<td>Process Name</td>
<td>Oraserv.exe</td>
</tr>
<tr>
<td>service</td>
<td>ServiceName</td>
<td>Service Name (W2K)</td>
<td>ORCL</td>
<td></td>
</tr>
<tr>
<td>tcpports</td>
<td>port</td>
<td>TCP Ports</td>
<td>1521, 1810, 2481</td>
<td></td>
</tr>
<tr>
<td>Command line</td>
<td>arguments</td>
<td>Command Line</td>
<td>C:\oracle9\bin\oradim</td>
<td></td>
</tr>
<tr>
<td>File</td>
<td>path</td>
<td>Files</td>
<td>init.ora, tnsnames.ora</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Topology Description</th>
<th>Label</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>Class</td>
<td>DatabaseServer</td>
</tr>
<tr>
<td>Instance</td>
<td>Instance Name</td>
<td>ACCT-Receiver-DB</td>
</tr>
<tr>
<td>VendorAttribute</td>
<td>Vendor</td>
<td>Oracle Int. Corp</td>
</tr>
<tr>
<td>DescriptionAttribute</td>
<td>Description</td>
<td>RDBMS</td>
</tr>
<tr>
<td>MajorVersionAttribute</td>
<td>Major Ver.</td>
<td>9</td>
</tr>
<tr>
<td>MinorVersionAttribute</td>
<td>Minor Ver.</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Monitoring Action</th>
<th>Parameter</th>
<th>Language</th>
<th>Label</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>OraclePing</td>
<td>Perl</td>
<td>Health Check</td>
<td>OraclePing</td>
<td></td>
</tr>
</tbody>
</table>