Computer-implemented system for analysis, administration, control, management and monitoring of a complex hardware/software architecture

Description

Background of the invention

Many organizations base their business on networked computers, in which many application programs, which are called by users, access the resources (hardware or software or infrastructure) of a network of computers via middleware, operating systems and other computer program components. But IP-based (IP = Internet Protocol) access to resources in the computer network is also becoming increasingly widespread for implementing and executing business processes in an IT environment.

This situation also results from that fact that, for a specific application, computer programs containing the complete required functionality have hitherto usually been used. Thus it is possible that another application needs a similar or identical functionality. In this case this functionality was implemented in parallel or redundantly. The great advantage of this solution is the independence. Each application program for implementing a process can be enhanced and operated individually.

In contrast, work is increasingly done today according to the concept of redundancy-free applications, in which a software system is split up into reusable components. In addition, interfaces which are offered by these components make them more suitable for reuse / multiple use. The consequence and the aim here are redundancy-free and highly reusable systems, with complex dependencies of the reusable components. In other words, the required functionalities are implemented in autonomous components, which can be called and used from a different side. As well as the lower maintenance costs, it is thereby advantageously ensured that only one version of a functionality exists. This means that two application program components 'A' and 'B' obtain the identical result, which is supplied by only one autonomous component 'C'. In addition to the advantages of this approach, an obvious result is especially the diverse

dependencies caused by increased complexity. The following questions are among those arising: What happens with 'A' and 'B' if the autonomously developed component 'E' is modified? Which application program components are affected by a change? What happens if the autonomous component 'C' contains an error?

Technical problems underlying the invention

Especially with larger and also geographically widely distributed components of the computer network, individual resources are multiply provided even with the concept of the redundancy-free application software system, not least on grounds of performance and of minimizing outage of the computer network. It is essential here that the multiply provided resources actually are identical to one another. Otherwise, the result would be faults in operation that were difficult to trace: Imagine a computer program for a complicated computation process is only supposedly identically present on two servers in the computer network. An application program whose request for a computation is directed (routed) by the network protocol at one time to a first instance of the computer program, and at a different time to a second instance of the computer program, will return differing results, dependent on random factors that are practically untraceable. The situation is comparable with hardware components which are only apparently identical (processors, network channels etc.). The access of a first computer program over a heavily used network channel to a server with a second computer program for executing a computation or a database query can lead to a time-out for the requesting computer program, while the access of the first computer program over a less heavily used network channel to another server with a different instance of the second computer program returns the computation or the database query to the requesting computer program without problems (since within the expected timeframe).

Furthermore, it becomes increasingly difficult, as the dependencies of the hardware and software systems and their interdependencies become ever more complex, to recognize effects of a (hardware or software) change on other software or hardware.

A further problem is the fact that currently, in the planning and implementation of computer programs, these are often equipped with functionalities whose benefit/relevance for the actual use can only be included in the plans with difficulty.

Thus for example it is difficult to estimate the intervals at which certain functions or autonomous components are called, how long their runtimes are, what other autonomous components they in turn call, what data flows they cause / need, and similar.

EP 1 079 320 A1 describes a technique for the visualization of data, which is generated in a process-management-system. For this purpose, a number of software-agents are used, of which in particular a "Management Information Agent" reverts to a process-management-data in a database. Then, these data are visualized by means of a visualization-agent. The process-management-system is also based on software-agents. A data analysis can be carried out on the layer of arrangements among agents as well as on the resources layer. For example, information about the execution of processes, the behaviour of self-adapting agents and the utilization of resources within the system can be presented.

Ambras et. Al. "MicroScope: An Integrated Program Analysis Toolset", Hewlett-Packard Journal, Nr. 4, August 1988, Page 71-82, describes a program-analysis-system, which combines a plurality of analysis-tools in a common user interface. This system analyses the static structure of a program, for example interactions among functions of programs and variables. Further, certain predetermined events could be activated for the analysis of the execution of the program. It is possible to visualize the results of the analysis.

The US-Patent Nr. 5,748,881 describes a technique for the analysis of objects in a object-orientated distributed computer-system. A generic monitoring and display-system contains a generic data-acquisition-system, which adopts a suitable monitoring method on an selected object. Thus, the data determined by this method are displayed to the user.

The US-Patent Nr. 5,758,077 describes a monitoring-system to monitor distributed services in a computer-network. A host-computer is used as a service-monitor and contains a "Discovery"-function, which is forwarded to another host-computer and receives a set of requests, which refer to the function of the host as a server or as a client. Functional units, which are provided to the respective host, are generated from the responds. Further, the services are determined, which are provided by this host.

It is known for certain factors to be estimated within simulations, and the information from highly specialized software tools to be included in the planning, creation, installation, monitoring etc. However, these known software tools each return only limited, non-networked aspects.

Solution according to the invention

To solve these problems, according to the invention, a computer-implemented system is provided in a network of several node computers and a plurality of program components, which are distributed instanced on one of the node computers and have a predetermined interface in each case, it being specified according to the interface what inputs should be applied to address a functionality of this software program component and what outputs the respective functionality (this software program component) will supply after its execution, the system according to the invention providing a software program / program component Application Landscape, which by communication with these software program components records state information from software program resources and/or hardware resources in the computer network, namely about the architecture of the computer network, the configuration status of the hardware resources of the computer network, event and error messages of the hardware resources of the computer network, the development status or the version of the software program components, the runtime behaviour of the software program components, events and errors of the software program components, the data flow between individual software program components and/or hardware resources of the computer network, and/or introduction and change requirements and information for the hardware resources of the computer network or the software program components; the system according to the invention recording this state information with time, space, functional and/or structural descriptors, preferably storing it in a database, and, dependent on queries, outputting it in one or more standardized output formats.

According to the invention, an integral multi-stage approach is selected here:
(i) Even in the outline planning / structuring of a program component and its relationships to other program components, the relevant (static) data is electronically structured and documented.

(ii) This is continued in the programming / coding of a program component, this relevant data being accessed on the one hand, and on the other hand this data being refined in presentation and supplemented with function calls, parameter transfer, dynamic attributes etc..

(iii) During the lifetime of the program component, this data is also further supplemented with version information, runtime behaviour, etc..

(iv) At the runtime of the program component, this data is drawn upon, in order for the static and dynamic behaviour of the program component to be evaluated with its output log file entries by the computer program component Application Landscape, and verified and output in a suitable manner.

Especially in complex IT environments, this computer-implemented system allows a more efficient and safer exposure of errors, along with the analysis/correction of causes of errors, than the previous approach. In particular, this computer-implemented system according to the invention supports the architecture conception, the development, the integration and the debugging, the operation, the event and error management, and a monitoring of complex IT environments.

In other words, the invention supplies a concept, as well as standardizing all necessary / available information about the software, software versions, the hardware, data flows, development information, monitoring information etc., from the existing tools and from new ones, and, to manage the complexity, represents them graphically with their dependencies within a computer program component Application Landscape. A uniform static or dynamic data structure is used here for the entry of the IT assets (hardware, software, infrastructure) and their operation (processes, services, resources etc.). Data streams and requirements for their volume, the properties of their occurrence/appearance (times, from where to where), and the executions of programs are automatically entered and monitored. Using the data thereby obtained, errors and breakdowns are investigated and minimized, and dependencies between separate program components and/or hardware resources of the computer network are detected. Based on this, change requirement can be determined and changes can be designed. It can be detected, for example, whether a program component or a hardware asset is required from several sides within a certain timeframe, so that wait times ensue for the individual requirements. As a remedy, one or more further

instances of the program component can then be made available on different servers, for example, so that the wait times are reduced and the overall capacity of the system increases more than proportionately.

One result that this system can return is that dependencies between individual components or assets can very quickly be detected and analyzed, so that their influences on planned or desired changes are foreseeable; it is furthermore possible, with the analysis of the accumulating statistical data and the accumulating event data, to raise or optimize the efficiency of programs or the usage of hardware assets.

Depending on its complexity, a program component can be a simple function or procedure, a technical component or similar, but also a multilayered and sophisticated business service, which in turn accesses many simpler program components.

The output occurs in a form which reproduces the relationships to / dependencies on other resources in the computer network.

There can be a static, dynamic and/or multilevel output of this state information. The communication between the computer program component Application Landscape with the program components can take place in batch processing and/or in real time for the state information.

The display of the program component for which an operational error was communicated to the computer program component Application Landscape, as state information, is preferably together with the computer program components which are dependent on this program component, and/or with its involved hardware resources in the computer network. Computer program components dependent on this computer program component are for example those which supply information on request to the computer program component. Involved hardware resources are such as are called upon to execute the computer program component, or for its communication with dependent computer program components.

The display of the program component for which a user error was communicated to the computer program component Application Landscape, as state information, is preferably together with its dependent computer program components, current data records and/or its involved hardware resources in the computer network.

The display can include a measure of quality of service (QoS) for a program component, and performance statistics including the average response time (target – actual), the minimum response time (target – actual), the maximum response time (target – actual), the time of last access, the number of accesses per time unit, and so on.

Detailed description of the invention

The computer-implemented system according to the invention serves in complex IT environments for a more efficient and safer exposure of errors, along with the analysis/correction of causes of errors, than the previous approach. In particular, this computer-implemented system according to the invention supports the architecture conception, the development, the integration and the debugging, the operation, the event and error management, and a monitoring of complex IT environments. Events that are recorded, edited and prepared for graphic display, by the computer-implemented system according to the invention, cover the graphic visualization of a complex IT environment with its hardware, software and infrastructure components including its state, operational conformity (= governance), IT service management, development, interdependencies of the hardware / software components, software version management, those responsible (owner, service, support) for the individual components, and their relationships to one another.

As globally illustrated in Fig. 1, several software programs 1 ... n are present in an IT environment, and themselves access different hardware resources 1 ... p, and also each communicate with different program components 1 ... q, i.e. supply output data and receive input data over specified interfaces. Data is either exchanged directly, i.e. a software program needs a result of a computation and therefore sends the code for the required functionality to a library of functionalities, for example, along with the necessary input data for this functionality. The library selects the corresponding functionality and passes the input data. The input data is processed by the functionality

and the output data is returned from the library to the interface of the software program. According to the invention, both the software program and the program component in which the library is instanced notify the Application Landscape about this operation. Alternatively, the data exchange can also take place using files, which are stored in databases and accessed alternately for reading/writing. In this case too, both the software program and the program component send a corresponding message to the Application Landscape about their respective activities.

In the Application Landscape, these messages are recorded, sorted, stored in a database, evaluated using rules from a rule pool, and sent on request and dependent on the choice of output format, to a graphic output (screen, printer and/or plotter).

As illustrated in Fig. 2, the output of an object occurs always in the same view to enable clear and unique classification. An object can be any IT asset, a program, a hardware component, or an infrastructure component, or a property/description of the same. Depending on the desired representation, it is possible to choose between a detail view and a hierarchy representation, in which the object of interest is shown optionally as a subordinate object (detail view) or as a priority (hierarchy representation). The entry point into the choice of representation is determined in each case by the object of interest.

A further criterion for the choice of output format is the nature of the question concerned. The following questions are possible:

1.) Display of the cause of an error, and its dependencies, in a service interruption
    a. Which component shows faulty behaviour, and which services does this component use, and by which services is this component used
    b. Which component shows sub-optimal behaviour, and which transactions are affected by this
2.) Display of the dependencies and networkings (for orientation, training etc.)
3.) Display of the IT environment for performing analyses of influences
    a. Software
        i. Which components / which services are affected when a component fails

ii. Which components / which services are affected when a component's availability is restricted

iii. What is the effect on the components / services i, j, k from making the component m multiply available

iv. What time does a component need for supplying a service

b. Hardware / Infrastructure

i. Which areas are affected if the computer system X fails

ii. Which areas are affected if the network Y has only a reduced capacity

iii. What is changed if an expanded capacity is provided

Figs. 3-8 illustrate various representations, dependent on the choice of output format, illustrating from an organizational angle the domain of a software program, the program solution and the program components, and differing in the level of detail and focus of the view. In each case, a large window shows a magnified detail view and a smaller window shows an overview. Fig. 3 shows for example the detail view of the governance of a font-end component "Card banking" with its immediate environment, i.e. the components which access it and which it accesses. Fig. 4 shows for example a hierarchical view of the governance of several components with their immediate environment, i.e. the components which access them and which they access in a wider environment. Fig. 5 shows for example a hierarchical view of the IT Service Management of several hardware components with their immediate environment, i.e. the components which access them and which they access. Fig. 6 shows for example a hierarchical view of the IT Service Management of several software components with their immediate environment, i.e. the components which access them and which they access. Fig. 7 shows the highest-level component of a program solution, in this case "Calendar Host" in the development stage. Finally, Fig. 8 illustrates release stages (Software Release Management) of different program components at the business application detail level. It goes without saying that these views are only examples and further views can also be implemented within the conception according to the invention. The different views also allow individual aspects to be shown / hidden dependent on the entry point and on context-dependent filter categories ([Solution / IT Service], [Software Component], [Environment], etc.)

Fig. 9 illustrates the interaction in the computer-implemented system, in which software program components enter state information about the execution in log files (or agents) during their execution. These log files undergo statistical evaluation. In addition, all database accesses and service accesses to a database environment and to the other components are registered and checked in an event evaluation, a log being kept of the individual program components, the number of their calls, and also the program components calling them and the result of the processing (OK/NOK), duration of processing etc.. The version of the software program components, their runtime behaviour, events and/or errors connected with them, the data flow between individual software program components and/or hardware resources of the computer network, etc. are recorded and edited with time, space, functional and/or structural descriptors. This information from the statistical evaluation and from the event evaluation (in each case up to maybe 10 attributes for each event / statistical entry) is fed to a source code analysis. The results of the source code analysis are augmented with software development data and forwarded for output through the Application Landscape. In the Application Landscape, a save can also take place, so that the results can be output in one or more standardized output formats dependent on queries. The Application Landscape also contains a rule pool, using which it is determined what should be rated as a fault, when what data is accepted from the individual data sources or is queried by these, and so on.

Claims

1. Computer-implemented system in a network of several node computers and a plurality of program components, which are distributed instanced on one of the node computers and have a predetermined interface in each case, it being specified according to the interface what inputs should be applied to address a functionality of this software program component, and what outputs the respective functionality of this software component will supply after its execution, the system providing a computer program component Application Landscape, which by communication with these program components records state information from software program resources and/or hardware resources in the computer network, namely about

- the architecture of the computer network,
- the configuration status of the hardware resources of the computer network,
- event and error messages of the hardware resources of the computer network,
- the development status or the version of the software program components,
- the runtime behaviour of the software program components,
- events and errors of the software program components,
- the data flow between individual software program components and/or hardware resources of the computer network, and/or
- introduction and change requirements and information for the hardware resources of the computer network or the software program components;

the system according to the invention recording this state information with time, space, functional and/or structural descriptors, preferably storing it in a database, and, dependent on queries, outputting it in one or more standardized output formats, wherein

(i) in the outline planning / structuring of a program component and its relationships to other program components, the relevant (static) data is electronically structured and documented;

(ii) in the programming / coding of a program component, this relevant data is accessed on the one hand, and on the other hand this data is refined and supplemented with function calls, parameter transfer, dynamic attributes etc.;

(iii) during the lifetime of the program component, this data is supplemented with version information, runtime behaviour, etc.;

(iv) at the runtime of the program component, this data is drawn upon in order for the static and dynamic behaviour of the program component with its output log file entries to be evaluated by the computer program component Application Landscape, verified and output in a suitable manner.

2. Computer-implemented system according to Claim 1, in which the system standardizes necessary / available information about the IT assets such as software, software versions, the hardware, the hardware versions, data flows, development information, monitoring information etc. and, to manage the complexity, represents them graphically with their respective dependencies within a computer program component Application Landscape for evaluation, using a uniform static or dynamic data structure for the entry of the IT assets and their operation.

3. Computer-implemented system according to Claim 1 or 2, in which the system automatically records and monitors data streams and requirements for their volume, the properties of their occurrence/appearance and the executions of programs, in order to investigate and optimize errors and breakdowns with the data thereby obtained, and to detect dependencies between separate program components and/or hardware resources of the computer network.

4. Computer-implemented system according to one of Claims 1 to 3, in which a program component, depending on its complexity, can be a simple function or procedure, a technical component or similar, but also a multilayered and sophisticated business service, which in turn accesses many simpler program components.

5. Computer-implemented system according to one of Claims 1 to 4, in which the output occurs in a form which reproduces the relationships to / dependencies on other resources in the computer network, and/or includes a static, dynamic and/or multilevel output of the state information.

6. Computer-implemented system according to one of Claims 1 to 5, in which the communication between the computer program component Application Landscape

with the program components takes place in batch processing and/or in real time for the state information.

7. Computer-implemented system according to one of Claims 1 to 6, in which the program component for which an operational error was communicated to the computer program component Application Landscape, as state information, is displayed together with its dependent computer program components and/or its involved hardware resources in the computer network.

8. Computer-implemented system according to one of Claims 1 to 7, in which the program component for which a user error was communicated to the computer program component Application Landscape, as state information, is preferably displayed together with its dependent computer program components, current data records and/or its involved hardware resources in the computer network.

9. Method for execution in a computer network of several node computers and a plurality of program components, which are distributed instanced on one of the node computers and have a predetermined interface in each case, it being specified according to the interface what inputs should be applied to address a functionality of this software program component, and what outputs the respective functionality of this software component will supply after its execution, the system providing a computer program component Application Landscape, which by communication with these program components records state information from software program resources and/or hardware resources in the computer network, namely about

- the architecture of the computer network,
- the configuration status of the hardware resources of the computer network,
- event and error messages of the hardware resources of the computer network,
- the development status or the version of the software program components,
- the runtime behaviour of the software program components,
- events and errors of the software program components,
- the data flow between individual software program components and/or hardware resources of the computer network, and/or

introduction and change requirements and information for the hardware resources of the computer network or the software program components, is recorded with

time, space, functional and/or structural descriptors, stored in a database, and, dependent on queries, is output in one or more standardized output formats, wherein

(i) in the outline planning / structuring of a program component and its relationships to other program components, the relevant (static) data is electronically structured and documented;

(ii) in the programming / coding of a program component, this relevant data is accessed on the one hand, and on the other hand this data is refined and supplemented with function calls, parameter transfer, dynamic attributes etc.;

(iii) during the lifetime of the program component, this data is supplemented with version information, runtime behaviour, etc.;

- (iv) at the runtime of the program component, this data is drawn upon in order for the static and dynamic behaviour of the program component with its output log file entries to be evaluated by the computer program component Application Landscape, verified and output in a suitable manner.

10. Method according to Claim 9, in which necessary / available information about the IT assets such as software, software versions, the hardware, the hardware versions, data flows, development information, monitoring information etc., is standardized and, to manage the complexity, graphically represented with respective dependencies within a computer program component Application Landscape for evaluation, a uniform static or dynamic data structure being used for the entry of the IT assets and their operation.

11. Method according to Claim 9 or 10, in which data streams and requirements for their volume, the properties of their occurrence/appearance and the executions of programs are automatically recorded and monitored, in order to investigate and minimize errors and breakdowns with the data thereby obtained, and to detect dependencies between separate program components and/or hardware resources of the computer network.

12. Method according to one of Claims 9 to 11, in which a program component, depending on its complexity, can be a simple function or procedure, a technical

component or similar, but also a multilayered and sophisticated business service, which in turn accesses many simpler program components.

13. Method according to one of Claims 9 to 12, in which the output occurs in a form which reproduces the relationships to / dependencies on other resources in the computer network.

14. Method according to one of Claims 9 to 13, in which the output includes a static, dynamic and/or multilevel output of the state information.

15. Method according to one of Claims 9 to 14, in which the communication between the computer program component Application Landscape with the program components takes place in batch processing and/or in real time for the state information.

16. Method according to one of Claims 9 to 15, in which the program component for which an operational error was communicated to the computer program component Application Landscape, as state information, is displayed together with its dependent computer program components and/or its involved hardware resources in the computer network.

17. Method according to one of Claims 9 to 16, in which the program component for which a user error was communicated to the computer program component Application Landscape, as state information, is preferably displayed together with its dependent computer program components, current data records and/or its involved hardware resources in the computer network.

18. Computer program product, which is designed to execute the method according to one of the preceding claims if it is executed by a computer or a computer network.