

(19) World Intellectual Property  
Organization  
International Bureau



(43) International Publication Date  
10 September 2004 (10.09.2004)

PCT

(10) International Publication Number  
**WO 2004/077212 A2**

- (51) International Patent Classification<sup>7</sup>: **G06F**
- (21) International Application Number:  
PCT/IN2004/000023
- (22) International Filing Date: 29 January 2004 (29.01.2004)
- (25) Filing Language: English
- (26) Publication Language: English
- (30) Priority Data:  
118/MUM/2003 30 January 2003 (30.01.2003) IN
- (71) Applicant (for all designated States except US): **VAMAN TECHNOLOGIES (R & D) LIMITED** [IN/IN]; Pawani Plot, Near Vipul Apartment, Bhakti Marg, Mulund (West), Mumbai 400 080, Maharashtra (IN).
- (72) Inventor; and
- (75) Inventor/Applicant (for US only): **RAO, Vinayak, K.** [IN/IN]; A/4 Vishwakarma Jyoti, Subhash Lane, Malad (East), Mumbai 400 097, Maharashtra (IN).
- (81) Designated States (unless otherwise indicated, for every kind of national protection available): AE, AG, AL, AM,

AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NA, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, YU, ZA, ZM, ZW.

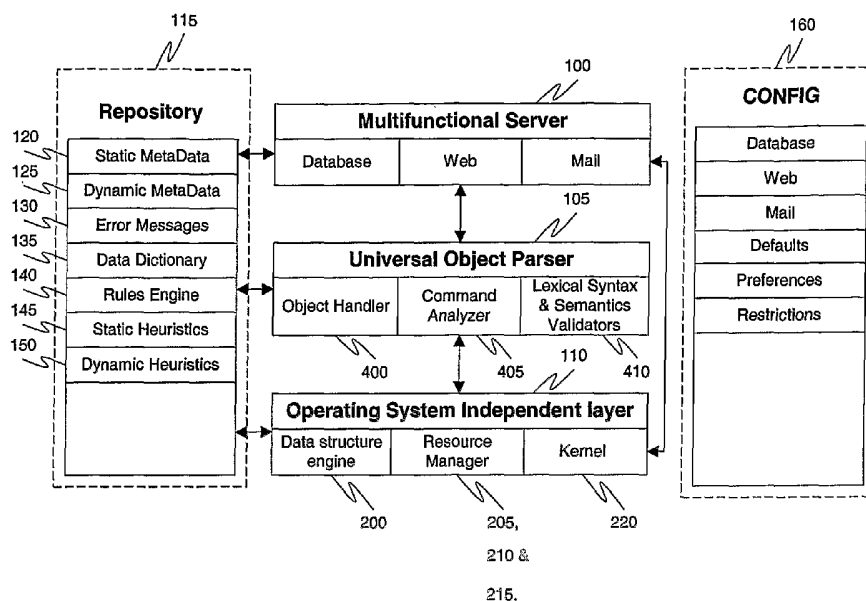
- (84) Designated States (unless otherwise indicated, for every kind of regional protection available): ARIPO (BW, GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IT, LU, MC, NL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

**Declarations under Rule 4.17:**

- as to applicant's entitlement to apply for and be granted a patent (Rule 4.17(ii)) for the following designations AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE,

[Continued on next page]

(54) Title: DATA SERVER INDEPENDENT OF COMMUNICATION PROTOCOLS, OPERATING SYSTEMS, FORMATS, FEATURES AND SYNTAXES



(57) Abstract: The present invention relates generally to the field of a data archival and retrieving mechanism, servicing clients irrespective of functionality, features, communication protocols, operating systems ("OS"), formats, semantics and syntaxes. More particularly the present invention relates to a system and method of data mapping and functionality mapping which enables an application specific server system to function as a database with the functionality extensions of middleware, web servers, mail servers etc. simultaneously allowing applications written for these servers to talk independent of operating systems, vendors or development tools without programming effort.



EG, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NA, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, UZ, VC, VN, YU, ZA, ZM, ZW, ARIPO patent (BW, GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IT, LU, MC, NL, PT, RO, SE, SI, SK, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG)

— as to the applicant's entitlement to claim the priority of the earlier application (Rule 4.17(iii)) for all designations

**Published:**

— without international search report and to be republished upon receipt of that report

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

**5 TITLE OF INVENTION**

Data Server independent of communication protocols, operating systems, formats, features and syntaxes

**BACKGROUND OF THE INVENTION**

- 10 Over the years, need based applications saved data the best way they could support them. The solution providers often did not foresee the rapid changes in technology that would follow, and have an impact on their own products and the scope of the projects they were involved in. Further, the increase in specialization of the skills led to the development of restricted, specialized knowledge and domains of work, rather than the best solutions required. This gave rise to heterogeneous data and data formats.
- 15 For example, applications like MS Excel, MS Word and MS Outlook could have been developed to save their data in a table or column format rather than formats such as '.XLS', '.DOC', '.EML' formats. Further, such applications could have been developed to use a presentation logic that would display a document or spreadsheet in a layout that would be easy for users to understand, while saving the data in a database format. In addition, applications were typically desktop or single user and as automation
- 20 progressed, the need arose to make such applications support multiple-users. As a result of this need, servers such as mail servers, web servers and database servers came into existence.

The most generic and popular methodology of sharing information and functionality is the client - server architecture, typically in which a machine designated as 'Server' has the best hardware and software,

25 configured to be shared across clients. Generally the resources that are sharable across clients includes the network hardware, the Hard Disk Space, the Central Processing Unit's processing time, server memory and data generated by application(s). The application-specific functionality classifies these servers as database servers, web servers, mail servers etc. Each of these servers have different functionality, which understands specific sources of request patterns and responds in a specific pattern.

30 Further, technology evolution created different architectures, syntaxes, features and vendors for these servers. The disparity in the hardware includes the endian dependencies of the central processing unit. These disparities in both the data and the functionality were due to different development tools being

5 used by different vendors. Moreover, the underlying operating system differed along with the operating system specific features used in implementation. For example, consider a system with Microsoft Windows NT as the OS has Microsoft SQL Server 7 or SQL 2000 installed on it. Any application or database instance uses the underlying file system of the OS for persisting the data. Further such examples are LDAP on Linux, HPFS on IBM OS2. Some of the additional reasons were the  
10 implementation design, which captured or used data and the mode of communication used to capture data. Further, the associated hardware used for communication and archiving data differed and there were limitations of resources such a software, hardware, time and economics. The ignorance to changes in technology with respect to skill sets of users and tool manufacturing industries also contributed in a significant manner. In other words alternative paths were available, but were never  
15 used.

However with globalization and increasing mergers and acquisitions, the need arose to built applications to communicate, exchange, share data and share functionality too, based on these different technologies. There did exist solutions in the form of several application tools, which solved the above-  
20 mentioned disparities in architectures, syntaxes, features and vendors. However, these application tools were too complex to use and required reprogramming effort because they never supported seamless migration or data exchange without compromising certain aspects of data and server architecture. Further, they were expensive and required tremendous resources and technical expertise. Some of the other known solutions include middleware, which allowed data and functionality exchange between two  
25 or more applications for more than one database and operating system servers. However, once a middleware is written for a particular application, it cannot be used for another, as it is very application specific. In addition, middleware had a limitation because no middleware could map functionality, features and syntax integration across functional servers such a database servers, web servers and mail servers without reprogramming effort. Further middleware requires knowledge of the functionality  
30 and the data format of applications required to communicate and if needed the applications need to have an import export exchange format available.

5 Accordingly, a need exists for a system and method by which we can make applications and their data talk and interact on a common platform, in a common format so as to understand, translate, respond and behave as per application requests without any changes. There also exists a need for a solution, which functions as a data server, which is independent of operating systems and can communicate with other relational and non-relational database management systems despite of differing syntaxes and  
10 data formats. There exists a further need for data server to function as a middleware, manipulate data from different sources, and carry out data and application migration.

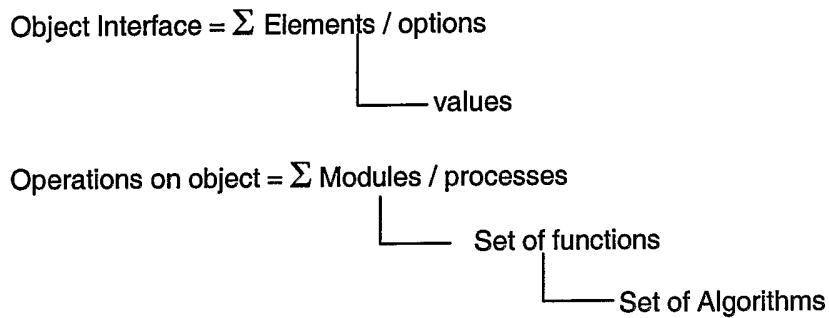
### SUMMARY OF THE INVENTION

15 To meet the foregoing needs, the present invention provides a software-implemented process, system, and method for use in a computing environment. In the history of computing the best multi-user server application with the capability to manipulate client data requests is the database server. Typically a database server works on certain objects created to service set of functionalities expected for a set pattern of data with a predefined scope of features. Also any database server is architecturally best  
20 designed to search, sort, manipulate and archive business intelligence and application objects. In the present invention, a database server is perceived as a collective functionality of the server sub-components delivering individual functionality in a linked or relational way. Typically any application built on the database server creates or uses these sub-objects as per the business intelligence required. Applications built similarly on like database servers from different vendors still could not communicate  
25 because either the sub objects supported by the server architecture had functionality / feature / syntax differences or the object itself was non existent in the server architecture.

The present invention decomposes server functionality into component objects, maps functionalities, features, related syntax and semantics under various architectural constraints so that this decomposed  
30 object entity encapsulates all vendor deliverable functional features and accessing mechanism criteria's without compromising any performance or functional issues with respect to any resource usage where the state entities of the objects are tracked. This methodology "OOOV" (Operation, Object, Option

5 Value) ensures that per object we have a virtual entity derived, which we term "Object Interface". The interface specification designed has member "elements" which are based on the features and functionalities expected from the object under various criteria's of operations, which can be performed on the object i.e. each element primarily maps to an "object option", which serves to contain "option values" defined by user in a query at run time. The methodology creates a functional linkage or map, 10 which associates a "process" or a callback modules / function to service the member element, which can have arguments as values provided by the user in input query. The decomposition also derives sequences of "processes" or modules to be executed with a set of options for an object, to deliver the operations specific functionality. The invention also "maps" or quantifies server resources associated with each "state entity" of the decomposed object under various stages right from beginning of an 15 operation till completion. Basically it means we perceive any object under any state of transition – right from creation or instantiation of the object till destruction as a composition of various resources i.e. RAM (virtual or in a static state of transition), DISK (final persistence or in a state of secondary swapped storage), CPU (in a state of dynamic transition), Network (in a state of communication), Timer (in a state to enforce predefined harmonics during transition). The methodology of perceiving an object as a 20 resource composition (RAM/DISK/CPU/Network/Timer) in various ratios helps to achieve best resource utilization in a concurrent environment. The objects and its interface specifications can be dynamic and defined in the Look Up Table changeable by the user. The entire design is based on state machines and modules comprising of various events communication via messages that is it is event driven using Finite State Machine (FSM) concept the functionality is broken down into a series of events scheduled by 25 kernel. This design approach of execution of modules or processes in an event driven mechanism also optimizes runtime the code execution as chaining of these state machine entities to execute callback function or modules to service each request as per options and values specified in the input query can be decided runtime as per user input query for delivering an option functionality the callback function may choose more than one algorithm ("workflow") to deliver the functionality in optimum resources. 30 Hence the program decided the execution flow rather than the programmer who has coded the functional logic. The design of these callback functions strictly maintains ACID (Atomicity, Consistency, Isolation, Durability) properties and guarantees that any dynamic sequence change does not affect

functionality. These callback functions also maintain the state of the object (in RAM or Disk) and its linkages with other objects during phases of transition. This approach of object and object interface design gives guaranteed portability of applications written and developed for the any functional servers independent of vendor, operating systems, syntax and semantics, server architecture, features and functionality, without reprogramming effort and no additional resource requirement.



## BRIEF DESCRIPTION OF THE DRAWINGS

The various objects and advantages of the present invention will become apparent to those of ordinary skill in the relevant art after reviewing the following detailed description and accompanying drawings, wherein:

Fig 1(a) is a block diagram of the architecture of the data server.

Fig 1(b) is a screenshot depicting the Error Repository.

Fig 2 is a block diagram depicting the configuration of the OS Independent Layer.

Fig 3 is a flow diagram depicting the functioning of the OS Independent Layer.

Fig 4 is a block diagram depicting the configuration of the Universal Object Parser.

5

Fig 5 is a flow diagram depicting the functioning of the Universal Object Parser.

Fig 6 is a block diagram depicting the configuration of the Multifunctional Server.

10 Fig 7(a) and 7(b) is a flow diagram depicting the functioning of the Multifunctional Server.

Fig 8 depicts whereby the end-user can predict the pattern of resource usage in the most effective way and can tune, customize or derive a newer server from the framework of events and state map entities.

15 Fig 9 illustrates the State Machine Map of an Agent-wise list to display the current settings and can be used to configure the events corresponding to the states.

Figure 10 depicts a screenshot to accomplish a complete multi-functional server as a set of agents doing / executing specific tasks, Resource wise.

20

Figure 11 depicts a screenshot to accomplish a Database server as a set of agents doing / executing specific tasks, Resource wise.

25 Figure 12 depicts a screenshot to accomplish a Web server as a set of agents doing / executing specific tasks, Resource wise.

Figure 13 depicts a screenshot to accomplish a Mail server as a set of agents doing / executing specific tasks, Resource wise.

30 **DETAILED DESCRIPTION OF THE INVENTION**

5 While the present invention is susceptible to embodiment in various forms, there is shown in the drawings and will hereinafter be described a presently preferred embodiment with the understanding that the present disclosure is to be considered an exemplification of the invention and is not intended to limit the invention to the specific embodiment illustrated.

10 In the present disclosure, the words "a" or "an" are to be taken to include both the singular and the plural. Conversely, any reference to plural items shall, where appropriate, include the singular.

Referring now to the drawings, more particularly Fig. 1, there is shown a System of data archival and retrieving mechanism, servicing clients irrespective of functionality, features, communication protocols,  
15 operating systems, formats, semantics and syntaxes. The System is a three layered structured comprising of a Multifunctional Server 100, a Universal Object Parser 105, an Operating System (OS) independent layer 110, Repository 115 and Config module 160 to configure Database, Web and Mail etc.

20 The Multifunctional Server 100 is configured to support the features and functionalities for every smallest sub object entity in the data server architecture. The Multifunctional Server 100 analyses the operations for the object with options and parameters specified in the request command, executes them taking care of multi-user constraints and generates the resultant response buffer. The Universal Object Parser 105 is configured to validate and parse syntaxes and semantics, which are vendor specific or OS  
25 specific to create an Interface Object for the Multifunctional Server 100. The Universal Object Parser 105 is also responsible for the sub objects, which are specific for vendor server architectures. For example Synonyms and Tablespace are database objects supported in ORACLE but similar sub objects are missing in SQL server. The Universal Object Parser 105 architecture consists of a server object dictionary, which encapsulates all sub objects independent of vendors and their related reserved SQL  
30 words. The OS independent layer 110 is configured to isolate any operating system dependencies, which are used by applications specific for the OS, including network protocols, file formats or any kernel objects, which form a part of the OS architecture.

5

The Repository 115 is a collection of compressed or hashed persistent index data, so as to achieve fastest way of accessing whenever needed. This repository is used for mapping search algorithms according to the resources, encryption algorithm, includes meta-data for the database, Error ID of an errors, Heuristics tables etc. The logical segregation of data as per persistence is depicted in the figure 1 that handles the sequencing and isolation of varying or dynamic data to the end so as not to disturb the existing static data structure.

10

Static Metadata 120 is the basic fixed pattern of persistence, which is the design assumption when creating a database with bare minimum necessities of compulsory objects. The location of these assumed objects in terms of blocks and extents are hard coded with respect to start of the database file.

15

Dynamic Metadata 125 is a delimited SQL file, which is parsed and executed at run time to create other metadata objects based on the assumption of the fixed static base objects. The reason of this design consideration was that the basic architecture supported dynamic agent creation and binding as per various functional servers may require newer SQL scripts to persist data as per each functional server added or plugged later.

20

Error Messages 130 is basically a lookup table of error strings and its presentation logic, which is partly static and has some dynamic contents derived later as per Object, Operation, Option and Value (OOOV) principle. Consider an option(s) specified in the user query failed and triggered an error. This error could either be a logical, Environmental or User Error. Figure 1 (b) is an Error Repository screenshot. As explained in this architecture methodology wherein each option is associated with a callback module, such a callback module gets executed as specified in the query. For example, If the user fired a query such as "Create Database XYZ size 10G; " then as per the OOOV principle where the Operation is Create, the Object is Database, the Options are "Database Name" with Value XYZ and "Size" with Value 10M. As per the state machine design an associated module is executed relevant to each option hence in case function named F1() delivers size option and fails due to "insufficient disk

25

30

5 space" or "security rights" the error code 174 generated by the code serves to highlight the exact operation on an object with a specific option which failed and what parameters (10G) in the command was the cause of error. Hence the end user gets a very precise description of the cause of error and the developer gets the precise location in the source code that clearly isolates the module, which serviced the operation and object 170 for related value. The error code 174 generation and sequencing has been  
10 designed considering various operations (such as CREATE / ALTER / DROP / SELECT etc), Objects 170 (such as Database / TableSpace / Table / Index etc), the type of notification 186 required (such as a Message Box, Error Log, Beep etc) when the error is triggered with various levels of warning intensity right from a basic popup to halting of the server. These values of operands are defined using various enumerations and bit-wise error codes are generated which can be decomposed into type of notification  
15 186, operation 190, object 194, module 198 in which the error occurred and corresponding value of string used for formatting user parameters.

Data Dictionary 135 and Rule Engine 140 are typically collection of reserved words and any other syntax and semantic script repository, which is user definable and can be used to parse, SQL, XML, or  
20 any web scripts.

Static Heuristics 145 is a Look-Up Table (multi-dimensional matrix) of resource utilization of a specific data structure algorithm versus performance or versus degradation of performance under extremities of resource and loads of data and specific data types. These may include searching, sorting, hashing,  
25 compression and encryption of data streams.

Dynamic Heuristics 150 is the statistics that are gathered related to real-time operation on specific server objects and are generally persisted virtual tables which maybe required for historical analysis. There are many database rule engines which optimize or re-write queries to enhance performance. The  
30 statistical conditions and rules to optimize the query rewrites are defined here are based on which database query operations are to be performed.

5 The Config file 160 is used to configure the servers or server instances. As per the settings defined in the config file 160, the various servers use this defined information on booting up. In the event no value is defined for a particular parameter in the config file 160, then the system takes the default value. As depicted in the figure, the config file 160 further includes configuration details as per various functional servers. For example this file includes details of Database, Web and Mail server.

10

As illustrated in Fig 2, the OS Independent Layer 110 comprises of a Data Structure Algorithm and manipulation engine 200, Resource Manager such as a Memory Manager 205, a Network Manager 210, a Disk Manager 215 and Kernel Objects 220. The OS Independent Layer 110 encapsulates all the hardware resources required by any application such as Random Access Memory (RAM), Hard Disk  
15 Drive (HDD), network, Central Processing Unit and any kernel objects used for implementing multi-user functionality. The OS independent layer helps to achieve OS independence.

The Memory Manager 205 is configured to support basic memory allocation and de-allocation functions with extended support for virtual secondary storage. Hence allocations larger than the memory available  
20 are easily supported. The Memory Manager 205 has a built-in debugger which tracks any loose unreleased allocations and even defragments the non-contiguous memory blocks. The Network Manager 210 is configured to support all network protocols used for client-server network communications independent of the nature of client. For example, database clients use Named Pipes, TCPIP, IPX etc, web clients use HTTP / FTP etc and the mail clients use SMTP, POP. The Disk  
25 Manager 215 is configured to allow all extended file seeks, reads, writes for large files (64 bit). It helps optimize disk access and manages disk caching for most frequently used data locks.

The OS Independent layer 110 also supports OS portable Kernel objects (KO) 220 like processes, threads, events, queues, mutexes, semaphores, timers etc. These KO are independent of OS  
30 architecture and endian dependencies. The entire architecture implements FSM based kernel objects, which work on the principle of co-operative multithreading. Wherever the server sub object functionality / feature demands synchronous and asynchronous behavior the implementation design uses these KO

5 220 in a unique or hybrid model of asynchronous or synchronous architecture. For managing data manipulations across various concurrent request sessions the OS Independent Layer 110 supports every possible data structure algorithm in a unique wrapped way which isolates algorithm specific functionality from the pattern of arguments and data expected. Hence in case a better algorithm is devised for searching, sorting, hashing etc there is no change in the layers tiered over the OS layer. So  
10 without disturbing existing functionality any future enhancements in algorithms need not require a recompilation of tiers written over the OS Independent layer 110.

The OS Independent Layer 110 accepts any request packet irrespective of the source of generation. It is responsible for the compatibility of the major and popularly used network protocols widely supported  
15 by most OS. Once the protocol compatibility is achieved any request reaches the multifunctional server. The Network Manager 210 in the present invention is designed to support almost all protocols irrespective of server functionality. For example Database servers usually use Named Pipes for small number of clients. Generally the default protocol is dictated by the OS on which the server functionality is supported like TCP/IP for Linux, IPX / SPX for Netware, NETBEUI for Windows XX, AppleTalk for  
20 Macs. Web servers have HTTP and FTP based communication support, Mail servers uses the SMTP / POP protocol.

As illustrated in Fig 3, as soon as the Multifunctional Server 100 is initialized 300, the Network Manager 210 checks the hardware and associated protocols bound with the hardware 305. When a Client sends  
25 a request 310 to the Server 100, the OS Independent Layer 110 analyzes the request as per the source of request 315, to identify the nature of the Client to ascertain whether the request has been sourced from a database client, an ODBC client, web clients or a browser. Once the source of the request is analyzed, the OS Independent Layer 110 isolates the Protocols and Standards of the request and identifies the limitations of the Client from which the request is sourced 320. When the OS Independent  
30 Layer 110 successfully receives the request, it is verified for data integrity 330 by the Cyclic Redundancy Check method as per the source of request. On verification, if the request cannot be sufficiently verified, an error code is generated 325 as per the operation and the object. The OS

5 Independent Layer 110 then proceeds to isolate the source Client and verify whether the request from the Client is an existing Client 340 with which the Server 100 already has an established interactive session or whether the Client is a first-time Client. In case the request is not from an existing Client, the OS Independent Layer 110 proceeds to allocate Server 100 resources 345 such as memory block (Random Access Memory) or persistent data (Hard Disk Data) to the request till the session instance is  
10 active. The OS Independent Layer 110 then proceeds to create Kernel Objects 350, which are associated with the Client session instance to manage any data transactions between the Client and the Server till the session dies. The OS Independent Layer 110 then proceeds to pass the request to the Parser for further processing 360. However, on isolation of the source Client and verification, if the Client is found to be an existing Client, with which the Server already has an established interactive  
15 session, then the OS Independent Layer 110 associates the machine identification, session identification, transaction identification and the query identification of the Client, with its existing session 365. The Server then proceeds to pass the request to the Universal Object Parser 105 for further processing 360.

20 As illustrated in Fig 4, the second step is to analyze and respond to the query in the format expected so that the client applications written for a specific server do not feel any change and continues to work on the server as if was using the original features and syntaxes of the vendor server. where the Universal Object Parser 105 comes into picture. Major issues of portability arise as apart from syntax / semantic differences the objects and features supported by the server cause porting discrepancies.

25 The Universal Object Parser 105 comprises of an Object Handler 400, a Command Analyzer 405 and a Lexical Syntax and Semantics Validator (LSS Validator) 410. As soon as request packets are successfully received, the Universal Object Parser 105 comes into play to process the request. The Object Handler 400 is configured for each sub object supported by the server functionality. The Object  
30 Handler isolates the vendor dependent syntaxes, semantics, object features. Further, any future enhancements in these sub object(s) supported can be incorporated in the Object Handler 400 without disturbing the Server architecture. The Command Analyzer 405 is configured to isolate the nature of

5 request and is also responsible for associating it with any previous session data if needed. The Lexical Syntax and Semantics (LSS) Validator 410 is configured to validate the request for syntax and semantics specified irrespective of vendor syntax.

As illustrated in Fig 5, the requests can be sent from the Server or the operating system independent  
10 layer to the Universal Object Parser 500. The options specified in the request are counted 501, if there is more than one option 502 then proceed to analyze option in the query with respect to objects and operation 503. Further after analyzing the option 503, analyze the option values if any as specified 504. Further mapping the options to object interface's elements as per syntax map defined in the LUT 505. After this mapping is complete, then decrease the count 506. After decreasing the count 506, then  
15 check whether the count is greater than zero 502. In the event the count is zero 502, then assume the defaults for the remaining values 507. Further after assuming the remaining options 507, then the Lexical Syntax and Semantic (LSS) Validator proceeds to validate the request to check whether it is lexically compliant 509. If the request is found to be lexically non-compliant, the LSS Validator 410 proceeds to generate an error code 510 based on the operations and objects. However, if the request if  
20 found to be lexically compliant, the LSS Validator 410 proceeds to check the syntax, which have been specified, irrespective of vendor syntax 515. If the request is syntactically non compliant, the LSS Validator 410 proceeds to report an error by generating an error code 510. However, if the request is found to be syntactically compliant, the LSS Validator 410 proceeds to check for the semantics of the request 520. If the packet data is found to be semantically non-compliant, then the LSS Validator 410  
25 proceeds to generate an error code 510. However, if the request is found to be semantically compliant, the LSS Validator passes the request 525 to the Command Analyzer 405. The Command Analyzer 405 isolates the nature of the request 530 and then checks if the request is associated with any previous session data 540. If the request is not associated with any previous session data, the Object Handler 400 proceeds to create 545 an Object Interface for each sub object supported by the Server  
30 functionality. However, on checking if the request is associated with any previous session data, if the Command Analyzer 405 finds that the request is associated with any previous session data, the Command Analyzer proceeds to associate the options and values as per the object and operation

5 specified 550. The Object Handler 400 then proceeds to create an Object Interface for each sub object supported by the Server functionality 545.

As illustrated in Fig 6, the Server 100 supports sub-object features and consists of different agents, which support generating responses. The Dispatcher Agent 600 is configured to wrap protocol layer and  
10 creates the response buffer in the format expected as per the nature of client (for example either in ODBC / http / ftp formats) and is also responsible for cursor management caching requests and coordinating bandwidth distribution based on nature of query, protocol limitations and response priority, size of data etc.

15 The Scheduler Agent 605 is configured to schedule the various functions carried out by the Server 100. The Scheduler Agent 605 forms the kernel of the Server 100. The Scheduler Agent 605 performs the tasks of getting the environment status, checking for hardware and OS restrictions, starting the Server 100 boot process and communicating with all the other agents in the network. The scheduler agent combined with the OS Independent Layer 110 schedules and creates KOs 220 across OS architectures,  
20 which helps the Server and data generated migrate across OS. The Server 100 is purely based on FINITE STATE MACHINE approach hence any agent or thread communication is via messaging scheduled by the Scheduler Agent 605, which raises events and these events are implemented as per object features and functioning required. The implementation takes care of deadlocking and resource conflicts using object specific semaphores and event chaining mechanism hence one gets a very  
25 effective CPU utilization through co-operative multithreading. It is also responsible to dynamically activate any functional agent based on need and configuration.

The Network Agent 610 is configured to receive and transmit concurrent requests and responses, irrespective of their protocol, with optimal utilization of available resources. The Network Agent 610 is  
30 responsible for reading incoming data from one or a plurality of Clients, reading and writing data and establishing a connection with a Server 100 irrespective of its functionality. These could be any clients irrespective of type of clients, their protocol and protocol functionality.

5

The Disk Agent 615 is configured to map patterns of data irrespective of the server protocols and data formats in which requests are received, in a relational database management system. The Disk Agent 615 is responsible for reading and writing different patterns of data from a database, initializing log files, writing logs into log files and to persist committed data blocks. The Disk Agent 615 is primarily responsible to read or persist any logical request / response data taking care of data integrity using journaling mechanism to guarantee fault tolerance and disaster recovery. It is also responsible to optimize disk access and manage secondary storage memory in case any query requires more storage space than available RAM for processing.

10

15

The Timer Agent 620 manages any periodic event or resource monitoring across process running concurrently on the OS that is System Monitor (SMON) and across kernel objects within the server instance that is Process Monitor (PMON). The present invention uses co-operative multithreading the entire kernel scheduling is asynchronous but a lot of tasks are time bound. Hence timer agent triggers the events and the sub objects that required time specific functionalities. This makes our architecture a hybrid model or asynchronous and synchronous event driven architecture. It also manages periodic data and cache flushing and hence delivers checkpointing functionality.

20

25

As illustrated in Fig 7, when a request is sent 700 from the Universal Object Parser 105 to the Server 100, the Server 100 checks if the request from the Client is an existing Client 701 with which the Server 100 already has an established interactive session or whether the Client is a first-time Client. In case the request is not from an existing Client, the Server 100 proceeds to allocate Server 100 resources 702 such as memory block (Random Access Memory) or persistent data (Hard Disk Data) to the request till the session instance is active. However, on isolation of the source Client and verification, if the Client is found to be an existing Client, with which the Server already has an established interactive session, then the Server associates the machine identification, session identification, transaction identification and the query identification of the Client 703, with its existing session. The Server 100 then proceeds to isolate the operations from the request 704 and checks if there exist objects in the request 705. If there

30

5 do not exist objects in the request, it generates an error code 706. If there exist objects it connects using the objects 707. However, if there do not exist objects in the request, it connects using an object environment 708. Any command or operation to be executed on any server object requires an environment prerequisite – which maybe related to authentication, operation rights etc For example, in a query void of server objects. For example “CONNECT INTERNAL/XYZ”. When parsed the string  
10 passes syntax validation but is void of any object specified on which operation “CONNECT” has to be performed.

The Operation Validator classifies it as a valid operation “CONNECT” with option “INTERNAL/XYZ”. Therefore, if the objects are missing, the missing entity OBJECT for the operation “CONNECT” is  
15 assumed to be the Server itself as CONNECT operation can be executed only on objects – SERVER (the machine configured to work as server) and Database (only if the server architecture supports multiple database options per server instances). This is also a valid proposition only when there is a concept of a single server environment. In case the network environment has more than one server supporting the same functionality, the connect command has to be very explicit like “CONNECT/  
20 APPDATA@INTERNAL/XYZ”. The Server 100 then proceeds to translate the Options or attributes 709 and classifies the names of the database, users and passwords for a specific server in a multi-server environment. The option value validator 710 verifies permissible characters and lengths of username or password permitted 711. If it finds that the same are not permitted, it proceeds to generate an error code 706. The value translator 712 converts the strings to a case (upper / lower) as required and  
25 verifies if the value is valid 713 with requisite parameters is executed the Scheduler Agent 605. The translated value 712 is then verifies if the value is valid then using the License Manager 714 check the purchasing restrictions of the server product for the operation, count of operations, options and values of options specified for the object (ex: concurrent users license, database size permitted etc) and next the operation is executed 715. The request / privilege validation 716 verifies whether the user name  
30 specified in the operation is valid and has a permissible scope to execute the operation. If it finds that the user name specified in the operation is not valid or does not have the permissible scope to execute the operation, it generates an error code 706. However, if the username specified in the operation is

5 found to be valid and has a permissible scope to execute the operation, it carries out metadata/Object schema validation 717 as per the scope of the identifier resolved by the object attribute translator or it generates an error code 706. In the event the operation is analyzed 718, then proceed to check for available resources 719. In the event the operation analyzed is not valid then the error handler 706 is gives an error message.

10 The Multifunctional Server 100 then proceeds to analyze if there are any resources available 719 and verifies 719 if is not available then an error code 706 is generated and checks if the constraints permit resource acquisition 720. If not it then proceeds to check if there are any available resources or resources can be acquired 721. If it finds that there are no available resources, it generates an error  
15 code 706. The Multifunctional Server 100 then proceeds to check for availability of resources and checks for the allocation or acquisition of resources 721. It then proceeds to verify if there are no resources available and if resources cannot be acquires, it proceeds to generate an error code 706. The Multifunctional Server 100 then proceeds to check for shared resources 722. If there exists shared resources, it checks if there is a lock available 723. In case a lock does not exist, it waits for a command  
20 timeout 724 and then sends a timeout error 725. If there is a lock available, it locks the partial or full object as per query option 726. However, if the resources are not shared, it acquires the required resources and proceeds to execute the requests 727. Further the checking is carried out for ascertaining the dependencies that are to be executed prior to the command execution 729. In the event there are no pre-dependencies, check whether dependencies has to be derived, evaluated or persisted  
25 730. In the event when no prior dependencies have to be derived, evaluated or persisted then proceed to execute pre-dependencies 731. In the event of pre-dependencies existing 729 then execute them 731. After the successful execution of pre-dependencies 731, the command is executed and the results are calculated 732. Further the post dependencies are executed 733 and after the successful execution 734. The result is checked whether it requires persistence 735. If persistence is not required then  
30 prepare the notification buffer 736. Also after the dependencies are derived, evaluated or persisted 735 then the notification buffer is prepared 736. This notification buffer then proceeds to notify results or status 740. If the event the result requires persistence 735 then prepare the result buffer 737. Then the

5 result is persisted as per the transactional needs 738. On successful persistence 739, the notification of the result or the status is carried out 740. After the result or status notification request processed and responded 741.

Figure 8 illustrates the server configuration map. The configuration file comprises of server tuning and customization parameters. The figure depicts a resource such as a disk being consumed by a server instance across multiple databases. The figure depicts the graphical interpretation of the server resources allocation irrespective of concurrent server functionality instances. The left-hand side of the figure 8 lists the configurations details such as the running instances including the Mumbai HO 800 and Bangalore Branch 805. Each of these Instances further shows settings. Under the Mumbai HO 800 instance, the settings is possible for resources 810 or for functionality 815, various resources such as Disk 820, Memory 830, CPU 840, Network 850 and Timer 852 etc as depicted in the figure. Similarly other parameters like memory in various functional needs like cache or static allocation or rollback segment can be tuned. The best part of configuration is tuning of each agent specific resource usage and its variance in performance with a specific server load. The figure depicts that the Database server instance named FactoryDB 855 that is configured as depicted in the figure. The FactoryDB 855 has the following database files 860 i.e. the Production table, QualityControl table and Warehouse table. The filenames, filepath, filesize, InitialSize, Next Size and AutoExtend are explicitly defined. The Log files 865, RollBack Segment files 870 and swap files 875 are shown. Further the right bottom image of the figure depicts the percentage of Database disk utilization 880 by the various database files.

25 As depicted in the figures 8, this design approach whereby the end-user can predict the pattern of resource usage in the most effective way. This prediction of the resource usage can tune, customize or derive a newer server from the framework of events and state map entities provided by the architecture design.

30 Based on licensing policy or the query functional need the agents can be invoked and activated dynamically or can be swapped and made defunct. This optimizes resources and saves a lot of CPU

5 activity from doing or executing functionality, which currently is undesired. The event chain, which is sequenced as per the agent design can be changed dynamically as per the query operation, object or options specified.

Figure 9 illustrates the State Machine Map of an Agent-wise list 900 to display the current settings and  
10 can be used to configure the events 905 corresponding to the states 910. As shown in the figure 9 the breakdown of various functional server or agents and associated sub objects (i.e. for database the sub-objects are tablespace, Table, Index etc) and subsequent associations of these objects into features and functionalities (as per OOOV) are mapped as a set of modules executed as events 905 as per options specified by the user during the query execution. These functional patterns across vendor  
15 syntaxes and server features are finally using only four resources - Disk, Network, RAM and CPU. Hence using the OOOV principle of Operation, Object, Options and Values, finally they were mapped to these resource utilizations and patterns derived. These functional patterns were mapped in a state diagram and their sequencing was saved in a repository outside the server in a Repository 115. There are resource specific agents to handle set of resources such as Disk, Network, RAM, CPU and other  
20 agents that as per operational functionality demanded by the various servers needs having events 905 and modules 915 to consume CPU and deliver the desired output. These agents are actual functional derivatives and each agent modules, if sequence and functionally wired (event chain) for handling specific operations and objects. The figure illustrates the Scheduler Agent 920 having the events 905 and states 910 as depicted in the diagram.

25 Figure 10 depicts a screenshot to accomplish a complete multi-functional server as a set of agents doing / executing specific tasks, Resource wise 1005, resources 1010 like DISK, N/W, MEMORY, TIMER are common agents which handle operations like disk management, concurrency and connection pooling, memory and transaction management and synchronous tasks like checkpoint,  
30 scheduled jobs etc respectively irrespective of any functional server.

5 The CPU operations which deliver specific functionalities as demanded by the relevant server such as Database 1015, Web 1020, Mail 1025 etc are further sub divided into agents delivering / rendering specific tasks such as DML agent, Index agent etc. Further three figure 11, 12 and 13 depict these functionalities as a set of three diagrams which have nodes of different functional servers expanded to display associated set of agents programmed to deliver server specific functionalities.

10

There are various common agents 1030 which have functional competence irrespective of server functionalities such as scheduler, dispatcher etc.

The right side of the figure 10 shows the relationship between various states / events and modules  
15 associated to deliver agent specific functionalities. These have been designed and implemented as per OOOV methodology.

The Figure 11 depicts the same state machine map of diagram 10 where the Database 1015 node has been expanded. This expanded node now displays associated set of agents programmed to deliver the  
20 specific server functionality. The Database comprises of various agent such as the Client Agent 1105, Distributor Agent 1110, DML Agent 1115, Index Agent 1120, Job Agent 1125, Server Agent 1130 etc. The right side of the figure 11 shows the relationship between various states 1135, events 1140 and modules 1145 associated to deliver agent specific functionalities. There are various common agents 1030 which have functional competence irrespective of server functionalities such as scheduler,  
25 dispatcher etc.

The Figure 12 depicts the same state machine map of diagram 10 where the Web Server 1020 node has been expanded. This expanded node now displays associated set of agents programmed to deliver the specific server functionality. The Web Server 1020 comprises of various agent such as the DNS  
30 Agent, HTTP Agent, Parse Agent, Script Agent, SSI Agent etc. The right side of the figure 12 shows the relationship between various states 1230, events 1235 and modules 1240 associated to deliver agent

5 specific functionalities. There are various common agents 1030 which have functional competence irrespective of server functionalities such as scheduler, dispatcher etc.

The Figure 13 depicts the same state machine map of diagram 10 where the Mail Server 1025 node has been expanded. This expanded node now displays associated set of agents programmed to deliver the specific server functionality. The Mail Server 1025 comprises of various agents such as the POP Agent 1300, SMTP Agent 1310 etc. The right side of the figure 13 shows the relationship between various states 1330, events 1335 and modules 1340 associated to deliver agent specific functionalities. There are various common agents 1030 which have functional competence irrespective of server functionalities such as scheduler, dispatcher etc.

15

20

25

30

35

**5 What is claimed is:**

1. A method of processing data independent of server functionality, communication protocols, data formats, features and syntaxes, comprising:

10       creating an object interface for each object required to execute a plurality of pre-determined operations wherein said object interface has all elements to provide functionally for each operation;

15       isolating a plurality of operation, objects, options and option values from each instruction and data received from said input memory until all instructions and data are isolated;

      mapping said object to said object interface until said operation are mapped;

20       mapping said option and option value to the elements of said object interface till said objects and operations are mapped; and

      processing said operation as per said option and value in the said object interface;

25       whereby said data processor will be compatible with any functional server such as a database, web, mail etc with any server architecture, independent of the vendor or its version and works;

      whereby said data processor can achieve homogeneity of persistent data formats amongst disparate functional servers; and

30       whereby said data processor can achieve cross-server functionality by extending the rules of one server applicable for other.

2. The method according to claim 1, where said operations are from disparate functional servers.

5

3. The method according to claim 1, where said operations are from functional servers provided by multiple vendors including different versions, syntaxes and features.

10

4. The method according to claim 1, where said data can be processed on any operating system and any CPU based hardware.

15

5. The method according to claim 1, where the data processing can be made compatible with any new versions, vendors, operating systems, just by upgrading the definition of the object interface and elements are in said repository.

6. The method according to claim 1, where said elements can be defined dynamically.

7. The method according to claim 1, where said object interface and elements are stored in said repository.

20

8. The method according to claim 1, where said repository can be used to define object interface and element dynamically.

25

9. The method according to claim 1, where defaults are assumed from said repository, if options and values are not defined.

30

10. The method according to claim 1, where further said repository can dynamically be upgraded whereby making said data processor capable of being compatible with additional vendors, versions of functional servers thereby making new additions and update possible.

5 11. The method according to claim 1, where said invention receives and replies to request from the any other functional server thus provides dynamic access to the data in real-time irrespective of all or any combination of server vendor, operating system, versions, protocols, hardware, etc.

12. A method of optimizing operations execution time and resources, comprising:

10 storing a plurality of pre-determined algorithms in said repository for each pre-determined processes along with required resources;

15 selecting a plurality of optional processes from the pre-determined set of said processing required for execution of said operation for each object based on option values and available resources;

tracking the state of said object with respect to resource as required by each of said process; and

20 determining an optimal algorithm to execute said process based on the state of said object.

13. A system for providing a data processor independent of communication protocols, operating systems, formats, features and syntaxes, comprising:

25 a input means to accept instructions and data from the user

a parsing means to validate and parse syntaxes and semantics independent of vendor specific syntax and features, independent of operating system whereby creating said object interface;

30 a repository means to store collections of data definitions, metadata, rules, objects and heuristics; and

5

a processing means to analyze said operations for the said object with said options and parameters specified in the request command, executes them taking care of multi-user constraints and generates the resultant response buffer;

10 14. A system according to claim 13, where said input means could be the operating system independent library capable of accepting any request packet irrespective of source of generation including any protocol, establishing or associating a connection, analyzing the integrity, verifying the request and proceed to allocate the server resource.

15 15. The system according to claim 13, where said multifunctional server can have an established connection and then proceeds to pass the request to the Universal Object Parser.

16. The system according to claim 13, where said repository comprises:

20 a first memory means to fixed pattern with bare minimum necessities of compulsory objects;

a second memory means to supported dynamic agent creation and binding as per various functional servers may require newer SQL scripts to persist data as per each functional server added or plugged later;

25

a third memory means handling error strings and its presentation logic;

a fourth memory means to act as a collection of reserved words;

30 a fifth memory means to act as a syntax and semantic script repository;

a sixth memory means to capture resource utilization and other performance parameters;

5

a seventh memory means to gathered statistical information related to real-time operation on specific server objects.

10

17. The system according to claim 16, where said repository's error message module acts to generate unique error ID which helps to identify the precise location of the error and supplements reporting the appropriate error from the entire string.

15

18. The system according to claim 13, where said multifunctional server comprises of various different agents such as dispatcher, Scheduler, Network agent, Disk, Timer and Job agents.

19. The system according to claim 13, where said parsing means aids to respond to the query in the format expected by various disparate client.

20

20. The system according to claim 13, where said parsing means comprises:

Object handler to isolates the vendor dependent syntaxes, semantics, object features;

Command Analyzer to isolate the nature of request and for associating it with any previous session data if needed; and

25

Lexical, syntax and Semantics Validator to validate the request for syntax and semantics specified irrespective of vendor syntax.

30

21. The system according to claim 13, where said repository is outside the main bin.

22. The system according to claim 13, where said invention can provide for seamless data exchange without any reprogramming effort.

5

23. The system according to claim 13, where said invention can provide a configuration file to be used along with the said system.

10

15

20

25

30

Fig. 1a

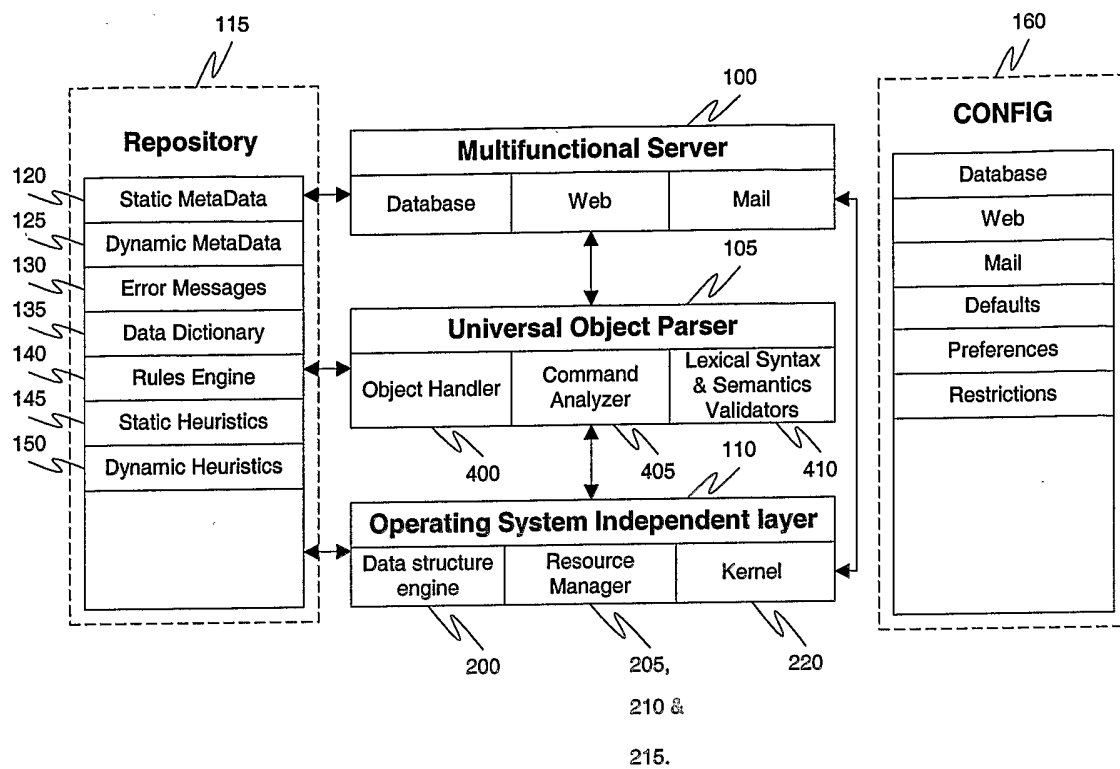


Fig. 1b

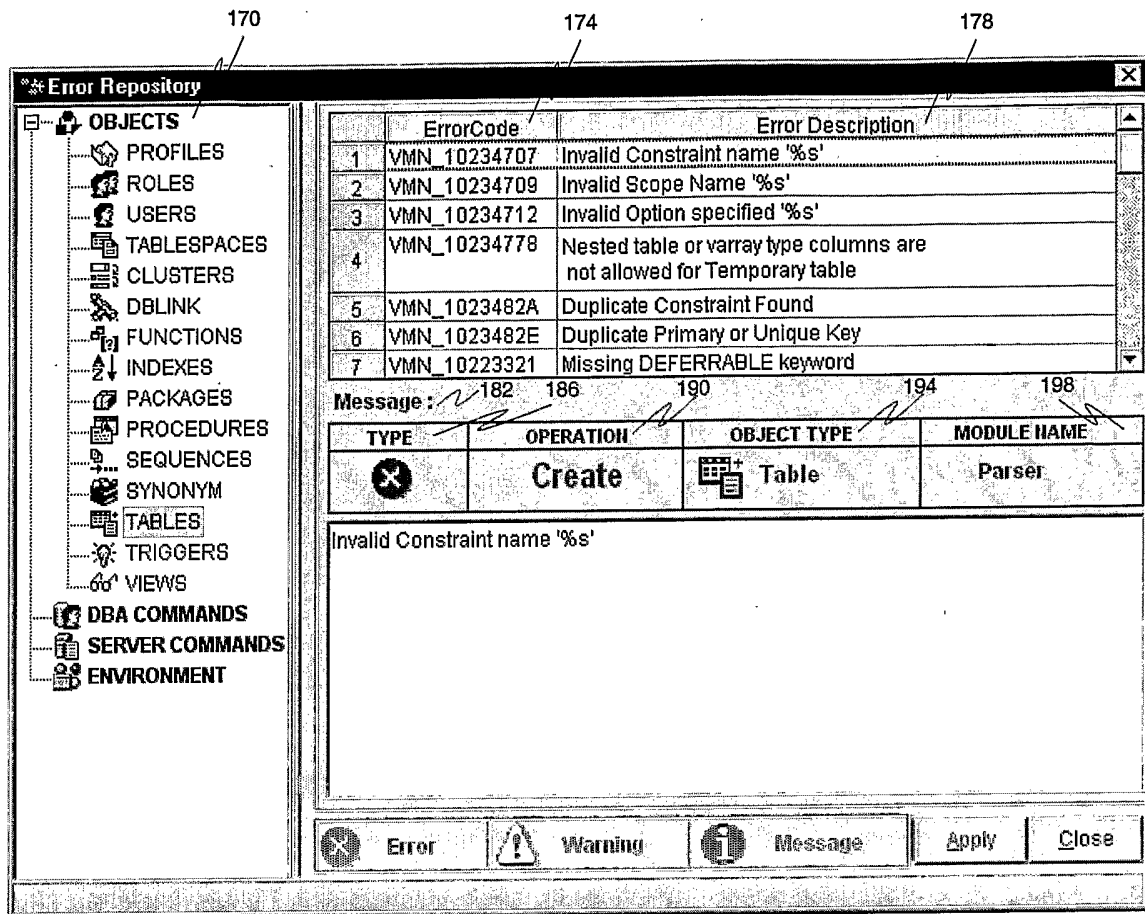


Fig. 2

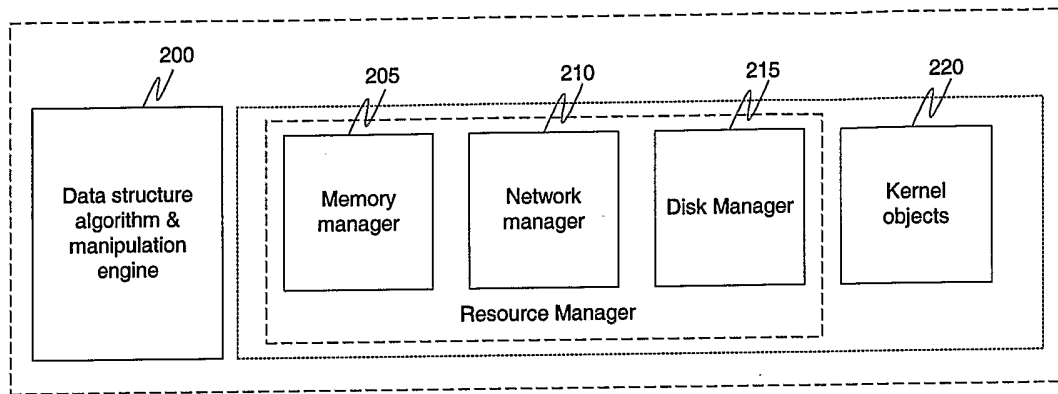


Fig. 3

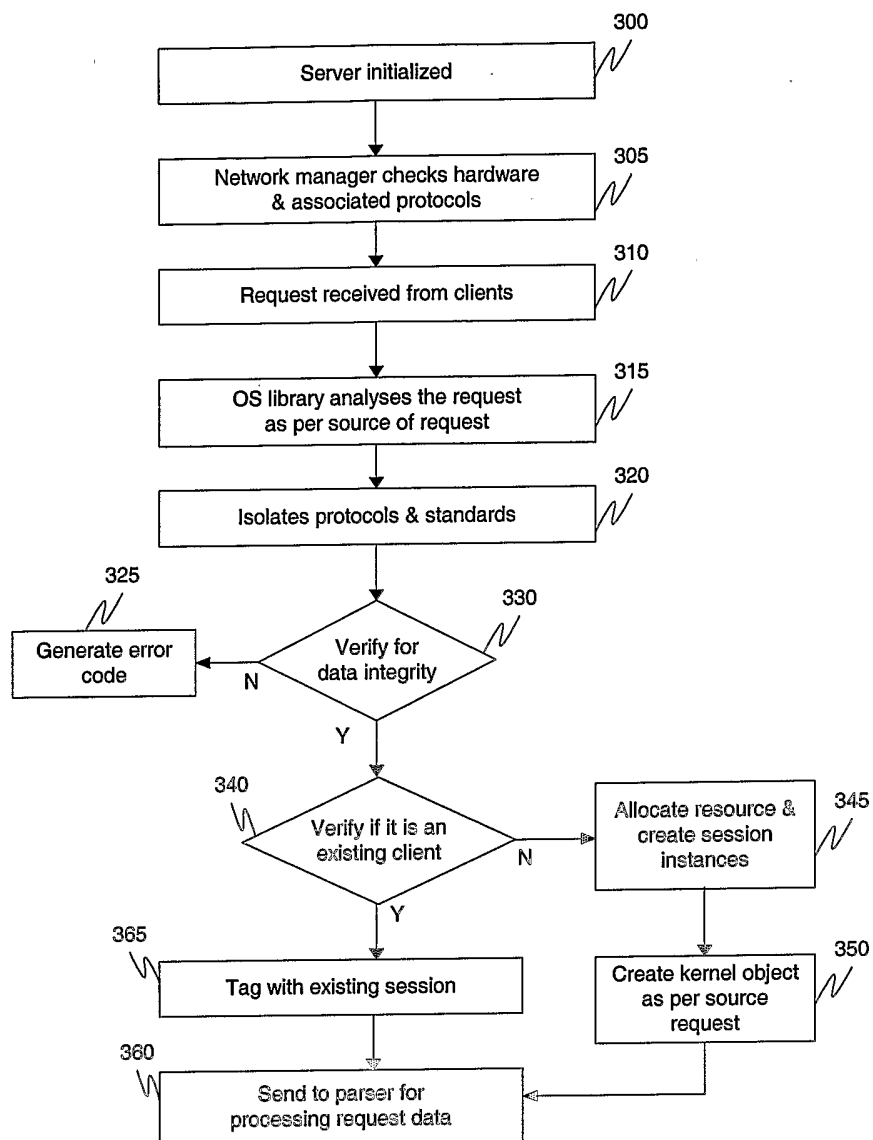


Fig. 4

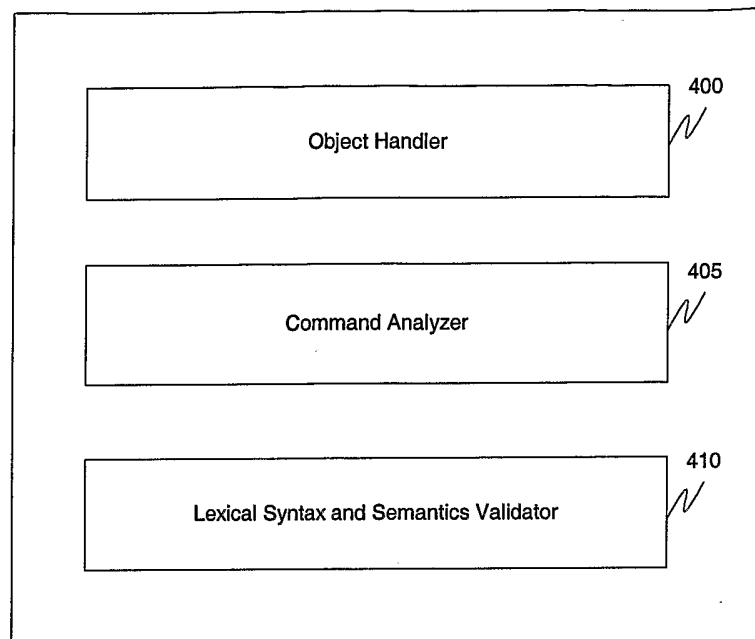


Fig. 5a

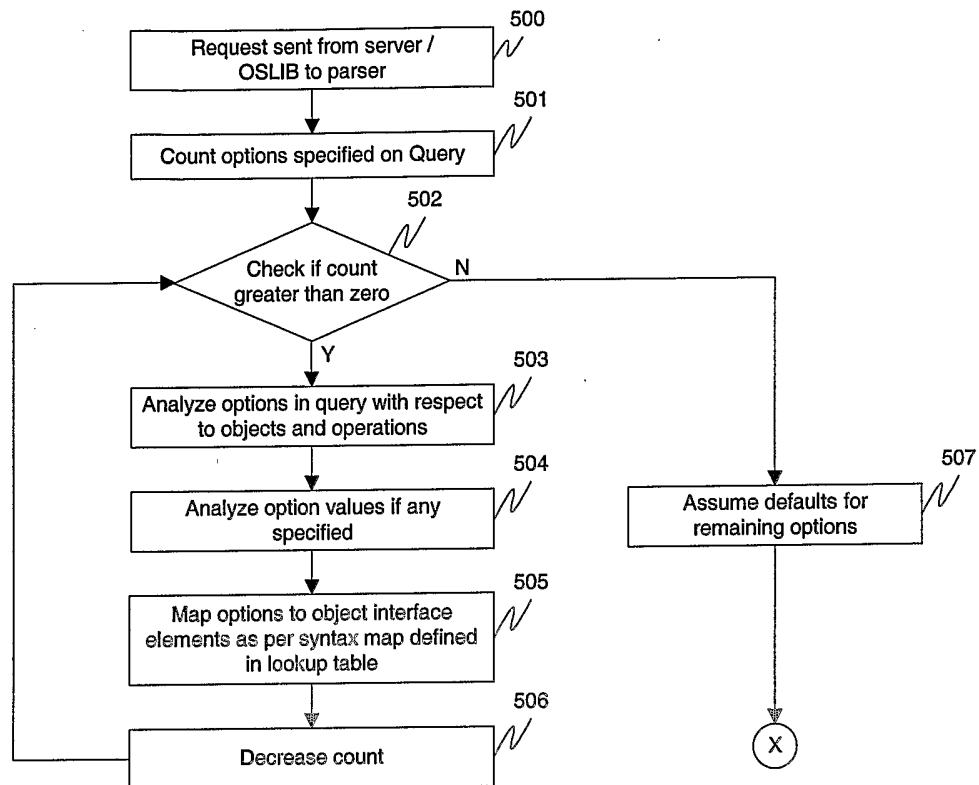


Fig. 5b

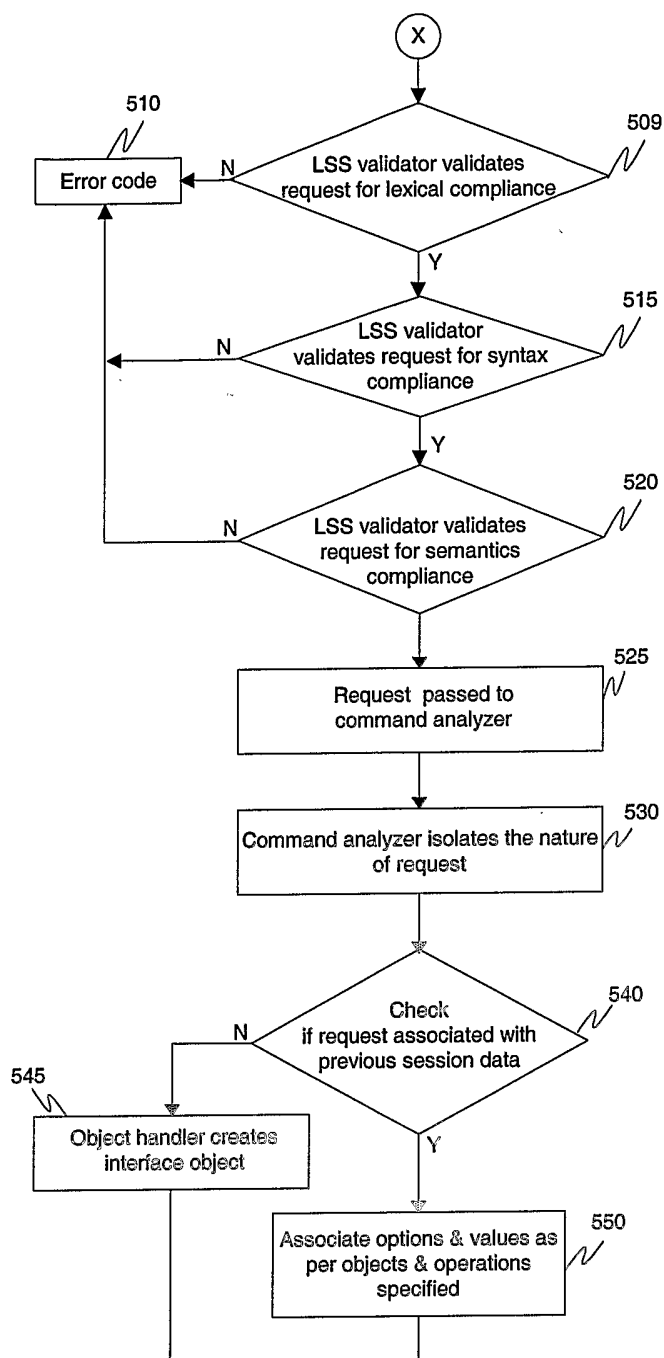


Fig. 6

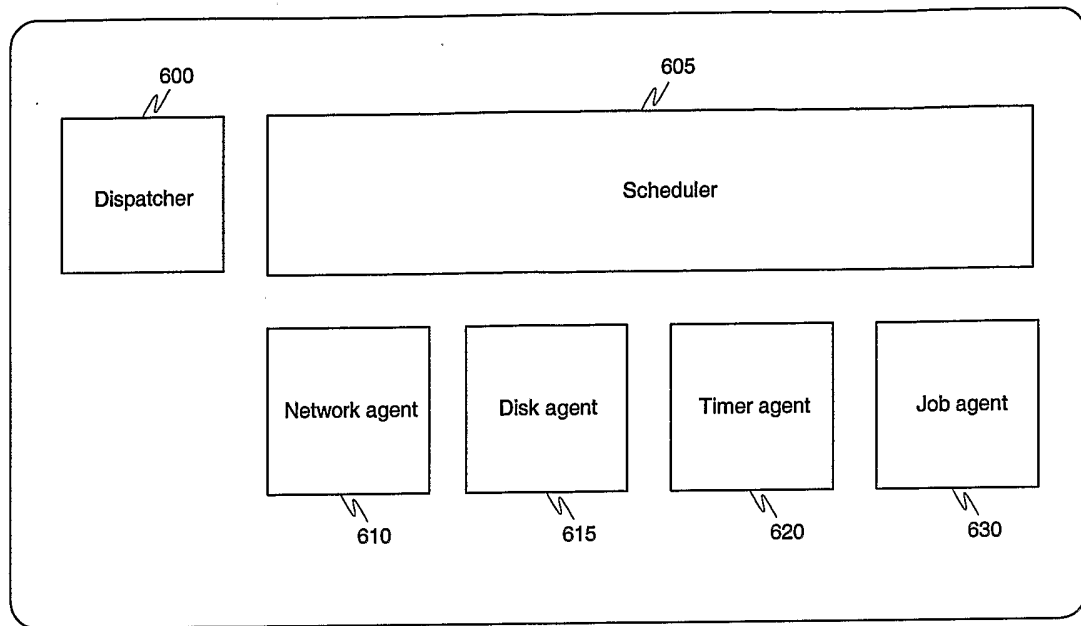


Fig. 7a

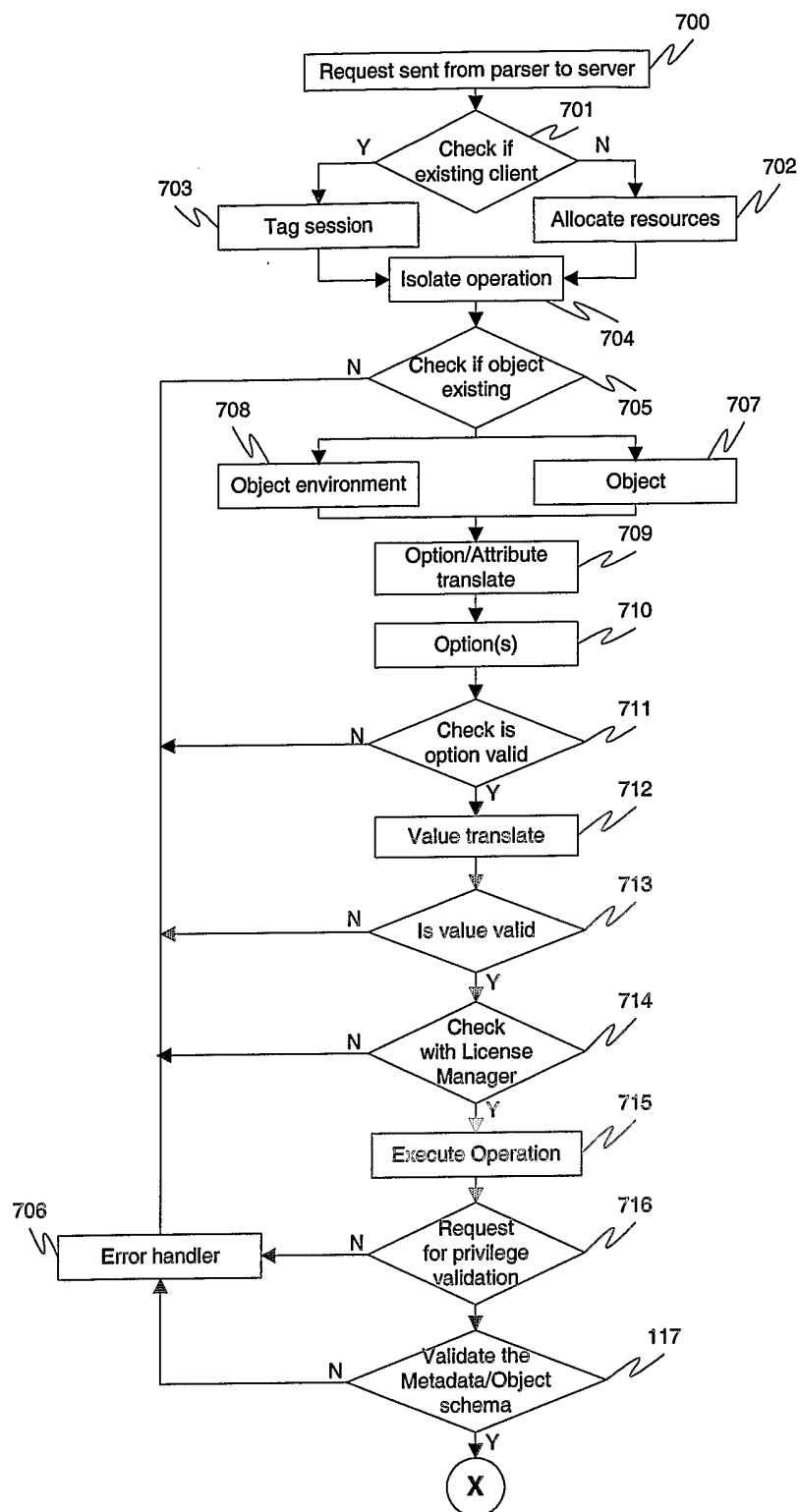


Fig. 7b

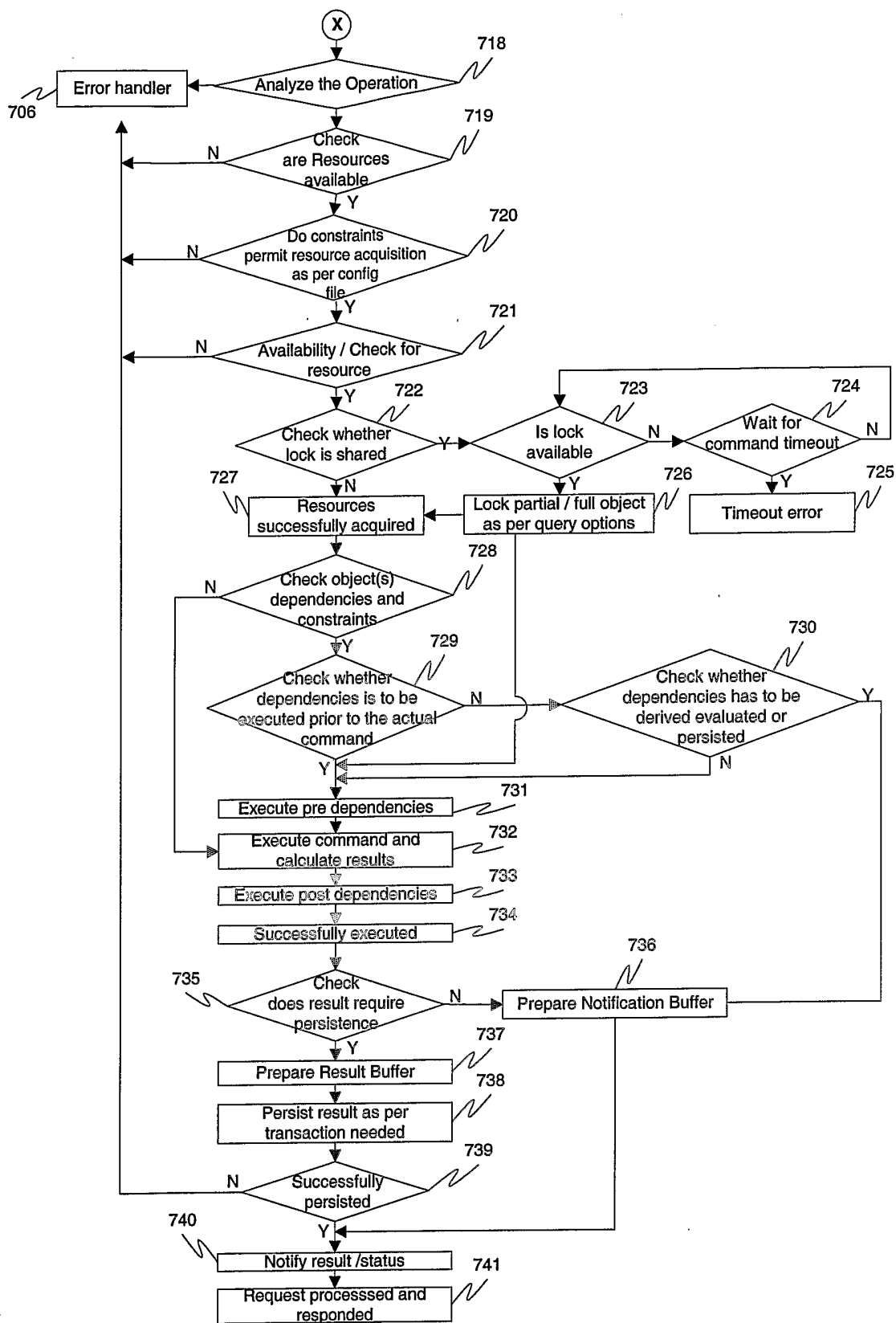


Fig. 8

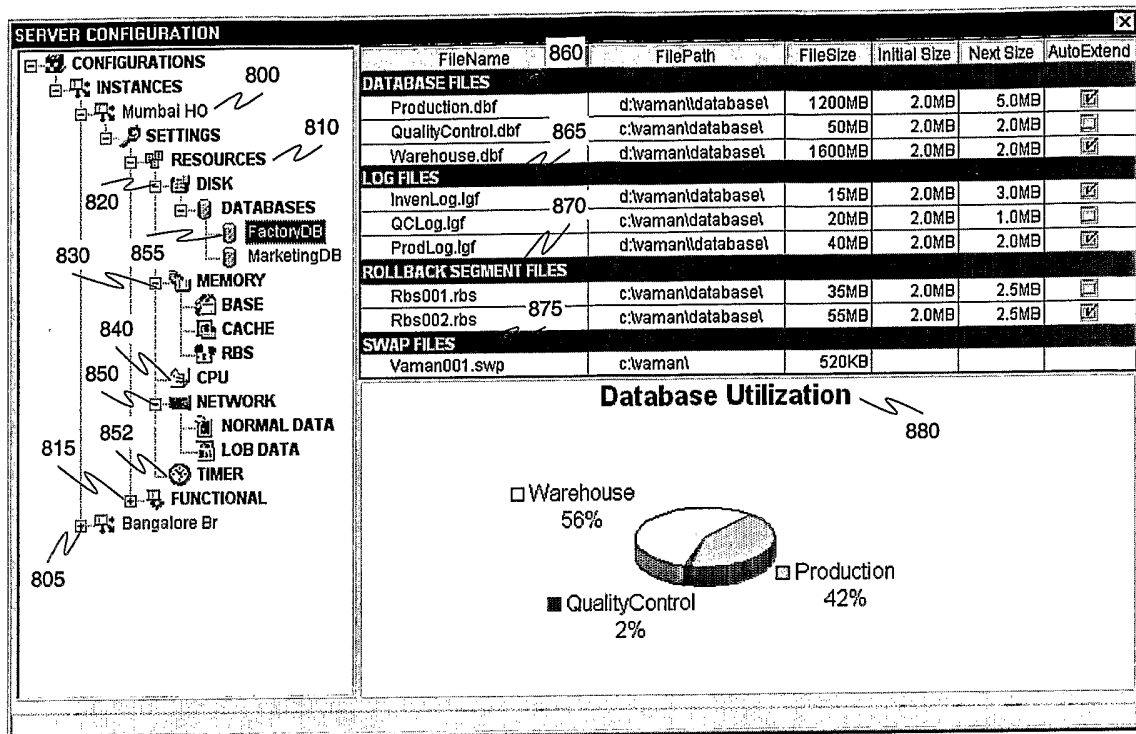


Fig. 9

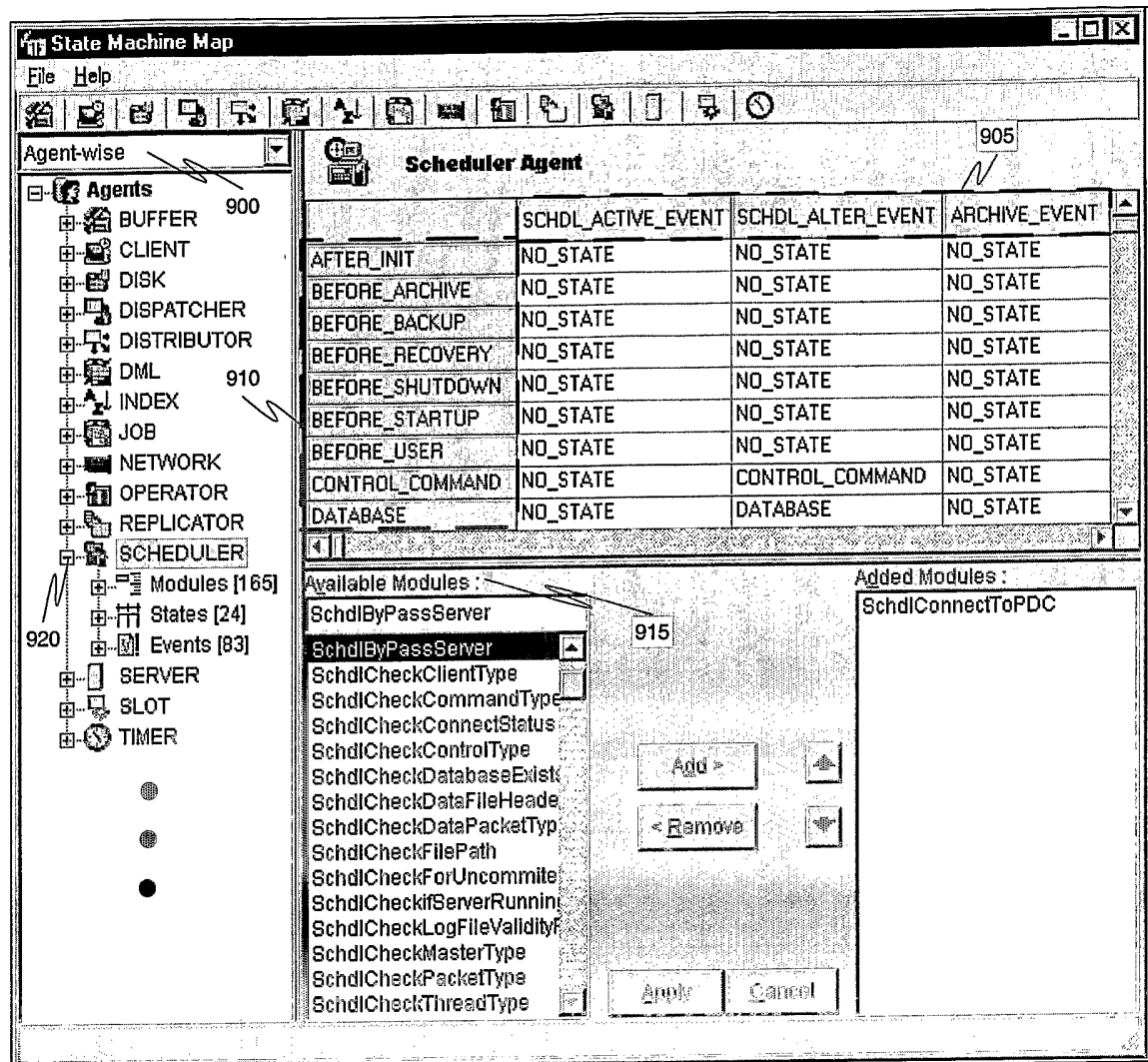


Fig. 10

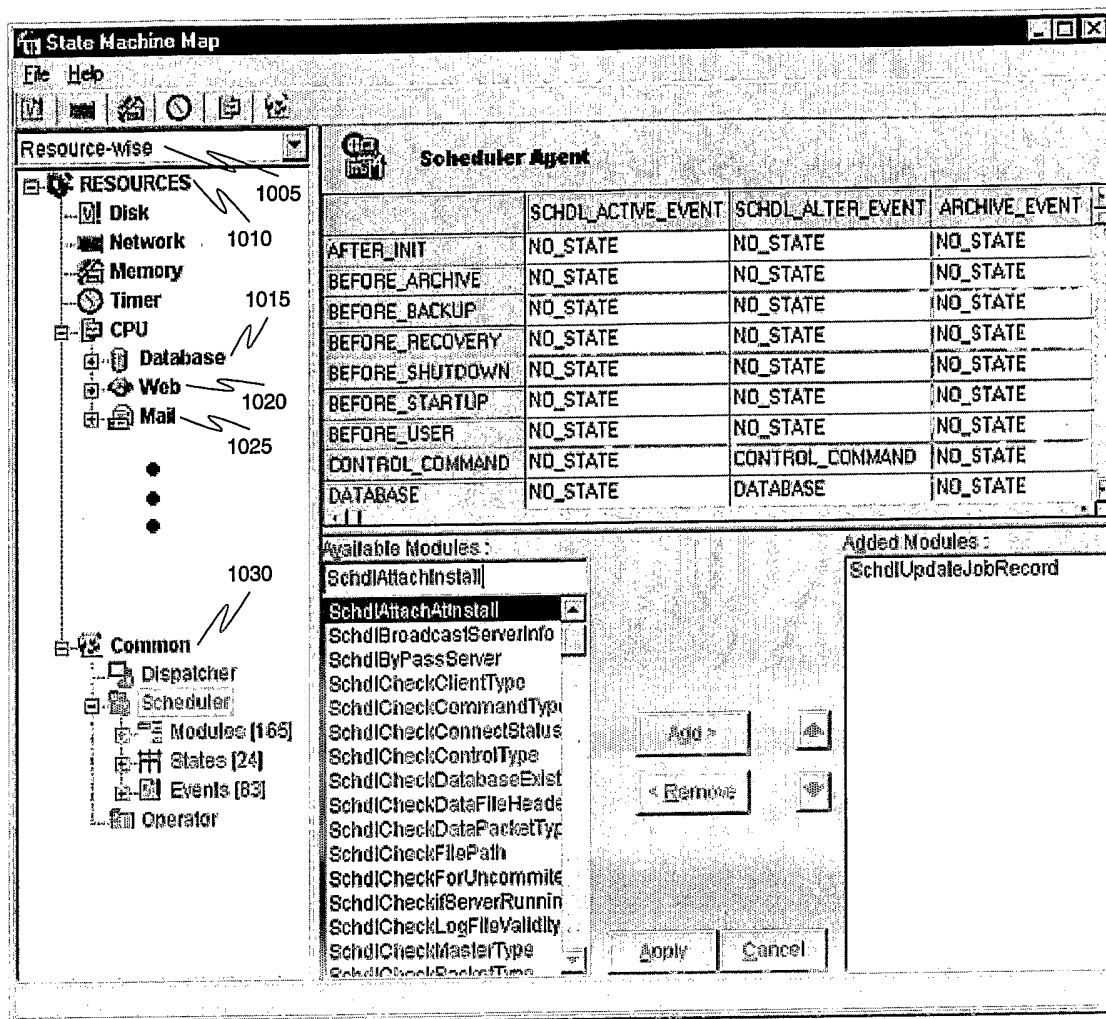


Fig. 11

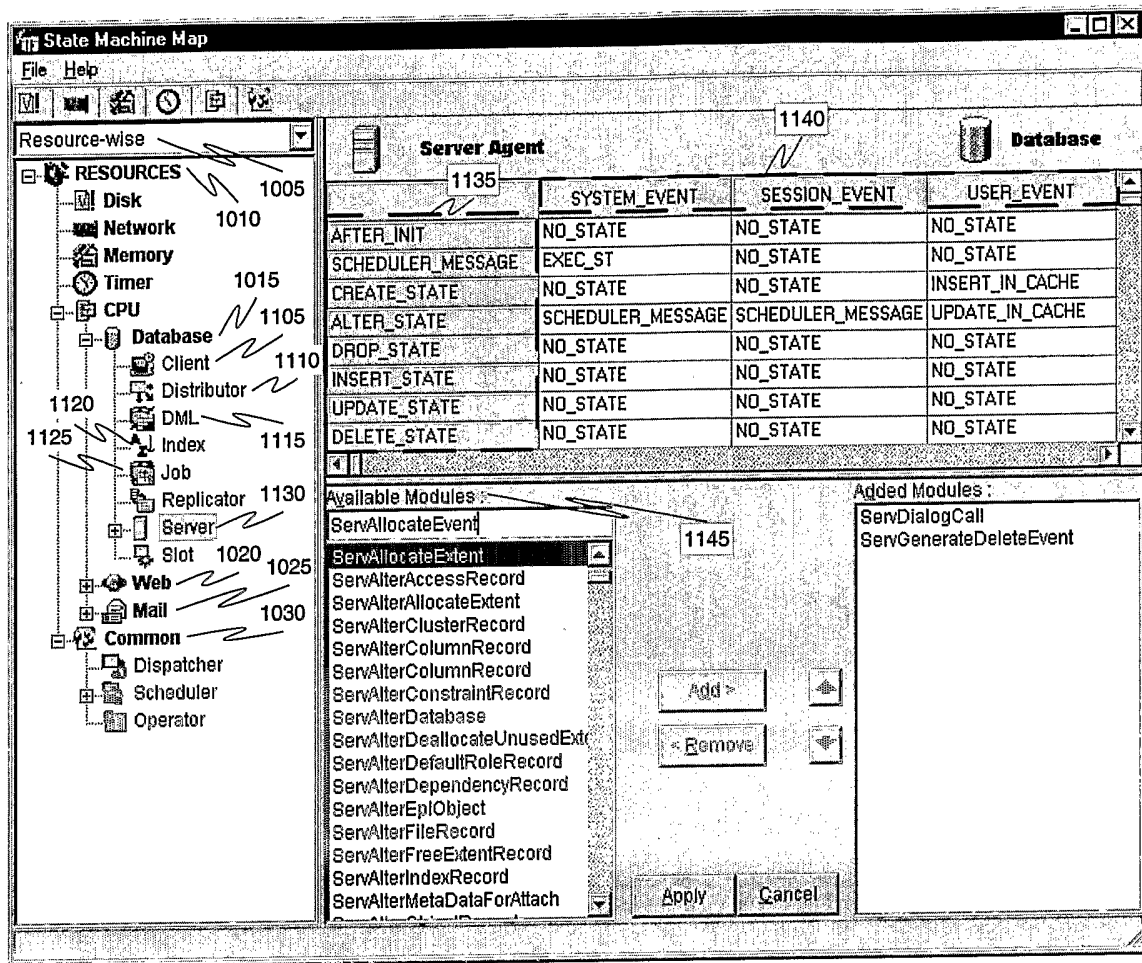


Fig. 12

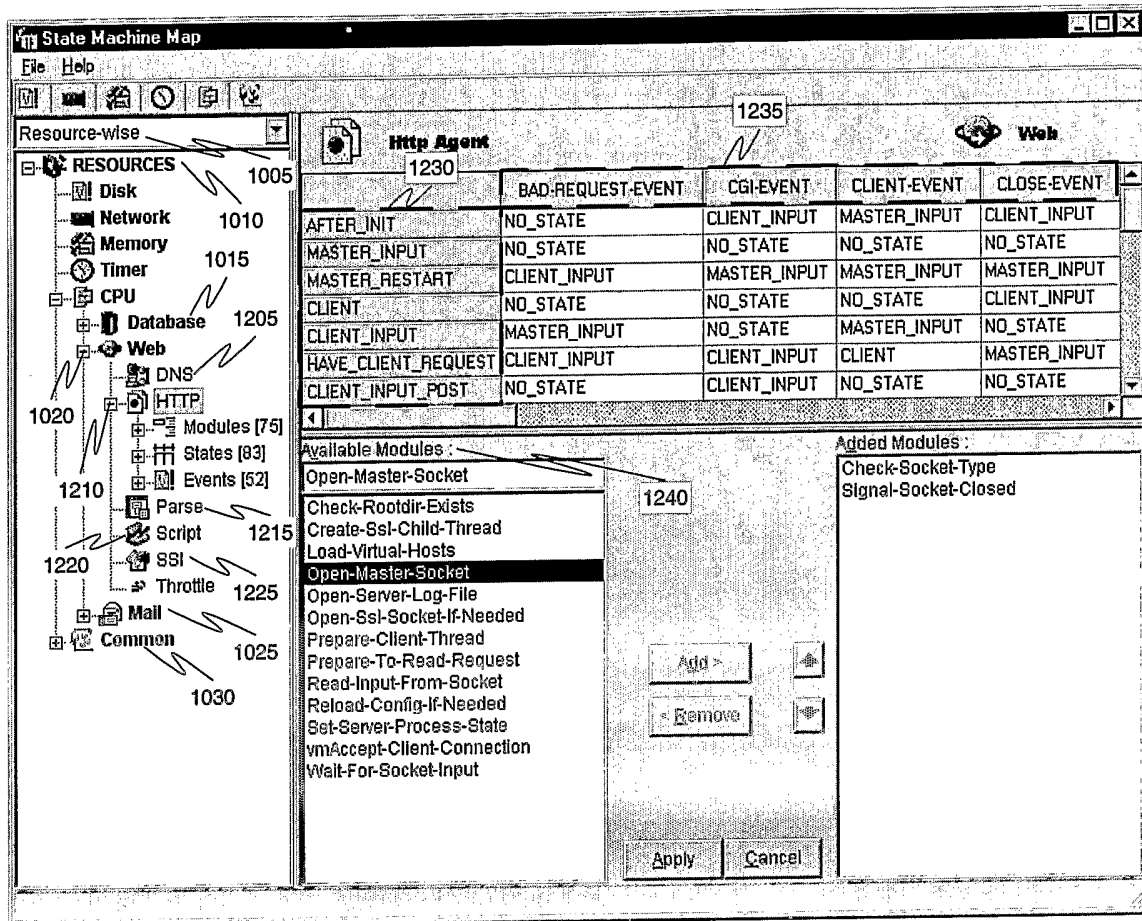


Fig. 13

**State Machine Map**

File Help

Resource-wise

**RESOURCES**

- Disk 1005
- Network 1010
- Memory 1015
- Timer 1025
- CPU 1020
- Database 1020
- Web 1310
- Mail 1320
  - POP
  - SMTP
  - Modules [54]
  - States [8]
  - Events [11]
- Common 1030
  - Dispatcher
  - Scheduler
  - Operator

**Smtp Agent**

1330

	MASTER_EVENT	COMMIT_EVENT	CLIENT_EVENT	KILL_THREAD_EVENT
AFTER_INIT	NO_STATE	MASTER_INPUT	NO_STATE	MASTER_INPUT
MASTER_STARTUP	MASTER_INPUT	NO_STATE	MASTER_INPUT	NO_STATE
MASTER_INPUT	NO_STATE	SMTP_SERVER	NO_STATE	MASTER_INPUT
CHECK_AUTHORITY	SMTP_SERVER	NO_STATE	SMTP_SERVER	NO_STATE
EXPECT_COMMAND	NO_STATE	SMTP_SERVER	NO_STATE	NO_STATE
SMTP_SERVER	SMTP_SERVER	NO_STATE	SMTP_SERVER	NO_STATE
SMTP_CLIENT	NO_STATE	SMTP_SERVER	NO_STATE	DEFAULT
DEFAULT	NO_STATE	SMTP_SERVER	NO_STATE	NO_STATE

1335

**Available Modules:**

- SMTPLoadConfigParameter
- SMTPAcceptClientConnection
- SMTPCheckClientIPAllowed
- SMTPCheckIfSMTPEnabled
- SMTPGetSMTPCommand
- SMTPInitialiseClientThread
- SMTPInitialiseTheThread
- SMTPLoadConfigParameter
- SMTPReadMailRequest
- SMTPSendReadToNetForSocketInput
- SMTPTerminateTheThread
- SMTPWaitForSocketInput
- SMTPWriteServiceReady
- SMTPWriteServiceUnavailable

1345

**Added Modules:**

- SMTPSendHelpMessage
- SMTPSendMailContent

Apply Cancel