(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
9 February 2006 (09.02.2006)

PCT

(10) International Publication Number
WO 2006/014733 A1

(51) International Patent Classification[7]: G06F 9/46, H04L 12/24

(21) International Application Number: PCT/US2005/025741

(22) International Filing Date: 21 July 2005 (21.07.2005)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/590,405 22 July 2004 (22.07.2004) US
11/186,280 20 July 2005 (20.07.2005) US

(71) Applicant *(for all designated States except US)*: **COMPUTER ASSOCIATES THINK, INC.** [US/US]; One Computer Associates Plaza, Islandia, NY 11749 (US).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: **LY, An, V.** [US/US]; 4235 Placid Drive, Sarasota, FL 34243 (US). **PADMANABHAN, Arun** [IN/US]; 6813 Waterton Drive, Riverview, FL 33569 (US). **CHEN, Edward, F.** [US/US]; 1183 Fraser Pine Blvd., Sarasota, FL 34240 (US).

(74) Agent: **STALFORD, Terry, J.**; Fish & Richardson P.C., 1717 Main Street, Suite 5000, Dallas, TX 75201-4605 (US).

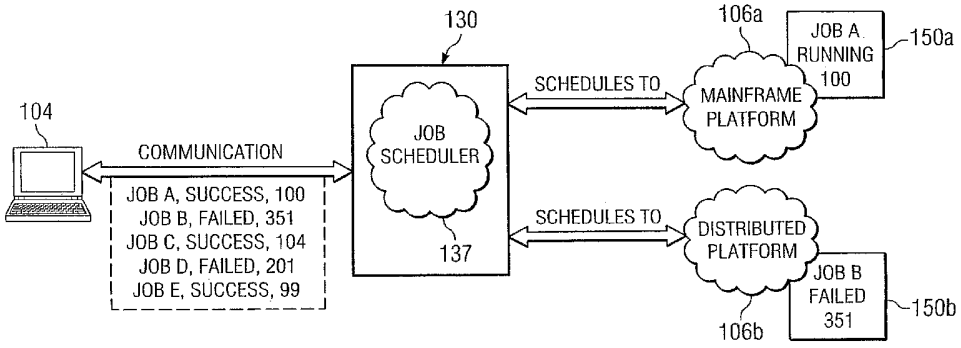
(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NA, NG, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, SM, 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, NA, 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, IS, IT, LT, LU, LV, MC, NL, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report*
— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments*

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.

(54) Title: SYSTEM AND METHOD FOR PROVIDING ALERTS FOR HETEROGENEOUS JOBS



(57) Abstract: This disclosure provides a system and method for summarizing jobs for a user group. In one embodiment, a job manager is operable to invoke an alert filter. The alert filter is compatible with a plurality operating environments. One or more properties of a first job associated with a first operating environment is identified. One or more properties of a second job associated with a second operating environment is identified. The first operating environment and the second operating environment are heterogeneous. A first alert object is generated in response to a first match between the alert filter and the identified properties of the first job. A second alert object is generated in response to a second match between the alert filter and the identified properties of the second job.

WO 2006/014733 A1

SYSTEM AND METHOD FOR PROVIDING ALERTS FOR HETEROGENEOUS JOBS

RELATED APPLICATION

This application claims the benefit of U.S. Provisional Application No. 60/590,405 filed July 22, 2004 and U.S. application entitled "System and Method for Providing Alerts for Heterogeneous Jobs" filed July 20, 2005.

TECHNICAL FIELD

This invention relates to enterprise job scheduling and, more particularly, to a system and method for providing alerts for heterogeneous jobs.

BACKGROUND

There are numerous heterogeneous operating environments for jobs, applications or other processes. Typically, each of these operating environments comprise one of disparate operating systems including UNIX, Windows or Windows Server, Linux, z/OS or other mainframe OS, and others. Generally, these jobs or applications, whether enterprise or consumer, are compatible or optimized for one of these heterogeneous operating systems. Some properties of these jobs are similar across the heterogeneous systems, while others are unique to each operating system, job type, or job dependencies. For example, the status property of a job residing in an enterprise job scheduler for a mainframe system may indicate one of the following example states: "Abend," "Requeued," "JCL Error," and others. But the status of a second job residing in an enterprise job scheduler for a Unix-based system may indicate one of the following example states: "Exited," "Running," "Suspended," "Failed," and such.

SUMMARY

This disclosure provides a system and method for summarizing jobs for a user group. In one embodiment, a job manager is operable to invoke an alert filter. The alert filter is compatible with a plurality operating environments. One or more properties of a first job associated with a first operating environment is identified. One or more properties of a second job associated with a second operating environment is

identified. The first operating environment and the second operating environment are heterogeneous. A first alert object is generated in response to a first match between the alert filter and the identified properties of the first job. A second alert object is generated in response to a second match between the alert filter and the identified properties of the second job.

The details of one or more embodiments of the disclosure are set forth in the accompanying drawings and the description below. Particular features, objects, and advantages of the disclosure will be apparent from the description and drawings and from the claims.

DESCRIPTION OF DRAWINGS

FIGURE 1 illustrates a job filtering system in accordance with one embodiment of the present disclosure;

FIGURES 2A-E illustrate various configurations of an enterprise system for executing jobs in heterogeneous operating environments;

FIGURE 3 illustrates one embodiment of the job manager of FIGURE 1;

FIGURES 4A-F illustrate an example alert filter object of FIGURE 1 and associated processing in accordance with one embodiment of the present disclosure;

FIGURES 5A-F are example displays for presenting various properties of heterogeneous jobs as executed in the system of FIGURE 1 in accordance with one embodiment of the present disclosure;

FIGURES 6A-E are example displays associated with alert filter objects of FIGURE 1 in accordance with one embodiment of the present disclosure;

FIGURE 7 is a flowchart illustrating an example method for processing a job request in one of a plurality of heterogeneous environments in accordance with one embodiment of the present disclosure; and

FIGURE 8 is a flowchart illustrating an example method for filtering jobs in response to a user request in accordance with one embodiment of the present disclosure.

DETAILED DESCRIPTION

FIGURE 1 illustrates a job management system 100 for generating job alerts in an enterprise in accordance with one embodiment of the present disclosure. Generally, job management system 100 identifies environment-independent alert filter objects 142, determines one or more matches between alert filter objects 142 and job objects 140, and generates alert objects 144 in accordance with alert filter objects 142. As a result, job management system 100 may monitor properties of jobs 150 in heterogeneous operating environments 106 and notify a user or application of, for example, a job's transition to a certain state. Job management system 100 may provide event notification in response to state transitions of jobs 150 in heterogeneous operating environments 106. Heterogeneous operating environments 106 may include job schedulers that are at least partially incompatible so monitoring and managing alerts across these heterogeneous job schedulers can be difficult. For example, operating environment 106a may include a job scheduler for a mainframe system that indicates a job failure as "JCL Error" and operating environment 106b may include a job scheduler for a Unix-based system that indicates a job failure as "Failed." In overcoming these difficulties, a user (directly or indirectly) invokes alert filter objects 142 to identify transitions in disparate job properties. Generally, users may include any user of system 100 or one of its components such as, for example, job scheduling personnel with the ability to schedule jobs, forecast future scheduling requirements, analyze and measure the effectiveness of the job flow, automated job management policies, and/or manage jobs on distributed networks.

At a high level, system 100 is all or a portion of the enterprise that includes or is communicably coupled with server 102, one or more clients 104, and a plurality of heterogeneous operating environments 106. For example, system 100 may be associated with the entire enterprise, a geographical or logical location within the enterprise, or any other portion of the enterprise. It will be understood that the enterprise may be a corporation, non-profit organization, government agency, or any other person or entity that includes, utilizes, or receives the results from multiple computing devices and operating environments 106. In other words, job management system 100 is typically a distributed client/server system that allows users of clients 104 to submit jobs 150 for execution on any of the plurality of operating environments

106. But system 100 may be any other suitable environment without departing from the scope of this disclosure. Generally, "dynamically," as used herein, means that certain processing is determined, at least in part, at run-time based on one or more variables. Whereas the term "automatically," as used herein, generally means that appropriate processing is substantially performed by at least part of job management system 100. It should be understood that "automatically" further contemplates any suitable administrator or other user interaction with system 100 without departing from the scope of this disclosure.

Returning to the illustrated embodiment, system 100 includes, invokes, executes, references, or is communicably coupled with a plurality operating environments 106. Each operating environment 106 is any system or subsystem operable to at least partially or fully execute or process jobs 150. For example, each operating environment 106 is one of a plurality of heterogeneous environments including Unix, Linux, Windows, or mainframe environments, as well as others. In another example, an operating environment 106 may represent a particular application. Moreover, each operating environment 106 may include one server or may be distributed across a plurality of computers. For example, illustrated system 100 includes three operating environments 106a, 106b, and 106c respectively. In this example, first operating environment 106a is server environment executing UNIX, second operating environment 106b is a mainframe environment executing z/OS, and third operating environment is a distributed processing environment including a plurality of clients executing Windows. In another example, two operating environments 106 may be executing the same operating system, but may include different storage capabilities, file systems, or computing devices. In yet another example, two operating environments 106 may be substantively similar or identical, except for executing two disparate cyclical releases or versions of the same operating system. As illustrated in FIGURES 2A-E, each operating environment 106 typically includes one or more job schedulers 137, each of which may be tailored to, designed for, or at least partially compatible with job executing in the associated operating environment 106. In this case, "operating environment 106" and "job scheduler 137" may be used interchangeably as appropriate. Of course, illustrated operating environments 106 are for example purposes only. Indeed, while illustrated separately, server 102 may represent, include, or execute one of the operating environments 106

or one of the operating environments 106 may include or utilize server 102 without departing from the scope of the disclosure.

Illustrated server 102 includes memory 120 and processor 125 and comprises an electronic computing device operable to receive, transmit, process and store data associated with system 100. For example, server 102 may be any computer or processing device such as, for example, a blade server, general-purpose personal computer (PC), Macintosh, workstation, Unix-based computer, or any other suitable device. Generally, FIGURE 1 provides merely one example of computers that may be used with the disclosure. For example, although FIGURE 1 illustrates one server 102 that may be used with the disclosure, server 102 can be implemented using computers other than servers, as well as a server pool. Server 102 may be adapted to execute any operating system including Linux, UNIX, Windows Server, z/OS or any other suitable operating system. But, the present disclosure contemplates servers other than general purpose computers as well as servers without conventional operating systems. According to one embodiment, server 102 may also include or be communicably coupled with a web server and/or a data server.

Memory 120 may include any memory or database module and may take the form of volatile or non-volatile memory including, without limitation, magnetic media, optical media, random access memory (RAM), read-only memory (ROM), removable media, or any other suitable local or remote memory component. In the illustrated embodiment, illustrated memory 120 includes job objects 140, alert filter objects 142, and alert objects 144, but may also include any other appropriate data such as a job history, normalization policies, a security or audit log, print or other reporting files, HTML files or templates, and others. Job objects 140 are representations of enterprise jobs and their associated properties. These jobs may be update or report batch jobs, database processing utilities, commands, or other tasks. Each job object 140 comprises at least a mapping of property names to values that represent the parameters, variables, output format, or other details of the associated job. For example, job object 140 typically comprises at least a job identifier and a pointer or other reference to the appropriate or associated operating environment 106. The environment pointer may be automatically, dynamically, or manually populated based on operating system compatibility, data storage location, application, utilization, priority, department or business rules, geography, other criteria or characteristics, or

any combination thereof. In another example, each job object may include job predecessor, job successor, triggers, calendar, VRM requirements, dataset predecessors, user requirements, and network predecessors. In certain embodiments, the constituent data may be dynamically populated based on the particular type of job. For example, in the case of a distributed job, job object 140 may include two or more identifiers of the associated operating environments, while a standalone job merely includes one environment pointer. Job object 140 may be in any appropriate logical or physical format including an executable, a Java object, text, Structured Query Language (SQL), eXtensible Markup Language (XML), and such. Indeed, job object 140 may be a default job or a particular instance of a job as appropriate. Moreover, job object 140 may be keyed on or associated with a user, a user or security group, a department, or any other variable or property. In addition, job object 140 may include normalized properties or any properties operable to be normalized.

Alert filter objects 142 include any parameters, variables, algorithms, instructions, rules, objects or other directives for filtering transitions of jobs 150 to specific states in heterogeneous operating environments 106. For example, alert filter object 142 may be used to identify jobs objects 140 whose properties have transitioned to or from specified states and to generate alert objects 144 in response to an event. Such an event may include completion of a specific job or jobset, failure of a specific job or jobset, failure rate of a job type exceeding a threshold, or any other suitable event. As a result, alert filter objects 142 may provide a non-intrusive way to generate alerts based on the state of jobs and other objects in associated job schedulers. In some embodiments, alert filter objects 142 may comprise one or more tables stored in a relational database described in terms of SQL statements or scripts. In another embodiment, alert filter objects 142 may store or define various data structures such as Java objects, text files, XML documents, comma-separated-value (CSV) files, internal variables, SQL, or one or more libraries. An alert filter object 142 may comprise one table, file, or object or a plurality of tables, files, or objects stored on one computer or across a plurality of computers in any appropriate format. Moreover, alert filter objects 142 may be local or remote without departing from the scope of this disclosure and store any type of appropriate data. Alert filter objects 142 may be dynamically created by server 102, a third-party vendor, or any suitable user of server 102, loaded from a default file, or received via network 112. In the case of a user creating alert

filter object 142, the user may be able to view alerts from jobs 150 that match specified criteria across operating environments 106. In other words, the user may be able to limit alerts that are processed and/or presented by server 102 to those relevant to the user's task regardless of the particular processing environment, operating environment, or application. Alert filter object 142 may be associated with a specific job, a type of job, a job-scheduler type, a specific operating environment 106, a specific job scheduler, or other suitable elements in the enterprise.

In the illustrated embodiment, each alert filter objects 142 implements filtering directives using filter criteria 146 and includes an alert template 148 for generating alert objects 144. In some embodiments, filter criteria 146 are compared to job properties encapsulated in job objects 140 in order to determine matches. Filter criteria 146 typically includes one or more values that are compared against one or more job properties (*e.g.*, status) contained in job objects 140. In some embodiments, prior to comparing with filter criteria 146, server 102 may normalize the one or more job properties. In this case, the filter criteria 146 may be compared with job properties from heterogeneous operating environments 106. For example, one or both of the job properties indicating a job failure from mainframe operating environment 106a and UNIX-based operating environment 106b, respectively, may both be converted to the normalized property "Failure." After normalization, server 102 may compare filter criteria 146 to the normalized job properties from both operating environments 106a and 106b for determining whether to generate alert objects for each operating environment 106. In the case that the job properties are not normalized, filter criteria 146 may be associated with a specific job scheduler and/or operating environment 106. For example, filter criteria 146 may be associated with UNIX-based operating environment 106b and may only be used to generate alerts for that operating environment 106b. Filter criteria 146 may include values to match against one or more of the following properties: name of job, name of jobset, status of job, name of operating environment 106, or others. In some embodiments, filter criteria 146 includes one or more tuples for comparing with job properties. Alert filter object 142 may require that all tuples are matched or one or more tuples are matched before associated alert objects 144 are generated. Each tuple may include a property name, an operator, and a value. (discussed in more detail below) In this case, the operator instructs server 102 how to compare the value with the associated job property. For

example, the operator may be a greater than sign indicating that filter criteria 146 matches a job property if the job property exceeds the value. The operator may be an equal to sign, a less than sign, a greater than or equal sign, a less than or equal to sign, or any other logical or mathematical operator. In the event that filter criteria 146 is matched, server 102 may use alert template 148 to generate alerts representing transitions of job states. Alert template 148 may comprise a script, executable, template or any other suitable description such that server 102 may quickly instantiate an appropriate alert object 144. In some embodiments, alert template 148 is a class definition for one or more alert object 144. In other words, alert template 148 may include properties that are used to efficiently create, instantiate, or invoke one or more alert objects 144. The properties included in template 148 may be determined by a user of system 100 and/or a process running in system 100. In some embodiments, a user of system 100 selects or inputs these properties through GUI 116 when generating alert filter object 142.

Typically based on alert template 148, server 102 instantiates at least one alert object 144. Alert objects 144 are representations of enterprise alerts and their associated properties. As discussed above, the alerts may be a transition in a job state such as, for example, success of a specific job or jobset, failure of a specific job or jobset, or other events. Each alert object 144 typically comprises a mapping of property names to values that represent the parameters, variables, or other details of the associated alert. For example, alert object 144 typically includes at least a job identifier and a pointer or other reference to the appropriate or associated operating environment 106. In some embodiments, each alert object 144 includes one or more the following properties: identification number, class name, severity level, queue name, timestamp, description, and others. In certain embodiments, the constituent data may be dynamically populated based on the particular type of alert. For example, in the case of a jobset, alert object 144 may include two or more job identifiers. Alert object 144 may be in any appropriate logical or physical format including an executable, a Java object, text, SQL, XML, and such. For example, alert object 144 may be an instantiated object based on a class defined in any appropriate object-oriented programming language such as C++, Java, or any other suitable language.

Server 102 also includes processor 125. Processor 125 executes instructions and manipulates data to perform the operations of server 102 such as, for example, a

central processing unit (CPU), a blade, an application specific integrated circuit (ASIC), or a field-programmable gate array (FPGA). Although FIGURE 1 illustrates a single processor 125 in server 102, multiple processors 125 may be used according to particular needs and reference to processor 125 is meant to include multiple processors 125 where applicable. In the illustrated embodiment, processor 125 executes job manager 130, which performs at least a portion of the management of heterogeneous jobs 150 and/or the normalization of their properties.

Job manager 130 could include any hardware, software, firmware, or combination thereof operable to allow users access to operating environments 106, submit jobs 150, query the status or other job properties, normalize some or all of these properties, or any other appropriate job management processing. For example, job manager 130 may be written or described in any appropriate computer language including C, C++, Java, J#, Visual Basic, assembler, Perl, any suitable version of 4GL, another language, or any combination thereof. It will be understood that while job manager 130 is illustrated in FIGURE 1 as a single multi-tasked module, the features and functionality performed by this engine may be performed by multiple modules. For example, job manager 130 may be a job scheduler and a plurality of adapters 135 (*see* FIGURE 2). In another example, job manager 130 may comprise a connection listener 304, a request controller 308 communicably coupled with a plurality of job parsers and managers, a view controller 314, a session manager 318, a template manager 320, an adapter manager 322, and a profile manager 324 (as shown in more detail in FIGURE 3). Further, while illustrated as internal to server 102, one or more processes associated with job manager 130 may be stored, referenced, or executed remotely such as GUI 116 and one or more agents residing in the appropriate operating environments 106. Moreover, job manager 130 may be a child or sub-module of another software module (not illustrated) without departing from the scope of this disclosure. In certain embodiments, job manager 130 may include or be communicably coupled with an administrative workstation 104 or graphical user interface (GUI) through interface 114. In these embodiments, job manager 130 may run as a persistent process (*e.g.*, a daemon or service) operable to listen on a particular port through or in interface 114.

Server 102 may also include interface 114 for communicating with other computer systems, such as clients 104, over network 112 in a client-server or other

distributed environment. In certain embodiments, server 102 receives job submissions or customizations from internal or external senders through interface 114 for storage in memory 120 and/or processing by processor 125. Generally, interface 114 comprises logic encoded in software and/or hardware in a suitable combination and operable to communicate with network 112. More specifically, interface 114 may comprise software supporting one or more communications protocols associated with communications network 112 or hardware operable to communicate physical signals.

Network 112 facilitates wireless or wireline communication between computer server 102 and any other local or remote computer, such as clients 104. Illustrated network 112 comprises two sub-nets or virtual LANS, 112a and 112b, respectively. Indeed, while illustrated as two networks, network 112 may be a continuous network without departing from the scope of this disclosure, so long as at least portion of network 112 may facilitate communications between job manager 130 and one or more of the operating environments 106. In other words, network 112 encompasses any internal or external network, networks, sub-network, or combination thereof operable to facilitate communications between various computing components in system 100. Network 112 may communicate, for example, Internet Protocol (IP) packets, Frame Relay frames, Asynchronous Transfer Mode (ATM) cells, voice, video, data, and other suitable information between network addresses. Network 112 may include one or more local area networks (LANs), radio access networks (RANs), metropolitan area networks (MANs), wide area networks (WANs), all or a portion of the global computer network known as the Internet, and/or any other communication system or systems at one or more locations.

Client 104 is any local or remote computing device operable to receive job submissions 150 and present output (such as properties or reports) via a GUI 116. At a high level, each client 104 includes at least GUI 116 and comprises an electronic computing device operable to receive, transmit, process and store any appropriate data associated with system 100. It will be understood that there may be any number of clients 104 communicably coupled to server 102. For example, illustrated clients 104 include one directly coupled client 104 and two communicably coupled clients to the illustrated server 102. Further, "client 104," "job owner," and "user" may be used interchangeably as appropriate without departing from the scope of this disclosure. Moreover, for ease of illustration, each client 104 is described in terms of being used

by one user. But this disclosure contemplates that many users may use one computer or that one user may use multiple computers to submit or review jobs 150 via GUI 116. As used in this disclosure, client 104 is intended to encompass a personal computer, touch screen terminal, workstation, network computer, kiosk, wireless data port, wireless or wireline phone, personal data assistant (PDA), one or more processors within these or other devices, or any other suitable processing device or computer. For example, client 104 may comprise a computer that includes an input device, such as a keypad, touch screen, mouse, or other device that can accept information, and an output device that conveys information associated with the operation of server 102 or clients 104, including digital data, visual information, or GUI 116. Both the input device and output device may include fixed or removable storage media such as a magnetic computer disk, CD-ROM, or other suitable media to both receive input from and provide output to users of clients 104 through the display, namely GUI 116.

GUI 116 comprises a graphical user interface operable to allow the user of client 104 to interface with at least a portion of system 100 for any suitable purpose. Generally, GUI 116 provides the user of client 104 with an efficient and user-friendly presentation of data provided by or communicated within system 100. For example, GUI 116 may be a front-end of job manager 130 and provide functionality to monitor jobs and alerts, as well as a summary of the jobs and alerts. GUI 116 may provide an alternate to a Business Scheduling View (BSV) graphical interface for monitoring. Further, GUI 116 may help the user by providing certain advantages including ease-of-use, compatibility with Java and non-Java browser platforms, and performance. Conceptually, the user logs into job manager 130 through GUI 116, which then presents a list of jobs or job schedulers 137. By selecting a particular job scheduler, GUI 116 displays the list of active jobs on that scheduler with the appropriate normalized or raw properties. Using GUI 116, the user can define filters in order to configure his (or his group's) view to a specific set of jobs and/or job properties. After configuration, the user can save this view for later reuse. When a view is saved for later use, it may show up on a list of available, pre-configured views during login. This feature may give the user the ability to quickly see the same type of information from where he left off last time. Alternatively, the user can start on a new view by selecting from the list of job schedulers in the view. From an example "Job Status" view, the user can select a job 150 and zoom into its details, thereby easily locating or

viewing the specific properties for each desired job 150. The user can also manage job 150 using this particular view of GUI 116. For example, the user can start, stop, or suspend the job, often according to the particular job scheduler 137 capabilities. In addition to the Job Status view, GUI 116 may provide "Alert" and "Dashboard" views. The example Alert view may show alerts that have been generated by job manager 130 or job scheduler 137 in response to a particular filter. In the Alert view, the alert objects 144 may be sorted based a property. For example, when displayed via GUI 116 in a tabular format, alert objects 144 may be sorted according to the severity level property or column. In the event that the number of alert objects 144 is large, the Alert view may provide a scrolling function to enable a user to scroll between alert objects 144. In addition, the user of GUI 116 may perform actions on alert objects 144. For example, the Alert view may enable a user to acknowledgement and closure actions allowing the user or application to acknowledge or close alarms. In this example, the state property of an alarm object may start as open, until it is acknowledged or closed. The example Dashboard view may provide a statistical summary of the jobs and alerts. Moreover, the filters may be applied in the Dashboard view to set the overall severity level of the view. When multiple filters are applied to the Job Status, Alert, or Dashboard views, information from various heterogeneous job schedulers may be collected into one view. This view shows the selected job and all its direct dependencies including its immediate predecessors, successors, triggers, resource and other requirements, and the current status of each. The consolidated data is often presented in a single way in an example "Enterprise" view. Thus, the Job Status, Alert and Dashboard views (as well as others) may be types or children of certain Enterprise views. Another view may be a Map view, which graphically displays the details of a selected job or jobset. Yet another view may be a Server Configuration view, in which the administrator or other authorized user can add, edit, and delete servers or operating environments 106 that are available to job manager 130. This view does not typically create back-end servers. Instead, it creates or populates the configuration information to access the environments 106 based on information supplied by the user. Of course, this configuration information may be automatically retrieved, received, or polled as appropriate. Each view may be static and or dynamic as appropriate. Generally, static views do not change over time, while dynamic views automatically change at a regular update interval or dynamically update according to other criteria. In certain

embodiments, GUI 116 may also present a "Credentialed User" view, allowing the user or administrator to add, edit, and delete credentialed users. The credentialed user information provides login credentials to back-end servers or operating environments 106. Credentialed users are set up to simplify access to the back-end servers/environments 106 and to provide an additional level of security. The portal user ID may be used as a key to access the credentialed user information. In addition to the portal user ID, the system administrator can set an environment password, which can be different than the Portal password. This feature is for users who have access to multiple back-end servers with the same user ID but different passwords for each. In addition, for each user ID in the credentialed user information, an alias ID can be established. The alias ID can be either a group ID (one-to-many or many-to-one) or can be a user's personal ID for the back-end server. The alias ID has an associated password for the back-end server. In addition, a group user/group ID can be set to provide the credentials.

Regardless of the particular view or data, GUI 116 may comprise a plurality of customizable frames or windows having interactive fields, pull-down lists, and buttons operated by the user. In one embodiment, GUI 116 presents information associated with jobs 150, including job status, and associated buttons and receives commands 170 from the user of client 104 via one of the input devices. This information may be presented in tabular, graphical, and any other suitable format. Moreover, it should be understood that the term graphical user interface may be used in the singular or in the plural to describe one or more graphical user interfaces and each of the displays of a particular graphical user interface. Therefore, GUI 116 contemplates any graphical user interface, such as a generic web browser or touch screen, that processes information in system 100 and efficiently presents the results to the user. Server 102 can accept data from client 104 via the web browser (*e.g.*, Microsoft Internet Explorer or Netscape Navigator) and return the appropriate HTML or XML responses using network 112. For example, server 102 may receive a job submission from client 104 using the web browser, execute the particular job 150 in the appropriate operating environment 106, and present the results in the web browser.

In one aspect of operation, a user logs into job manager 130 using GUI 116 and is presented with the following example functionality or views: Administration, Monitoring, Configuration, and Event Management. Both the Administration and

Monitoring views normally includes an applet deployed in an HTML page. The Configuration view is provided by a series of HTML pages that communicate with a Configuration servlet or process. The applets graphically display the objects defined in the job management system. The applet communicates with the appropriate servlet or process to send and receive data to the job management system. Event Management provides web-enabled access to the log facility. Job manager 130 may use the Jacada Terminal Emulator (JTE) to provide host emulation capabilities. In certain embodiments, the user may be provided access to certain functionality based on assignments to Portal workgroups. Based on the particular functionality selected by the user, job manager 130 may invoke a particular module from a Server/Web Server tier. This example level includes applets, servlets, servlet engines, and adapters.

Each servlet serves as a central point of communication and management between the GUI 116 (Applet and/or Portlet) and the one or more operating environments 106. The servlet is generally operable to expose a callable interface to GUI 116 to allow the end-user to configure and monitor jobs. The servlets, in turn, are operable to forward those calls into the various adapters that link with the particular environment 106. The servlets may be further operable to control client sessions. This session control typically involves session management, authentication, and persistency. As described in more detail in the embodiments of FIGURE 2, each individual adapter 135 communicates with the servlets and the associated operating environment 106 and/or job scheduler 137. Adapters 135 encapsulate the user command into an object 150 it for the particular operating environment 106 and/or job scheduler 137. After any suitable amount of processing or job management, job scheduler 137 communicates output or job details to job manager 130 via the appropriate adapter 135 (perhaps in response to worry or at automatically upon completion or error).

At this point, adapter 135 may contain unmodified or native data from each job scheduler 137. However, normalization profiles may give the user, administrator, or job manager 130 the ability to generate, select, or otherwise identify a set of normalized properties to be shown from the possible properties of the various types of jobs and operating environments 106. Job manager 130 may then utilize one or more normalization profiles, which in this example includes a plurality of job status property objects, to normalize the encapsulated job properties. Job manager 130 applies the job

status property objects on each set of jobs in the results 160. The outcome of this process is a set of values, for each job, that are ordered and normalized as needed. This information may be used to generate job objects 140. It could also be displayed in a tabular format at this point for the convenience of the user using GUI 116. In short, there is no specific limitation on how the resulting set of values may be displayed or stored.

The generated job objects 140 may also be filtered or monitored for alerts. For example, system 140 may generate an alert when a specific jobset is successfully completed. As a result of alert filtering, a user may view alerts that are relevant to the user's tasks. In response to a request from the user, job manager 130 identifies one or more alert filter objects 142 using the request. In the jobset example, job manager 130 may identify an alert filter object 142 associated with a job scheduler 137 that is processing the jobset. The user may communicate a request to job manager 130 by making a selection through GUI 116. The request may include information that identifies or is operable to identify one or more alert filter objects 142. In some embodiments, job manager 130 identifies a single alert filter object 142 operable to filter alerts from two or more heterogeneous operating environments 106. After identifying the one or more alert filter objects 142, job manager 130 selects one or more job objects 140 using the one or more identified alert filter objects 142. For example, alert filter object 142 may include information that matches or is operable to match a specific job, job type, jobset, all jobs, or other groups of jobs. Using such information, job manager 130 collects or processes one or more job objects 140 using the one or more alert filter objects 142. Returning to the jobset example, job manager 130 select job objects 140 because they contain a reference identifying the specific jobset. Moreover, job manager 130 may identify one or more job properties for each job object 140. In the example, job manager 130 identifies the job status of each identified job object 140 for determining their success. Job manager 130 identifies alert filter criteria 146 for filtering job objects 140 for alerts. In some embodiments, alert filter criteria 146 comprise a tuple. In this case, the tuple may include a property name, operator, and a value. Turning to the jobset example, alert filter criteria 146 includes the following tuple: Job Status, =, and Success. Job manager 130 then compares the identified job properties with associated alert filter criteria 146. In the case of tuples, job manager 130 may then compare the value to the identified job

property in accordance with the operator. For example, job manager 130 may determine whether the job property equals the value. As in the jobset example, job manager 130 determines whether the job status of each identified job object 140 is equal to "Success." In the event of a match, job manager 130 identifies alert template 148 and instantiates alert object 144. In the jobset example, job manager 130 instantiates an alert object 144 representing that the jobset was successfully executed by the associated job scheduler 137.

FIGURES 2A-E illustrate various configurations of enterprise system 100 for executing jobs in heterogeneous operating environments 106. Generally, these figures illustrate a job manager 130 communicating with a job scheduler 137, resident in one of the operating environments 106, via an associated adapter 135. Put another way, job manager 130 may use adapters 135 to interface, normalize, or otherwise process communications from various heterogeneous job schedulers 137.

Each adapter 135 is an object or other module that encapsulates one or more types of job schedulers 137. Adapters 135 may be written or described in any particular format or programming language. For example, adapter 135 may be a Java object. Regardless of the particular format, adapter 135 is generally operable to provide APIs to job manager 130 for communication with each job scheduler 137 to manage and monitor job information. Put another way, adapter 135 may be logically located between job manager 130 and at least the associated job scheduler 137, thereby allowing job manager to be communicably coupled with the job scheduler 137. In certain embodiments, each adapter 135 may provide this compatibility by invoking, including, exposing, or executing one or more of the following example methods:

Name-	Description
List getJobStatus(List filters)	Returns the job status data according to the given filters
Job getJobDetails(Map params)	Returns the job details
void updateJobDetails(Job job)	Updates the job details
List getRunLog(RunLogFilter filter)	Returns the run log data according to the given filter
List getPriorRun(	Returns the prior run data according to the given

PriorRunFilter filter)	filter
void actionJob(Job job, Map params)	Perform action on the specified job
List getJobPreds(Job job, Map params)	Returns the job predecessors of the specified job
List getJobVRMs(Job job, Map params)	Returns the job VRMs of the specified job
void actionVRM(VRM vrm, Map params)	Perform action on the specified job VRM
void actionPred(Pred pred, Map params)	Perform action on the specified job predecessor

There may be any number of adapters 135, each compatible with any appropriate number of job schedulers 137. For example, system 100 may include a mainframe job adapter 135 that provides APIs to allow communication with a mainframe-based job scheduler 137. These APIs allow the caller to read and write to different objects that exist within the mainframe job scheduler 137. These objects may include jobs, calendars, datasets, ARFSets, ARFs, JCL, triggers and predecessors. In another example, system 100 may include a distributed job adapter 135 that provides APIs to allow communication with a distributed job scheduler 137. This example distributed job scheduler 137 may run on any distributed platform such as Windows and Unix-based operating systems. As with the mainframe adapter 135, the APIs allow the caller to read and write to different objects (such as jobs, calendars and global variables) that exist within the distributed job scheduler 137.

Job scheduler 137 is any executable, routine, service, daemon, or other module or process that is operable to execute, monitor, or otherwise manage jobs 150 in at least one operating environment 106. Typically, job scheduler 137 is associated with a particular type, format, or compatibility of job 150. But, as illustrated in the various embodiments, any job scheduler 137 may be also be configured to run as a more varied job scheduler or even a master job scheduler 137 managing a plurality of slave job schedulers 137. Moreover, while job scheduler 137 is illustrated as residing within a particular operating environment 106, it will be understood that is for example

purposes only and merely illustrates that job scheduler 137 is associated with the particular environment 106. Indeed, job scheduler 137 may be distributed across a plurality of environments, computers, or data stores, without departing from the scope of the disclosure. Job scheduler 137 may be proprietary, off-the-shelf, customized, or any other type of job scheduler. Moreover, enterprise 100 may purchase, download, or otherwise obtain job scheduler 137 using any appropriate technique.

For example, FIGURES 2A-C illustrates at least a portion of system 100 that includes server 102 communicably coupled to first and second operating environments 106. In this example, each operating environment 106 includes one job scheduler 137, each operable to manage jobs 150 for that particular operating environment 106. Job manager 130, illustrated as executing on server 102, is communicably coupled to first job scheduler 137a through a first adapter 135a and to second job scheduler 137b through a second adapter 135b. But, as illustrated in the respective figures, adapters 135 may reside on server 102 and/or the associated operating environment 106 as appropriate. For example, as illustrated in FIGURE 2A, job manager 130 locally includes, loads, or otherwise invokes adapter 135a for executing job 150a, receiving or retrieving job status 160a, or other communications, commands, instructions, and such to first job scheduler 135a. In another example, as illustrated in FIGURE 2B, one or more of the adapters 135 may act as an agent, service, or daemon residing within the operating environment 106 for the appropriate job scheduler 137. In this example, job manager 130 may invoke or interact with remote adapter 135 using a particular port, socket, or method. In yet another embodiment, illustrated in FIGURE 2D, job manager 130 may include one of the job schedulers 137 operable to schedule heterogeneous jobs 150 to a plurality of operating environments 106. In this embodiment, job manager 130 may be considered a logical all-in-one module with internal job scheduling, adapting, and normalizing processes and capabilities.

As illustrated in FIGURE 2E, a particular job scheduler 137 or other application (job manager 130 or other non-illustrated application) may be designed or implemented as a "metascheduler" (137a) that caters to more than one type of job 150 or is compatible with more than one operating environment 106. In this scenario, job scheduler 137a can manage heterogeneous jobs on different platforms, operating systems, or other environments 106. When job scheduler 137a provides the information about such jobs, it may automatically normalize the properties of these

jobs. As illustrated, the "metascheduler" 137a could also control subordinate schedulers 137b and 137c, respectively. "Metascheduler" 137a may be operable to consolidate and normalize the information obtained from the subordinates 137b and 137c as appropriate.

In one aspect of operation, illustrated in FIGURE 2C, when retrieving the details or properties of jobs, adapter 135 communicates with the job scheduler 137 to get the raw values of these job properties. After adapter 135 receives the information, it then translates and normalizes certain properties into a common set of values. In particular, the status property of job 150 is mapped from the set of job scheduler-specific values into a common or customized set of values. In some cases, more than one raw value may be used to map to the common set of values. For example, a mainframe job may include three properties that determine the normalized job status value. These example properties are: queue name, status and specific status. In this example, the raw values are used in combination to map to a common normalized value.

Normalized Value	Mainframe Raw Value (Queue/Status/Specific Status)	Windows/Unix Raw Value
Running	ACT/*any status except WARN/any specific status*	Running
Waiting	REQ/*any status except WARN or ERRX/any specific status except RESTART* RDY/*any status except WARN/any specific status*	Starting, Inactive, Activated, Queue Wait
Success	CMP/*any status except CANCEL/any specific status*	Success
Failure	REQ/ERRX/*any specific status* REQ/*any status*/RESTART	Failure
Cancel	CMP/CANCEL/*any specific status*	Terminated

Restart		Late to Start
On Hold	REQ/WARN/HOLD	Hold
Late to Start	REQ/WARN/*any specific status* RDY/WARN/*any specific status*	
Running Late	ACT/WARN/*any specific status*	
Inactive	FOR/*any status*/*any specific status*	
Unknown	*other values or combinations*	*other values*

In other words, the normalization of job properties can also be performed in job manager 130 instead of the associated adapter 135. Indeed, an example Job Status Console of GUI 166 may also be operable to normalize of the status property of the jobs. That is, adapter 135 may perform no normalizing translation when the raw data is retrieved from the job scheduler 137. This information is then returned to the caller, which is job manager 130 or GUI 116 as appropriate. The calling application then normalizes the job properties using, for example, a technique of mapping the raw values into a set of common values using normalization policies.

FIGURE 3 illustrates one embodiment of the job manager 130. At a high level, this embodiment of job manager 130 includes a connection listener 304, a request controller 308 communicably coupled with a plurality of job parsers and managers, a view controller 314, a session manager 318, a template manager 320, an adapter manager 322, and a profile manager 324. But, of course, these sub-modules are for example purposes only and job manager 130 may include none, some, or all of (as well as other) the illustrated sub-modules. Moreover, one or more of the sub-modules may be remote, dynamically linked, or invoked as appropriate.

Connection listener 304 is any module, library, object, or other process operable to listen (such as on a known port(s)) for connections from clients 104. For example, connection listener 304 may include or implement the following example properties:

Name	Description
portList	List of server ports
serverSocketList	List of server sockets
connectionThreadManager	Connection thread manager

Connection listener 304 may also execute or invoke the following example methods:

Name	Description
void init()	Initialize the server listener
void destroy()	Destroy the server listener
void addPort(int portNumber)	Add a listener port
void removePort(int portNumber)	Remove a listener port

Connection listener 304 may include or be communicably coupled with connection pool 302. Connection pool 302 may be any thread manager module or data structure operable to dispatch outgoing messages to the connection threads for processing. In certain embodiments, connection pool 302 is at least partially responsible for maintaining connection threads for communications between job manager 130 and clients 104. The following table shows example properties of connection pool 302:

Name	Description
threadList	List of connection threads
workerPool	Worker pool

And the following table shows example methods of the connection pool:

Name	Description
void init()	Initialize the manager
void destroy()	Destroy the manager
void addConnection(Socket socket)	Add a new socket connection and instantiate a connection thread to handle it
void	Destroy the socket connection thread

destroyConnection(Socket socket)	
void sendMessage (ResponseMessage msg)	Send a message to the first available connection

After a connection is established, it is assigned to a connection thread in the connection pool, for processing communications. Generally, a connection thread manages a particular connection. For example, the connection may be keyed on or assigned a socket. For example, when an outgoing message is to be sent out, the thread sends the message through the connection using the appropriate socket. When an incoming request is received from client 104, the thread reads the message and unpacks it into a message object. This object is then handed off to the worker pool 306 for processing.

Worker pool 306 is any object or data structure representing the pool of worker threads. Generally, each worker thread object represents a thread that can perform a particular task. For example, the worker thread may accept a unit of work and perform or execute it. When the task is completed, the worker is typically released back into worker pool 306. Worker threads are handed out to perform tasks on behalf of client 104. In certain embodiments, worker pool 306 can be configured to start with a particular number of threads and automatically grow to handle higher loads as necessary. Worker pool 306 may include the following example properties:

Name	Description
workerThreadList	List of worker threads
connectionPool	Connection pool

and implement the following example method:

Name	Description
void process(RequestMessage message)	Process the given request message

Illustrated worker pool 306 is communicably coupled with request controller 308.

Request controller 308 is any module, object, or other process operable to route incoming messages to the appropriate objects 310 and 312. For example, the message may first be sent to the appropriate parser object 310 so that the message may be parsed into a request object. There may be many kinds of request objects, such as one for each type of request. For example, the following table illustrates a number of example request objects:

Type	Parameter	Description
Server Connect	server	Name of server
	user	User name credential
	password	Password credential
Server Disconnect	server	Name of server
Get Job Status	server	Name of server
	view	Name of job status view
Scroll Job Status	view	Name of job status view
	scroll size	Scrolling size
	scroll direction	Direction (forward or back)
Sort Job Status	view	Name of job status view
	property	Which property to scroll by
	direction	Ascending or descending
Save Job Status	view	Name of job status view
	old view name	Previous name of job status view (if any)
	other parameters	Other configuration parameters
Delete Job Status	view	Name of job status view
Get Job Details	job name	Name of job
	server	Server name
	job number	Job number
	job properties	Other job parameters

Update Job Details	job name	Name of job
	server	Server name
	job number	Job number
	job properties	Other job parameters
Job Action	job name	Name of job
	server	Server name
	job number	Job number
	job properties	Other job parameters
	action properties	Action parameters
Get Run Log	server	Server name
	view	View name
Save Run Log	view	View name
	old view name	Previous name of run log view, if any
Delete Run Log	view	Delete the run log view
Get Prior Run	server	Server name
	view	View name
Save Prior Run	view	View name
	old view name	Previous name of prior run view, if any
Delete Prior Run	view	Delete the prior run view
Get Alerts	server	Server name
	view	View name
Update Alerts	server	Server name
	view	View name
	alert properties	Alert properties
	filter properties	Filter properties
Alert Action	alert properties	Alert properties
	action parameters	Parameters to alert action
Get Dashboard	server	Server name
	view	View name
Update Dashboard	server	Server name

	view	View name
	dashboard properties	Dashboard properties
	filter properties	Filter properties
Open session		Create new session
Close session	session ID	Destroy session

In certain embodiments, the request object encapsulates data pertinent to the request, including ID, session, request parameters, and more. For example, the request object may have the following properties:

Name	Description
requestId	Request ID
session	Session
response	Response to this request, if any

Each request object may implement or invoke the following example methods:

Name	Description
int getRequestId()	Returns the request ID
void setRequestId(int id)	Sets the request ID
Session getSession()	Returns the session
void setSession(Session session)	Sets the session
IResponse getResponse()	Returns the response associated with this request, if any
void setResponse(IResponse response)	Sets the response

Based on the incoming message's request ID, a parser manager provides the appropriate parser object 310 to unpack the message into the request object. Parser object 310 is then invoked to unpack the message. It will be understood that there may be any number of parser objects 310, such as one for each type of request. For

example, the parser manager may include or be coupled with one or more of the following example parser objects 310:

Name	Description
JobStatusParser	Parses requests pertaining to job status
JobDetailsParser	Parses requests pertaining to job details
JobActionsParser	Parses requests pertaining to job actions
ServerParser	Parses requests pertaining to server actions
OEParser	Parses requests pertaining to operating environment specific objects
AlertParser	Parses requests pertaining to alerts
DashboardParser	Parses requests pertaining to dashboard
SessionParser	Parses requests pertaining to session

These example parser objects 310 may implement, execute, or produce the following request messages:

Parser	Request Message
JobStatusParser	Get Job Status
	Scroll Job Status
	Sort Job Status
	Save Job Status
	Delete Job Status
JobDetailsParser	Get Job Details
	Update Job Details
JobActionsParser	Job Action
ServerParser	Server Connect
	Server Disconnect
CA7Parser	Get Run Log
	Save Run Log
	Delete Run Log
	Get Prior Run

	Save Prior Run
	Delete Prior Run
AlertParser	Get Alerts
	Update Alerts
	Alert Action
DashboardParser	Get Dashboard
	Update Dashboard
SessionParser	Open Session
	Close Session

After the request object is produced by parser object 310, the request is routed to one of the handler objects 312 for subsequent processing. The handler manager processes a request object, which often includes the object ID. Based on the request object ID and other information, the handler manager routes the request object to the correct handler object 312. Each handler 312 is responsible for processing the request using operating environment 106, adapters 135, and job schedulers 137 as appropriate. As with parser objects 310, there are typically many handler objects 312, such as one for each type of request. In certain embodiments, each handler 312 is responsible for performing or requesting the work that is requested. For example, each handler may be operable to load, invoke, or communicate with the appropriate adapter 135 based on the request object. As a result of its processing, a response object is produced. This response object is returned along with the request object, after processing (typically through adapter 135). The following table shows an example list of handlers 312:

Name	Description
JobStatusHandler	Processes requests pertaining to job status
JobHandler	Processes requests pertaining to job details
JobHandler	Processes requests pertaining to job actions
ServerHandler	Processes requests pertaining to server actions
OEHandler	Processes requests pertaining to OE specific objects
AlertHandler	Processes requests pertaining to alerts
DashboardHandler	Processes requests pertaining to dashboard

SessionHandler	Processes requests pertaining to the session

The following table maps requests to example handlers:

Handler	Request
JobStatusHandler	Get Job Status
	Scroll Job Status
	Sort Job Status
	Save Job Status
	Delete Job Status
JobHandler	Get Job Details
	Update Job Details
	Job Action
OEHandler	Get Run Log
	Save Run Log
	Delete Run Log
	Get Prior Run
	Save Prior Run
	Delete Prior Run
AlertHandler	Get Alerts
	Update Alerts
	Alert Action
DashboardHandler	Get Dashboard
	Update Dashboard
SessionHandler	Open session
	Close session

As described above, the processing by each handler object 312 results in a response object. This example response object is then fed into view controller 314 to produce the response that, in turn, is returned as the outgoing message to client 104.

View controller 314 routes a processed request object (along with its response object, if any) to the correct objects. First, the request is fed to a view manager, which

is operable to generate a view for use by GUI 116. The view manager provides, calls, or other executes view handlers to process requests into views. For example, it may route the request to the correct view handler. There are any number of handler objects, such as one for each type of view.

Name	Description
JobStatusHandler	Processes responses pertaining to job status
JobDetailsHandler	Processes responses pertaining to job details
JobActionsHandler	Processes responses pertaining to job actions
ServerHandler	Processes responses pertaining to server actions
CA7Handler	Processes responses pertaining to CA-7 specific objects
AlertHandler	Processes responses pertaining to alerts
DashboardHandler	Processes responses pertaining to dashboard

The view manager is responsible for processing the given request into an end-user view. As a result of its processing, a view object is produced or updated. This view object is returned along with the request object after processing. After the view is produced, the response object is then sent back to client 104. It will be understood that the response object may comprise any particular data or instruction in any suitable format. There may be any number of types or sub-classes of response objects. For example,

Type	Property	Description
Server Connect		
Server Disconnect		
Get Job Status	jobStatus	Job Status object
Scroll Job Status	jobStatus	Job Status object
Sort Job Status	jobStatus	Job Status object
Save Job Status	jobStatus	Job Status object
Delete Job Status		
Get Job Details	Job	Job object
Update Job Details	Job	Job object

Job Action	job	Job object
	action returns	Other action return values
Get Run Log	RunLog	Run Log object
Save Run Log	RunLog	Run Log object
Delete Run Log		
Get Prior Run	PriorRun	Prior Run object
Save Prior Run	PriorRun	Prior Run object
Get Alerts	alerts	Alerts object
Update Alerts	alerts	Alerts object
Alert Action	alert	Alert object
	action returns	Other action return values
Get Dashboard	dashboard	Dashboard object
Update Dashboard	dashboard	Dashboard object
Open session	session	Session object

In certain embodiments, the response object encapsulates most or all of the data pertinent to the response, such as output information, errors, and more. This response object may include some or all of the following example properties:

Name	Description
Request	Request associated with this response, if any
Buffer	Buffer containing response
Exception	Exception, if any
errorMessage	Error message, if any
errorCode	Error code, if any

Moreover, in certain embodiments, the response object may include the following example methods:

Name	Description
IRequest getRequest()	Returns the request associated with this response
void setRequest(IRequest request)	Sets the request
StringBuffer getBuffer()	Returns the response buffer
void setBuffer(StringBuffer buffer)	Sets the response buffer

Illustrated job manager 130 also includes session manager 318. In this embodiment, session manager 318 is any module generally responsible for handling sessions. In other words, it creates, stores, and destroys sessions that are assigned to each unique client 104, often utilizing a map of the current sessions. The session typically maintains persistent information for a unique client 104 for the lifetime of the connection. Certain back-end objects specific to client 104 are stored and reachable from the client's session. In certain embodiments, session manager 318 implements the following example methods:

Name	Description
Session createSession()	Creates a new session
void destroySession(Session session)	Destroy the given session
Session findSession(String sessionId)	Return the session that matches the given session ID, if any
void init()	Initialize the session manager
void destroy()	Destroy the session manager

Session manager 318 may automatically cull inactive or abandoned sessions that exceed a timeout period. For example, certain sessions are governed by an idle timeout. If this session is kept idle beyond a configurable timeout period, then session manager 318 may clean it up automatically. In this example, all objects – views, models, adapters, etc. – associated with the session are destroyed. A next request by the user may result in an error indicating an unknown session or bad session. But, if

the view is dynamic, then the view may be responsible for sending the timeout event at the point where the manager cleans up the session (and its views).

Template manager 320 may be any module operable to manage templates, which are generally stored as objects in HTML files with placeholder variables representing dynamic sections. But in certain circumstances, templates may not be complete <html> blocks. Some may represent small sections of a complete page such as a frame, table, graph, etc. At runtime, the component sections are typically replaced by the actual data. Template objects are identified by their file names. Since they are often uniquely named on the file system, there may be no need to invent a new tagging scheme to identify them. Once requested, executed, or otherwise located, a transformation of the template yields the output that is returned to the user through GUI 116. During startup, initialization, or at any other appropriate time, job manager 130 reads in or loads the desired templates. Templates are often preprocessed after they are read from the file system. Each template may be encapsulated inside an object that uses a vector. Each entry in the vector contains an internal object that is either a static portion of the template or a dynamic portion represented by a variable name. When the entries are traversed in order and printed out, the resulting output resembles the template file. This process may be called printing. The template object exposes the printing functionality with a parameter. The caller provides a map that contains variable names and values as its parameter. When the template object encounters a variable name in the vector while printing, it uses the map to resolve the variable name into a value. That value is then printed in lieu of the variable; otherwise, the variable may be deemed empty. Sometimes, template manager 320 executes code in response to a variable entry in the vector. The caller can register callbacks with the object for this scenario. Callbacks can be registered for specific variable name, index number, or all variables. Parameters to a callback include the current vector entry and working buffer of the printing process. Template manager 320 hands these objects out to transformers as necessary. Transformers can use the same template object simultaneously. In this scenario, the template object is responsible for safely supporting multiple callers.

Adapter manager 322 is responsible for handling adapter wrappers, often utilizing a map of adapters. The adapter wrapper encapsulates a local or back-end adapter 135. By providing a high-level interface layer on top of each adapter 135, the

wrapper provides a consistent and semantic set of methods to each type of job scheduler. Typically, adapter manager 322 creates, stores, and destroys wrappers that are assigned to each unique back-end connection or environment 106. In certain embodiments, adapter manager 322 implements the following example methods:

Name	Description
AdapterWrapper getAdapter(String server)	Creates or returns the adapter wrapper for this server
void init()	Initialize the adapter manager
void destroy()	Destroy the adapter manager

Profile manager 324 is responsible for handling profile objects such as, for example, servers, users, groups and views. In this example, the server profile object encapsulates a configured server, the user profile object encapsulates a user record, the group profile object encapsulates a Portal group record, and the view profile object encapsulates a view record. The profile manager 324 communicates with configuration, Portal, and its own data store to create, update and delete these objects. In certain embodiments, profile manager 324 includes the following example methods:

Name	Description
ServerProfile getServer (String serverName)	Returns the server profile matching the given name
UserProfile getUser(String userName)	Returns the user profile matching the given name
GroupProfile getGroup (String groupName)	Returns the group profile matching the given name
ViewProfile getView (String userName, String viewName)	Returns the view profile matching the given name, that is accessible to the user
List getServers()	Returns the list of servers
List getUsers()	Returns the list of users
List getGroups()	Returns the list of groups

List getViews(String username)	Returns the list of views that are accessible to the user

It will be understood that the foregoing sub-modules, properties, and methods are for illustration purposes only. Indeed, each of these sub-modules, properties, and methods may or may not be present in certain job managers 130. Put another way, job manager 130 includes any processes or data storage operable manage jobs 150 and may include none, some, or all of the foregoing example embodiments without departing from the scope of the disclosure.

In one aspect of operation, a flow describes a path of execution of a client request through job manager 130. The request typically originates from GUI 116 and results in a new or updated page that is returned to the browser. When the servlet receives a request, it is routed the request controller 308. This controller 308 produces a request object that encapsulates the HTTP request and response. Request controller 308 then forwards this object to parser manager 310. Parser manager 310 is comprised of one or more parsers. Each parser inspects the request and breaks it down into various pieces of information as appropriate. For example, the session ID and request ID are extracted. The parser may use this information to look up objects that are relevant to the request. For example, the session ID translates to a session object. When control returns to request controller 308 from the parser, the request object is forwarded to handler manager 312.

Handler manager 312 is comprised of one or more handlers. Based on information in the request object such as the request ID, handler manager 312 forwards the request to the corresponding handler. Each handler may be considered an "atomic" piece of business logic dedicated to servicing a request. A handler often depends on other objects to accomplish its work. Some of these objects include adapters 135, model objects, and other manager objects. For example, when a job status handler executes, it uses the correct adapter instance 135 in conjunction with the job status model object to accomplish its work. When the handler finishes its work, it produces a response object. A response object can contain different pieces of information such as output data, error codes, and others. Handler manager 312 returns this response object to request controller 308.

Request controller 308 forwards the response object to view controller 314. View controller 314 is comprised of one or more view objects. Each object is dedicated to producing a specific view such as job status. The job status response object provides the information to the view to produce the output for the browser. Views are normally closely tied to templates. Template manager 320 provides HTML templates that form the basis for the output. The final output is a combination of data from a response object and a template. After the output is composed, view controller 314 sends it to client 104. Control then returns to request controller 308 and out of the servlet.

FIGURE 4A illustrates an example alert filter object 142 and child or associated job property object in accordance with one embodiment of the present disclosure. As illustrated, alert filter object 142 may include a number of child records or objects. For example, one record or object (such as a Java job status model object) represents the job status view for a particular job 150. In certain embodiments, each object 142 contains a collection of alert filters 402. Alert filter 402 represents a filter that job manager 130 may apply to job objects 140 for identifying transitions in job states and generating alert objects 144. Alert filter 402 may be associated with a particular operating environment 106. Alert filter 402 may be a collection of property tuples 404 and alert definitions 406. Each example property tuple 404 comprises three values: property name, property operator, and property value. The property name contains the name or alias of the property that is matched from the job definition or instance. For example, the name could be "Job ID" representing the ID number of the job. The property operator contains the type of comparison to perform on the property value. For example, the operator could be "=" representing the "equals" comparison. The property value contains the value to match or compare. For example, the value could be "100." In certain embodiments, alert filter 402 may allow multiple property operators and/or property values in the same property tuple 404. For example, there could be multiple values specified in a special format as the property value, such as "100,101,102." The interpretation of the specification of multiple property operators and/or property values in the filter may depend on the property name in question. In addition to the collection of property tuple 404, alert filter 402 may contain a reference, identifier, or other pointer that identifies or is operable to identify one or more job objects 140. In some embodiments, the reference may be identified an

instance of a job scheduler 137 by a machine name, network address, database name, or a combination of system, network, database and proprietary identifiers that represent a unique installation of the associated job scheduler 137 or operating environment 106. This reference may later be resolved to the set of information in order to perform a network connection or API call into that instance of a job scheduler 137. In one aspect of operation, job manager 130 first groups the alert filters 402 according to their references to instances of job schedulers 137. (*see* FIGURE 4B). For example, if there are five alert filters 402 and three of them refer to Job Scheduler "S1" and two refer to job scheduler "S2," then two filter groups may be formed: one for S1, which contains three alert filters 402, and another for S2, which contains two alert filters 402. Then alert filter object 142 distributes each filter group to a unique worker thread for processing. In this example, each worker thread invokes an adapter 135 for job scheduler S1 and S2 that is associated with the type of job scheduler 137. (*see* FIGURE 4C). Moreover, each worker thread passes filter criteria to the associated adapters API which in turn converts requests to forms that are compatible to the job schedulers. Job schedulers S1 and S2 return job properties to the worker threads via associated adapters 135. (*see* FIGURE 4D). Job manager 130 then waits for the threads to complete the processing before continuing. In the event of a match, job manager 130 instantiates associated alert objects 144 using alert definition 406. (*see* FIGURE 4E). Alert objects 144 may then be stored in system 100. For example, alert objects 144 may be recorded to disk via a file system or database. (*see* FIGURE 4F).

Based, at least in part, on alert definition 406, job manager 130 instantiates alert objects 144. Alert definition 406 is one example of alert template 148. Each alert definition 406 includes one or more the following properties in the table below.

Property	Type	Description
Job	String	Name of job
Jobset	String	Name of jobset
Status	Enum	Status of job
Server	String	Name of server

Then, the following properties may need to be provided in order to generate alert object 144.

Property	Description
Class	Class of alert
Queue	Name of alert queue
Status	Status of alert
Text	Text description for alert
Severity	Severity level of alert
Creation Time	Creation date-time
Update Time	Last update date-time
Job Properties	Other job properties to be included in alert
URL	URL reference to content related to alert

The properties identified above are for illustration purposes only. Alert definition 406 may use the some, all, or different properties that are used to generate alert objects 144.

FIGURES 5A-F are example displays for presenting various normalized properties of heterogeneous jobs as executed in accordance with one embodiment of system 100. It will be understood that illustrated web pages 116a-116f, respectively, are for example purposes only. Accordingly, GUI 116 may include or present data, such as normalized or raw job properties, in any format or descriptive language and each page may present any appropriate data in any layout without departing from the scope of the disclosure.

Turning to the illustrated embodiments, FIGURE 5A illustrates an example job requirements or job properties view 116a. In this view 116a, the user may be able to view or modify various properties of job 150 or jobset. In other words, job properties view 116a is a graphical representation of the objects that can be included in the definition of the job. Job objects may include: job predecessor; job successor; triggers; calendar; VRM requirements; dataset predecessors; user requirements; and network predecessors. The dialog may be a modeless frame that contains a context sensitive panel for displaying the graphical view of the selected item's objects. This frame may contain a palette on the left side that has a list of objects that can be created for the

selected object. On the right may be the graphical layout of the objects for the selected item. Users may have the option to drag items from the palette and drop them onto the graphical layout. Dragging and dropping an object may create a new object, but the user often fills in the properties for that object in the main view. Upon dropping the object, an icon may appear in the graphical layout. Also, the main view may select the new object and display its properties so the user may fill in any missing attributes. Until the user fills in required properties, all icons representing the new object may have a graphical design that alerts the user that the object is incomplete.

Accordingly, job properties view 116a gives the user the ability to drag existing objects into the job properties view 116a from the main panel's tree view. Job properties view 116a may not allow invalid objects to be dropped and the cursor may change to a "No" symbol to notify the user. When a valid object is dropped, an icon may appear in the job properties view 116 layout and the main view may select the dropped object and display its properties. Job properties view 116a may always be locked onto the object that was selected when it was launched. Users may have the ability to select objects in the main view without job properties view 116a changing. When the user is finished changing the requirements for job 150 or jobset, the applet may provide the option to either close the dialog or change the job properties view 116a's selection to edit another object's requirements. Job properties view 116a may display a blank panel if the user deletes the selected job 150 or jobset from the view. When the user selects an object in job properties view 116a, the main view may select the same object and display its properties.

FIGURE 5B illustrates an example job status view 116b. In certain embodiments, job status view 116b consolidates the jobs that are filtered for that particular view and displays associated properties. Of course, this view may be customizable across the enterprise or individually. In other words, the particular view may display properties as selected by the user. This view can be sorted by column (or property). The user is also allowed to scroll between sets of jobs when the number of jobs exceeds the window size set for the view. Also, the user is allowed to jump directly into a specific set, the starting set, and the ending set. The order of the properties can also be defined. Often, job status view 116b displays the normalized properties, opposed to the raw data. This allows the user to sort, filter, and otherwise view heterogeneous jobs in a consistent interface. For example, the first two displayed

jobs "testjob" and "WhereDidAllTheBoxJobsGo" may be a UNIX job and a mainframe job, respectively. Yet, view 116b presents a common property, "Success," for both jobs after normalizing the native values.

FIGURE 5C provides an example overall view 116c of all the various BSVs, to which the particular user has access, in list type view. For each BSV, view 116c shows the number of objects in various pre-defined statuses. When user navigates through rows of the list view by pressing the left mouse button or the up/down arrow keys in the keyboard, the toolbar will show the corresponding enabled icons for the selected BSV. When there is a selected row and user sorts the rows, the toolbar icons will be enabled/disabled according to the newly selected row object after the sort. If user right-clicks any row, a context menu will appear that shows the same enabled menu items, and the toolbar icons will be enabled/disabled accordingly. As with the other views, view 116c may display or process the various properties after appropriate ones have been normalized.

FIGURES 5D, 5E, and 5F illustrates various graphical or tabular views (116d 116e, and 116f) of the various jobs and job properties. For example, the user may select a loaded BSV from the tree, resulting in the BSV details in multiple tabs in the right pane. In this case, this view summarizes the status of the jobs and jobsets included in this BSV and can be displayed as a bar chart or pie chart. These charts show the number of jobs at different status. Each status is represented with a color and this helps in understanding the overall health of the system at a glance. The user can typically switch between these two chart styles using the toolbar option. As with the other views, views 116d 116e, and 116f may each display or process the various properties after appropriate ones have been normalized.

FIGURES 6A-6E are example displays 116g through 116k associated with alert filter objects 142 in accordance with one embodiment of system 100. As with views 116a-f, it will be understood that illustrated web pages 116g through 116k are for example purposes only. Accordingly, GUI 116 may include or present data, such as statistical information of jobs states and alerts states, in any format or descriptive language and each page may present any appropriate data in any suitable layout without departing from the scope of the disclosure.

Turning to the illustrated embodiments, FIGURE 6A illustrates an example Alert view 116g (discussed above). View 116g provides a summary of alert objects

144 in tabular form including the following properties: ID, server, severity, time, type, and status. In this view, alert objects 144 may be sorted based on a property. For example, when displayed via GUI 116 in a tabular format, alert objects 144 may be sorted according to the severity level property or column. In the event that the number of alert objects 144 is large, view 116g may provide a scrolling function to enable a user to scroll between alert objects 144. In addition, the user may select an alert object 144 from the display in order to view additional information and/or perform actions.

As to view 116h in FIGURE 6B, the user may be able to view available alert filter objects 142 and select desired filters such as through a checkbox. As illustrated, view 116h includes "Filter 1" and "Filter 2" as available alert filters. Both or a single alert filter may be selected and applied to associated job objects 140. Regarding view 116i in FIGURE 6C, a user of system 100 may create an alert filter thereby adding an available alert filter to the list display in view 116h. In the process of adding a filter to system 100, a user may provide values for the various fields displayed in view 116i. These steps may be performed by entering information in available fields and/or selecting values from dropdown menus. In addition, a user may provide and/or select the information to be included in an alert object in the event of a match. As with the filter criteria, the alert properties entered in available fields and/or selecting from a dropdown menu. In the process of performing these tasks and/or other tasks, job manager 130 may use one or more of the following methods in the table below.

Name	Description
List getAlertFilters()	Returns the filters
AlertFilter getAlertFilter(String name)	Returns a specific alert filter
void setAlertFilter(AlertFilter filter)	Updates a specific alert filter
void addAlertFilter(AlertFilter filter)	Add a specific alert filter
void removeAlertFilter(String name)	Remove a specific alert filter
List getAlerts(Filter filter)	Return alerts based on given filter
void setAlert(Alert alert)	Updates a specific alert
List getAlarms(Filter filter)	Returns alarms based on given filter
void setAlarm(Alarm alarm)	Updates a specific alarm

It will be understood that these methods are for illustration purposes only. Job manager 130 may use some, none, or all of the illustrated methods without departing

from the scope of the disclosure. Regarding view 116j in FIGURE 6D, in the process of generating an alert filter object 142, the user may select properties in which to base the filter on. In other words, view 116j enables the user to select the properties to be included in the generated alert filter object 142. In the illustrated embodiment, view 116j presents to tables to the user: Available Properties and Selected Properties. The table labeled Available Properties identifies the properties that the user may add to alert filter object 142. The table labeled Selected Properties identifies the properties that the user has already selected to be included in alert filter object 142. As to view 116k in FIGURE 6E, view 116k display properties of an associated alert object 144 and enables a user perform an action such as Acknowledge or Open on the associated alert object 144. In addition, the user may enter comments in the comment field.

FIGURE 7 is a flowchart illustrating an example method 700 for submitting a job 150 in one or more of a plurality of heterogeneous operating environments 106 in accordance with one embodiment of the present disclosure. At a high level, method 700 includes receiving a job submission from a user and executing job 150 in the appropriate operating environment 106 (or operating environments 106). The following description focuses on the operation of job manager 130 in performing method 700. But system 100 contemplates using any appropriate combination and arrangement of logical elements implementing some or all of the described functionality.

Method 700 begins at step 702, where job manager 130 receives a job request from the user, typically using client 104. But, as described above, the user may submit job request directly to server 102 without departing from the scope of method 700. The job request may comprise one or more of the following types of jobs: an update job, a command, a task, or any other appropriate enterprise job or request. Next, at step 704, job manager 130 authenticates the user. This authentication may include verifying that the user can submit this particular type of job, can update the requested or associated data, or any other security or authentication procedure or technique. Of course, while not illustrated, modules other tha job manager 130 may perform this authentication and communicate the results to job manager 130. Job manager 130 then identifies a job object 140 using the received job request at step 706. For example, the job request may include a job identifier or other pointer. In this example, job manager 130 queries the plurality of job objects 140 to determine the particular job object 140

associated with the request based on the pointer. Once the appropriate job object 140 is identified, Job manager 130 identifies operating environments 106 for the job at step 708. As described above, in the case of a distributed job, there may be more than one operating environment 106 associated with the job. Job manager 130 may identify the appropriate operating environment 106 using any suitable technique. For example, job manager 130 may determine the appropriate operating system to execute the job. In another example, job manager 130 may identify the location of the data storage associated with the job request. In yet another example, job manager 130 may identify the appropriate virtual location for executing the job request. Next, at step 710, job manager 130 invokes a job scheduler 137 in the identified operating environment 106. Once job manager 130 has invoked job scheduler, it may execute the job using the invoked job scheduler 137 at step 712. It will be understood that this execution may include an immediate submission, adding the job to queue associated with the invoked job scheduler, or any other appropriate technique.

FIGURE 8 is a flowchart illustrating example method 800 for filtering alerts in accordance with one embodiment of the present disclosure. Generally, method 800 describes one example technique for job manager 130 to filter one or more jobs objects 140 associated with heterogeneous operating environments 106 for state transitions and generate alert objects 144 in response to determining a match. The following descriptions will focus on the operation of job manager 130 in performing this method. But, as with the previous flowchart, system 100 contemplates using any appropriate combination and arrangement of logical elements implementing some or all of the described functionality.

Method 800 begins at step 802, where job manager 130 receives a status request from a user, typically at client 104. Next, at step 804, job manager 130 identifies a first alert filter object 142 using the status request. For example, job manager 130 may identify a first alert filter object 142 associated with the research and development (R&D) department. At step 806, job manager 130 identifies a property name, an operator, a value, an alert definition, and one or more references using alert filter object 142. In the R&D example, alert filter object 142 may include a failed state expression and an identifier operable to identify job objects 140 associated with the R&D department. At step 808, job manager 130 identifies job objects 140 using the one or more references. In some embodiments, job objects 140 are associated with

heterogeneous operating environments 106. Next, a step 810, a property of each identified job object 140 is identified in accordance with the property name. Returning to the example, job manager 130 may identify the job state of each identify job object 140. The property and the value are compared in accordance with the operator a step 812. If a match is determined that decisional step 814, then job manager 130 instantiates an associated alert object 144 using alert template 148. In the example, job manager 130 instantiates an alert object for those job objects associated with the R&D group that contain a failed state. Otherwise, execution proceeds to decisional step 818. If additional job objects 140 are available, then the next job object 140 is identified at step 820. Otherwise, execution proceeds to decisional step 822. If additional alert filter objects 142 are identified using the status requests, job manager 130 identifies the next alert filter object 142 at step 824. Otherwise, job manager 130 generates a presentation of the alert objects 144 at step 826. Next, at step 828, job manager 130 communicates the presentation to the requesting user.

The preceding flowcharts and accompanying description illustrate exemplary methods 700 and 800. System 100 contemplates using any suitable technique for performing these and other tasks. It will be understood that method 800 is for illustration purposes only and that the described or similar techniques may be performed at any appropriate time, including concurrently, individually, or in combination. In addition, many of the steps in these flowcharts may take place simultaneously and/or in different orders than as shown. Moreover, system 100 may use methods with additional steps, fewer steps, and/or different steps, so long as the methods remain appropriate.

Although this disclosure has been described in terms of certain embodiments and generally associated methods, alterations and permutations of these embodiments and methods will be apparent to those skilled in the art. Accordingly, the above description of example embodiments does not define or constrain this disclosure. Other changes, substitutions, and alterations are also possible without departing from the spirit and scope of this disclosure.

WHAT IS CLAIMED IS:

1. A job manager operable to:

invoke an alert filter, wherein the alert filter is compatible with a plurality operating environments;

identify one or more properties of a first job associated with a first operating environment;

identify one or more properties of a second job associated with a second operating environment, wherein the first operating environment and the second operating environment are heterogeneous;

generate a first alert object in response to a first match between the alert filter and the identified properties of the first job; and

generate a second alert object in response to a second match between the alert filter and the identified properties of the second job.

2. The job manager of Claim 1, at least one of the identified properties of the first job and at least one of the identified properties of the second job are normalized to a same value.

3. The job manager of Claim 1, further operable to:

retrieve the one or more properties of the first job from the first operating environment;

normalize the one or more properties of the first job at a third operating environment; and

compare the normalized properties of the first job to the alert filter.

4. The job manager of Claim 1, wherein the job manager operable to generate a first alert object comprises the job manager operable to:

retrieve the one or more properties of the first job from the first operating environment; and

instantiate the first alert object by population a portion of the alert filter with the retrieved data.

5. The job manager of Claim 4, wherein each selection includes a property name, an operator, and a value.

6. The job manager of Claim 1, further operable to communicate the first and second alert objects to one of a client, a database, or a user.

7. The job manager of Claim 1, wherein the job manager comprises:

a first worker thread associated with the first operating environment, the first worker thread operable to process the alert filter;

a second worker thread associated with the second operating environment, the second worker thread operable to process the alert filter;

a first adapter associated with the first worker thread, the first adapter operable to convert information to a form compatible with the first operating environment; and

a second adapter associated with the second worker thread, the second adapter operable to convert information to a form compatible with the second operating environment.

8. A system for providing alerts for heterogeneous jobs, comprising:

memory storing a plurality of alert filters, each alert filter compatible with a plurality operating environments; and

one or more processors operable to:

periodically invoke one of the alert filters;

identify one or more properties of a first job associated with a first of the plurality of operating environments;

identify one or more properties of a second job associated with a second of the plurality of operating environments, wherein the first operating environment and the second operating environment are heterogeneous;

generate a first alert object in response to a first match between the alert filter and the identified properties of the first job; and

generate a second alert object in response to a second match between the alert filter and the identified properties of the second job.

9. The system of Claim 8, at least one of the identified properties of the first job and at least one of the identified properties of the second job are normalized to a same value.

10. The system of Claim 8, the processors further operable to:

retrieve the one or more properties of the first job from the first operating environment;

normalize the one or more properties of the first job at a third operating environment; and

compare the normalized properties of the first job to the alert filter.

11. The system of Claim 8, wherein the processors operable to generate a first alert object comprises the processors operable to:

retrieve the one or more properties of the first job from the first operating environment; and

instantiate the first alert object by population a portion of the alert filter with the retrieved data.

12. The system of Claim 11, wherein each selection includes a property name, an operator, and a value.

13. The system of Claim 8, the processors further operable to communicate the first and second alert objects to one of a client, a database, or a user.

14. The system of Claim 8, the processors further operable to execute:

a first worker thread associated with the first operating environment, the first worker thread operable to process the alert filter;

a second worker thread associated with the second operating environment, the second worker thread operable to process the alert filter;

a first adapter associated with the first worker thread, the first adapter operable to convert information to a form compatible with the first operating environment; and

a second adapter associated with the second worker thread, the second adapter operable to convert information to a form compatible with the second operating environment.

15. A method for providing alerts for heterogeneous jobs, comprising:

invoking an alert filter, wherein the alert filter is compatible with a plurality operating environments;

identifying one or more properties of a first job associated with a first of the plurality of operating environments;

identifying one or more properties of a second job associated with a second of the plurality of operating environments, wherein the first and the second of the plurality of operating environments are heterogeneous;

generating a first alert object in response to a first match between the alert filter and the identified properties of the first job; and

generating a second alert object in response to a second match between the alert filter and the identified properties of the second job.

16. The method of Claim 15, at least one of the identified properties of the first job and at least one of the identified properties of the second job are normalized to a same value.

17. The method of Claim 15, further comprising:

retrieving the one or more properties of the first job from the first operating environment;

normalizing the one or more properties of the first job at a third operating environment; and

comparing the normalized properties of the first job to the alert filter.

18. The method of Claim 15, the method further comprising generating the alert filter based, at least in part, on one or more inputs from a user.

19. The method of Claim 18, wherein each selections includes a property name, an operator, and a value.

20. The method of Claim 15, the method further comprising communicating the first and second alert objects to one of a client, a database, or a user.

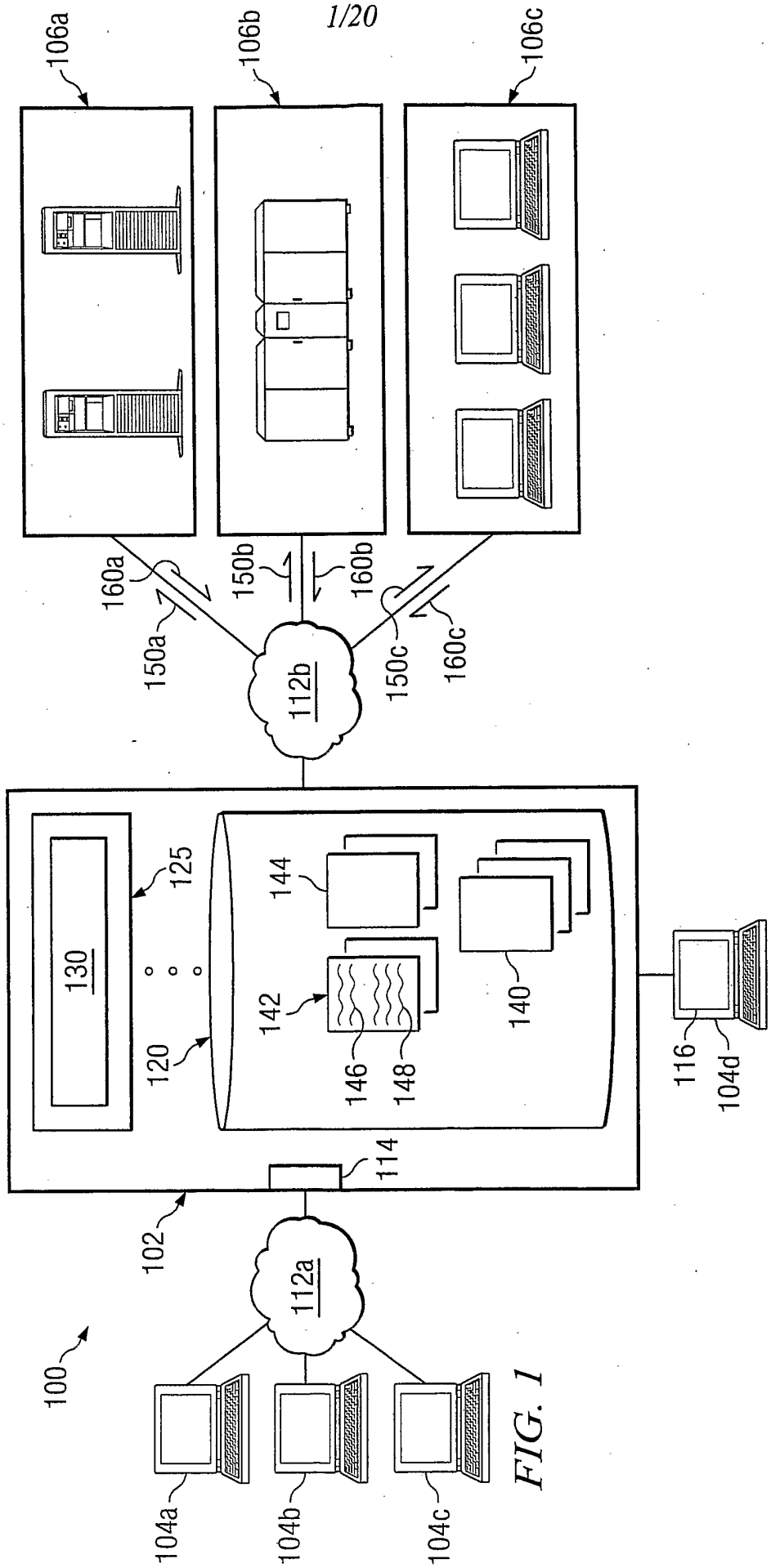

100
102
104a
104b
104c
106a
106b
106c
112a
112b
114
116
104d
120
125
130
140
142
144
146
148
150a
150b
150c
160a
160b
160c

FIG. 1

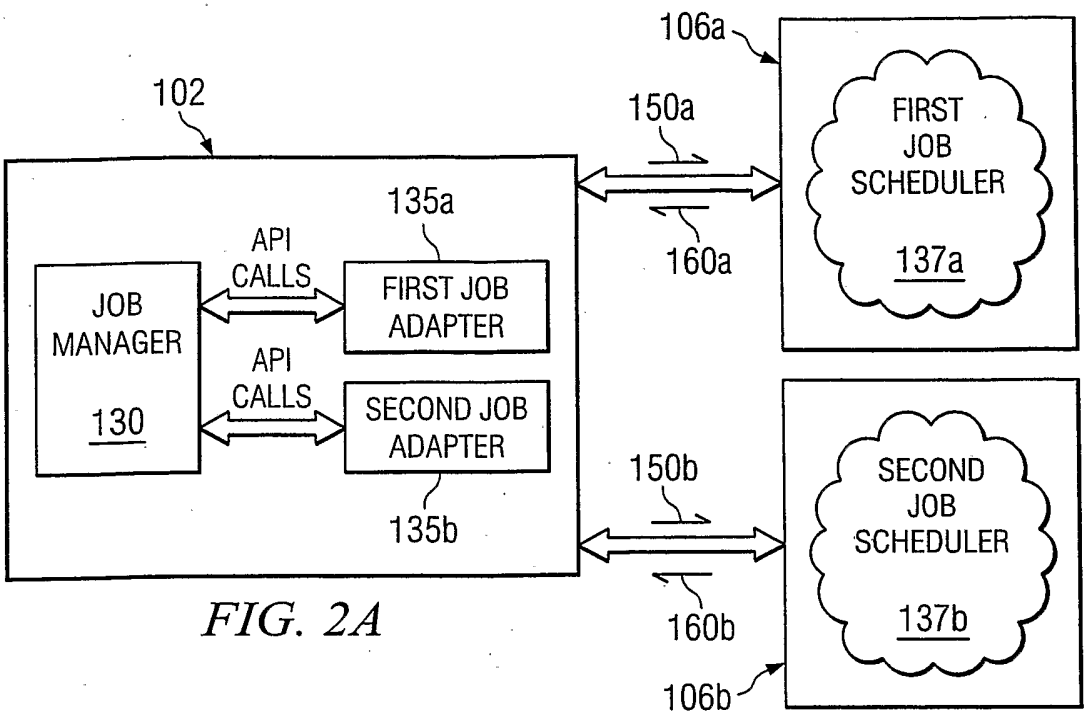

106a
102
150a
FIRST
JOB
SCHEDULER
137a
135a
API
CALLS
JOB
MANAGER
130
FIRST JOB
ADAPTER
160a
API
CALLS
SECOND JOB
ADAPTER
150b
SECOND
JOB
SCHEDULER
137b
135b
160b
106b

FIG. 2A

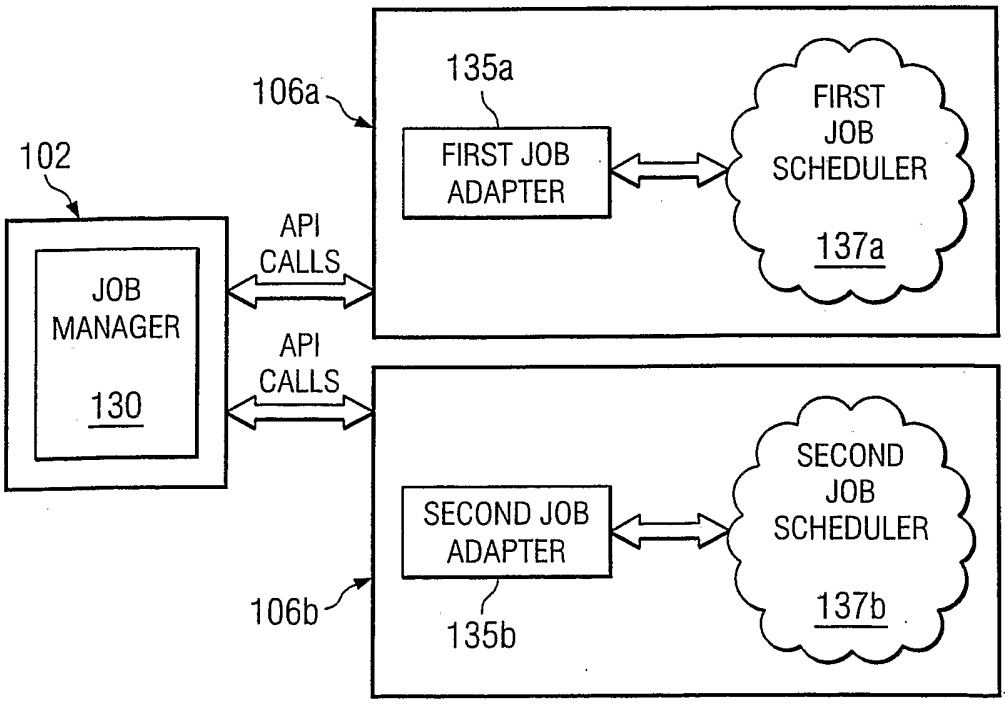

135a
106a
FIRST JOB
ADAPTER
FIRST
JOB
SCHEDULER
137a
102
API
CALLS
JOB
MANAGER
130
API
CALLS
SECOND JOB
ADAPTER
SECOND
JOB
SCHEDULER
137b
106b
135b

FIG. 2B

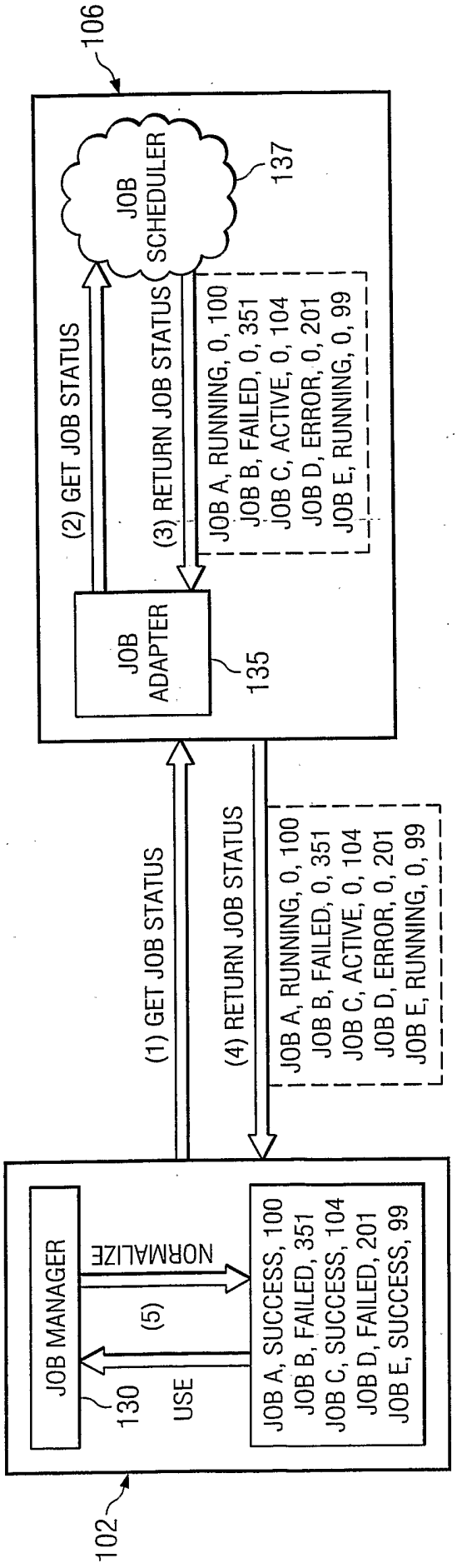

102
106
JOB SCHEDULER
137
(2) GET JOB STATUS
(3) RETURN JOB STATUS
JOB A, RUNNING, 0, 100
JOB B, FAILED, 0, 351
JOB C, ACTIVE, 0, 104
JOB D, ERROR, 0, 201
JOB E, RUNNING, 0, 99
JOB ADAPTER
135
(1) GET JOB STATUS
(4) RETURN JOB STATUS
JOB A, RUNNING, 0, 100
JOB B, FAILED, 0, 351
JOB C, ACTIVE, 0, 104
JOB D, ERROR, 0, 201
JOB E, RUNNING, 0, 99
JOB MANAGER
NORMALIZE
(5)
130
USE
JOB A, SUCCESS, 100
JOB B, FAILED, 351
JOB C, SUCCESS, 104
JOB D, FAILED, 201
JOB E, SUCCESS, 99

FIG. 2C

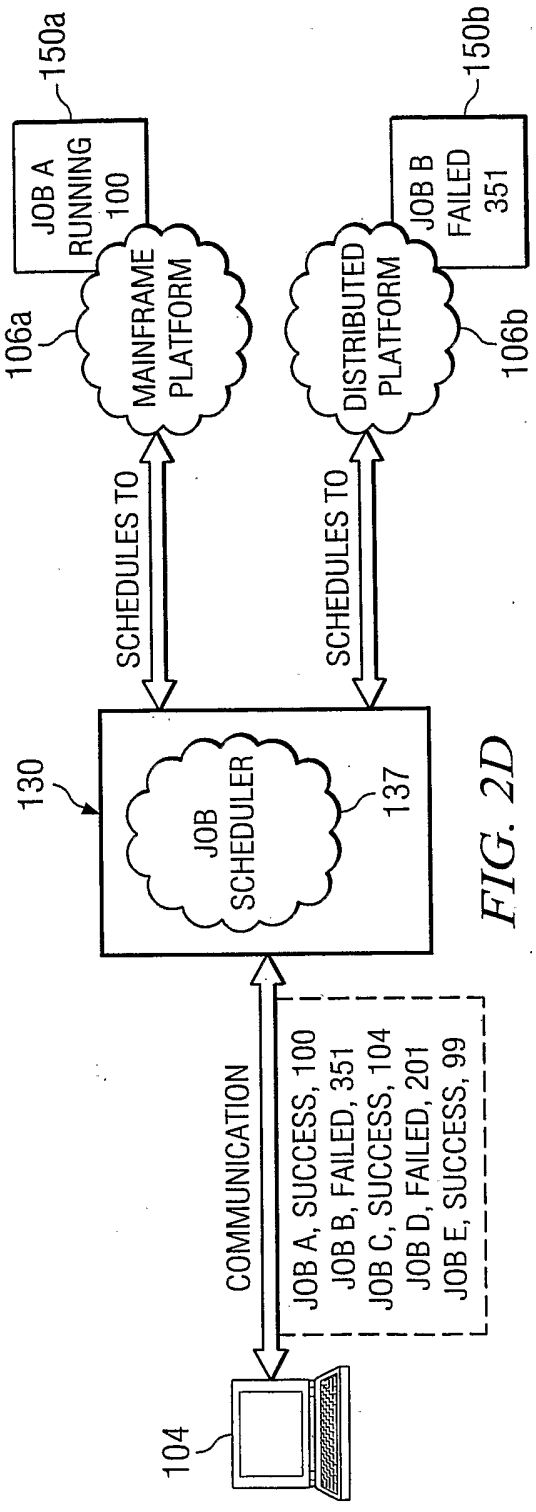

150a
JOB A RUNNING 100
150b
JOB B FAILED 351
106a
MAINFRAME PLATFORM
DISTRIBUTED PLATFORM
106b
SCHEDULES TO
SCHEDULES TO
130
JOB SCHEDULER
137
COMMUNICATION
JOB A, SUCCESS, 100
JOB B, FAILED, 351
JOB C, SUCCESS, 104
JOB D, FAILED, 201
JOB E, SUCCESS, 99
104

FIG. 2D

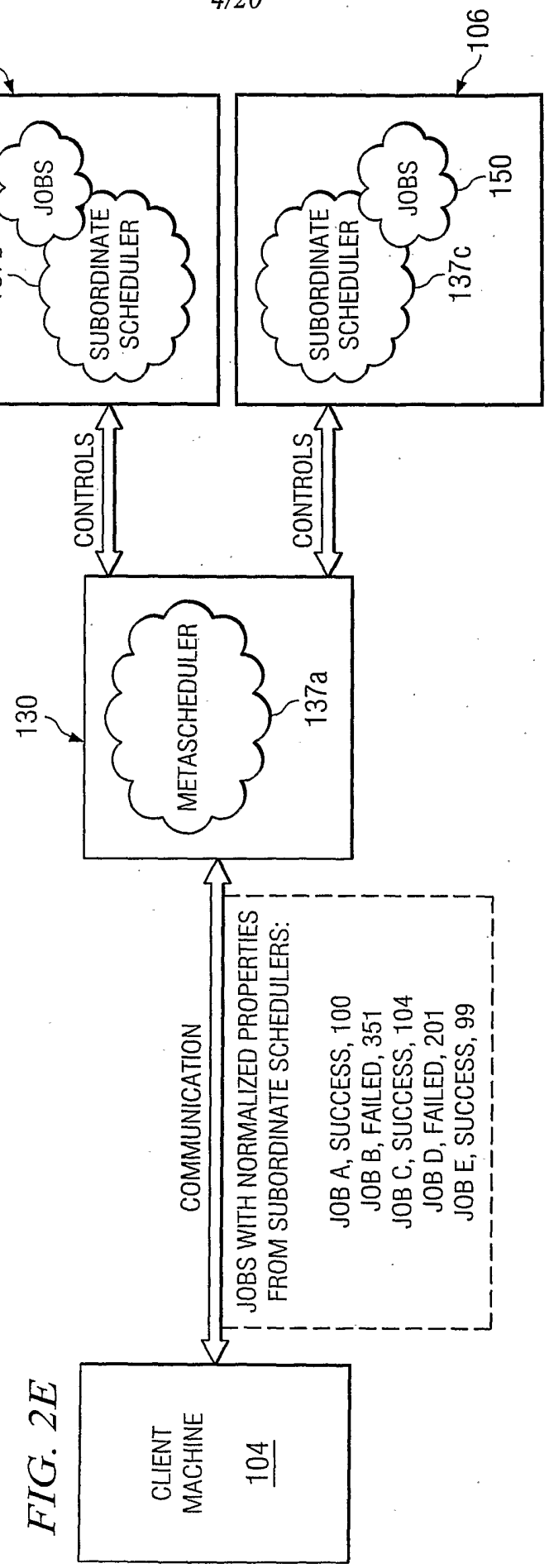

106
106
150
JOBS
137b
SUBORDINATE SCHEDULER
SUBORDINATE SCHEDULER
JOBS
150
137c
CONTROLS
CONTROLS
130
METASCHEDULER
137a
COMMUNICATION
JOBS WITH NORMALIZED PROPERTIES FROM SUBORDINATE SCHEDULERS:
JOB A, SUCCESS, 100
JOB B, FAILED, 351
JOB C, SUCCESS, 104
JOB D, FAILED, 201
JOB E, SUCCESS, 99
CLIENT MACHINE
104

FIG. 2E

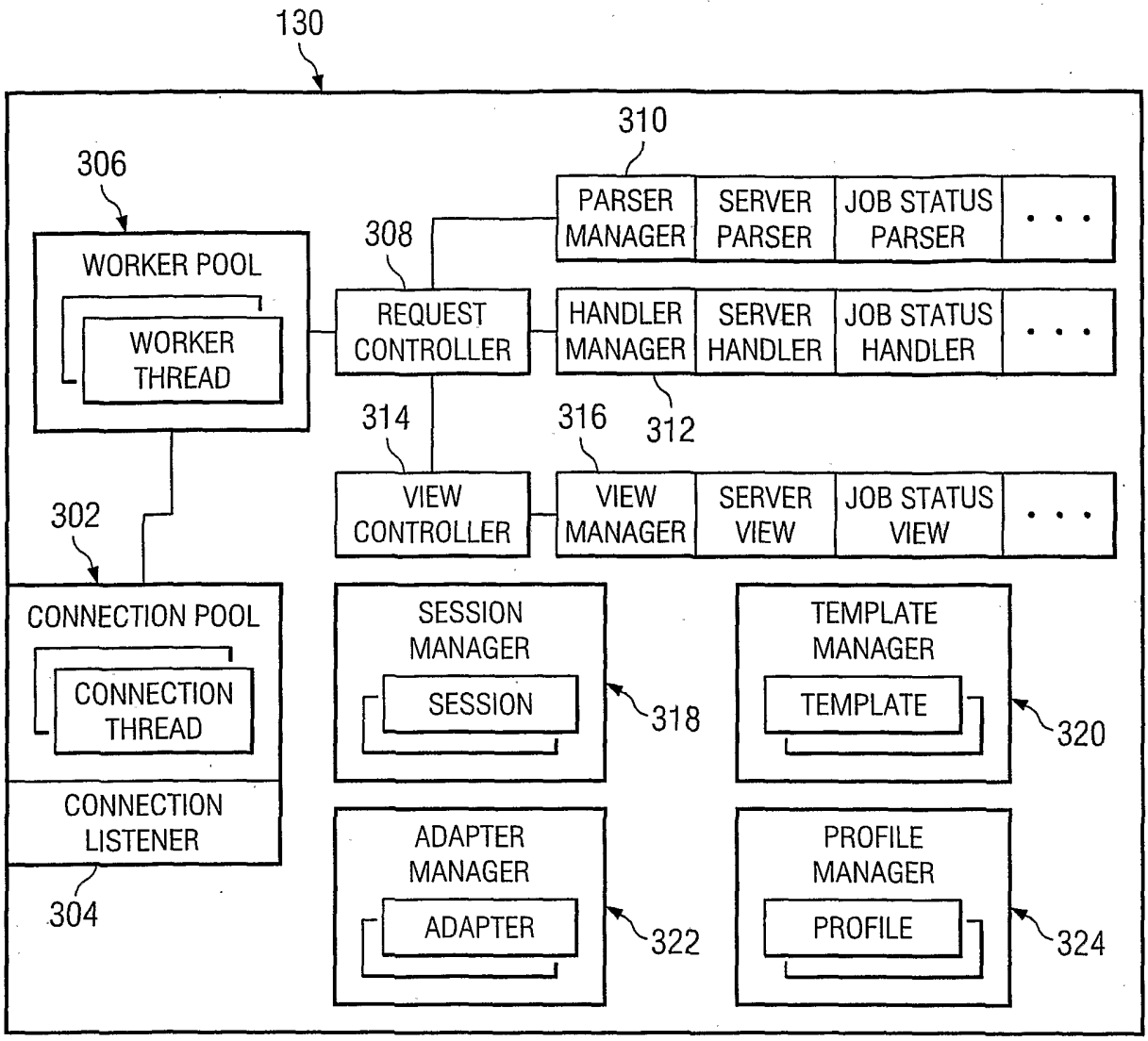

130
310
306
PARSER MANAGER
SERVER PARSER
JOB STATUS PARSER
308
WORKER POOL
WORKER THREAD
REQUEST CONTROLLER
HANDLER MANAGER
SERVER HANDLER
JOB STATUS HANDLER
314
316
312
VIEW CONTROLLER
VIEW MANAGER
SERVER VIEW
JOB STATUS VIEW
302
CONNECTION POOL
CONNECTION THREAD
SESSION MANAGER
SESSION
318
TEMPLATE MANAGER
TEMPLATE
320
CONNECTION LISTENER
304
ADAPTER MANAGER
ADAPTER
322
PROFILE MANAGER
PROFILE
324

FIG. 3

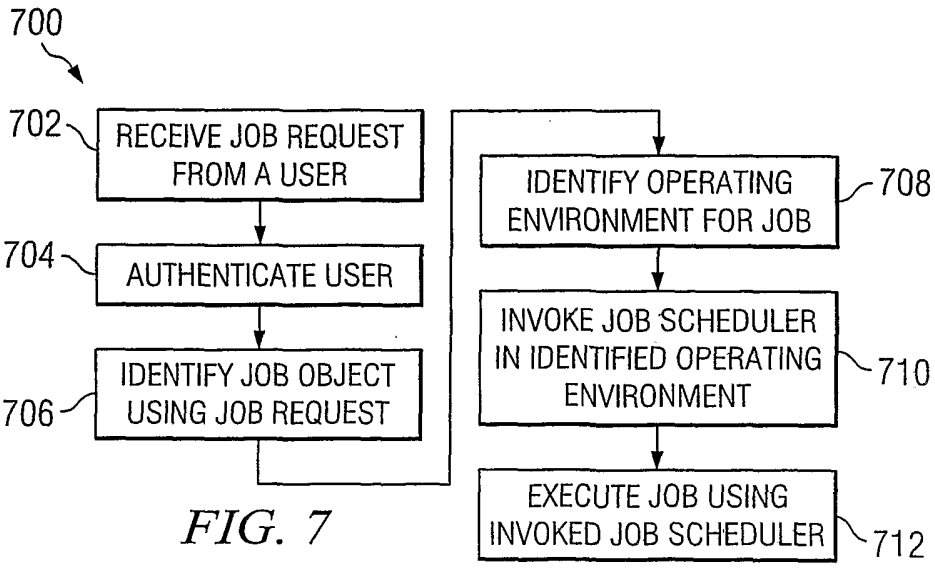

700
702
RECEIVE JOB REQUEST FROM A USER
704
AUTHENTICATE USER
706
IDENTIFY JOB OBJECT USING JOB REQUEST
708
IDENTIFY OPERATING ENVIRONMENT FOR JOB
710
INVOKE JOB SCHEDULER IN IDENTIFIED OPERATING ENVIRONMENT
712
EXECUTE JOB USING INVOKED JOB SCHEDULER

FIG. 7

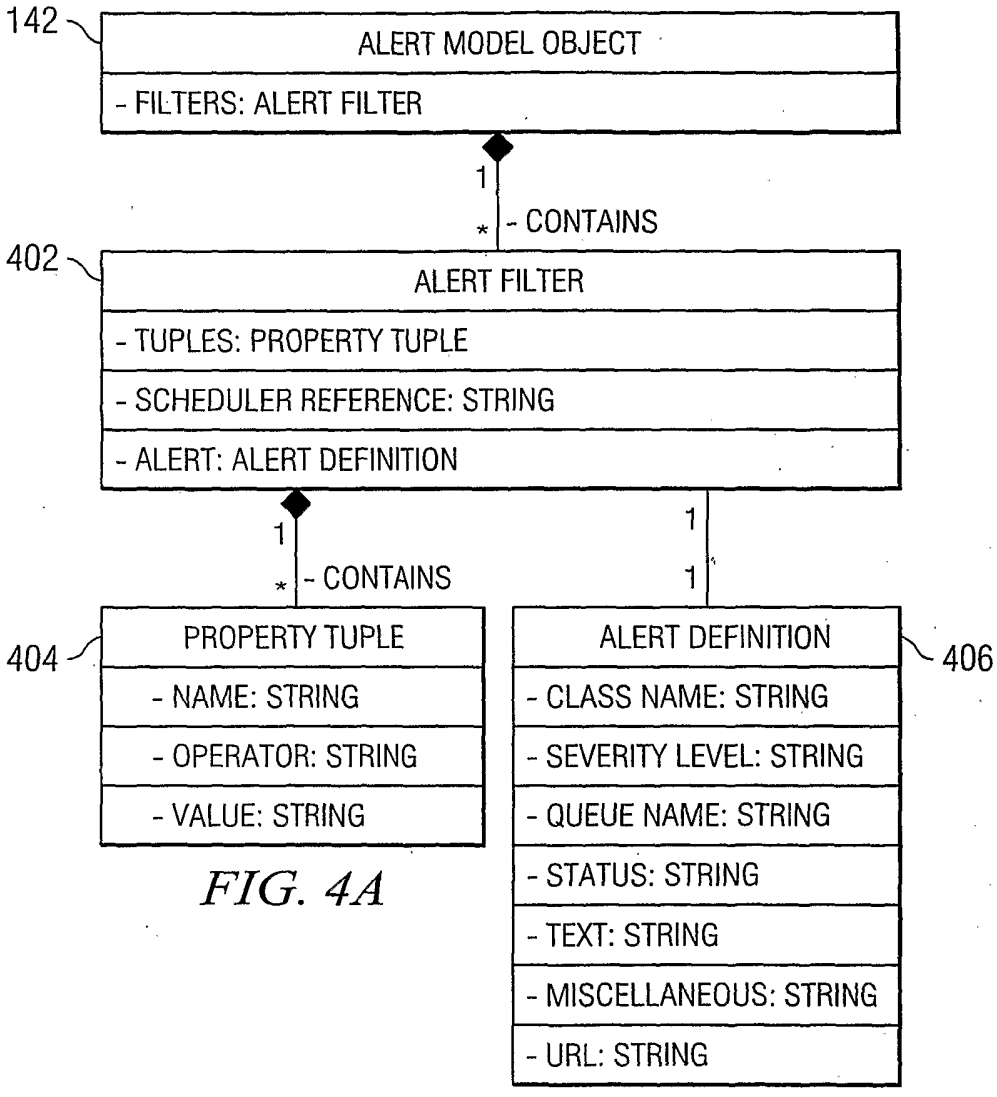

142
ALERT MODEL OBJECT
- FILTERS: ALERT FILTER
1
*
- CONTAINS
402
ALERT FILTER
- TUPLES: PROPERTY TUPLE
- SCHEDULER REFERENCE: STRING
- ALERT: ALERT DEFINITION
1
*
- CONTAINS
1
1
404
PROPERTY TUPLE
- NAME: STRING
- OPERATOR: STRING
- VALUE: STRING
ALERT DEFINITION
406
- CLASS NAME: STRING
- SEVERITY LEVEL: STRING
- QUEUE NAME: STRING
- STATUS: STRING
- TEXT: STRING
- MISCELLANEOUS: STRING
- URL: STRING

FIG. 4A

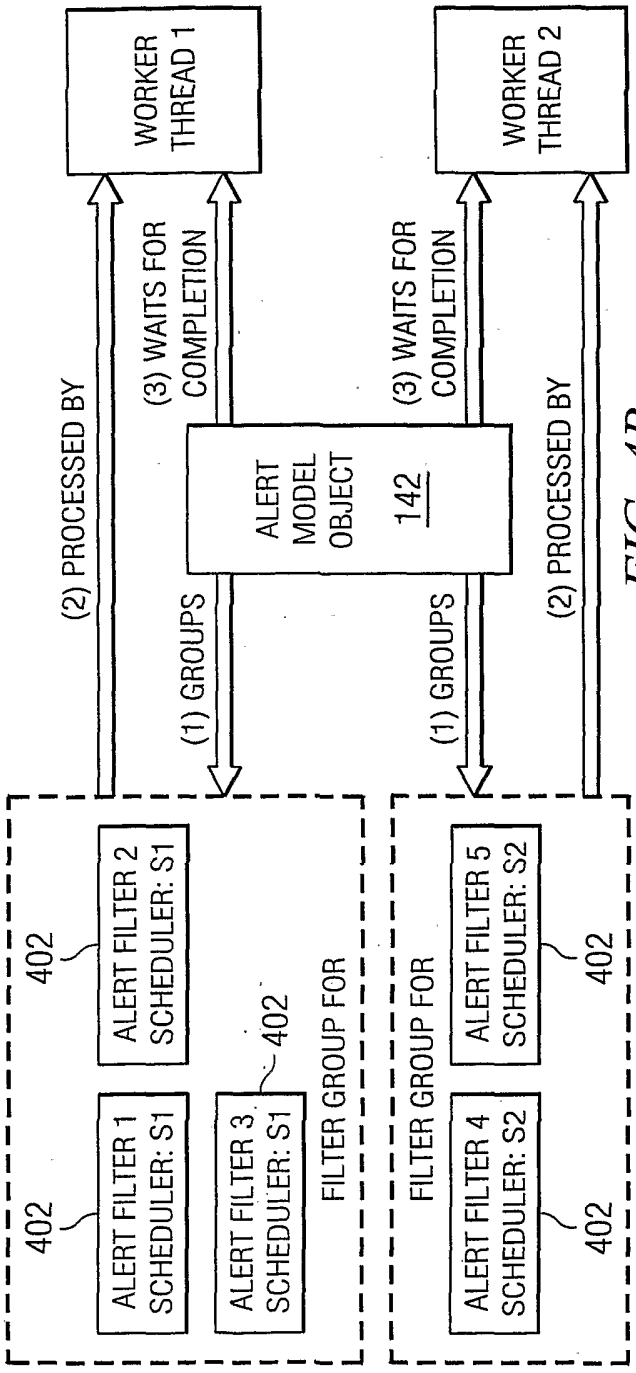

WORKER THREAD 1
WORKER THREAD 2
(2) PROCESSED BY
(3) WAITS FOR COMPLETION
(1) GROUPS
ALERT MODEL OBJECT
142
FILTER GROUP FOR
ALERT FILTER 1 SCHEDULER: S1
ALERT FILTER 2 SCHEDULER: S1
ALERT FILTER 3 SCHEDULER: S1
ALERT FILTER 4 SCHEDULER: S2
ALERT FILTER 5 SCHEDULER: S2
402

FIG. 4B

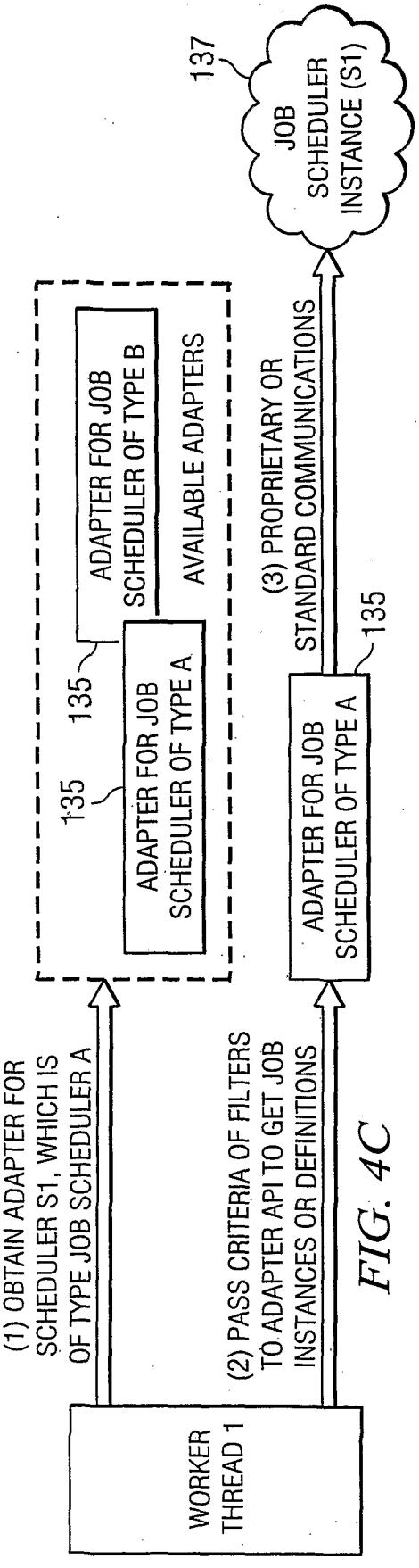

137
JOB SCHEDULER INSTANCE (S1)
(3) PROPRIETARY OR STANDARD COMMUNICATIONS
135
ADAPTER FOR JOB SCHEDULER OF TYPE A
ADAPTER FOR JOB SCHEDULER OF TYPE B
AVAILABLE ADAPTERS
(1) OBTAIN ADAPTER FOR SCHEDULER S1, WHICH IS OF TYPE JOB SCHEDULER A
(2) PASS CRITERIA OF FILTERS TO ADAPTER API TO GET JOB INSTANCES OR DEFINITIONS
WORKER THREAD 1

FIG. 4C

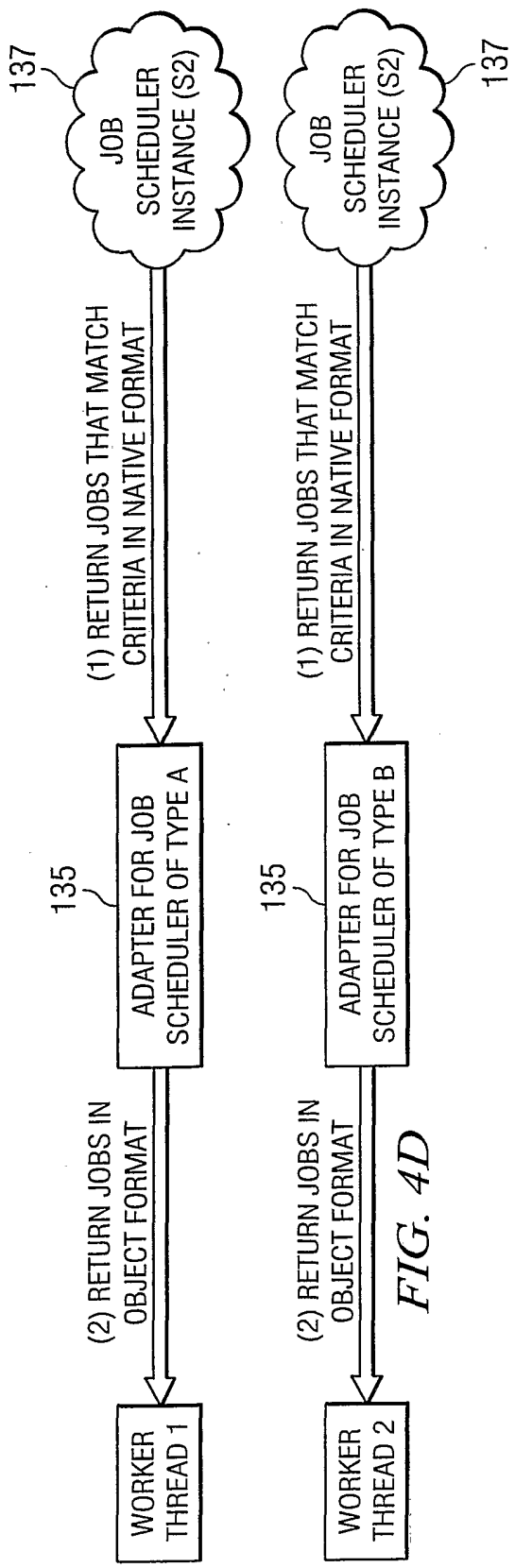


FIG. 4D

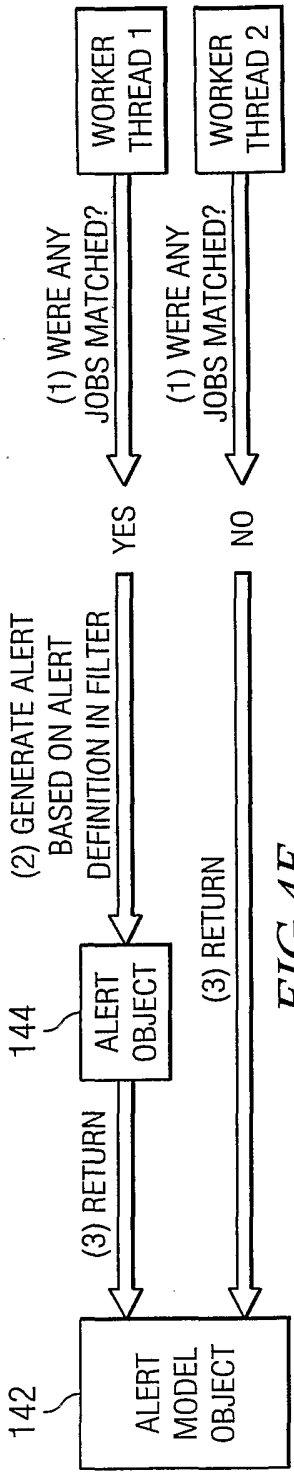


FIG. 4E

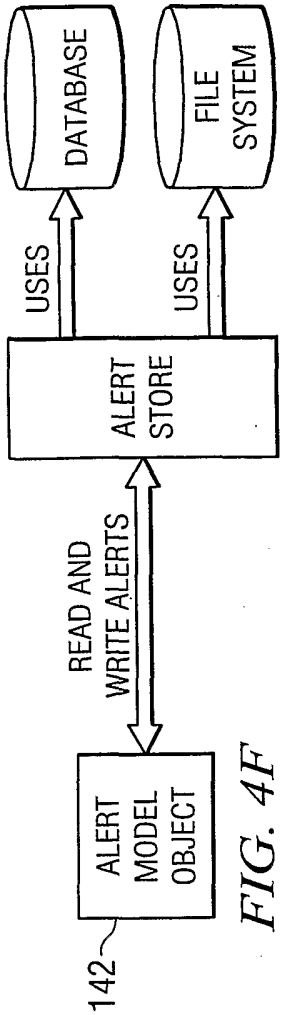


FIG. 4F

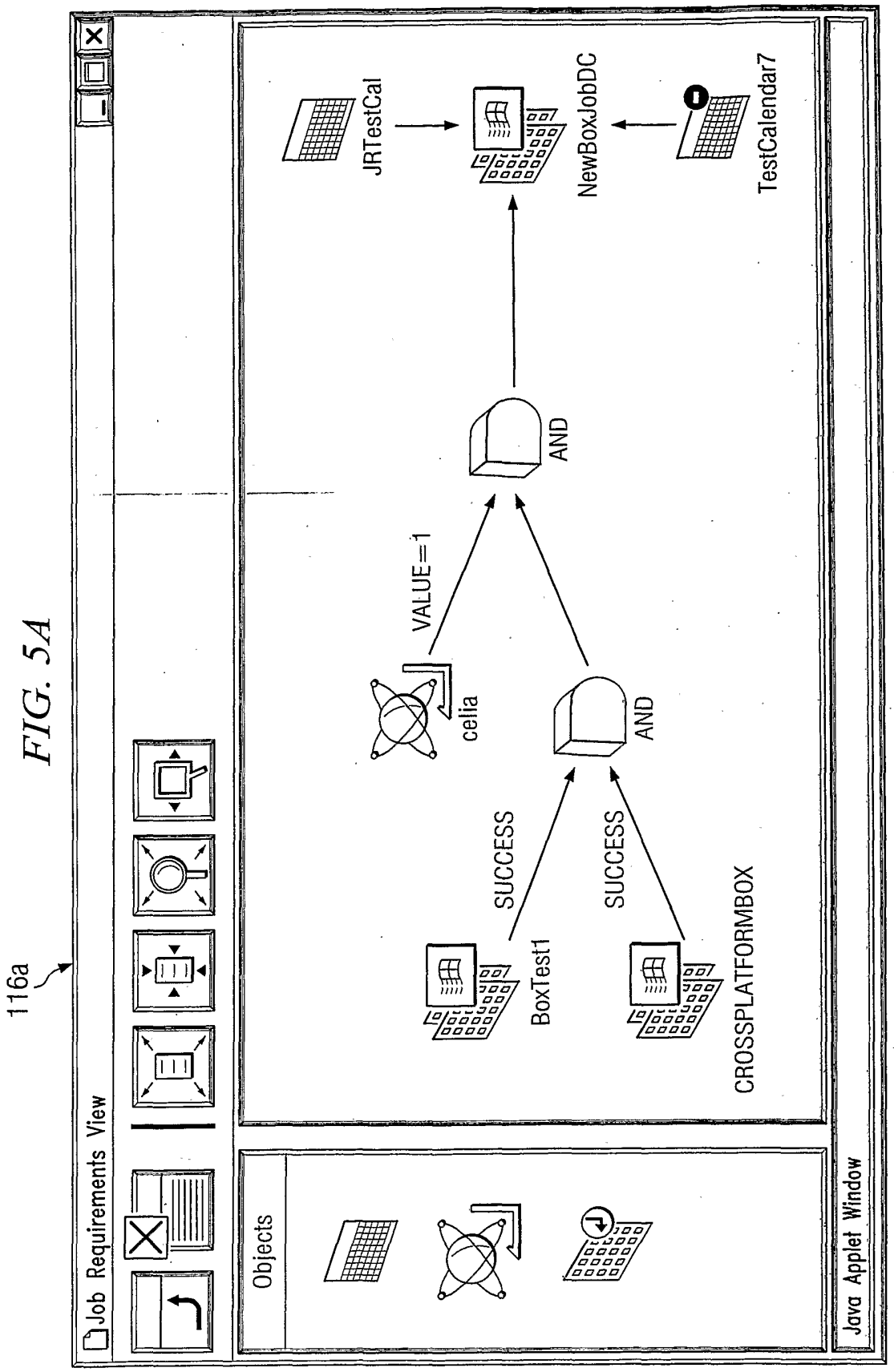
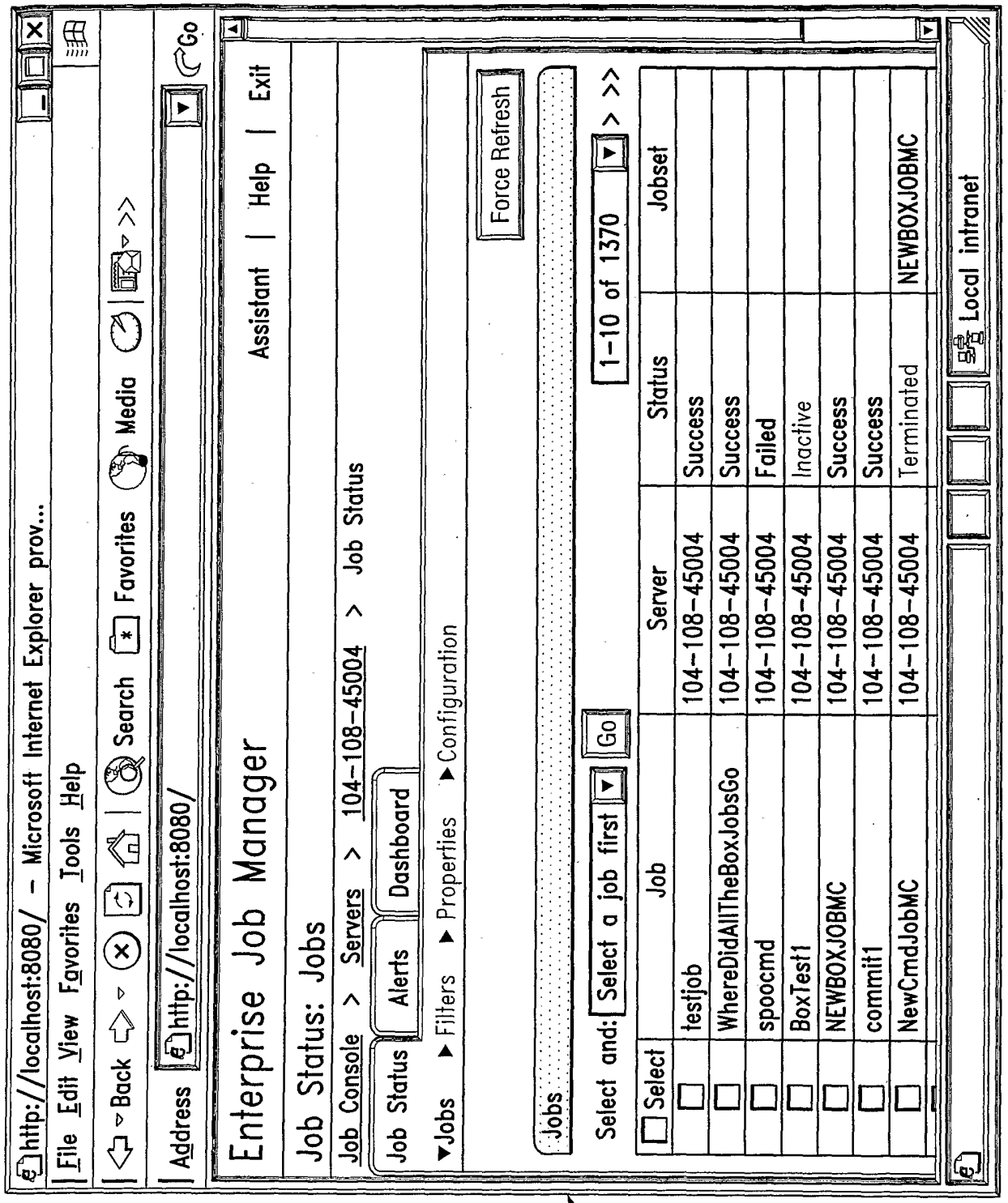

116a
Job Requirements View
Objects
Java Applet Window
JRTestCal
NewBoxJobDC
TestCalendar7
AND
VALUE=1
celia
AND
SUCCESS
SUCCESS
BoxTest1
CROSSPLATFORMBOX

FIG. 5A


http://localhost:8080/ - Microsoft Internet Explorer prov...
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8080/
Go
Enterprise Job Manager
Assistant | Help | Exit
Job Status: Jobs
Job Console > Servers > 104-108-45004 > Job Status
Job Status
Alerts
Dashboard
Jobs ▶ Filters ▶ Properties ▶ Configuration
Force Refresh
Jobs
Select and: Select a job first
Go
1-10 of 1370
> >>
Select
Job
Server
Status
Jobset
testjob 104-108-45004 Success
WhereDidAllTheBoxJobsGo 104-108-45004 Success
spoocmd 104-108-45004 Failed
BoxTest1 104-108-45004 Inactive
NEWBOXJOBMC 104-108-45004 Success
commit1 104-108-45004 Success
NewCmdJobMC 104-108-45004 Terminated NEWBOXJOBMC
Local intranet

116b

FIG. 5B

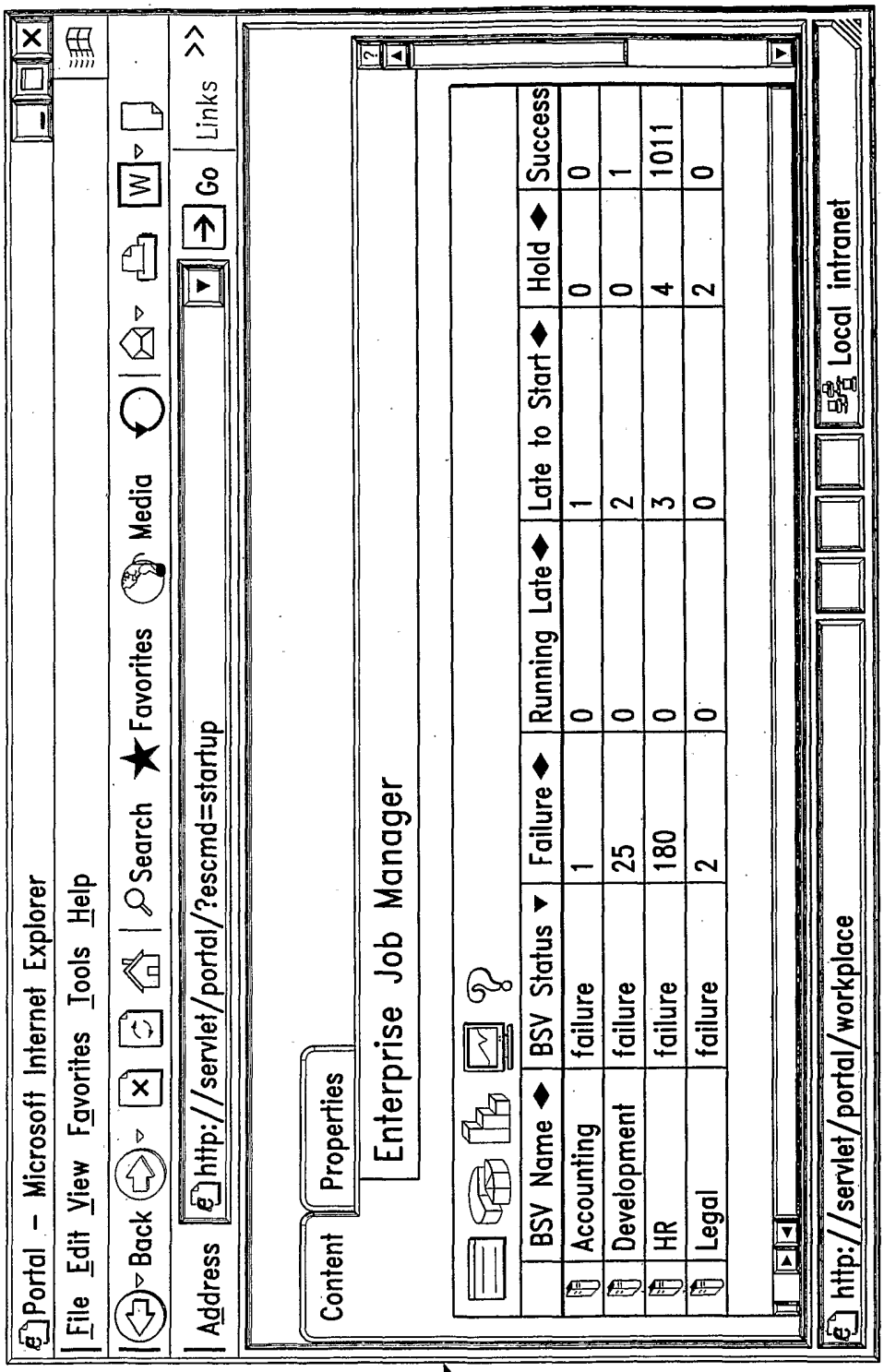

Portal - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://servlet/portal/?escmd=startup
Go Links
Content Properties
Enterprise Job Manager
BSV Name BSV Status Failure Running Late Late to Start Hold Success
Accounting failure 1 0 1 0 0
Development failure 25 0 2 0 1
HR failure 180 0 3 4 1011
Legal failure 2 0 0 2 0
http://servlet/portal/workplace
Local intranet
116c

FIG. 5C

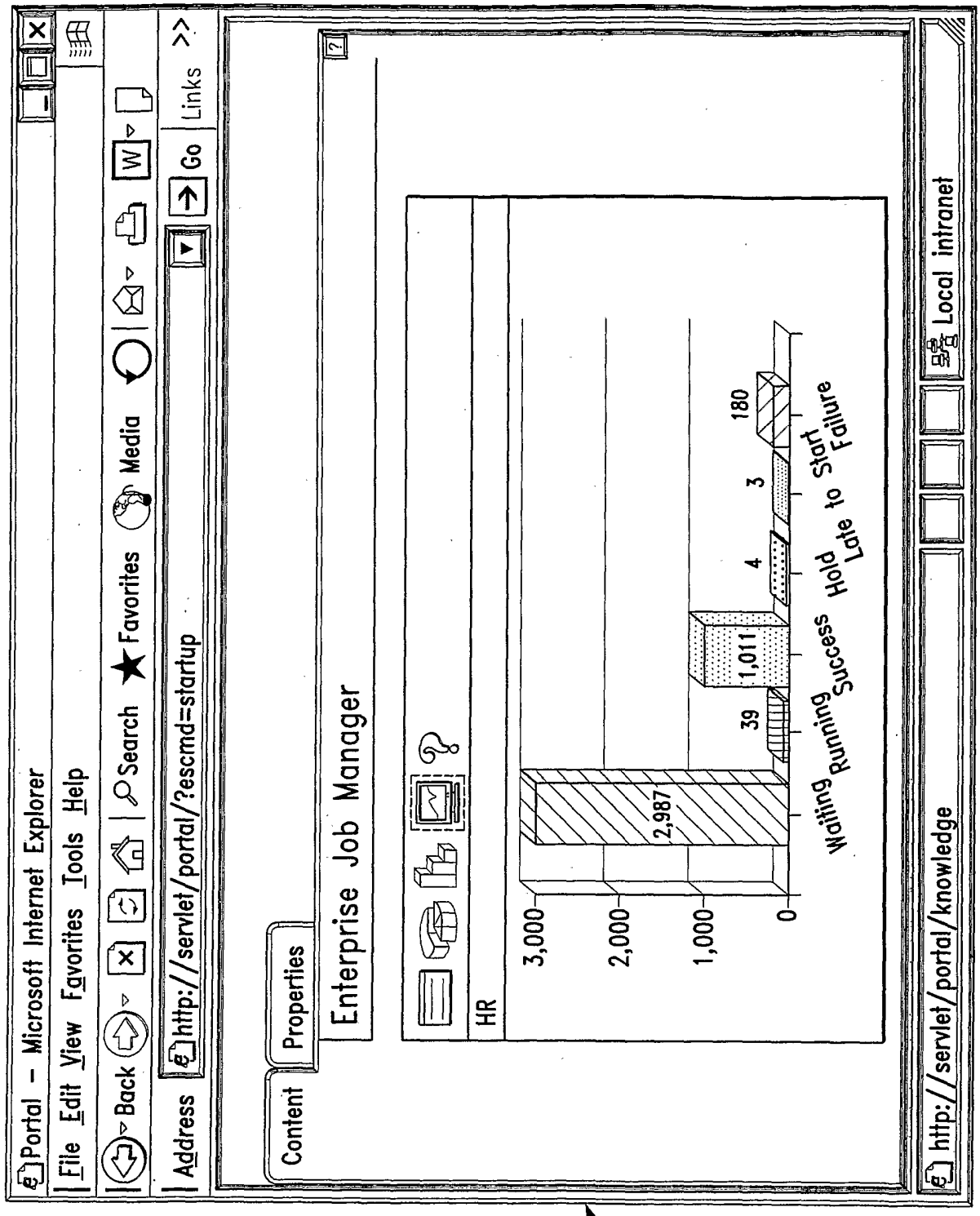

Portal - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://servlet/portal/?escmd=startup
Go Links
Content
Properties
Enterprise Job Manager
HR
3,000
2,000
1,000
0
2,987
39
1,011
4
3
180
Waiting
Running
Success
Hold
Late to Start
Failure
http://servlet/portal/knowledge
Local intranet
116d

FIG. 5D

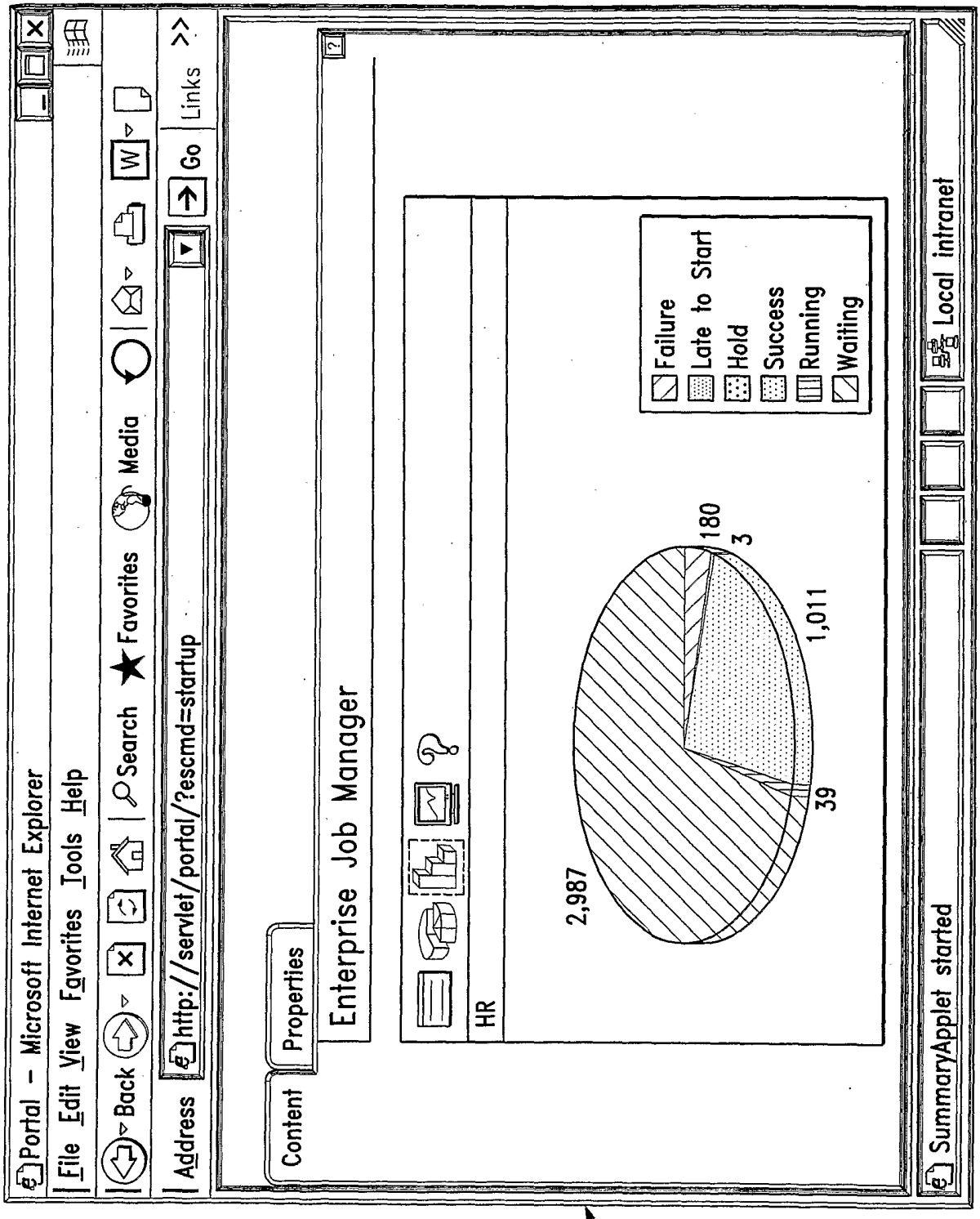

Portal - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Favorites
Media
Address
http://servlet/portal/?escmd=startup
Go
Links
Content
Properties
Enterprise Job Manager
HR
2,987
180
3
1,011
39
Failure
Late to Start
Hold
Success
Running
Waiting
Local intranet
SummaryApplet started
116e

FIG. 5E

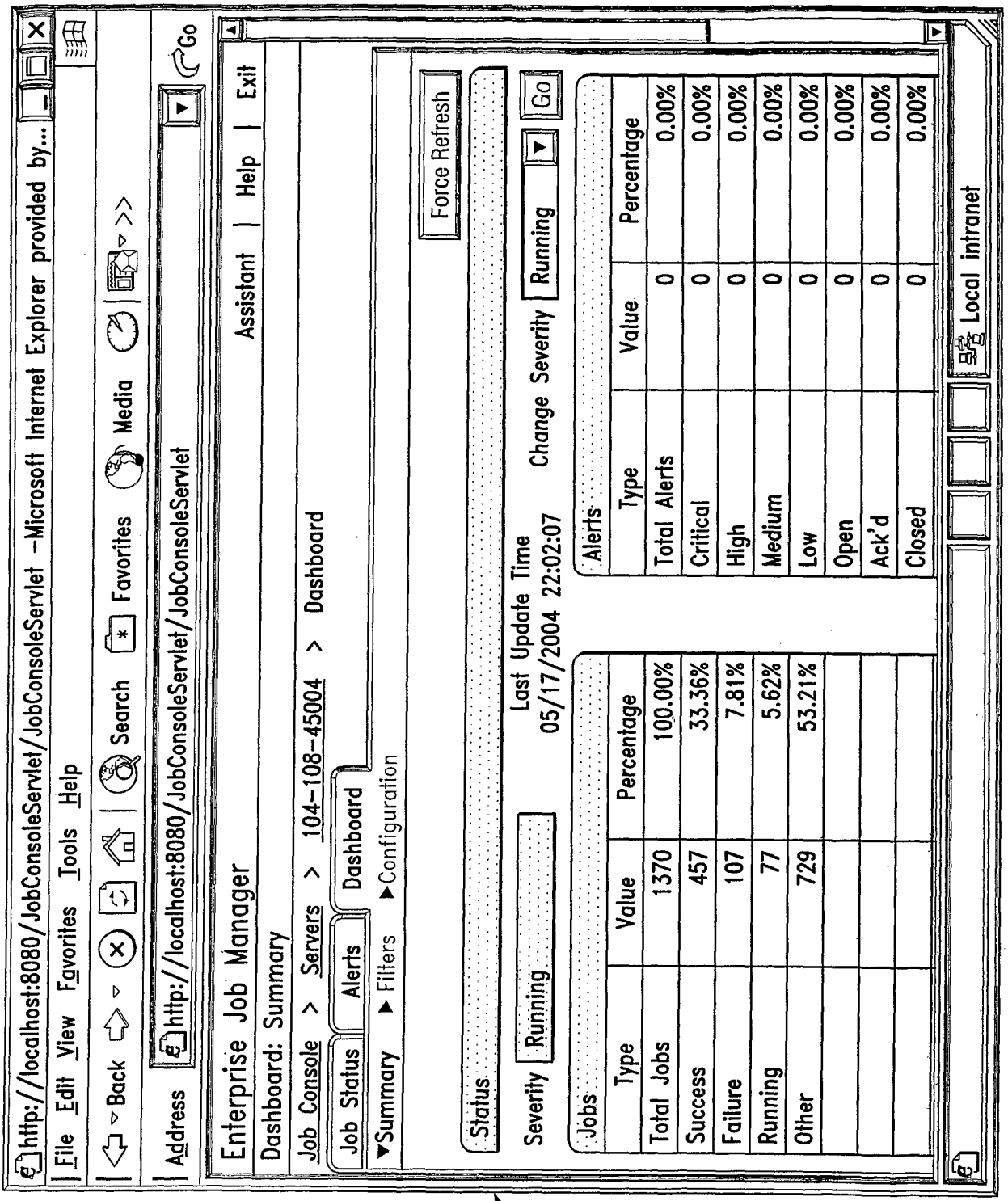

http://localhost:8080/JobConsoleServlet/JobConsoleServlet -Microsoft Internet Explorer provided by...
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8080/JobConsoleServlet/JobConsoleServlet
Go
Enterprise Job Manager
Assistant | Help | Exit
Dashboard: Summary
Job Console > Servers > 104-108-45004 > Dashboard
Job Status Alerts Dashboard
Summary ▶ Filters ▶ Configuration
Force Refresh
Status
Severity Running
Last Update Time 05/17/2004 22:02:07
Change Severity Running Go
Jobs
Type Value Percentage
Total Jobs 1370 100.00%
Success 457 33.36%
Failure 107 7.81%
Running 77 5.62%
Other 729 53.21%
Alerts
Type Value Percentage
Total Alerts 0 0.00%
Critical 0 0.00%
High 0 0.00%
Medium 0 0.00%
Low 0 0.00%
Open 0 0.00%
Ack'd 0 0.00%
Closed 0 0.00%
Local intranet
116f

FIG. 5F

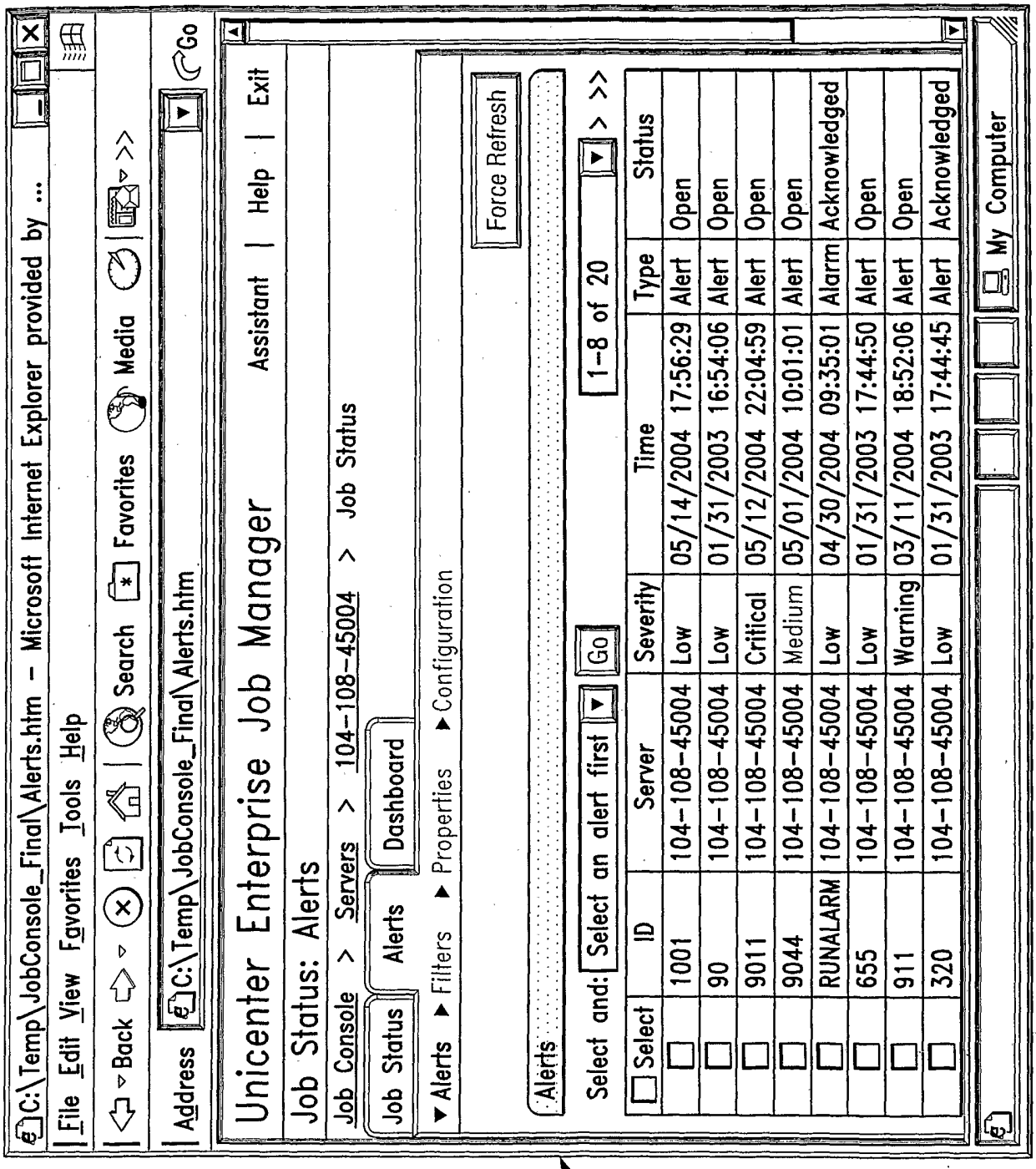

C:\Temp\JobConsole_Final\Alerts.htm - Microsoft Internet Explorer provided by ...
File Edit View Favorites Tools Help
Back Search Favorites Media
Address C:\Temp\JobConsole_Final\Alerts.htm
Go
Unicenter Enterprise Job Manager
Assistant | Help | Exit
Job Status: Alerts
Job Console > Servers > 104-108-45004 > Job Status
Job Status
Alerts
Dashboard
Alerts ▸ Filters ▸ Properties ▸ Configuration
Force Refresh
Alerts
Select and: Select an alert first
Go
1-8 of 20
> >>
Select
ID
Server
Severity
Time
Type
Status
1001 104-108-45004 Low 05/14/2004 17:56:29 Alert Open
90 104-108-45004 Low 01/31/2003 16:54:06 Alert Open
9011 104-108-45004 Critical 05/12/2004 22:04:59 Alert Open
9044 104-108-45004 Medium 05/01/2004 10:01:01 Alert Open
RUNALARM 104-108-45004 Low 04/30/2004 09:35:01 Alarm Acknowledged
655 104-108-45004 Low 01/31/2003 17:44:50 Alert Open
911 104-108-45004 Warning 03/11/2004 18:52:06 Alert Open
320 104-108-45004 Low 01/31/2003 17:44:45 Alert Acknowledged
My Computer
116g

FIG. 6A

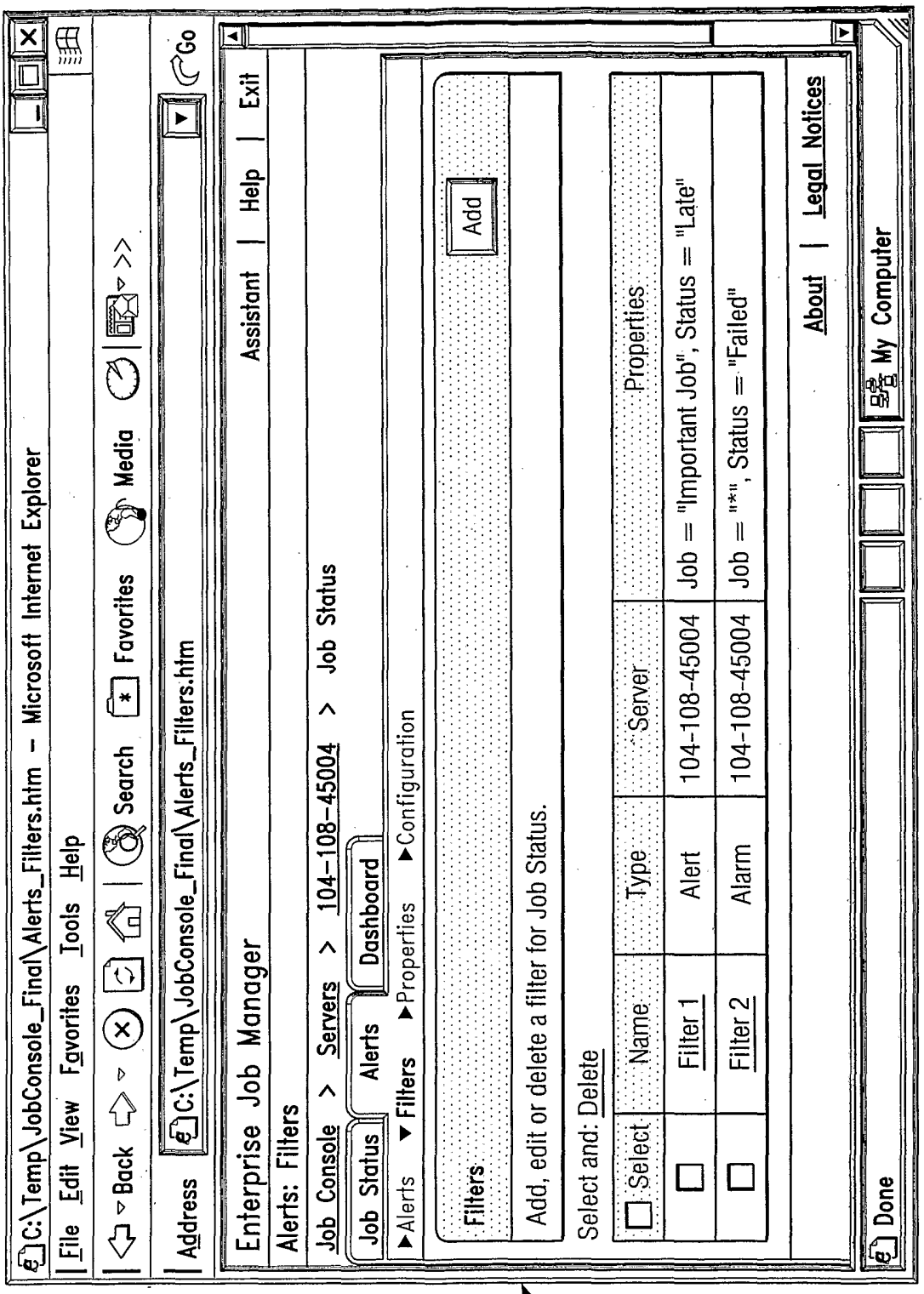
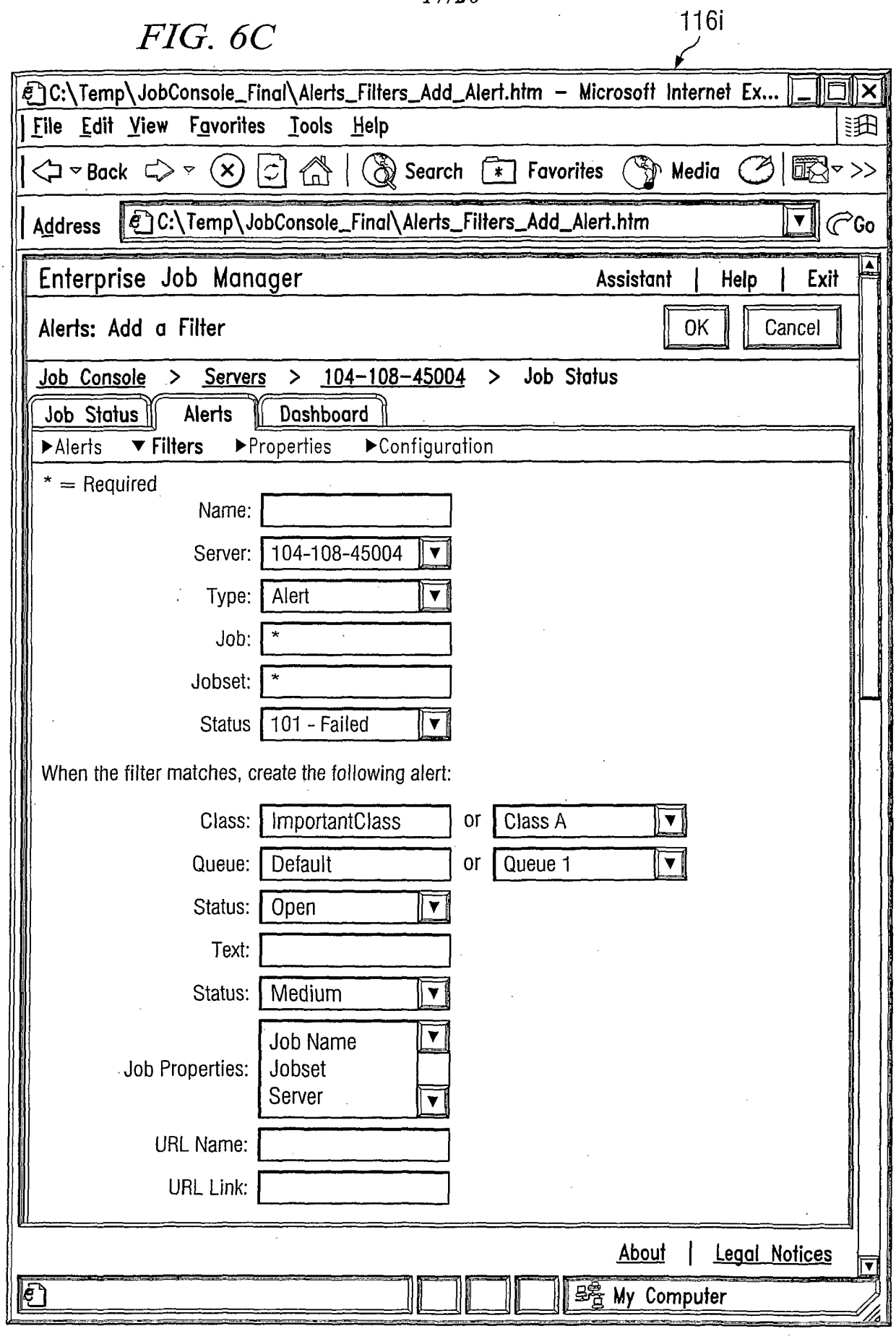

C:\Temp\JobConsole_Final\Alerts_Filters.htm - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address C:\Temp\JobConsole_Final\Alerts_Filters.htm
Go
Enterprise Job Manager
Assistant | Help | Exit
Alerts: Filters
Job Console > Servers > 104-108-45004 > Job Status
Job Status Alerts Dashboard
▶Alerts ▼Filters ▶Properties ▶Configuration
Filters
Add
Add, edit or delete a filter for Job Status.
Select and: Delete
Select Name Type Server Properties
Filter 1 Alert 104-108-45004 Job = "Important Job", Status = "Late"
Filter 2 Alarm 104-108-45004 Job = "*", Status = "Failed"
About | Legal Notices
Done
My Computer
116h

FIG. 6B

FIG. 6C


116i
C:\Temp\JobConsole_Final\Alerts_Filters_Add_Alert.htm - Microsoft Internet Ex...
File Edit View Favorites Tools Help
Back Search Favorites Media
Address C:\Temp\JobConsole_Final\Alerts_Filters_Add_Alert.htm
Go
Enterprise Job Manager
Assistant | Help | Exit
Alerts: Add a Filter
OK
Cancel
Job Console > Servers > 104-108-45004 > Job Status
Job Status
Alerts
Dashboard
▶Alerts ▼Filters ▶Properties ▶Configuration
* = Required
Name:
Server: 104-108-45004
Type: Alert
Job: *
Jobset: *
Status 101 - Failed
When the filter matches, create the following alert:
Class: ImportantClass or Class A
Queue: Default or Queue 1
Status: Open
Text:
Status: Medium
Job Properties: Job Name Jobset Server
URL Name:
URL Link:
About | Legal Notices
My Computer

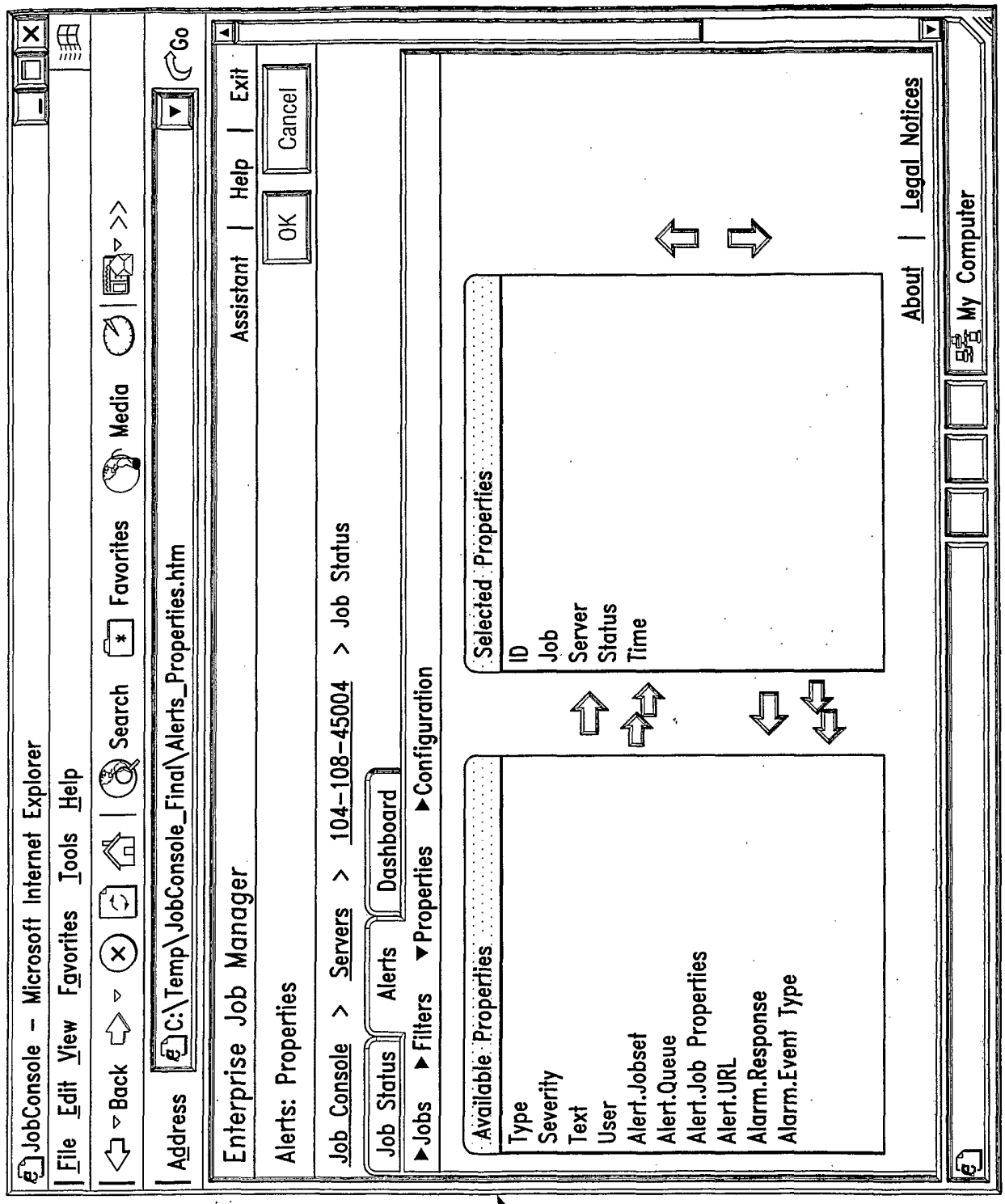

JobConsole - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address C:\Temp\JobConsole_Final\Alerts_Properties.htm
Go
Enterprise Job Manager
Assistant | Help | Exit
Alerts: Properties
OK
Cancel
Job Console > Servers > 104-108-45004 > Job Status
Job Status
Alerts
Dashboard
▶Jobs ▶Filters ▼Properties ▶Configuration
Available Properties
Type
Severity
Text
User
Alert.Jobset
Alert.Queue
Alert.Job Properties
Alert.URL
Alarm.Response
Alarm.Event Type
Selected Properties
ID
Job
Server
Status
Time
About | Legal Notices
My Computer
116j

FIG. 6D

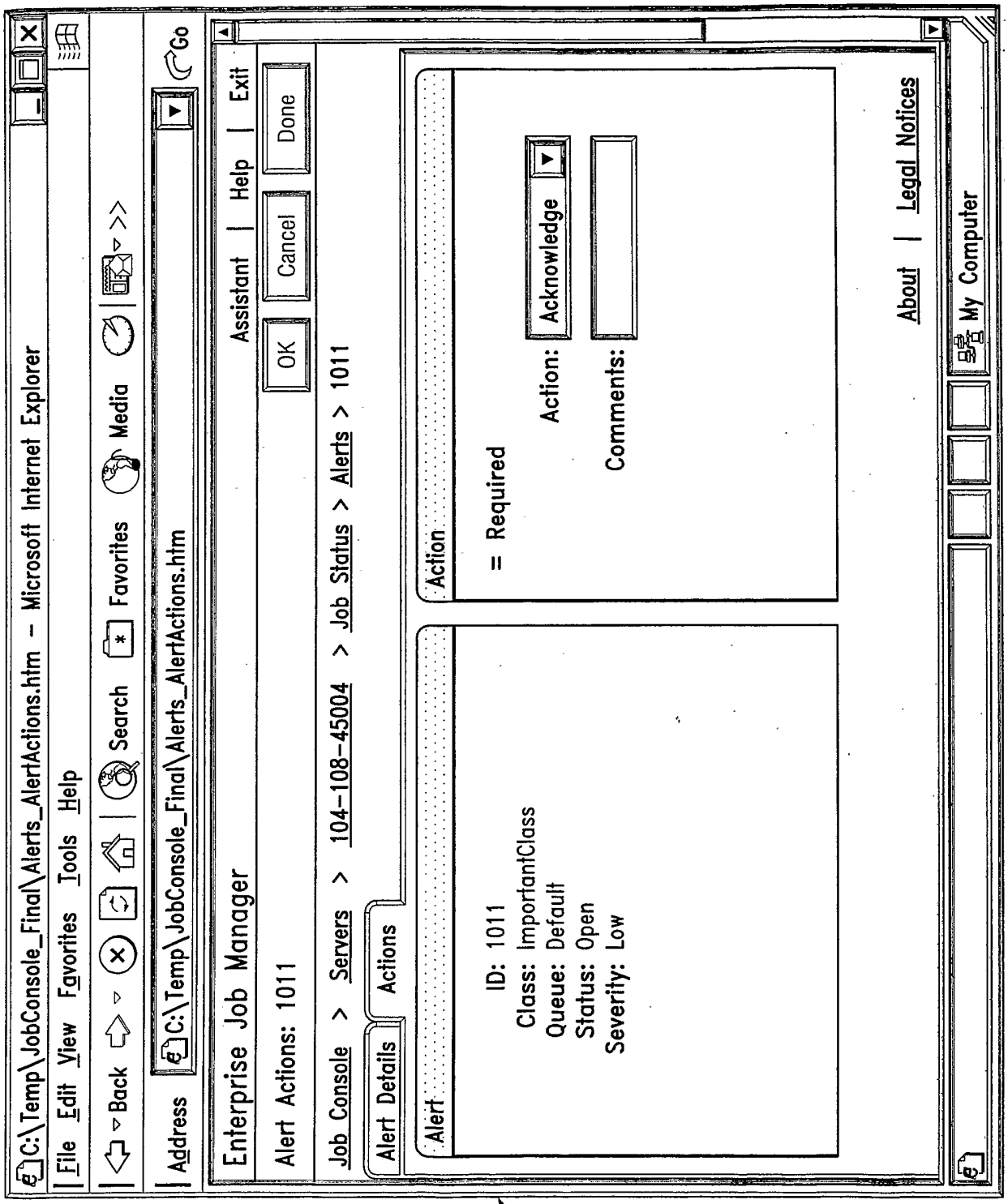
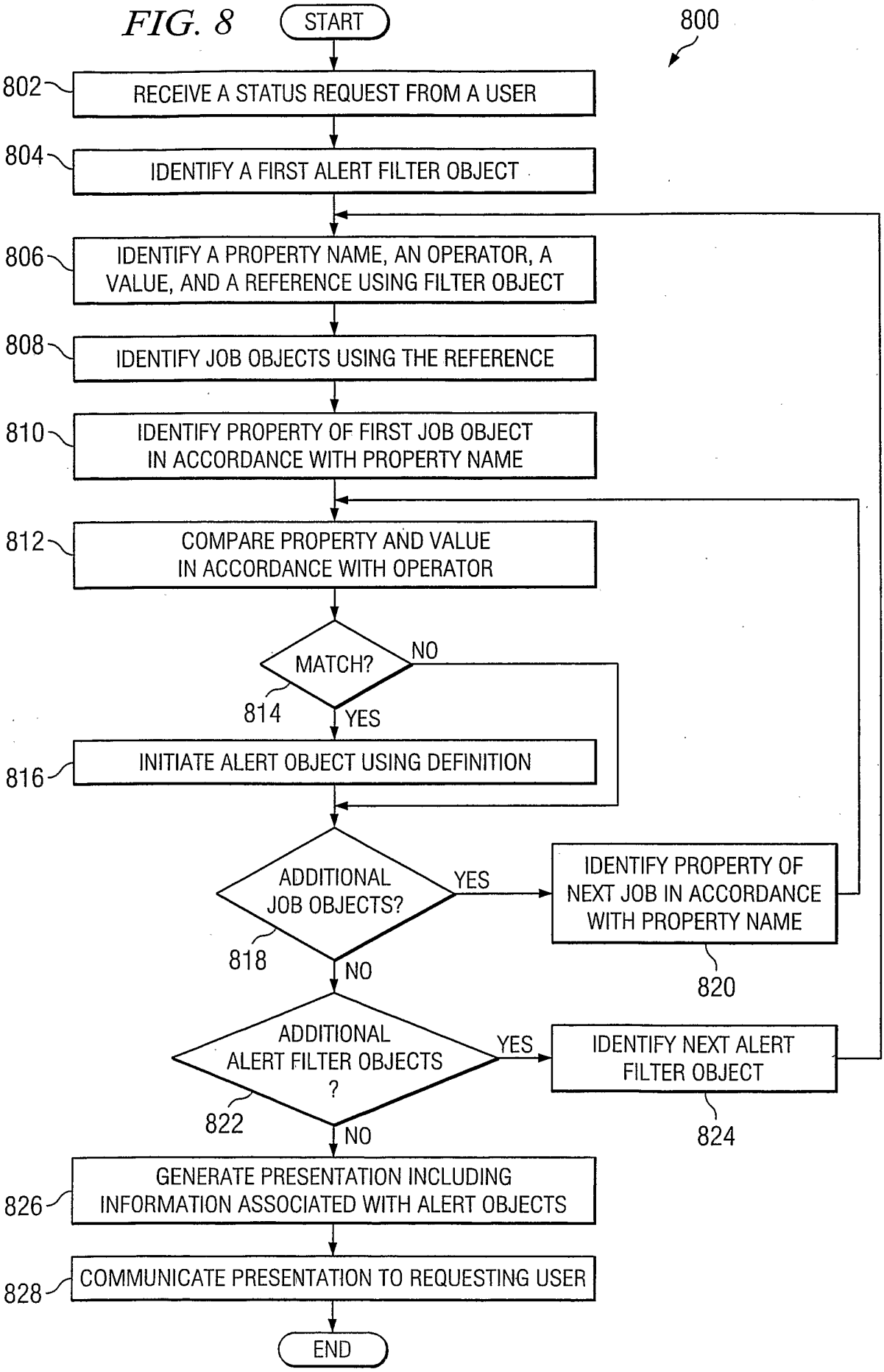

C:\Temp\JobConsole_Final\Alerts_AlertActions.htm - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address C:\Temp\JobConsole_Final\Alerts_AlertActions.htm
Go
Enterprise Job Manager
Assistant | Help | Exit
Alert Actions: 1011
OK Cancel Done
Job Console > Servers > 104-108-45004 > Job Status > Alerts > 1011
Alert Details Actions
Alert
ID: 1011
Class: ImportantClass
Queue: Default
Status: Open
Severity: Low
Action
= Required
Action: Acknowledge
Comments:
About | Legal Notices
My Computer
116k

FIG. 6E

FIG. 8


START
800
802
RECEIVE A STATUS REQUEST FROM A USER
804
IDENTIFY A FIRST ALERT FILTER OBJECT
806
IDENTIFY A PROPERTY NAME, AN OPERATOR, A VALUE, AND A REFERENCE USING FILTER OBJECT
808
IDENTIFY JOB OBJECTS USING THE REFERENCE
810
IDENTIFY PROPERTY OF FIRST JOB OBJECT IN ACCORDANCE WITH PROPERTY NAME
812
COMPARE PROPERTY AND VALUE IN ACCORDANCE WITH OPERATOR
MATCH?
NO
814
YES
816
INITIATE ALERT OBJECT USING DEFINITION
ADDITIONAL JOB OBJECTS?
YES
IDENTIFY PROPERTY OF NEXT JOB IN ACCORDANCE WITH PROPERTY NAME
818
NO
820
ADDITIONAL ALERT FILTER OBJECTS ?
YES
IDENTIFY NEXT ALERT FILTER OBJECT
822
NO
824
826
GENERATE PRESENTATION INCLUDING INFORMATION ASSOCIATED WITH ALERT OBJECTS
828
COMMUNICATE PRESENTATION TO REQUESTING USER
END

INTERNATIONAL SEARCH REPORT

Application No
PCT/US2005/025741

A. CLASSIFICATION OF SUBJECT MATTER
G06F9/46 H04L12/24

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
G06F H04L

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

EPO-Internal, INSPEC, PAJ, COMPENDEX

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	MARTIN D: "Job Scheduling - What's old is new again... sort of" INTERNET ARTICLE, 'Online! 24 February 2003 (2003-02-24), pages 1-2, XP002353991 Retrieved from the Internet: URL:http://www.branhamgroup.com/article_print.php?id=24> 'retrieved on 2005-11-08! the whole document	1-20

-/--

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
15 November 2005	29/11/2005
Name and mailing address of the ISA European Patent Office, P.B. 5818 Patentlaan 2 NL – 2280 HV Rijswijk Tel. (+31–70) 340–2040, Tx. 31 651 epo nl, Fax: (+31–70) 340–3016	Authorized officer de Man, A

Form PCT/ISA/210 (second sheet) (January 2004)

INTERNATIONAL SEARCH REPORT

Application No
PCT/US2005/025741

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	ANONYMOUS: "ISA Server 2004 Monitoring Features" INTERNET ARTICLE, 'Online! 16 June 2004 (2004-06-16), pages 1-10, XP002353992 Retrieved from the Internet: URL:http://www.microsoft.com/technet/prodtechnol/isa/2004/plan/monitoringfeatures.mspx> 'retrieved on 2005-11-14! page 1 page 4	1-20
A	WO 96/31035 A (CABLETRON SYSTEMS, INC) 3 October 1996 (1996-10-03) abstract page 3, lines 20-31 page 5, lines 3-25 page 10, line 13 - page 11, line 13 page 13, lines 27-29	1-20
A	WO 02/086750 A (COMPUTER ASSOCIATES THINK, INC) 31 October 2002 (2002-10-31) abstract page 2, lines 9-30 page 5, line 25 - page 6, line 32	1-20
P,A	ANONYMOUS: "Unicenter Enterprise Job Manager r1 SP3" INTERNET ARTICLE, 'Online! 30 June 2005 (2005-06-30), pages 1-4, XP002353993 Retrieved from the Internet: URL:http://www3.ca.com/Files/DataSheets/uni_ejm_r1_sp3_ds.pdf> 'retrieved on 2005-11-14! the whole document	

INTERNATIONAL SEARCH REPORT

Application No
PCT/US2005/025741

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
WO 9631035 A	03-10-1996	AT 219874 T	15-07-2002
		AU 720061 B2	25-05-2000
		AU 5325896 A	16-10-1996
		DE 69622026 D1	01-08-2002
		DE 69622026 T2	27-02-2003
		EP 0818096 A1	14-01-1998
WO 02086750 A	31-10-2002	CA 2439911 A1	31-10-2002
		EP 1386245 A1	04-02-2004
		JP 2004535624 T	25-11-2004