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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
25 June 2009 (25.06.2009)

PCT

(10) International Publication Number
WO 2009/079036 A1

(51) International Patent Classification:
H04L 12/22 (2006.01) *H04L 12/12* (2006.01)

(21) International Application Number:
PCT/US2008/072727

(22) International Filing Date: 9 August 2008 (09.08.2008)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/955,007 9 August 2007 (09.08.2007) US

(71) Applicant *(for all designated States except US)*: **VIALOGY LLC** [US/US]; 2400 Lincoln Avenue, Altadena, California 91001 (US).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: **GULATI, Sandeep** [US/US]; 5467 La Forest Drive, La Canada, California 91011 (US). **HILL, Robin, Anthony** [GB/US]; 1800 State Street #71, South Pasadena, California 91030 (US). **DAGGUMATI, Vijay** [US/US]; 2400 Lincoln Avenue, Altadena, California 91001 (US). **BREAUX, Jim** [US/US]; 2400 Lincoln Avenue, Altadena, California 91001 (US). **MORRISETT, Alan** [US/US]; 2400 Lincoln Avenue, Altadena, California 91001 (US). **BOYSEN, Cecilie** [US/US]; 2400 Lincoln Avenue, Altadena, California 91001 (US). **HA, Tong** [US/US]; 2400 Lincoln Avenue, Altadena, California 91001 (US). **MANDUTIANU, Dan** [US/US]; 2400 Lincoln Avenue, Altadena, California 91001 (US).

(74) Agent: **LEE, Hwa, C.**; Fish & Richardson P.C., P.O. Box 1022, Minneapolis, Minnesota 55440-1022 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RS, RU, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, 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, HR, HU, IE, IS, IT, LT, LU, LV, MC, MT, NL, NO, 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*

[Continued on next page]

(54) **Title:** NETWORK CENTRIC SENSOR POLICY MANAGER FOR IPV4/IPV6 CAPABLE WIRED AND WIRELESS NETWORKS

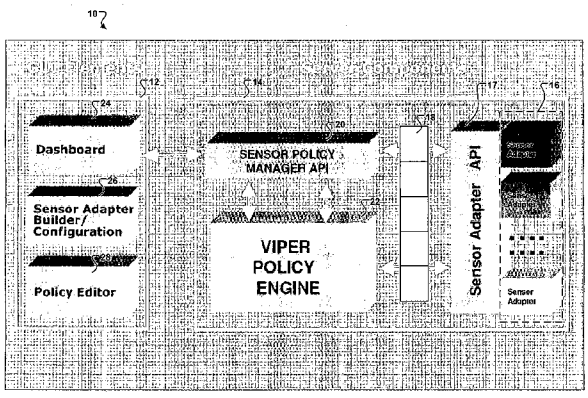


FIG. 1

(57) **Abstract:** Techniques, system and computer program products are disclosed for mapping sensor data form sensors. For example, a computerized system for managing sensor data from sensors include a plurality of sensor adapter modules that receive digital sensor data from sensors and convert received sensor data in a selected sensor data format; a sensor adaptor application programming interface (API) in communication with the sensor adapter modules to transmit the sensor data from the sensor adapter modules; a sensor policy engine comprising sensor management policies and in communication with the sensor adaptor API; a graphic user interface (GUI) to provide user control tools enabling a user to manage and control sensor adapter and sensor management policies in the sensor policy engine; and a sensor policy manager API in communication with the GUI, the sensor adaptor API and the sensor policy engine, the sensor policy manager API operable to communicate instructions and messages between the GUI and the sensor policy engine and the sensor adapter API.

WO 2009/079036 A1

— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments*

NETWORK CENTRIC SENSOR POLICY MANAGER FOR IPV4/IPV6 CAPABLE WIRED AND WIRELESS NETWORKS

CLAIM OF PRIORITY

[0001] This application claims priority under 35 USC §119(e) to U.S. Patent Application Serial No. 60/955,007, filed on August 9, 2007, the entire contents of which are hereby incorporated by reference.

TECHNICAL FIELD

[0002] This application relates to managing sensors.

BACKGROUND

[0003] Various sensors can be used to receive various sensor data. Example sensors include biosensors and radiation sensors.

SUMMARY

[0004] In one aspect, a computerized system for managing sensor data from sensors include a plurality of sensor adapter modules that receive digital sensor data from sensors and convert received sensor data in a selected sensor data format; a sensor adaptor application programming interface (API) in communication with the sensor adapter modules to transmit the sensor data from the sensor adapter modules; a sensor policy engine comprising sensor management policies and in communication with the sensor adaptor API; a graphic user interface (GUI) to provide user control tools enabling a user to manage and control sensor adapter and sensor management policies in the sensor policy engine; and a sensor policy manager API in communication with the GUI, the sensor adaptor API and the sensor policy engine, the sensor policy manager API operable to communicate instructions and messages between the GUI and the sensor policy engine and the sensor adapter API.

[0005] Implementations can optionally include one or more of the following features. A sensor database can be implemented to receive and store sensor data from the sensors.

[0006] The subject matter described in this specification potentially can provide one or more of the following advantages. The Sensor Policy Manager (SPM) solution can provide immediate integration and data commonality. SPM is a standards based product that can access and operate on all Chemical, Biological, Radiological, Nuclear and High-Yield Explosives (CBRNE) and physical security sensors that are part of the Institute for Defence and Strategic

Studies (IDSS) solution. The Commonality of the data transfer protocols embedded/enabled by SPM allows better integration of response involving local, state and federal agencies in major CBRN terrorist attack or incident, for example. Furthermore, this commonality can enhance integration of assets, response, resupply, and reconstitution.

[0007] Also, SPM can provide tested solution. The SPM solution has been tested through several Department of Homeland Security (DHS) deployments as well as in four US Department of Defense (DoD) JFPASS exercises. SPM also has certification from the U.S. National Incident Management System (NIMS). SPM can provide for future upgrade path. SPM can enable future sensor interoperability upgrades because (i) SPM includes an existing library of over 150 sensor adaptors, including many common and unique CBRN sensors, and (ii) a capability to timely create new sensor adaptors (e.g., 72-hours). The overall advantage is a reduction in life cycle management costs because the simplified information management structure provided by the IDSS creates a single integrated decision system to support Tier 1 and Tier 2 Installation Protection Program (IPP) requirements.

[0008] The subject matter described in this specification can be implemented as, etc.

BRIEF DESCRIPTION OF THE DRAWINGS

[0009] FIG. 1 illustrates and example SPM system.

[0010] FIG. 2 shows an example IDSS Enhancement to an IPP Mission.

[0011] FIG. 3 shows an example graphical user interface (GUI) for a SPM console.

[0012] FIG. 4 shows another example GUI for a SPM console.

[0013] FIG. 5 shows another example GUI for a SPM console.

[0014] FIG. 6 shows an example channel class model.

[0015] FIG. 7 shows an example mapper statement in SPM.

[0016] FIG. 8 shows an example interaction diagram for publishing a message.

[0017] FIG. 9 shows an example configuration.

[0018] FIG. 10 shows an example configuration entry.

[0019] FIG. 11 shows a table 600 that explains the configuration properties

[0020] FIG. 12 shows an example message type.

[0021] FIG. 13 shows an example mapping entry from a configuration file.

[0022] FIG. 14 shows an example declaration of the channel which uses this mapping entry.

[0023] FIG. 15 shows an example declaration for the SMCData message type.

[0024] FIG. 16 shows an example published data generated from this configuration using the FieldServer / SMC sensor at the bottom end.

[0025] FIG. 17 shows an example SPM/SPM Proxy.

[0026] FIG. 18 shows an example design of the SPM Proxy to SPM interface with reference to the issues and proposed solution.

[0027] FIG. 19 shows an example of publishing data from SPM Proxy to SPM.

[0028] FIG. 20 is a diagram showing an example class model.

[0029] FIG. 21 shows an example Viper sever.

[0030] FIG. 22 shows an example diagram for generating a new policy template from condition and action templates.

[0031] FIG. 23 shows an example use case where access is granted.

[0032] FIG. 24 shows an example use case where a broken glass alarm is triggered.

[0033] FIG. 25 shows an example use case of stolen equipment.

[0034] FIG. 26 shows an example use case of seismic event detection.

[0035] FIG. 27 shows example class diagrams

[0036] FIG. 28 shows an example data and control flow.

[0037] FIG. 29 shows an example sequence diagram.

[0038] FIG. 30 shows an example sensor proxy framework.

[0039] FIG. 31 shows an example sensor proxy framework for protocol support.

[0040] FIG. 32 is a diagram showing a simplified example representation of a SPM communications infrastructure.

[0041] FIG. 33 shows example component dependencies.

[0042] FIG. 34 is a table showing different layers of an example TCP/IP Stack.

[0043] FIG. 35 shows an example protocol class model.

[0044] FIG. 36 shows example underlying transport protocol for determining inheritance.

[0045] FIG. 37 shows an example ViaSensorProxyProtocol base class.

[0046] FIG. 38 is a table showing example ViaAceSocketThread methods.

[0047] FIG. 39 s a table showing example ViaSensorProxyProtocol methods.

[0048] FIG. 40 shows an example message type section in a configuration file.

[0049] FIG. 41 shows an example sensor proxy entry.

[0050] FIG. 42 shows example entries for the two single-head sensor proxies.

[0051] FIG. 43 shows example composite sensor proxies.

[0052] FIG. 44 shows an example configuration section in XML.

[0053] FIG. 45 shows an example construction for a sensor proxy.

[0054] FIG. 46 showsn an example packet assembler.

[0055] FIG. 47 an example config section

[0056] FIG. 48 shows a sample XML.

[0057] FIG. 49 shows an example XML configuration file section.

[0058] FIG. 50 shows an example code fragment.

[0059] FIG. 51 shows an example SPM / SPM Edge Configuration.

[0060] FIG. 52 shows an example message.

[0061] FIG. 53 shows a sample XML.

[0062] FIG. 54 shows a sample XML.

[0063] FIG. 55 an example code fragment

[0064] FIG. 56 shows an example SPM / SPM Edge Configuration.

[0065] FIG. 57 shows an example SPM architechture.

[0066] FIG. 58 shows an example main panel.

[0067] FIG. 59 shows another example main panel.

[0068] FIG. 60 shows an example user interface panel for using a WinSCP Client.

[0069] FIG. 61 shows an example PuTTY SSH Client Login interface.

[0070] FIG. 62 shows an example browser.

[0071] FIG. 63 shows an example browser.

[0072] FIG. 64 shows an example Control Center Explorer with New Database Folder.

[0073] FIG. 65 65 shows an example Newly Added SPM Database.

[0074] FIG. 66 shows an example operation panel 6100 for Managing Sensor Instances Operations.

[0075] FIG. 67 shows highlighting an example record.

[0076] FIG. 68 shows an example panel.

[0077] FIG. 69 shows an example panel.

[0078] FIG. 70 is a panel showing example templates

[0079] FIG. 71 shows an example Sensor Adapter Instance Channel Listing.

[0080] FIG. 72 shows example Channel Property Definitions.

[0081] FIG. 73 shows an example list of message types.

[0082] FIG. 74 shows example instance message properties.

[0083] FIG. 75 is a diagram showing an example of communicating from a

remote SPM server.

[0084] FIG. 76 is a diagram showing an example of Sending Controls and Commands from a Central SPM.

[0085] FIG. 77 shows an example Sensor Emulation.

[0086] FIG. 78 shows an example of sensor emulation alert.

[0087] FIG. 79 is a panel showing an example alert messages.

[0088] FIG. 80 shows an example operation panel.

[0089] FIG. 81 shows an example Manage SPM Configuration form.

[0090] FIG. 82 shows an example Database Manager.

[0091] FIG. 83 shows example Database Backup Functions.

[0092] FIG. 84 shows example Database Restore Functions.

[0093] FIG. 85 shows example database information.

[0094] FIG. 86 shows an example panel for Managing a Configuration.

[0095] FIG. 87 shows an example panel for assembling a viper server.

[0096] FIG. 88 shows an example Policy Wizard that provides step-by-step guidance in creating a new SPM policy or modifying a copy of an existing one.

[0097] FIG. 89 shows example SPM Policy Wizard Triggers.

[0098] FIG. 90 shows an example SPM Policy Wizard Alerts.

[0099] FIG. 91 shows an example Publish form that contains a number of alternatives for publishing or displaying alerts.

[00100] FIG. 92 shows an example SPM policy wizard showing an action tab.

[00101] FIG. 93 shows an example SPM policy wizzar with a final tab.

[00102] FIG. 94 shows an example Viper Configuration File Editor.

[00103] FIG. 95 shows a list of SPM operators.

[00104] FIG. 96 shows a list of SPM operators.

[00105] FIG. 97 shows a list of reserved SPML keywords.

[00106] FIG. 98 shows example SPML functions.

[00107] FIG. 99 shows an example File Editor Tool.

[00108] FIG. 100 shows an example panel for naming a new viper configuration.

[00109] FIG. 101 shows example tools for a main menu reporting panel.

[00110] FIG. 102 shows an example reporting panel for viewing logs.

[00111] FIG. 103 shows example color codes for network nodes.

[00112] FIG. 104 shows a graphical representation of the existing SPM network.

[00113] FIG. 105 shows an example of event messages generated by an SPM policy as it evaluates data coming from a sensor via a sensor adapter.

[00114] FIG. 106 shows example Events Message Retrieval and Event Cleanup panels.

[00115] FIG. 107 shows example communications alarm event messages.

[00116] FIG. 108 shows an example SOAP message alert.

[00117] FIG. 109 shows a channel list status panel.

[00118] FIG. 110 shows example advanced search filters and functions.

[00119] FIG. 111 shows an example real time dashboard.

[00120] FIG. 112 shows a typical examples of Monitor displays are the BMS video camera.

[00121] FIG. 113 shows a monitor.

[00122] FIG. 114 shows a monitor display for gas detector.

[00123] FIG. 115 shows an example monitor.

[00124] FIG. 116 shows an example JMS Log Viewer that displays the status of the traffic through the JMS Messaging Server.

[00125] FIG. 117 shows an example monitoring.

[00126] FIG. 118 shows an example monitoring.

[00127] FIG. 119 shows an example Cepheid SmartCycler Monitor with Data Table.

[00128] FIG. 120 shows example network options for network video.

[00129] FIG. 121 shows example operations for account management.

[00130] FIG. 122 shows an alert dialog.

[00131] FIG. 123 shows example user roles and permissions.

[00132] FIG. 124 shows an example listing of SPM roles.

[00133] FIGs. 125 and 126 are tables showing example Permission Rankings for Config Objects and Predefined User Roles.

[00134] FIG. 127 shows an attempt to access an unallowed config object.

[00135] FIGS. 128 and 129 show panels for selecting permission to add to a user role.

[00136] FIG. 130 shows a listing of current SPM users and their roles.

[00137] FIG. 131 shows example fields.

[00138] FIG. 132 shows all roles tab.

[00139] FIG. 133 a role added to a user profile.

[00140] FIG. 134 shows a table of SPM users.

[00141] FIG. 135 shows a panel for SPM Roles and Permissions.

[00142] FIG. 136 shows Role/Permissions for this User.

[00143] FIG. 137 is a table showing descriptions of message data structure properties.

[00144] FIG. 138 shows an example data structure for AlienRFID_OwnedTagGoneMessage.

[00145] FIG. 139 shows example Alien Sensor Adapter (RFID) properties as implemented in the sensor adapter template.

[00146] FIG. 140 shows example specified Barix Barionet Sensor Adapter properties.

[00147] FIG. 141 shows example properties for the Canberra (Nuclear) Sensor Adapter.

[00148] FIG. 142 shows properties specified for Dragon Force Sensor Adapter.

[00149] FIG. 143 shows example message types for GeneXpertSampleInfo.

[00150] FIG. 144 shows an example GeneXpertAssayInfo Message type.

[00151] FIG. 145 shows example GeneXpertIndivisualAssayResult Message types.

[00152] FIG. 146 shows example GeneXpertAssayResults Data Structure.

[00153] FIG. 147 shows an example GeneXpertMessage Data Structure.

[00154] FIG. 148 shows example GeneXpertMessage Template properties.

[00155] FIG. 149 shows example gases detected by the Honeywell-Midas sensor adapter.

[00156] FIG. 150 shows example properties of the Honeywell-Midas Sensor Adapter.

[00157] FIG. 151 shows example ICD-compatible message properties.

[00158] FIG. 152 shows example properties for the Inova display.

[00159] FIGS. 153 and 154 show example properties for a lightweight chemical detector message data structure.

[00160] FIG. 155 shows example properties of the ModbusRegisterMessage Data Structure.

[00161] FIG. 156 shows example properties of the Modbus sensor adapter with typical values.

[00162] FIG. 157 shows example properties of Ortec Sensor Adapter Message Data Structure.

[00163] FIG. 158 shows example properties of Ortec (Radiation).

[00164] FIG. 159 shows examples of ITrans Sensor Adapter properties.

[00165] FIG. 160 shows examples of the specified QTL Biosensor 2000 properties.

[00166] FIG. 161 shows examples of the specified SNMP Net Guardian properties.

[00167] FIG. 162 shows examples of the specified Voxeo properties.

[00168] FIG. 163 shows example errors.

[00169] FIG. 164 an example SPM event traceability.

[00170] Like reference symbols and designations in the various drawings indicate like elements.

DETAILED DESCRIPTION

[00171] Sensor Policy Manager (SPM)

[00172] Techniques, systems and computer readable medium are provided for managing Sensor, Policy, and Communications Interoperability. Sensor Policy Manager (SPM) is a network-centric framework that provides interoperability among disparate, multi-vendor sensor and digital cameras, remotely controlled robotic devices, intelligent data processing and management, and communication systems. SPM integrates sensor, video, and bi-directional communication data for higher level analysis, representation, and execution of rule based policies that in turn will drive other instruments, policies, or individuals to take action. Applying a web-based user interface, users from remote sites can observe all connected sensors, cameras, and other input channels concurrently, or retrieve the data at a later time from archived databases.

[00173] Using external communications channels, such as phone, radio, e-mails, and electronic displays, SPM provides for a hierarchical alerting system, improving enterprise and government operations, business continuity as well as emergency safety and security management. SPM can potentially provide the user with one or more of the following benefits.

- Scalable Installation Protection (IP)-based network-centric platform capable of incorporating legacy and future sensors and camera systems.
- Integration, processing, and display of sensor and video data for automated emergency notification and easy visualization and interpretation by non-expert decision makers.

- Policy engine with configurable, logical, decision-tree schemas that when acting upon specific conditions result in distinct actions.
- Configurable, hierarchical alerting infrastructure, including delivery of unique messages to different groups of stakeholders via external communication devices and receipt notification ability.
- Fusion of sensor data and alerts over multiple sensors, locations, and time points for monitoring purposes as well as minimizing false positives by orthogonal confirmation.
- Configurable to user applications and deployment sites, including the ability to add additional sensors, as well as add or modify policies remotely.
- All network elements are controlled by an easy-to-use graphical interface, allowing web-based monitoring from anywhere in the world.

[00174] The processing of the data happens automatically according to policies that have been tailored for the particular sensors in the SPM system. A policy is a condition-action pair. When a certain event occurs, appropriate actions are taken and the corresponding messages are automatically published via a number of different communications channels. Actions may consist of commands to instruments to perform a certain task or alerts dispatched via phone messaging, e-mail, video displays and the like. This automated implementation is intended to reduce or even eliminate the need for on-site human intervention, thus minimizing human error. Policy-based data interpretation is crucial where humans are not or cannot be present.

[00175] FIG. 1 shows an example SPM system 10. The SPM system 10 can

include graphical user interface (GUI) elements 12 and server components 14. The server components 14 can include sensor adapters 16, a sensor adapter API, channels 18, a policy engine 22, a sensor policy manager API 20 and Simple Object Access Protocol (SOAP) command architecture. The GUI elements 12 on the client side are provided to a user to enable the user interface with the server components. The GUI elements 12 include a dashboard 24, a sensor adapter builder/configuration component 26 and a policy editor 28.

[00176] The sensor adapters 16 are component that communicate directly with physical sensors. The sensor adapters 16 can also communicate with other management systems such as Building Management Systems (BMSes), Access Control Systems, Supervisory control and data acquisition (SCADAs), etc. The sensor adapters 16 can receive commands and publish data and present a normalized interface to the rest of the SPM components.

[00177] The Sensor Adaptors normalize the sensor protocols, and information to Sensor ML. The SPM sensor policies analyze the predefined conditions and trigger actions, based on the policies configured by the SPM administrator. The policy framework is highly flexible, and allows aggregating the information as channels. These channels carry the information such as data, video and control information, which can be leveraged by other SPM policies for the real-time Sensor Dashboards or for archival of the desired event.

[00178] The SPM Policy framework allows for publishing and subscribing to the desired information based on the customer needs. This unique policy framework of SPM enables multi-threaded and real-time correlation of the sensor data. SPM

Policies use SPM Language derived out of Sensor ML. SPM Application Programming Interface (API) enables system integrators to subscribe the desired channels.

[00179] Channels 18 are communication pipes or paths for forwarding data to the Policy Engine 22, to a Java Message Service (JMS) server or over point-to-point connections using SOAP or raw sockets. The channels 18 may also be internal to a process (i.e., a local channel.) In this case, the channel 18 is used merely to forward an event from one architectural component to another (typically, from a sensor adapter 16 to the policy engine 22.)

[00180] Both sensor adapters 16 and the policy engine 22 can publish using the channels 18. The sensor adapters 16 and the policy engine 22 do not contain any knowledge relating to the type of the channel published to.

[00181] The SPM system 10 can support the following channel types.

- JMS to Publish and subscribe.
- SOAP for Point-to-point
- Local channel for on process only and to publish and subscribe.
- Permanent TCP connection for point-to-point.

[00182] The policy engine 22 evaluates various conditions and executes a list of actions when the conditions are passed. Supported actions of the policy engine 22 include: (1) Publishing an alert message on a channe; and (2) Sending a command to a sensor adapter.

[00183] The Policy Engine 22 includes a collection of individual policies. Each policy contains a condition template and an action template. The input to a policy is a channel, usually an output channel from a sensor proxy, but it could also be

a channel from another policy. A condition template is used to determine whether the policy should fire or not. A condition template could provide a simple threshold. For example, when the concentration of a gas monitored by sensor "A" exceeds threshold "X", the policy can fire. However, even for a simple case like this, the condition template will often need to be more complex. For example, one might have a rule to prevent the policy from firing more than once per minute in order to avoid "Event Storms." This means that Conditions Templates also have the capability to evaluate some very complex rules. When a policy fires (by the conditions being matched), the action template is executed.

[00184] The Action Template includes a sequence of actions. Examples of actions include transforming the input message(s) to a derived message type and publish to JMS on an output channel (such as publishing an alert.) Also, a command can be issued to a command-type sensor proxy. A specific example can result in a Text message appearing or a web page being displayed. The videos obtained for the past 30 seconds can be played back. A list of persons can be called and an audio message delivered to them. A user can start video retrieval occurring and publish it on a channel.

[00185] The SOAP command architecture is the only logical component that is not also a separate code component. The code for the SOAP command architecture resides mainly within the Sensor Adapter Framework and the SOAP framework.

[00186] The typical SPM configuration for IDSS includes

- A user-programmable general purpose Very hIgh density Policy EncodeR (VIPER) policy engine.

- Sensor adaptors (for different protocols and hardware devices).
- Sensor proxies within the Sensor Adapter API (embedded component within Sensor Adaptors for addressing variations from sensor to sensor for the same protocol.
- A web-based administration console (GUI) and dashboard that allows for editing of alerting and decision policies.

[00187] From an IDSS perspective, data and control elements (e.g. sensor, video, application program, signal device) are all "sensors" with some control and data attributes.

[00188] Systems and Services View (SV)

[00189] SV-1 Systems/Services Interface Description depicts systems nodes and the systems resident at these nodes to support organizations/human roles represented by operational nodes of the OV-2. SV-1 also identifies the interfaces between systems and systems nodes.

[00190] FIG. 2 shows example IDSS Enhancements to the IPP Mission. SPM-FPS, as the IDSS solution, interfaces and expands existing CBRN capabilities in a IPP by using an Integrated Information Management that ensures integration of the CBRN network with existing C4I capabilities to provide effective information management. For example, Single Common Operational Picture (COP) can be provided with a Layered View approach for viewing critical infrastructure including geolocations of assets and events. The layered vies can include views based on location, function and task grouping. The views can also be layered by category of assets. The IDSS COP can utilize either an ESRI or Google Map as the background. IP-based interoperability and policy framework can be provided for wired and wireless CBRN + Physical Security sensors and devices with FIPS

140-2 encryption. A Leader View provides first-level operators with the ability to escalate events to the attention of their Leader by pushing it on his/her screen. Tailorable message formats can be segmented into "Responder" priorities, i.e. 1st, 2nd, 3rd, Responders etc. Thus, subscribers can be segregated by priority and function. Workload Reduction can be achieved through integration and automation. Fusion of Communications and Sensor Data can be implemented to provide actionable information.

[00191] Physical Protection includes providing an effective CBRN protection, detection, identification and warning system for installation protection. For example, real time weather data can be integrated from multiple locations and HPAC-certified plume models. Adjustable Sensor Sensitivity provides the ability to dynamically tune sensors as well as combine orthogonal sensor inputs to minimize false positives and false negatives. Automated Video Tipping and Cueing is used to pan a camera to acquire video clip from the cued location of a chemical sensor. Both day and night vision is provided. Simple Video Analytics can be used to assist the operator by distinguishing between humans and other objects. GPS-enabled tracking of High Value-Low Density (HV-LD) assets, such as fire trucks, security vehicles etc. can be implemented. Also, Biometric monitoring (vital signs) can be implemented where the human is treated as a sensor through wireless means. Integration with robotic response is available for PACBOT and/or BAMS (Lockheed Martin). Other sensing capabilities include standoff explosive sensing devices, seismic and acoustic sensing.

[00192] CBRN Protection and Restoration provides a capability that allows for

Rapid Restoration of Critical Installation Operations. Distributed COP are available via Web applications and compatible with PDAs. Also, video and alerting via 2G/3G devices, SMS, voicemail and email can be implemented. Self-Diagnostics can be used to assess "up" time of System: Reliability, Maintainability (by personnel with 8th grade level of education) for a System Availability of 90% up time. Also, compatibility with civilian communications channels can be implemented. SPM can protect DoD civilians, contractors and other persons working or living on military installations and facilities. CAP 1.1 Message format is supported for communications between DoD and DHS COGs (Collaboration Groups). Radio communications interoperability is available to ACU-1000 encoded (P-25 standard) radios.

[00193] Graphical User Interface

[00194] FIGS. 3-5 show example GUIs for interfacing with the SPM server components. FIG. 3 shows an example GUI for use with the SPM system. The end user, supervisor, or administrator interacts with SPM-FPS through a web-based user interface. Upon immediately logging into the system via a web browser such as Internet Explore 7 or Firefox 3.0, the user can see a navigable and scalable Global Information System (GIS) representation of the location of the sensors currently being monitored by SPM, as illustrated in FIG. 3.

[00195] FIG. 4 shows an example display of a list of sensors being monitored. Information about alert conditions, as triggered by user-defined policies, as well as communications status with the sensors is presented in the two center panes. Furthermore, by clicking onto the "Sensor List View" tab in the right-hand pane,

the list of sensors being monitored is presented in a tiled format, as illustrated in FIG. 4.

[00196] FIG. 5 shows an example GUI for inputting user-defined policies. Choosing the appropriate icons in the view shown in FIG. 4 also allows the user to plot real-time data in order to observe data trends. Further, user-defined, condition-action policies can be inputted from the "Viper Config Editor," as illustrated in FIG. 5.

[00197] SPM Architecture and Main Components

[00198] FIG. 52 shows an example SPM architechture 5200. The sensor adapters 5210 provide a complete software wrapper around the physical sensor 5212. The function of the wrapper is to ensure that diverse sensor data and sensor message blocks are converted into messages whose formats can be understood by downstream processes in the policy engine (Viper) 5214. Similarly, a sensor adapter can be used to filter the data stream so that only information of interest is forwarded through the messaging channels 5216.

[00199] The creation of new sensor adapters or the modification of existing ones is typically not a task for the normal user, although SPM does include tools which allow administrative users to create and modify policies. Creation of a totally new sensor adapter often entails the addition of new look-up tables, algorithms, or formulas, and requires C++ programming experience. Services, documentation, and support for making new sensor adapters are available by contacting ViaLogy Technical Support.

[00200] The core of SPM is the Sensor Policy Engine 5214 known as the Viper

(**V**ery H**i**gh Density **P**olicy **E**ncode**r**). Each Viper 5214 contains a number of policies which assume the general form of an 'if' clause (the condition) and a 'then' clause (the action). Policies are constructed as templates that are generic for each type of sensor 5212. These templates can be customized to incorporate the characteristics of each sensor instance. Condition and action templates are joined to make up a policy template. Since the policy templates provided are generic or at best semi-specific, the system integrator or expert user creates specific instances of the policies to fit the particular real-world application. Specific policy instances are created using the Policy Wizard, or, alternatively, by editing the Viper configuration file. Input messages typically are data from sensors 5212 processed via their sensor adapters or messages which come from previously evoked policy actions. Templates for sensor adapter message types and examples of conditions and actions for different kinds of sensors and devices are available in a separate document, Sensor Adapter Templates.

[00201] Each Viper 5214 typically manages multiple sensor adapters 5210 and policies. Depending on the data and processing load it may be advantageous to distribute the sensor adapters 5210 and policies amongst separate Vipers 5214, each running its own processes. Alternatively, a light-weight implementation of the Viper 5214 called Viper Edge or SPM Edge can be constructed to operate high data rate devices such as video cameras by locating the Edge server in close proximity to the data device. The purpose of SPM Edge is to minimize the load on the network posed by such sensors and to only transmit processed information a majority of the time. Viper/SPM Edge implementations resemble

standard Vipers in many respects, but do not incorporate a database 5218.

[00202] The SPM Console 5220 is a Web-based Java application that interacts with the Sensor Policy Client API. The Console allows users to monitor data streams, results and events visually. Authorized users can log in to the SPM Console from any Internet location and view data and messages allowed by their permission level. The SPM Console 5220 is also used by the System Integrator to add or edit users and their roles and permissions as well as modify Viper configuration files.

[00203] SPM includes an internal database 5218 that stores the configuration and control operations that have been defined in the Viper configuration file. The internal database 5218 can store data and messages from the system for later retrieval via the SPM Console 5220.

[00204] SPM uses the Java Message Service (JMS) API as a messaging standard that allows application components to create, send, receive, and read messages. JMS enables distributed communication by providing asynchronous delivery of data between applications on a “store and forward” basis and is a useful method for time-independent or parallel information processing.

[00205] A particular strength of the SPM architecture is that any number of consumers can subscribe to the same channel in much the same way as tuning into a radio or television station. The multicast feature can easily be implemented without any involvement by the data producer. The multicast feature can be controlled in the SPM configuration file by adding or deleting recipients.

[00206] Another important advantage of using messaging middleware software 5216 is its ability to store data in a persistent repository before delivering it. Because SPM is a distributed network system, oftentimes the connection between data producers and data consumers can be temporarily broken. If guaranteed delivery is needed for some sensor readings or derived results, the corresponding channels can be configured as 'persistent.' In this way, once the data have been successfully published, they will be available to any subscriber even if that subscriber was not online at the time of the publication.

[00207] SOAP Communication Protocol is a simple XML-based protocol to let applications exchange information over http via the Internet. SOAP is platform- and language-independent that provides a way to communicate between applications running on different operating systems, with different technologies and programming languages. SOAP allows communications through firewalls in a safe and controlled manner. In SPM, SOAP is used to send and retrieve messages and commands between sensors and sensor adaptors, or between different instances of SPM with firewalls in between or running on two different platforms.

[00208] Lightweight Directory Access Protocol (LDAP) is used to store information about SPM users. This information includes user names, passwords, contact information, as well as their assigned roles and permissions.

[00209] Sensor Policy Client API is a JAVA API and provides the sole user interface to the other SPM components. Sensor Policy Client API is used by the SPM Console 5220 and by all communication applications which connect to

SPM. Sensor Policy Client API allows clients to configure and access the functionality of the VIPER Policy Engine 5214.

[00210] The Configuration Server 5220 has two primary responsibilities. The Configuration Server 5220 functions as the Database Proxy and provides the interface to the database for configuration and control operations. All changes to the configuration of SPM made using the SPM Console 5220 are routed through the Configuration Server, which writes the new configurations to the database before forwarding them to their assigned destinations.

[00211] SPM is available in either Linux or Windows XP based server applications utilizing TCP/IP. Normal communications with the SPM server are through SPM's web-based interface on IE6, IE7, or Firefox version 2.0.0.X. It can also be useful to have an SSH communication utility such as PuTTY for installation and monitoring. When used on a single subnet, there are no unusual networking requirements for SPM itself other than those required to support network communication between the primary SPM server, multiple SPM installations if used, the sensors, and the clients. However, when installations span multiple subnets TCP ports 443 (https) and 22 (SSH) should be open between the clients and the SPM server(s). Inter-SPM communications will require that port 1099 be open to support JMS messaging and port 15001 if TCP messaging is used. Other requirements may exist depending on which specific output devices are used.

[00212] SPM connects to any network device, sensor or otherwise, that directly supports TCP/IP or whose output can be converted to TCP/IP. Sensors that

support only serial RS-232, RS422 and RS485 output can usually be converted by serial-to-Ethernet bridge products such as those made by Digi System. For instructions on configuring specific setups, users must refer to the bridge vendor's product information. SPM also supports the MIDAS interface and protocol. Specific MIDAS information can be reviewed at http://midas-nms.sourceforge.net/. See http://www.comptechdoc.org/independent/networking/guide/ for a general review of networking principles and protocols.

[00213] Sample SPM Deployment

[00214] In the example of an SPM deployment shown in FIG. 52, an SPM Server 5202 is integrated with several sensors 5212 in different locations. The main SPM Server (center) 5202 constitutes a centralized site where data are incorporated. Data flows from the sensors 5212 to their respective sensor adapters 5210. From there, the processed data can be sent to policies or directly to a messaging service from where it can be published or subscribed to. The data is available for other internal or external services such as displays, evaluation by policies or database storage. The configuration server 5222, which may reside in a different server than those which execute the policies, can be used to set up and control these other servers.

[00215] SPM User Roles

[00216] Example predefined user roles assigned to permission include System Integrator, Expert User and End User. The System Integrator (SI) is responsible for setting up the Linux Server and installing SPM. The SI configures the SPM

system which consists of an initial set of sensor adapters, message types and policies, and links up the sensor adapters to the various sensors and to the computers which control them. The SI is also able to add new sensor adapter instances to SPM and define user roles.

[00217] The Expert User can edit existing policies and create new ones. In contrast to the SI, the Expert User cannot define user roles or add new users and assign user passwords. The Expert User can exercise all the functions of the End User.

[00218] The End User is restricted to monitoring and observing SPM data and alerts and listing the users and their roles. The End User can also view the wording of sensor adapter instances and policies, but has not editing privilets.

[00219] Accessing SPM

[00220] The SPM Console web application has been designed for optimal viewing on a 1280 x 1024 monitor. The application has been optimized for viewing using Firefox 2.0, for example. Minor inconsistencies in appearance may be experienced when viewed with other browsers.

[00221] The Dashboard is the default landing page after logging in to the console. FIG. 53 shows an example main panel 5300 of the dashboard. Several other panels of useful information can be accessed from the main panel. FIG. 54 shows an example Main Menu panel 5400 used to access the main menu navigation panel. SPM forms and functions can be accessed from this main menu.

[00222] FIG. 25 shows an example sensor proxy framework 2500. The Sensor

Proxy Framework is designed to allow the user to develop new sensor proxies very rapidly by establishing some standard patterns and hence allowing for a very high reuse of the existing code. For example, each sensor proxy communicates with an individual sensor or mediator and normalizes the data received. Also, data is published via channels through JMS, SOAP or locally. JMS is typically used to publish to an application server. SOAP is used to navigate through a firewall or where a high performance is required.

[00223] In addition, the SPM supports a completely distributed architecture. JMS and SOAP output channels can be inputs to other SPMs. The normalized format of all messages is standards-based. This format is SensorML for regular data and CAP1.1 for alerts and alarms. Each Sensor Proxy can publish to four kinds of channel: namely data, alert, communications alarm and equipment alarm. The writer of a new sensor proxy is isolated from the details of this including the underlying protocol supported by the channel. Any sensor proxy can be configured to have one or more of any of these channel types. The messages passed through each channel are configurable. These message types are configured so as to provide meaningful information about the underlying sensor. A special support is provided for sensors with many heads or for mediators where many different kinds of sensor are supported. Each of the heads has its own set of channels. Simple filters may also be configured for a sensor proxy so that only data satisfying certain criteria is forwarded on a channel. There is an in-built support for a wide range of IP-based protocols in the sensor proxy framework. This means that the writer of a new sensor proxy is

isolated from the underlying protocol and can concentrate on the customization required for the particular sensor proxy rather than the transport. Each sensor proxy also has the capability to receive commands from the policy engine or from other sensor proxies. These commands can be local (on-process) or remote (forwarded to another SPM over SOAP.) There are a set of tools built into the sensor proxy framework that assist the developer in writing a new sensor proxy. Examples are a token parser for binary protocols and a generic XML parser which converts an arbitrary XML document into a normalized format for use within the SPM. Each Sensor Proxy supports a simple simulation mode where the proxy forwards data based upon its internal configuration instead of establishing communications with a real sensor and sending live data.

[00224] FIG. 26 shows an example sensor proxy framework for protocol support 2600. The sensor proxy framework 2600 provides base classes which support a wide range of protocols. The writer of a new sensor proxy derives from one of these classes, which effectively allows him to concentrate on the customizations that are required for the particular sensor. The workings of the underlying transport or application layer protocol are automatically taken care of by the framework. The protocols supported by the framework 2600 include the following: (1) TCP client and server; (2) UDP; (3) HTTP client and server; (4) SOAP client and server; (5) Telnet client; (6) SNMP Manager and Agent; and (7) XML-RPC.

[00225] Sensor Proxy Framework

[00226] The protocols supported by the sensor proxy framework include: TCP

Client; TCP Server; UDP; Telnet Client; HTTP Post Client; HTTP Post Server; HTTP Get Client; SNMP Manager; SNMP Agent; SOAP; TCP Modbus; and various proprietary protocols. All of these protocols belong to the TCP/IP suite of protocols, or are application layers running aboveTCP/IP.

[00227] FIG. 27 is a diagram showing a simplified example representation of a SPM communications infrastructure 2700. The components of the SPM include a policy engine 2710 a sensor proxy framework 2720 and a channel architecture 2730. These components are autonomous subsystems. The policy engine 2710 and the sensor proxy framework 2720 are able to publish asynchronous data by means of the channel architecture 2730. The channel architecture includes different kinds of channel, such as JMS, SOAP and local channels.

[00228] Local channels are used when the source and end points of the channel are both on process. SOAP channels are typically used where the footprint of the process is important and JMS is not supported.

[00229] In addition to publishing, there is a command architecture which allows the Policy Engine 2710 to send commands to individual sensor proxies. The command architecture is a part of the sensor proxy framework 2720.

[00230] The command is forwarded locally if the target sensor proxy is on-process or remotely over SOAP if it is running on a remote process. This is transparent to the Policy Engine 2710. Sensor proxies are also able to use this command architecture to send commands to each other

[00231] FIG. 28 shows example component dependencies 2800. The sensor proxy framework and the policy engine are autonomous subsystems within the

SPM. The sensor proxy framework has no direct dependencies upon the policy engine. The policy engine has a single dependency upon the sensor proxy façade to issue commands to individual sensor proxies. The sensor proxy façade is the only interface into the sensor proxy framework that should be used by any component outside of the framework itself. The only functions required are initialization of the framework, shutdown of the framework and the issuing of commands to individual sensor proxies. The sensor proxy façade is the only SPM component that has a direct dependency upon the individual sensor proxies. Also, dynamic libraries for the individual sensor proxies can be generated and loaded dynamically. Once the dynamic libraries are implemented, this component dependency can be eliminated. There is a separate project (package) for every individual sensor proxy. Individual sensor proxies have no compile time dependencies between each other. In most cases, there are no run-time dependencies either. There is an exception to this for composite and single-headed sensors. The ViaSensorProxy component is the heart of the sensor proxy framework, containing the base classes and utilities which are used in order to write new sensor proxies. Both ViaSensorProxy and the policy engine have dependencies on the channel architecture, and use channels to publish data. ViaServiceDataObject is a metadata library, loosely based upon the "Service Data Object" specification. ViaServiceDataObject can be used with the sensor proxy framework but can also used by other components (not shown in the diagram.) ViaAce is a wrapper around the ACE framework. ACE is a cross-platform toolkit used for managing threads and sockets. ViaAce is used by

several other components besides ViaSensorProxy (not shown in the diagram.)

[00232] TCP/IP Communications Stack

[00233] The sensor proxy framework supports sensors that are IP-enabled. The sensor proxy framework does not support serial interfaces such as RS-232 and RS-485.

[00234] FIG. 29 is a table showing different layers of an example TCP/IP Stack 2900. The first column describes the name of different protocol layers. The second column shows the TCP/IP mappings and framework usage of each protocol layer. The third column shows the services corresponding to each protocol layer. Each protocol layer uses the services of the layer directly below it. The first three protocol layers are encapsulated by the transport layer within the programming API. The sensor proxy framework deals only with the transport and application layers.

[00235] Connection-Oriented and Connectionless Transport Protocols

[00236] The usage of some of the tools within the sensor proxy framework depends upon whether the underlying protocols are connection-oriented or connectionless. A connection-oriented protocol is defined as a protocol where a connection has to be established between a client and a server before an exchange of data can occur. When the data interchange is complete, the connection is closed.

[00237] A connectionless protocol is defined as one where no connection has to be established and data can be exchanged at any time. Where connectionless protocols are used, the distinction between client and server are

less clear-cut than in the connection-oriented case.

[00238] The Ethernet and IP layers are connectionless. The transport layer offers a choice between a connection-oriented and connectionless protocol.

[00239] TCP is the connection-oriented stream protocol. When a connection is established, the ordering of data sent and received is guaranteed as is delivery at the receipt point. Data packets can be concatenated or segmented. UDP is the connectionless datagram protocol. The ordering of packets received is not defined and some packets may be lost altogether. Packets are always discrete (i.e. there is no concatenation or segmentation.)

[00240] HTTP is a hybrid between the two, but for most purposes it can be treated as connectionless within the sensor-proxy framework. HTTP runs above TCP (connection-oriented), but after establishing a connection, there is a single data exchange and then the connection is closed. This means that the transmission of each HTTP Request is stateless (unlike raw TCP but similar to UDP.)

[00241] FIG. 30 shows an example protocol class model 3000. The main task in writing a new sensor proxy is to derive and write a protocol class. All protocol classes inherit from one of the base classes shown in FIG. 30. These protocol classes protect an engineer from having to deal directly with the underlying transport protocol used by the sensor proxies. Also, support for standard application protocols, such as Telnet and SNMP, are provided within these generic protocol objects. SNMP is discussed further below.

[00242] The base class, ViaSensorProxyProtocol provides the sole interface to

the channel architecture, the command architecture and basic filter logic.

[00243] Using the Protocol Classes to write a Sensor Proxy

[00244] The first step is to identify the underlying transport protocol to be used by the new sensor proxy and use the table in FIG. 31 to decide on the inheritance. A Sensor Array consists of a set of similar sensors that share a resource (such as a TCP Connection) and so have a common polling sensor proxy. For example, the Itrans gas sensor is double-headed with two gas detectors, one of which detects carbon monoxide and the other methane. Each of the heads is represented by a class derived from ViaSensorProxySingleHeadProtocol which depends for its data feed on an underlying polling sensor proxy.

[00245] A Sensor Group consists of a set of dissimilar sensors that do not share resources, but have some logical dependencies upon each other so that a higher-level (compound) sensor proxy manages them as a group. For example, a trigger to a compound sensor proxy requires it to read GPS and compass data, move a camera to point in the right direction and start acquiring and publishing photographs.

[00246] FIG. 32 shows an example ViaSensorProxyProtocol base class 3200. The ViaAceSocketThreadAction class is included here only because it provides some of the virtual methods that the programmer must override. All of the methods are described in the “C++” header files themselves.

[00247] FIG. 33 is a table showing example ViaAceSocketThread methods 3300. The table is provided for reference purposes only.

[00248] FIG. 34 is a table showing example ViaSensorProxyProtocol methods 3400.

[00249] Writing a Sensor Proxy using the Protocol Classes

[00250] The virtual methods to override depend upon the underlying transport protocol. A Sensor Proxy base class is selected to derive from based upon the underlying transport protocol. If the selected protocol is not supported or some custom behavior is needed, then the system inherits from ViaSensorProxyProtocol and overrides the Execute() method. The system performs the Override initialize() method if any extra initialization steps are needed prior to entering the main loop. The getType method is always overridden. A new enumerated type is added to the ViaSensorProxyEnums in the header. The Clone() pure virtual constructor method is also overridden. The acceptRunValues() is the virtual method used to load values from the configuration object SDO. If the sensor proxy can receive commands then the acceptUpdate() method is overridden. The configuration file is set up.

[00251] In addition, certain methods are overridden depending upon the underlying transport protocol.

[00252] TCP Client:

[00253] Sensors may be active or passive. For both types of sensors, the receiveFromServer() method is overriden. This is used to process incoming data. If the sensor is active (requires polling) then also the sendPoll() method is also overridden.

[00254] TCP Server:

[00255] The receiveFromClient() method and the sendPoll() method are overridden if the sensor is passive.

[00256] HTTP Post Client:

[00257] The sendPost() method is overridden if extra actions are required. By default, this method sends the post string to the HTTP server once every poll period. The initializePostString() is also overridden. This sets up the HTTP post string based upon the configuration file. The receiveFromServer() method is overridden to process the HTTP response.

[00258] HTTP Get Client:

[00259] HTTP Get strings take the general form: hostname:port/applet?parm1;parm2;parm3 etc. The pure virtual method constructURL() is also overridden. This constructs the HTTP Get string based upon entries in the configuration file.

[00260] Telnet Client:

[00261] The Telnet Client is handled in the same way as for the TCP Clients.

[00262] FIG 35 shows an example Service Data Object Class Model 3500. Service Data Objects (SDOs) are used within the sensor proxy framework as metadata in two different ways. Firstly, SDOs are used to describe the messages that are sent out on data channels and on alert channels, and used to publish data internally within the process. At the process boundary, SDOs are converted to SensorML.

[00263] Secondly, the configuration of each sensor proxy instance is described by anSDO. The ViaSdoDataObject class uses the composite design pattern and

so has the capability to contain a collection of ViaSdoDataObjects to any depth. ViaSdoDataObject also contains a collection of ViaSdoProperty which can be any one of the supported types shown in the diagram. ViaSdoProperty can contain a single value or an array of values. Where there is an array, all of the values can be extracted into a ViaSdoPropertyList object.

[00264] FIG. 36 shows an example data object class 3600 and an example property class 3610.

[00265] FIG. 37 shows example Observer Design Pattern classes. Essentially, a model observer can register interest in a model object without publishing its own interface to the model object. When the changed method is invoked on the model object, either by the model object itself or externally, then the virtual method updateFromModel() is invoked on each of the model objects registered observers. The updateFromModel() method is overridden for each observer so as to get the information it needs from the model and update itself based upon the change.

[00266] This pattern is used where the model objects should not know much about who their observers are. All sensor proxy protocol objects are both models and observers and the observer pattern itself is used in several different ways within the sensor proxy framework.

[00267] Example of a Multi-headed sensor proxy: Itrans

[00268] The Itrans sensor is a gas sensor which has two heads. One head tests for Carbon Monoxide: the other for Methane. However, the two heads are not entirely separate since they share a single TCP connection.

[00269] If modeling as a single sensor proxy, each head will publish to different channels and the proxy has to apply some highly custom logic to decide on what to publish and to where. Suppose we have another sensor with twelve heads. This soon becomes unmanageable.

[00270] If modeling as two separate proxies, unwanted coupling between the two are introduced in order to share the TCP socket. The solution is to model an underlying polling sensor proxy that establishes the TCP connection and retrieves the data. This is a standard TCP client type sensor proxy, except that it does not publish to channels. Two single-head sensor proxies are modeled, which set up observer relationships with the TCP client proxy and each has their own channels. The single-head sensor proxies apply standard filters which filter out data not for them. Both filters are notified by the TCP client when any data is received. When the single-head sensors receive data that passes their internal filters they publish to their own channels.

[00271] Unlike almost every other type of sensor proxy, single-headed sensors are not run on their own threads. They receive notifications on the thread of their model object (the TCP client) and action the data synchronously. Single-headed sensor proxies have a do-nothing execute loop which exits immediately, since they depend upon the Execute loop of the associated model object.

[00272] To create a single-headed sensor proxy, the system can inherit from ViaSensorProxySingleHeadProtocol. In addition, the following methods are overridden: getType(); acceptRunValues(); Clone() and updateFromModel(). If the data passes the filter, the data is published. Some single-headed sensor

proxies also have to transform the data here.

[00273] XML Configuration Files

[00274] An XML configuration file is used to specify what sensor proxies to run and which what parameters. Configuration files corresponding to the Itrans sensor proxy are described to illustrate the usage.

[00275] In order to create a new XML tag for use in the configuration file, the XML tag is added to the XML schema, as a part of the subSectionEntry "choice" as shown in FIG. 38.

[00276] FIG. 39 shows an example sensor proxy type section 3900 in a configuration file. The sensor proxy type section 3900 is a required section in a configuration file. The sensor proxy type section 3900 contains the names of all sensor proxy types to be run within the application.

[00277] ITrans is the type name for the Itrans TCP client sensor proxy. ITransChannel is the type name for Itrans single-headed sensor proxies.

[00278] Message Type

[00279] FIG. 40 shows an example message type section 40000 in a configuration file. Any message types used by the sensor proxies must be specified in the configuration file. For example, Itrans uses GasMessage as its data type.

[00280] FIG. 41 shows an example sensor proxy entry 4100. The polling rate is in seconds. The IP address and port specify the socket to which the client opens a connection. The sensor name is the name which is used to address this sensor proxy for sending commands. The enabled flag is standard. For

example, 0 = disabled, I = enabled, and 2 = suspended. The data message type and alert message type names must match to entries in the Message Types section.

[00281] FIG. 42 shows example entries for the two single-head sensor proxies. The filter values are applied by these sensor proxies to evaluate whether the data passed in is for them. The "SP_Model" tag gives the name of the TCP client sensor proxy with which these register with to set up the observer relationship. Both of these proxies are configured to publish to a JMS channel.

[00282] Adding a new sensor proxy to the application

[00283] The sensor proxy static libraries can be converted to DLLs and dynamically loaded. In such instances, integration of new sensor proxies is implemented using the configuration file.

[00284] When not converting to DLLs, a new enumerated value is added to ViaSensorProxyEnums as the type of the new sensor proxy. The new type is added to the ViaSensorProxyNameConverter class. This associates the enumerated values with the strings used in the XML configuration file. In the sensor proxy façade, the header is included for the new sensor proxy type in the source file and the new sensor proxy type is registered in SensorProxyFacade::registerProxies(). In addition, the include paths are updated for ViaSensorProxyFacade to find the header file for the new sensor proxy. Further, the configuration file is updated.

[00285] SNMP Sensor Proxies

[00286] SNMP Manager (Trap Collector)

[00287] The SNMP Manager is a specialized sensor proxy that binds to the SNMP Trap Port (UDP 162.) Only one SNMP Manager can run on a particular station, but it is able to receive SNMP traps from any number of sensors.

[00288] The processing of these traps is specific to the kind of sensor that originated them and so the scenario here is one sensor proxy that receives the data from many sources and individual sensor proxies that transform and publish the data. In other words, this is another situation where the use of single-headed sensors is appropriate.

[00289] The SPM supports at least two kinds of device that send SNMP traps, such as a UPS sensor and a Seismic sensor. Hence, a total of three SNMP sensor proxies are provided that receive traps.

[00290] An example SNMP sensor proxy is the SNMP Manager that binds to the SNMP trap socket and receives the traps. Other examples include the UPS and Seismic sensor proxies which are both single-headed sensor proxies and register interest with the SNMP Manager. As the number of supported SNMP devices grows, many more of these single-headed sensor proxies may be available.

[00291] SNMP Agent (Trap Generator)

[00292] The SNMP Agent does no polling and receives no data from outside of the process. Nor does it publish any data. The SNMP agent is essentially a "Command" type of proxy which waits to receive a command from the policy engine or from another sensor proxy. The command specifies the SNMP Trap to be sent. This must be a trap that is specified in the SPM MIB file. Both the

SNMP Manager and Agent derive from ViaSensorProxyProtocol and use the UDP mode of operation.

[00293] Composite Sensor Proxies

[00294] The composite sensor proxies manage a group of dissimilar sensors that form a logical group, requiring some common processing. Composite sensor proxies do not have any underlying protocols of their own but rely upon the individual proxies comprising the group in order to obtain data from the physical sensors.

[00295] FIG. 43 shows example composite sensor proxies. This is a slightly simplified example of an actual scenario supported by the SPM. On receiving a trigger (command), the Image Acquisition sensor proxy retrieves the latest GPS position and compass bearing from two individual sensor proxies and issues a command to a camera sensor proxy to point the camera in the right direction and begin image acquisition. The Image Acquisition sensor proxy already has the GPS and compass readings required because the Image Acquisition sensor proxy has registered to receive this data via local channels. The Image Acquisition sensor proxy uses the standard command architecture in order to control the camera. By using the standard command architecture and using local channels, there's no need to introduce coupling between these four different sensor types. The GPS and compass sensor proxies have no intelligence about the Image Acquisition proxy. They only publish to channels.

[00296] Similarly, the Camera sensor proxy only has the intelligence to check its internal queue to see if there are any commands. The camera sensor proxy

knows nothing about the source of the commands.

[00297] The Image Acquisition sensor proxy has to understand something about the fields in the normalized messages it receives from the GPS and Compass sensor proxies and the format of the commands it sends to the Camera. The Image Acquisition sensor proxy has no knowledge about the sensor proxies themselves and the underlying protocols are completely hidden from it.

[00298] How to write a Composite Sensor Proxy

[00299] The new composite proxy protocol object inherits from the ViaSensorProxyCompositeProtocol and uses its execute loop. The usual methods are overridden, including getType(); Clone(); Initialize(); and acceptRunValues(). Composite proxies are not associated with a real protocol. The composite proxies receive there data by command and by input channel. In addition, the composite proxies override the following methods: onReceiveData() that is invoked when data from an input channel is received; and onReceiveCommand() that is invoked when a command is received from the policy engine or from another sensor proxy.

[00300] Registering interest in local channels is a configuration option requiring no coding. FIG. 44 shows an example configuration section in XML. The "localInputChannel" names must match the names of the local channels that the compass and GPS sensor proxies publish to which is also specified in the configuration.

[00301] Sensor Proxy Framework – Utilities

[00302] ViaSensorProxyPacketAssembler is used for proxies with connection-oriented protocols and is used to undo packet concatenation and segmentation. For example, the physical sensor can be a TCP server. The TCP server sends data asynchronously as a response to polls. However, responses get concatenated together so that several responses can be sent as a part of a single TCP packet. Also, the beginning and end of the TCP packet cannot be assumed to align with the beginning and end of responses from the sensor. Each response starts with an ACK character (hex 6) or a NAK character (hex 15) and ends with an ETX character (hex 3.)

[00303] The sensor proxy physically contains a packet assembler object with a construction as shown in FIG. 45. When the sensor proxy receives data from the sensor, the receiveFromServer method is invoked. The data received is not processed by the proxy but passed directly to the packet assembler as shown in FIG. 46.

[00304] Each time the packet assembler parses a completeresponse, it notifies the sensor proxy via the updateFromModel method, which is then able to process it as shown in FIG. 47. This utility is suitable for both binary and ASCII protocols.

[00305] Simple Filters

[00306] Simple filters can be set up within the configuration file. This filter operates on a Service Data Object (SDO), usually the data message SDO or the alert SDO. FIG. 48 shows a sample XML. The filter tags in FIG. 48 have the following meanings. FilterAttribute represents the name of the SDO property to

apply the filter to. FilterValue represents the value of the attribute to apply the filter to. FilterMode represents the valid values are AND and OR. FilterType can be 0 = less than; 1 = greater than; or 2 = equality.

[00307] The filter outlined in FIG. 48 provides an AND filter which is passed when the AlarmType field = 4 AND the AlarmSubType = 2. From a sensor proxy, a filter is evaluated with the following line of code:

```
ViaSdoDataObject sdo;
...
bool isPass = m_filterMgr.passFIlter(sdo);
```

[00308] If the sensor proxy is not configured with a filter then this will return "true." In other words, a null filter is always passed. If one or more of the SDO properties is not present in the SDO input parameter, then this statement will return "true."

[00309] Sensor Proxy Group Manager

[00310] The Sensor Proxy Group Manage is a utility that allows sensor proxies of the same type to be assigned to groups. The utility is driven from the configuration file. FIG. 49 shows an example XML configuration file section. Each sensor group is assigned a group name and a type. The type must match one of the entries in the "SensorProxyTypes" section of the configuration file. The Sensor names must match to corresponding names in the sections where the sensor proxies themselves are specified. No coding effort is required to load the sensor groups.

[00311] FIG. 50 shows an example code fragment. The code fragment shows

how to retrieve sensor groups from the Group Manager Singleton object.

[00312] FIG. 51 shows an example SPM / SPM Edge Configuration 5100. Each instance of the SPM Edge runs on a separate work station and managed a set of sensors using the sensor proxy framework. The SPM server has the capability of managing sensors directly, but for this configuration, the SPM server runs only the policy engine and acts as a hub. The arrows represent a two-way communication between the SPM server and SPM Edge processes.

[00313] Publishing Data from the SPM Edge to the SPM server

[00314] Each of the sensor proxies running on the SPM Edge is configured to publish data to SOAP channels. This is a configuration option. On the SPM server, one instance of the SPM Edge Server sensor proxy is run. This receives the SOAP packets from the SPM Edge and republishes to JMS channels or local channels.

[00315] Sending commands from the SPM server to the SPM Edge

[00316] The SPM server runs one instance of the SPM Edge Command sensor proxy. When the SPM server needs to send a command to a sensor proxy, the command consists of the name of the sensor proxy and an SDO specifying the command details. The command is passed to the sensor proxy framework. If the sensor proxy is running locally on the SPM server, the command is passed to the proxy directly. Otherwise, the command is passed to the SPM Edge Command sensor proxy. This proxy holds a table of sensor proxy names together with the SOAP end point for each (contained in the configuration file.) The Command proxy looks up the SOAP end point for the sensor proxy name it

has been given and sends the command to the SPM edge.

[00317] The SPM has the capability to manage and interact with other sensor management systems such as the Richards-Zeta mediator and Lenel On-Guard. Using the two SPM Edge sensor proxies, SPM Server is able to interact and manage a remote SPM Edge processes in much the same way. The only two differences are that the publishing and command patterns have been separated, and that the SPM Edge proxies are more flexible in that SPM server and SPM Edge processes are interchangeable in the interactions that are possible.

[00318]

[00319] CAP 1.1 Support and Flexible Mapping

[00320] Introduction

[00321] Channels provide communications pipes for published data between sensor proxies, policies (which may be on-process or off-process), the application server and third party applications. Channels are responsible for converting the input SDO to SensorML (for JMS channels) and to serialized SDO (for SOAP channels.)

[00322] FIG. 6 shows an example design 100 of a channel class model that shows the functionality of the channels. The channel class model includes a flexible mapper component 102, an abstract base channel class 110 and a protocol converter model 112.

[00323] ViaChannel represents the abstract base channel class 110 that includes concrete base classes, such as Java Message Service (JMS) channels 104, SOAP channels 106 and local channels 108. Local channels 108 are used

by on-process senders and receivers that use a non-serialized SDO format.

[00324] The protocol converter module 112 converts the application protocols, such as SOAP and JMS. The protocol converter module 112 supports three formats including SensorML, CAP1.1 and serialized Service Data Objects.

[00325] The flexible mapper component 102 performs conversions between the XML SensorML and CAP1.1 schemas (or rather the SDO representation of these.) The conversion is necessary because the two schema are very different and most sensor proxies hold message data in a format suitable for conversion to SensorML. To avoid needing to rewrite every sensor proxy, the translation/conversion mechanism (e.g., the flexible mapper 102).

[00326] The NC4 sensor proxy converts from CAP1.1 format to SensorML format. The conversion can operate in the opposite direction also. The flexible mapper component can be hard-coded or included into SPM files. FIG. 7 shows an example mapper statement 200 in SPM. The source and destination entries provide full SDO paths to the SDO properties. The final numeric index provides a subscript for an entry in an array.

[00327] Interaction Diagram

[00328] FIG. 8 shows an example interaction diagram 300 for publishing a message. The interaction diagram 300 shows the various processes involved in publishing a message using design 100 described with respect to FIG. 1 above. The attributes of the channels include a payload type, a flexible mapper name, and a data mapper. The payload type can include SensorML, CAP1.1 or SDO. When SensorML is assigned as the default, the payload type is optional for the

simple cases. The flexible mapper name is implemented when mapping between SDO formats, such as between SensorML and CAP1.1. The data mapper object can be supported by the compiler or the database.

[00329] The architecture or design 100 can potentially provide the following advantages. The design 100 supports CAP1.1 and other formats without rewriting the sensor proxies or policies. In addition, flexible mapping can enable CAP1.1 integration.

[00330] The Generic Sensor Adapters: Protocol Mediation

[00331] Introduction

[00332] This document explains the architectural principles relating to how the SPM has the capability to customize the format of published messages using configuration and without the need for "C++" coding.

[00333] All of the features discussed are already extant within the SPM unless it is explicitly stated otherwise. System testing of these features is ongoing.

[00334] Sample Configuration

[00335] FIG. 9 shows an example configuration 400. The example configuration includes a Generic Modbus Adapter 402 in communication with an SMC 404 through a FieldServer 406. The Generic Modbus Adapter 402 includes various channels 408, 410, 412 and 414. The channels 408, 410, 412 and 414 can publish to different platforms 416, 418, 420 and 422 using application protocols 424, 426, 428 and 430. The example configuration 400 can be implemented without any custom coding by using the configuration language (e.g., sensor policy markup language (SPML)) to instruct the Generic Modbus

Adapter and the channels how to behave.

[00336] Generic Modbus Sensor Adapter

[00337] Functions of the Generic Modbus Sensor Adapter are supported by Modbus TCP protocol. FIG. 10 shows an example configuration entry 500 that aligns with the configuration 400 shown in FIG. 9. The example configuration entry 500 can be used to communicate with the SMC sensor via the FieldServer gateway. The four channels declared in FIG. 10 correspond to the four channels 408, 410, 412 and 414 shown in the diagram.

[00338] FIG. 11 shows a table 600 that explains the configuration properties. The table 600 includes a column that describes the properties 610, such as Polling Rate, Enabled, IPAddress, Port, MBAddress, MBFunction, MBStartAddress, MBCount, MBSlave, Modbus Type, etc. The table 600 also includes a column describes the corresponding descriptions 620. The Polling Rate represents a rate for sampling data in units of seconds. The Enabled property represents a flag for enabling/disabling this sensor adapter. The IPAddress represents an IP address or hostname of the FieldServer Gateway. The Port property represents a port of FieldServer gateway. For example, port 502 is the standard port for Modbus. MBAddress can be ignored for Modbus TCP, but is required for Modbus RTU. The MBFunction property represents the name of the Modbus function to be used for the poll. The MBStartAddress property is used to register start address for the poll. The MBCount property represents the number of registers to read. The MBSlave property represents the Modbus slave address. The ModbusType represents the Modbus type, with

valid entries that include MODBUS_TCP and MODBUS_RTU. Modbus-RTU is only supported when IP-enabled via a gateway.

[00339] The Sensor Adapter does not include intelligence on the message formats such as SensorML, CAP1.1 or SEIWG-ICD-100. The sensor adapter is associated with a message type (also from the configuration file) shown in FIG. 7. The sensor adaptor uses the message type shown in FIG. 12 to publish. This message type may require some extension in order to gain the capability to deal with some of the other Modbus functions. The generic Modbus adapter dumps out the values of the registers it reads into the "RegisterValues" array. This does not result in any customization to SensorML, CAP1.1 etc. and the heavy lifting is done by the channels.

[00340] Channel and Message Customization

[00341] Referring back to FIG. 9, channels can be used as described in this specification with both generic and custom sensor adapters without any code modification to those adapters. As shown in FIG. 9 above, each sensor adapter includes different types of output channels. These channel types can include Data Channel, Alert Channel, Communications Alarm and Equipment Alarm. Each of these channels types consists of collections of 0-n channels. The adapters themselves contain no intelligence relating to the format or configuration of its channels (which may be heterogeneous.) The adaptors publish their raw messages to the channel collections as appropriate.

[00342] There are different kinds of transport mechanism available for channels, such as JMS, SOAP (mainly used for navigation through firewalls.),

Local (for routing data which in on-process.), and TCP Client permanent connection. In FIG. 9 above, the channel publishing to the policy engine 416 is local, the one publishing to the App Server 418 is JMS and the other two 420 and 422 are TCP clients (because this is the interface provided by the COBRA and TASS platforms.)

[00343] Also, other channel types can be included to the design. For example, a type of channel can be provided that emits SNMP traps. The added channel type can be provided without any coding required at the adapters.

[00344] In addition to the transport mechanism, channels also include an application protocol setting. Some of the example protocol settings supported include SensorML, CAP1.1, SEIWG-ICD-100, and Arbitrary XML format. CAP1.1 and SEIWG-ICD-100 are both special cases of the arbitrary XML support.

[00345] Channels and Message Mapping

[00346] When considering the format of the message being published by the generic Modbus sensor adapter and what it would take to convert this message to a format that is suitable for publishing in SensorML or anything else, more components may be needed to avoid some hard-coding. To prevent such had-coding, a mapping object (e.g., the flexible mapper 102) is provided to perform the needed conversion of the message between different formats.

[00347] FIG. 13 is shows an example mapping entry from a configuration file. This mapping entry contains the mappings from the raw Modbus message produced by the adapter to a more meaningful representation. FIG. 14 shows an example declaration of the channel which uses this mapping entry. It should be

noted that the designation of the message protocol as type CAP1.1 is a misnomer. This actually means arbitrary XML. While the design could be set up to be real CAP1.1 but both the target message type and the mapping object would be much more complex.

[00348] FIG. 15 shows an example declaration for the SMCData message type. The properties match the right-hand side of the mapping object.

[00349] FIG. 16 shows an example published data generated from this configuration using the FieldServer / SMC sensor at the bottom end.

[00350] Generic SOAP Sensor Adapter

[00351] The NC4 sensor adapter accepts NC4 messages without having any in-built knowledge about the schema. While the NC4 adapter just parses an arbitrary SOAP/XML input, the channels are configured to translate the kind of data that the adaptor receives. The channel architecture can be the same as the channels of the Modbus adaptor described with respect to FIG. 9 above. The NC4 adapter is a passive SOAP server that receives SOAP requests containing the NC4 CAP1.1 alerts and sends an appropriate response. Also, a generic SOAP client can be included to carry out polling.

[00352] The Modbus generic adapter can be implemented to combine different polling operations and process generic SET requests that can be passed to the sensor. The SOAP adapter can be developed against several other platforms to ensure the generic nature and provide sufficient functionality.

[00353] SPM / SPM Proxy

[00354] FIG. 17 shows an example SPM/SPM Proxy 1200. In the SPM – SPM

proxy scenario, there are different communications interfaces, such as SPM Proxy – SPM interface 1210 and Proxy – Sensor interface 1220. While these two interfaces share some of the same issues, their natures are quite different.

[00355] The SPM Proxy – SPM interface 1210 is one over which we have almost complete control since we control both ends. All differences in the sensor protocols are completely encapsulated at the Sensor Proxy layer, which presents a normalized sensor proxy interface to the rest of the SPM. This normalized interface is comprised of a standard metadata for publishing data and for sending commands, together with the channel infrastructure. At the SPM Proxy – SPM interface 1210, the client – server relationships between the various sensors and their proxies are completely transparent and does not need to be taken into account in any way.

[00356] On the other hand, the Proxy – Sensor Interface 1220 contains a variety of client-server type relationships depending upon the underlying sensor protocol. All such relationships are encapsulated within the associated sensor proxy. This is an interface over which we have very little control since we have to adhere to the capabilities of the underlying sensors.

[00357] Within the existing sensor proxies, there are very few third party libraries associated with the individual sensors. Most of the proxies use standard protocols such as SNMP, HTTP, SOAP, Telnet etc. The SPM core libraries can support protocols such as these.

[00358] IP address changes have an effect upon both interfaces 1210 and 1220. However, at the SPM Proxy – SPM interface 1210, these changes are

rather simple to manage since we have complete control. This is less true of the Proxy – Sensor Interface 1220 since we are constrained by the limitations both of the sensors themselves and by the management protocols used by them.

[00359] FIG. 18 shows an example design 1300 of the SPM Proxy to SPM interface with reference to the issues and proposed solutions. There are two SOAP connections between the SPM proxy and the SPM, one for control and command and one for publishing data.

[00360] The initialization sequence can be described as follows. The SPM proxy registers with the server. This registration consists of opening a TCP connection from the SPM proxy to the SPM and sending credentials including the name of the viper server. This TCP connection is held open. Also, this connection is used for sending heartbeats and for receiving commands from the SPM.

[00361] The connection is held open for the duration of the session. The SPM Proxy opens a second connection used for publishing data. This connection also carries SOAP messages and is held open. Also, the SPM subscribes to receive messages from this channel. The SPM Proxy periodically sends heartbeats to the SPM over the SOAP connection. The SPM expects to receive these heartbeats periodically and, if it fails to do so, it closes the connection and waits for the SPM proxy to reconnect. The SPM Proxy detects the connection going away by the heartbeat failing (or otherwise.) A communications alarm is generated and is receivable at the GUI (not shown in the diagram.)

[00362] There are situations where the IP address of the SPM proxy has been

reconfigured by DHCP. For example, the SPM Proxy obtains its new IP address using calls to the operating system. Also, the registration sequence restarts from the beginning. This means that a new TCP connection is opened, thus again allowing heartbeats to be sent and commands to be received. The publishing connection is also reopened. This means that the SPM Proxy has the responsibility for recovery in the event of a change in IP address or of a network failure. The SPM is the server and does not keep a reference to where the SPM proxy resides. It accepts requests from the SPM Proxy (client.) The SPM proxy does keep a reference to the address of the SPM.

[00363] The SOAP connection used for registration and heartbeats is also used to send commands. Commands are used for configuration and control. Commands are usually sent from the SPM to the SPM Proxy. E.g. from a Policy Engine situated at the SPM to a Sensor Proxy situated at the SPM Proxy. However, there is nothing in the architecture which would prevent commands traveling in the opposite direction.

[00364] FIG. 19 shows an example of publishing data from SPM Proxy to SPM. The "Publish" connection is made as a part of the initialization outlined in the last section. When the connection is established, the SPM subscribes to receive messages. Each Sensor Proxy publishes data to a set of channels. Channels exist for data, alerts, communications alarms and equipment alarms.

[00365] The pattern for publishing data from SPM Proxy to SPM follows the patterns for publishing over any other kind of channel (JMS, local.) When the IP address changes, the SPM Proxy will detect this (see last section) and reinitialize

the SOAP channel so that the connection is closed and re-opened using the modified IP address.

[00366] Proxy – Sensor Interface

[00367] The Proxy – Sensor interface is different to the interface described and has its own set of issues. Following examples delineate the two main ones so far identified.

[00368] Example 1 represents an IP address of SPM proxy changes. For a simple case of a seismic sensor proxy, this device sends SNMP traps to the SPM proxy when there is an error condition or change of state. In order to do this, the sensor is configured with the IP address of the SPM proxy as one of its associated SNMP Managers.

[00369] When the IP address of the SPM proxy changes, the traps from the seismic sensor will no longer arrive, because the configuration at the sensor has now become invalid. If the sensor supports remote configuration, then logic can be built into the sensor proxy to reconfigure the sensor with the correct address. If the sensor does not support remote configuration, a communications alarm can be sent to inform the System Administrator to reconfigure the sensor manually.

[00370] Example 2 represents IP Address of a physical sensor changes. A Tivella device has its IP address changed by DHCP. Since the sensor proxy is configured to send commands to the Tivella at a known address, this means that subsequent commands will fail. The SPM Proxy will notice that the Tivella can no longer be accessed from it because it sends regular polls and these polls will now fail. The Sensor Proxy will send a communications alarm to the GUI. When

the Systems Administrator sees the communications alarm, he can reconfigure the sensor proxy remotely with the new IP address.

[00371] SPM –Architecture

[00372] The main players (objects) are identified in the SPM that are visible to the user and to flush out issues that we need to address. In addition, integration of GUI and SOAP interfaces are described.

[00373] FIG. 20 is a diagram showing an example class model 1500. Each class in the model 1500 represents a logical object residing in the SPM architecture rather than a discrete class. All classes are modeled in various ways within the new compiler language.

[00374] Block 1510 represents a containment of channels within message servers with a multiplicity of 1-n. Block 1520 represents a dependency of policy instance on registers. Block 1530 represents an association which is a looser relationship than either containment or dependency. Block 1530 can be used to model indirect dependencies in the diagram.

[00375] Message Servers

[00376] FIG. 21 shows an example Viper sever 1600. The Viper server 1600 can include a set of policies (policy instances), sensor proxies (instances), message servers (for publishing data form sensors) and a SOAP command architecture (for issuing commands from policies to sensor proxies and between different sensor proxies.)

[00377] Each Viper server contains one or more message servers. Message servers can be of type JMS, SOAP or local. JMS and SOAP message servers

have an associated hostname and port number. For JMS, this specifies a JMS server. For SOAP, the server is a SOAP end point which exists as a SOAP server running on another Viper instance.

[00378] Local channels allow data to be published on-process from a policy to a sensor proxy or between sensor proxies. Message servers are the owners of channels. Channels are the pipes used to communicate published data between policies and sensor proxies. Each channel is associated with an SDO (Service Data Object) based message which specifies the data field that the channel carries.

[00379] Messages are SDO-based objects used to specify internally what the data on a channel consists of. Since Policy instances and sensor proxy instances use channels and so have an indirect relationship with messages, sanity checking is required to ensure that the channels assigned to a sensor proxy or policy are have the correct message type.

[00380] Sensor proxy templates specify the configuration properties (SDO-based) that a sensor proxy has and allows default values to be set on them. Sensor proxies support two types of publishing, namely publishing sensor data and publishing alerts. The sensor proxy template provides the optional message type for both of these (optional because not all proxies publish data and/or alerts.) There is a one sensor proxy template for each sensor proxy type, such as Itrans and Arecont.

[00381] Sensor Proxy Instances contain the instance variables for a particular sensor proxy. This includes the configuration properties, the message types and

the data and alert channels. Also, sensor proxies can contain filters which consist of a condition template.

[00382] Policy templates contain one of each of the following objects: Condition template, Action template, and Condition Template. The condition template contains the information relating to the condition that the policy evaluates. Additionally sensor proxies use a condition template in order to evaluate filters. Condition templates specify: Channel inputs and Registers used in evaluations.

[00383] Action Template contains the action list executed when the condition in the policy is passed. Actions are typically to publish or to send a command to a sensor proxy. A policy instance is a policy template plus a specific data binding. A data binding is a channel plus an SDO property name, in its simplest form.

[00384] Registers are simple objects that hold state information. Registers can be constant or updatable. Most of the registers we currently have hold a single variable.

[00385] Sensor proxies can receive commands from policies or from other sensor proxies. The source and destination do not need to reside in the same process. The command architecture is provided as a part of the sensor proxy framework. A command consists of the distinguished name of the sensor proxy and an SDO. The SDO must contain some combination of the configuration properties of the target sensor proxy.

[00386] The sensor proxy is looked for on the local process. If the sensor proxy is not found, then a table lookup is carried out and a SOAP command is issued to the Viper Service which has the sensor proxy.

[00387] The fact that an item appears here does not denote any decision to act upon it for the first release. This is to be decided. The model/architecture described in this specification is designed to reduce/avoid confusion relating to what objects have to appear inside a Viper server and what ones appear outside and what effect this has.

[00388] In addition, for a Message Server, a user can specify JMS, local and SOAP message servers. However, these use selection can be automated.

[00389] Sensor proxies can be enhanced to also support an equipment alarm type of channel which publishes data when there are problems with the sensor itself. The compiler can be enhanced to support input channels for sensor proxies.

[00390] For issuing commands to sensor proxies, a section can be added for command messages (which would be SDO-based.) Such addition can be used to specify fonts, display modes, colors etc. for sensor proxies such as Inova.

[00391] For the Policy Engine, functions for conditions are implemented as one giant switch statement in the evaluate visitor. The architecture can be revised for this feature.

[00392] The register objects can be removed in some implementations. Instead of using the registers, state information can be held in the policy instances themselves. Updatable registers are notified using channels (local channels where the usage is local to a single Viper Server.)

[00393] SPML can be supported at the GUI interface as a temporary measure or using a GUI of sufficient sophistication. SPML is only needed at the GUI for

(1) Action Template– a component of Policy Template; and (2) Condition Template – a component of Policy Template but also used for filtering by the sensor proxies.

[00394] FIG. 22 shows an example diagram for generating a new policy template from condition and action templates.

[00395] FIG. 23 shows an example use case where access is granted.

[00396] FIG. 24 shows an example use case where a broken glass alarm is triggered.

[00397] FIG. 25 shows an example use case of stolen equipment.

[00398] FIG. 26 shows an example use case of seismic event detection.

[00399] FIG. 27 shows example class diagrams. ViaInputQueue class 2210 is used as a buffer for all the data that have been published on the channels a server has subscribed to. This is a singleton class. The messages are sorted based on the associated timestamp. The ViaInputQueue class 2210 is the observed (as in the Observer pattern) for ViaChannelData.

[00400] The ViaPolicyInstance 2220 class is the representation of an active policy instance that is a candidate for execution. All the active policies register as observers to the ViaInputQueue class 2210.

[00401] ViaChannelProxy class 2230 is a class used as a wrapper of the channel for publishing and subscribing to JMS topics. It will register to its associated channel on behalf of the server and will be notified by JMS (onMessage()) when a message is published on the channel.

[00402] ViaChannelData class 2240 is the representation of a single data

message published on a channel. This class acts as the observed for the ViaPolicyInstance class 2220.

[00403] FIG. 28 shows an example data and control flow 2300. The input channel proxies 2310 are used to subscribe to channels. These input channels are collected from all the active policies. All the channels referenced in the condition parts of the policies include corresponding input channel proxies 2310. The action part of a policy will have channels referenced as input or output. All the input channels in the actions will be added to the input channel proxies 1310. The output channels references in the active policies are turned into output channel proxies 2310. The list of input ViaChannelProxy objects is reproduced in the list of ViaChannelData objects 2330. The purpose of the list of ViaChannelData object 2330 is to act as actual input data for the policies.

[00404] When an input channel proxy 2310 is notified by JMS of the arrival of a new message it immediately creates a ViaChannelData object 2330 and push it into the ViaInputQueue 2320. The ViaChannelData 2330 is just a ViaData (holding actual data) with a mark of its origin (a pointer to a ViaChannelProxy). The reason for the existence of the ViaInputQueue is to guarantee the proper sorting of the data based on the timestamp. Some messages may reach the server out of order because of network latency. As long as there is at least one element in the queue the ViaInputQueue object 2320 continuously tries to push the data out of the top of the queue into the appropriate ViaChannelData objects 2330 that have registered as observers. If the ViaChannelData object it is trying to push the data to has a read-lock it will hold off (see below how the

ViaChannelData gets read-locked by a policy instance).

[00405] An active policy instance 2340 registers (as an observer) all the ViaChannelData objects representing its input channels. When ViaChannelData gets refreshed (a new message arrived) it notifies all the policy instances (by calling update()). A policy does not trigger unless all the input channels have data available and the condition part is filled. The validation of the condition as well as checking of the data availability happens in a separate thread for each policy. The same thread will execute the action part of the policy.

[00406] All the corresponding input ViaChannelData objects are read-locked for the duration of the policy execution so that they cannot change which the execution is in progress. Multiple policies can read-lock a ViaChannelData object at the same time. Only when all the policies depending on a data object have been completed the object can be refreshed (if there is any new data available).

[00407] The execution of the action part of a policy may result in the publication of data on the output channels. This is done by calling the publish() method of the corresponding ViaChannelProxy objects. The channel proxy will do the actual JMS publication. Even if the same channel proxy is used as an input channel the data will not become available locally. It will be first published to JMS and then made available through the general mechanism.

[00408] FIG. 29 shows an example sequence diagram.

[00409] PowerPoint on SPM Sensor Policy Manager.

[00410] FIG. 30 shows an example sensor proxy framework 2500. The Sensor

Proxy Framework is designed to allow the user to develop new sensor proxies very rapidly by establishing some standard patterns and hence allowing for a very high reuse of the existing code. For example, each sensor proxy communicates with an individual sensor or mediator and normalizes the data received. Also, data is published via channels through JMS, SOAP or locally. JMS is typically used to publish to an application server. SOAP is used to navigate through a firewall or where a high performance is required.

[00411] In addition, the SPM supports a completely distributed architecture. JMS and SOAP output channels can be inputs to other SPMs. The normalized format of all messages is standards-based. This format is SensorML for regular data and CAP1.1 for alerts and alarms. Each Sensor Proxy can publish to four kinds of channel: namely data, alert, communications alarm and equipment alarm. The writer of a new sensor proxy is isolated from the details of this. including the underlying protocol supported by the channel. Any sensor proxy can be configured to have one or more of any of these channel types. The messages passed through each channel are configurable. These message types are configured so as to provide meaningful information about the underlying sensor. A special support is provided for sensors with many heads or for mediators where many different kinds of sensor are supported. Each of the heads has its own set of channels. Simple filters may also be configured for a sensor proxy so that only data satisfying certain criteria is forwarded on a channel. There is an in-built support for a wide range of IP-based protocols in the sensor proxy framework. This means that the writer of a new sensor proxy is

isolated from the underlying protocol and can concentrate on the customization required for the particular sensor proxy rather than the transport. Each sensor proxy also has the capability to receive commands from the policy engine or from other sensor proxies. These commands can be local (on-process) or remote (forwarded to another SPM over SOAP.) There are a set of tools built into the sensor proxy framework that assist the developer in writing a new sensor proxy. Examples are a token parser for binary protocols and a generic XML parser which converts an arbitrary XML document into a normalized format for use within the SPM. Each Sensor Proxy supports a simple simulation mode where the proxy forwards data based upon its internal configuration instead of establishing communications with a real sensor and sending live data.

[00412] FIG. 31 shows an example sensor proxy framework for protocol support 2600. The sensor proxy framework 2600 provides base classes which support a wide range of protocols. The writer of a new sensor proxy derives from one of these classes, which effectively allows him to concentrate on the customizations that are required for the particular sensor. The workings of the underlying transport or application layer protocol are automatically taken care of by the framework. The protocols supported by the framework 2600 include the following: (1) TCP client and server; (2) UDP; (3) HTTP client and server; (4) SOAP client and server; (5) Telnet client; (6) SNMP Manager and Agent; and (7) XML-RPC.

[00413] Sensor Proxy Framework

[00414] The protocols supported by the sensor proxy framework include: TCP

Client; TCP Server; UDP; Telnet Client; HTTP Post Client; HTTP Post Server; HTTP Get Client; SNMP Manager; SNMP Agent; SOAP; TCP Modbus; and various proprietary protocols. All of these protocols belong to the TCP/IP suite of protocols, or are application layers running aboveTCP/IP.

[00415] FIG. 32 is a diagram showing a simplified example representation of a SPM communications infrastructure 2700. The components of the SPM include a policy engine 2710 a sensor proxy framework 2720 and a channel architecture 2730. These components are autonomous subsystems. The policy engine 2710 and the sensor proxy framework 2720 are able to publish asynchronous data by means of the channel architecture 2730. The channel architecture includes different kinds of channel, such as JMS, SOAP and local channels.

[00416] Local channels are used when the source and end points of the channel are both on process. SOAP channels are typically used where the footprint of the process is important and JMS is not supported.

[00417] In addition to publishing, there is a command architecture which allows the Policy Engine 2710 to send commands to individual sensor proxies. The command architecture is a part of the sensor proxy framework 2720.

[00418] The command is forwarded locally if the target sensor proxy is on-process or remotely over SOAP if it is running on a remote process. This is transparent to the Policy Engine 2710. Sensor proxies are also able to use this command architecture to send commands to each other

[00419] FIG. 33 shows example component dependencies 2800. The sensor proxy framework and the policy engine are autonomous subsystems within the

SPM. The sensor proxy framework has no direct dependencies upon the policy engine. The policy engine has a single dependency upon the sensor proxy façade to issue commands to individual sensor proxies. The sensor proxy façade is the only interface into the sensor proxy framework that should be used by any component outside of the framework itself. The only functions required are initialization of the framework, shutdown of the framework and the issuing of commands to individual sensor proxies. The sensor proxy façade is the only SPM component that has a direct dependency upon the individual sensor proxies. Also, dynamic libraries for the individual sensor proxies can be generated and loaded dynamically. Once the dynamic libraries are implemented, this component dependency can be eliminated. There is a separate project (package) for every individual sensor proxy. Individual sensor proxies have no compile time dependencies between each other. In most cases, there are no run-time dependencies either. There is an exception to this for composite and single-headed sensors. The ViaSensorProxy component is the heart of the sensor proxy framework, containing the base classes and utilities which are used in order to write new sensor proxies. Both ViaSensorProxy and the policy engine have dependencies on the channel architecture, and use channels to publish data. ViaServiceDataObject is a metadata library, loosely based upon the "Service Data Object" specification. ViaServiceDataObject can be used with the sensor proxy framework but can also used by other components (not shown in the diagram.) ViaAce is a wrapper around the ACE framework. ACE is a cross-platform toolkit used for managing threads and sockets. ViaAce is used by

several other components besides ViaSensorProxy (not shown in the diagram.)

[00420] TCP/IP Communications Stack

[00421] The sensor proxy framework supports sensors that are IP-enabled. The sensor proxy framework does not support serial interfaces such as RS-232 and RS-485.

[00422] FIG. 34 is a table showing different layers of an example TCP/IP Stack 2900. The first column describes the name of different protocol layers. The second column shows the TCP/IP mappings and framework usage of each protocol layer. The third column shows the services corresponding to each protocol layer. Each protocol layer uses the services of the layer directly below it. The first three protocol layers are encapsulated by the transport layer within the programming API. The sensor proxy framework deals only with the transport and application layers.

[00423] Connection-Oriented and Connectionless Transport Protocols

[00424] The usage of some of the tools within the sensor proxy framework depends upon whether the underlying protocols are connection-oriented or connectionless. A connection-oriented protocol is defined as a protocol where a connection has to be established between a client and a server before an exchange of data can occur. When the data interchange is complete, the connection is closed.

[00425] A connectionless protocol is defined as one where no connection has to be established and data can be exchanged at any time. Where connectionless protocols are used, the distinction between client and server are

less clear-cut than in the connection-oriented case.

[00426] The Ethernet and IP layers are connectionless. The transport layer offers a choice between a connection-oriented and connectionless protocol.

[00427] TCP is the connection-oriented stream protocol. When a connection is established, the ordering of data sent and received is guaranteed as is delivery at the receipt point. Data packets can be concatenated or segmented. UDP is the connectionless datagram protocol. The ordering of packets received is not defined and some packets may be lost altogether. Packets are always discrete (i.e. there is no concatenation or segmentation.)

[00428] HTTP is a hybrid between the two, but for most purposes it can be treated as connectionless within the sensor-proxy framework. HTTP runs above TCP (connection-oriented), but after establishing a connection, there is a single data exchange and then the connection is closed. This means that the transmission of each HTTP Request is stateless (unlike raw TCP but similar to UDP.)

[00429] FIG. 35 shows an example protocol class model 3000. The main task in writing a new sensor proxy is to derive and write a protocol class. All protocol classes inherit from one of the base classes shown in FIG. 35. These protocol classes protect an engineer from having to deal directly with the underlying transport protocol used by the sensor proxies. Also, support for standard application protocols, such as Telnet and SNMP, are provided within these generic protocol objects. SNMP is discussed further below.

[00430] The base class, ViaSensorProxyProtocol provides the sole interface to

the channel architecture, the command architecture and basic filter logic.

[00431] Using the Protocol Classes to write a Sensor Proxy

[00432] The first step is to identify the underlying transport protocol to be used by the new sensor proxy and use the table in FIG. 36 to decide on the inheritance. A Sensor Array consists of a set of similar sensors that share a resource (such as a TCP Connection) and so have a common polling sensor proxy. For example, the Itrans gas sensor is double-headed with two gas detectors, one of which detects carbon monoxide and the other methane. Each of the heads is represented by a class derived from ViaSensorProxySingleHeadProtocol which depends for its data feed on an underlying polling sensor proxy.

[00433] A Sensor Group consists of a set of dissimilar sensors that do not share resources, but have some logical dependencies upon each other so that a higher-level (compound) sensor proxy manages them as a group. For example, a trigger to a compound sensor proxy requires it to read GPS and compass data, move a camera to point in the right direction and start acquiring and publishing photographs.

[00434] FIG. 37 shows an example ViaSensorProxyProtocol base class 3200. The ViaAceSocketThreadAction class is included here only because it provides some of the virtual methods that the programmer must override. All of the methods are described in the “C++” header files themselves.

[00435] FIG. 38 is a table showing example ViaAceSocketThread methods 3300. The table is provided for reference purposes only.

[00436] FIG. 39 is a table showing example ViaSensorProxyProtocol methods 3400.

[00437] Writing a Sensor Proxy using the Protocol Classes

[00438] The virtual methods to override depend upon the underlying transport protocol. A Sensor Proxy base class is selected to derive from based upon the underlying transport protocol. If the selected protocol is not supported or some custom behavior is needed, then the system inherits from ViaSensorProxyProtocol and overrides the Execute() method. The system performs the Override initialize() method if any extra initialization steps are needed prior to entering the main loop. The getType method is always overridden. A new enumerated type is added to the ViaSensorProxyEnums in the header. The Clone() pure virtual constructor method is also overridden. The acceptRunValues() is the virtual method used to load values from the configuration object SDO. If the sensor proxy can receive commands then the acceptUpdate() method is overridden. The configuration file is set up.

[00439] In addition, certain methods are overridden depending upon the underlying transport protocol.

[00440] TCP Client:

[00441] Sensors may be active or passive. For both types of sensors, the receiveFromServer() method is overriden. This is used to process incoming data. If the sensor is active (requires polling) then also the sendPoll() method is also overridden.

[00442] TCP Server:

[00443] The receiveFromClient() method and the sendPoll() method are overridden if the sensor is passive.

[00444] HTTP Post Client:

[00445] The sendPost() method is overridden if extra actions are required. By default, this method sends the post string to the HTTP server once every poll period. The initializePostString() is also overridden. This sets up the HTTP post string based upon the configuration file. The receiveFromServer() method is overridden to process the HTTP response.

[00446] HTTP Get Client:

[00447] HTTP Get strings take the general form: hostname:port/applet?parm1;parm2;parm3 etc. The pure virtual method constructURL() is also overridden. This constructs the HTTP Get string based upon entries in the configuration file.

[00448] Telnet Client:

[00449] The Telnet Client is handled in the same way as for the TCP Clients.

[00450] FIG 40 shows an example Service Data Object Class Model 3500. Service Data Objects (SDOs) are used within the sensor proxy framework as metadata in two different ways. Firstly, SDOs are used to describe the messages that are sent out on data channels and on alert channels, and used to publish data internally within the process. At the process boundary, SDOs are converted to SensorML.

[00451] Secondly, the configuration of each sensor proxy instance is described by anSDO. The ViaSdoDataObject class uses the composite design pattern and

so has the capability to contain a collection of ViaSdoDataObjects to any depth. ViaSdoDataObject also contains a collection of ViaSdoProperty which can be any one of the supported types shown in the diagram. ViaSdoProperty can contain a single value or an array of values. Where there is an array, all of the values can be extracted into a ViaSdoPropertyList object.

[00452] FIG. 41 shows an example data object class 3600 and an example property class 3610.

[00453] FIG. 42 shows example Observer Design Pattern classes. Essentially, a model observer can register interest in a model object without publishing its own interface to the model object. When the changed method is invoked on the model object, either by the model object itself or externally, then the virtual method updateFromModel() is invoked on each of the model objects registered observers. The updateFromModel() method is overridden for each observer so as to get the information it needs from the model and update itself based upon the change.

[00454] This pattern is used where the model objects should not know much about who their observers are. All sensor proxy protocol objects are both models and observers and the observer pattern itself is used in several different ways within the sensor proxy framework.

[00455] Example of a Multi-headed sensor proxy: Itrans

[00456] The Itrans sensor is a gas sensor which has two heads. One head tests for Carbon Monoxide: the other for Methane. However, the two heads are not entirely separate since they share a single TCP connection.

[00457] If modeling as a single sensor proxy, each head will publish to different channels and the proxy has to apply some highly custom logic to decide on what to publish and to where. Suppose we have another sensor with twelve heads. This soon becomes unmanageable.

[00458] If modeling as two separate proxies, unwanted coupling between the two are introduced in order to share the TCP socket. The solution is to model an underlying polling sensor proxy that establishes the TCP connection and retrieves the data. This is a standard TCP client type sensor proxy, except that it does not publish to channels. Two single-head sensor proxies are modeled, which set up observer relationships with the TCP client proxy and each has their own channels. The single-head sensor proxies apply standard filters which filter out data not for them. Both filters are notified by the TCP client when any data is received. When the single-head sensors receive data that passes their internal filters they publish to their own channels.

[00459] Unlike almost every other type of sensor proxy, single-headed sensors are not run on their own threads. They receive notifications on the thread of their model object (the TCP client) and action the data synchronously. Single-headed sensor proxies have a do-nothing execute loop which exits immediately, since they depend upon the Execute loop of the associated model object.

[00460] To create a single-headed sensor proxy, the system can inherit from ViaSensorProxySingleHeadProtocol. In addition, the following methods are overridden: getType(); acceptRunValues(); Clone() and updateFromModel(). If the data passes the filter, the data is published. Some single-headed sensor

proxies also have to transform the data here.

[00461] XML Configuration Files

[00462] An XML configuration file is used to specify what sensor proxies to run and which what parameters. Configuration files corresponding to the Itrans sensor proxy are described to illustrate the usage.

[00463] In order to create a new XML tag for use in the configuration file, the XML tag is added to the XML schema, as a part of the subSectionEntry "choice" as shown in FIG. 43.

[00464] FIG. 44 shows an example sensor proxy type section 3900 in a configuration file. The sensor proxy type section 3900 is a required section in a configuration file. The sensor proxy type section 3900 contains the names of all sensor proxy types to be run within the application.

[00465] ITrans is the type name for the Itrans TCP client sensor proxy. ITransChannel is the type name for Itrans single-headed sensor proxies.

[00466] Message Type

[00467] FIG. 45 shows an example message type section 40000 in a configuration file. Any message types used by the sensor proxies must be specified in the configuration file. For example, Itrans uses GasMessage as its data type.

[00468] FIG. 46 shows an example sensor proxy entry 4100. The polling rate is in seconds. The IP address and port specify the socket to which the client opens a connection. The sensor name is the name which is used to address this sensor proxy for sending commands. The enabled flag is standard. For

example, 0 = disabled, I = enabled, and 2 = suspended. The data message type and alert message type names must match to entries in the Message Types section.

[00469] FIG. 47 shows example entries for the two single-head sensor proxies. The filter values are applied by these sensor proxies to evaluate whether the data passed in is for them. The "SP_Model" tag gives the name of the TCP client sensor proxy with which these register with to set up the observer relationship. Both of these proxies are configured to publish to a JMS channel.

[00470] Adding a new sensor proxy to the application

[00471] The sensor proxy static libraries can be converted to DLLs and dynamically loaded. In such instances, integration of new sensor proxies is implemented using the configuration file.

[00472] When not converting to DLLs, a new enumerated value is added to ViaSensorProxyEnums as the type of the new sensor proxy. The new type is added to the ViaSensorProxyNameConverter class. This associates the enumerated values with the strings used in the XML configuration file. In the sensor proxy façade, the header is included for the new sensor proxy type in the source file and the new sensor proxy type is registered in SensorProxyFacade::registerProxies(). In addition, the include paths are updated for ViaSensorProxyFacade to find the header file for the new sensor proxy. Further, the configuration file is updated.

[00473] SNMP Sensor Proxies

[00474] SNMP Manager (Trap Collector)

[00475] The SNMP Manager is a specialized sensor proxy that binds to the SNMP Trap Port (UDP 162.) Only one SNMP Manager can run on a particular station, but it is able to receive SNMP traps from any number of sensors.

[00476] The processing of these traps is specific to the kind of sensor that originated them and so the scenario here is one sensor proxy that receives the data from many sources and individual sensor proxies that transform and publish the data. In other words, this is another situation where the use of single-headed sensors is appropriate.

[00477] The SPM supports at least two kinds of device that send SNMP traps, such as a UPS sensor and a Seismic sensor. Hence, a total of three SNMP sensor proxies are provided that receive traps.

[00478] An example SNMP sensor proxy is the SNMP Manager that binds to the SNMP trap socket and receives the traps. Other examples include the UPS and Seismic sensor proxies which are both single-headed sensor proxies and register interest with the SNMP Manager. As the number of supported SNMP devices grows, many more of these single-headed sensor proxies may be available.

[00479] SNMP Agent (Trap Generator)

[00480] The SNMP Agent does no polling and receives no data from outside of the process. Nor does it publish any data. The SNMP agent is essentially a "Command" type of proxy which waits to receive a command from the policy engine or from another sensor proxy. The command specifies the SNMP Trap to be sent. This must be a trap that is specified in the SPM MIB file. Both the

SNMP Manager and Agent derive from ViaSensorProxyProtocol and use the UDP mode of operation.

[00481] Composite Sensor Proxies

[00482] The composite sensor proxies manage a group of dissimilar sensors that form a logical group, requiring some common processing. Composite sensor proxies do not have any underlying protocols of their own but rely upon the individual proxies comprising the group in order to obtain data from the physical sensors.

[00483] FIG. 48 shows example composite sensor proxies. This is a slightly simplified example of an actual scenario supported by the SPM. On receiving a trigger (command), the Image Acquisition sensor proxy retrieves the latest GPS position and compass bearing from two individual sensor proxies and issues a command to a camera sensor proxy to point the camera in the right direction and begin image acquisition. The Image Acquisition sensor proxy already has the GPS and compass readings required because the Image Acquisition sensor proxy has registered to receive this data via local channels. The Image Acquisition sensor proxy uses the standard command architecture in order to control the camera. By using the standard command architecture and using local channels, there's no need to introduce coupling between these four different sensor types. The GPS and compass sensor proxies have no intelligence about the Image Acquisition proxy. They only publish to channels.

[00484] Similarly, the Camera sensor proxy only has the intelligence to check its internal queue to see if there are any commands. The camera sensor proxy

knows nothing about the source of the commands.

[00485] The Image Acquisition sensor proxy has to understand something about the fields in the normalized messages it receives from the GPS and Compass sensor proxies and the format of the commands it sends to the Camera. The Image Acquisition sensor proxy has no knowledge about the sensor proxies themselves and the underlying protocols are completely hidden from it.

[00486] How to write a Composite Sensor Proxy

[00487] The new composite proxy protocol object inherits from the ViaSensorProxyCompositeProtocol and uses its execute loop. The usual methods are overridden, including getType(); Clone(); Initialize(); and acceptRunValues(). Composite proxies are not associated with a real protocol. The composite proxies receive there data by command and by input channel. In addition, the composite proxies override the following methods: onReceiveData() that is invoked when data from an input channel is received; and onReceiveCommand() that is invoked when a command is received from the policy engine or from another sensor proxy.

[00488] Registering interest in local channels is a configuration option requiring no coding. FIG. 49 shows an example configuration section in XML. The "localInputChannel" names must match the names of the local channels that the compass and GPS sensor proxies publish to which is also specified in the configuration.

[00489] Sensor Proxy Framework – Utilities

[00490] ViaSensorProxyPacketAssembler is used for proxies with connection-oriented protocols and is used to undo packet concatenation and segmentation. For example, the physical sensor can be a TCP server. The TCP server sends data asynchronously as a response to polls. However, responses get concatenated together so that several responses can be sent as a part of a single TCP packet. Also, the beginning and end of the TCP packet cannot be assumed to align with the beginning and end of responses from the sensor. Each response starts with an ACK character (hex 6) or a NAK character (hex 15) and ends with an ETX character (hex 3.)

[00491] The sensor proxy physically contains a packet assembler object with a construction as shown in FIG. 50. When the sensor proxy receives data from the sensor, the receiveFromServer method is invoked. The data received is not processed by the proxy but passed directly to the packet assembler as shown in FIG. 51.

[00492] Each time the packet assembler parses a completeresponse, it notifies the sensor proxy via the updateFromModel method, which is then able to process it as shown in FIG. 52. This utility is suitable for both binary and ASCII protocols.

[00493] Simple Filters

[00494] Simple filters can be set up within the configuration file. This filter operates on a Service Data Object (SDO), usually the data message SDO or the alert SDO. FIG. 5 3 shows a sample XML. The filter tags in FIG. 53 have the following meanings. FilterAttribute represents the name of the SDO property to

apply the filter to. FilterValue represents the value of the attribute to apply the filter to. FilterMode represents the valid values are AND and OR. FilterType can be 0 = less than; 1 = greater than; or 2 = equality.

[00495] The filter outlined in FIG. 53 provides an AND filter which is passed when the AlarmType field = 4 AND the AlarmSubType = 2. From a sensor proxy, a filter is evaluated with the following line of code:

```
ViaSdoDataObject sdo;
...
bool isPass = m_filterMgr.passFIlter(sdo);
```

[00496] If the sensor proxy is not configured with a filter then this will return "true." In other words, a null filter is always passed. If one or more of the SDO properties is not present in the SDO input parameter, then this statement will return "true."

[00497] Sensor Proxy Group Manager

[00498] The Sensor Proxy Group Manage is a utility that allows sensor proxies of the same type to be assigned to groups. The utility is driven from the configuration file. FIG. 54 shows an example XML configuration file section. Each sensor group is assigned a group name and a type. The type must match one of the entries in the "SensorProxyTypes" section of the configuration file. The Sensor names must match to corresponding names in the sections where the sensor proxies themselves are specified. No coding effort is required to load the sensor groups.

[00499] FIG. 55 shows an example code fragment. The code fragment shows

how to retrieve sensor groups from the Group Manager Singleton object.

[00500] FIG. 56 shows an example SPM / SPM Edge Configuration 5100. Each instance of the SPM Edge runs on a separate work station and managed a set of sensors using the sensor proxy framework. The SPM server has the capability of managing sensors directly, but for this configuration, the SPM server runs only the policy engine and acts as a hub. The arrows represent a two-way communication between the SPM server and SPM Edge processes.

[00501] Publishing Data from the SPM Edge to the SPM server

[00502] Each of the sensor proxies running on the SPM Edge is configured to publish data to SOAP channels. This is a configuration option. On the SPM server, one instance of the SPM Edge Server sensor proxy is run. This receives the SOAP packets from the SPM Edge and republishes to JMS channels or local channels.

[00503] Sending commands from the SPM server to the SPM Edge

[00504] The SPM server runs one instance of the SPM Edge Command sensor proxy. When the SPM server needs to send a command to a sensor proxy, the command consists of the name of the sensor proxy and an SDO specifying the command details. The command is passed to the sensor proxy framework. If the sensor proxy is running locally on the SPM server, the command is passed to the proxy directly. Otherwise, the command is passed to the SPM Edge Command sensor proxy. This proxy holds a table of sensor proxy names together with the SOAP end point for each (contained in the configuration file.) The Command proxy looks up the SOAP end point for the sensor proxy name it

has been given and sends the command to the SPM edge.

[00505] The SPM has the capability to manage and interact with other sensor management systems such as the Richards-Zeta mediator and Lenel On-Guard. Using the two SPM Edge sensor proxies, SPM Server is able to interact and manage a remote SPM Edge processes in much the same way. The only two differences are that the publishing and command patterns have been separated, and that the SPM Edge proxies are more flexible in that SPM server and SPM Edge processes are interchangeable in the interactions that are possible.

[00506] Sensor Policy Management Introduction

[00507] Sensor Policy Manager (SPM) is a network-centric, policy-driven framework that provides sensor, video, and communication interoperability among disparate network-connected sensors. SPM processes data from a diverse assortment of sensors and cameras, converting data from each sensor into messages in a common language. A message from a given sensor that has been brought into the SPM system via a sensor adapter can be used as a trigger to communicate with other sensors, initiate actions in other sensors, or publish messages, alerts and warnings to concerned parties.

[00508] The processing of the data happens automatically according to policies that have been tailored for the particular sensors in the SPM system. A policy is a condition-action pair. When a certain event occurs, appropriate actions are taken and the corresponding messages are automatically published via a number of different communications channels. Actions may consist of commands to instruments to perform a certain task or alerts dispatched via phone messaging,

e-mail, video displays and the like. This automated implementation is intended to reduce or even eliminate the need for on-site human intervention, thus minimizing human error. Policy-based data interpretation is crucial where humans are not or cannot be present.

[00509] SPM Architecture and Main Components

[00510] FIG. 57 shows an example SPM architechture 5200. The sensor adapters 5210 provide a complete software wrapper around the physical sensor 5212. The function of the wrapper is to ensure that diverse sensor data and sensor message blocks are converted into messages whose formats can be understood by downstream processes in the policy engine (Viper) 5214. Similarly, a sensor adapter can be used to filter the data stream so that only information of interest is forwarded through the messaging channels 5216.

[00511] The creation of new sensor adapters or the modification of existing ones is typically not a task for the normal user, although SPM does include tools which allow administrative users to create and modify policies. Creation of a totally new sensor adapter often entails the addition of new look-up tables, algorithms, or formulas, and requires C++ programming experience. Services, documentation, and support for making new sensor adapters are available by contacting ViaLogy Technical Support.

[00512] The core of SPM is the Sensor Policy Engine 5214 known as the Viper (**V**ery H**i**gh Density **P**olicy **E**ncode**r**). Each Viper 5214 contains a number of policies which assume the general form of an 'if' clause (the condition) and a 'then' clause (the action). Policies are constructed as templates that are generic

for each type of sensor 5212. These templates can be customized to incorporate the characteristics of each sensor instance. Condition and action templates are joined to make up a policy template. Since the policy templates provided are generic or at best semi-specific, the system integrator or expert user creates specific instances of the policies to fit the particular real-world application. Specific policy instances are created using the Policy Wizard, or, alternatively, by editing the Viper configuration file. Input messages typically are data from sensors 5212 processed via their sensor adapters or messages which come from previously evoked policy actions. Templates for sensor adapter message types and examples of conditions and actions for different kinds of sensors and devices are available in a separate document, Sensor Adapter Templates.

[00513] Each Viper 5214 typically manages multiple sensor adapters 5210 and policies. Depending on the data and processing load it may be advantageous to distribute the sensor adapters 5210 and policies amongst separate Vipers 5214, each running its own processes. Alternatively, a light-weight implementation of the Viper 5214 called Viper Edge or SPM Edge can be constructed to operate high data rate devices such as video cameras by locating the Edge server in close proximity to the data device. The purpose of SPM Edge is to minimize the load on the network posed by such sensors and to only transmit processed information a majority of the time. Viper/SPM Edge implementations resemble standard Vipers in many respects, but do not incorporate a database 5218.

[00514] The SPM Console 5220 is a Web-based Java application that interacts with the Sensor Policy Client API. The Console allows users to monitor data

streams, results and events visually. Authorized users can log in to the SPM Console from any Internet location and view data and messages allowed by their permission level. The SPM Console 5220 is also used by the System Integrator to add or edit users and their roles and permissions as well as modify Viper configuration files.

[00515] SPM includes an internal database 5218 that stores the configuration and control operations that have been defined in the Viper configuration file. The internal database 5218 can store data and messages from the system for later retrieval via the SPM Console 5220.

[00516] SPM uses the Java Message Service (JMS) API as a messaging standard that allows application components to create, send, receive, and read messages. JMS enables distributed communication by providing asynchronous delivery of data between applications on a "store and forward" basis and is a useful method for time-independent or parallel information processing.

[00517] A particular strength of the SPM architecture is that any number of consumers can subscribe to the same channel in much the same way as tuning into a radio or television station. The multicast feature can easily be implemented without any involvement by the data producer. The multicast feature can be controlled in the SPM configuration file by adding or deleting recipients.

[00518] Another important advantage of using messaging middleware software 5216 is its ability to store data in a persistent repository before delivering it. Because SPM is a distributed network system, oftentimes the connection

between data producers and data consumers can be temporarily broken. If guaranteed delivery is needed for some sensor readings or derived results, the corresponding channels can be configured as 'persistent.' In this way, once the data have been successfully published, they will be available to any subscriber even if that subscriber was not online at the time of the publication.

[00519] SOAP Communication Protocol is a simple XML-based protocol to let applications exchange information over http via the Internet. SOAP is platform- and language-independent that provides a way to communicate between applications running on different operating systems, with different technologies and programming languages. SOAP allows communications through firewalls in a safe and controlled manner. In SPM, SOAP is used to send and retrieve messages and commands between sensors and sensor adaptors, or between different instances of SPM with firewalls in between or running on two different platforms.

[00520] Lightweight Directory Access Protocol (LDAP) is used to store information about SPM users. This information includes user names, passwords, contact information, as well as their assigned roles and permissions.

[00521] Sensor Policy Client API is a JAVA API and provides the sole user interface to the other SPM components. Sensor Policy Client API is used by the SPM Console 5220 and by all communication applications which connect to SPM. Sensor Policy Client API allows clients to configure and access the functionality of the VIPER Policy Engine 5214.

[00522] The Configuration Server 5220 has two primary responsibilities. The

Configuration Server 5220 functions as the Database Proxy and provides the interface to the database for configuration and control operations. All changes to the configuration of SPM made using the SPM Console 5220 are routed through the Configuration Server, which writes the new configurations to the database before forwarding them to their assigned destinations.

[00523] SPM is available in either Linux or Windows XP based server applications utilizing TCP/IP. Normal communications with the SPM server are through SPM's web-based interface on IE6, IE7, or Firefox version 2.0.0.X. It can also be useful to have an SSH communication utility such as PuTTY for installation and monitoring. When used on a single subnet, there are no unusual networking requirements for SPM itself other than those required to support network communication between the primary SPM server, multiple SPM installations if used, the sensors, and the clients. However, when installations span multiple subnets TCP ports 443 (https) and 22 (SSH) should be open between the clients and the SPM server(s). Inter-SPM communications will require that port 1099 be open to support JMS messaging and port 15001 if TCP messaging is used. Other requirements may exist depending on which specific output devices are used.

[00524] SPM connects to any network device, sensor or otherwise, that directly supports TCP/IP or whose output can be converted to TCP/IP. Sensors that support only serial RS-232, RS422 and RS485 output can usually be converted by serial-to-Ethernet bridge products such as those made by Digi System. For instructions on configuring specific setups, users must refer to the bridge

vendor's product information. SPM also supports the MIDAS interface and protocol. Specific MIDAS information can be reviewed at http://midas-nms.sourceforge.net/. See http://www.comptechdoc.org/independent/networking/guide/ for a general review of networking principles and protocols.

[00525] Sample SPM Deployment

[00526] In the example of an SPM deployment shown in FIG. 57, an SPM Server 5202 is integrated with several sensors 5212 in different locations. The main SPM Server (center) 5202 constitutes a centralized site where data are incorporated. Data flows from the sensors 5212 to their respective sensor adapters 5210. From there, the processed data can be sent to policies or directly to a messaging service from where it can be published or subscribed to. The data is available for other internal or external services such as displays, evaluation by policies or database storage. The configuration server 5222, which may reside in a different server than those which execute the policies, can be used to set up and control these other servers.

[00527] SPM User Roles

[00528] A complete tabulation of permissions is contained in Example predefined user roles assigned to permission include System Integrator, Expert User and End User. The System Integrator (SI) is responsible for setting up the Linux Server and installing SPM. The SI configures the SPM system which consists of an initial set of sensor adapters, message types and policies, and links up the sensor adapters to the various sensors and to the computers which

control them. The SI is also able to add new sensor adapter instances to SPM and define user roles.

[00529] The Expert User can edit existing policies and create new ones. In contrast to the SI, the Expert User cannot define user roles or add new users and assign user passwords. The Expert User can exercise all the functions of the End User.

[00530] The End User is restricted to monitoring and observing SPM data and alerts and listing the users and their roles. The End User can also view the wording of sensor adapters instances and policies, but has not editing privilets.

[00531] Accessing SPM

[00532] The SPM Console web application has been designed for optimal viewing on a 1280 x 1024 monitor. The application has been optimized for viewing using Firefox 2.0, for example. Minor inconsistencies in appearance may be experienced when viewed with other browsers.

[00533] The Dashboard is the default landing page after logging in to the console. FIG. 58 shows an example main panel 5300 of the dashboard. Several other panels of useful information can be accessed from the main panel. FIG. 54 shows an example Main Menu panel 5400 used to access the main menu navigation panel. SPM forms and functions can be accessed from this main menu.

[00534] In FIG. 59, the Main Menu panel 5400 includes various action buttons including Session 5404, Manage SPM 5406, Policy 5408, Reporting 5410, Accounts 5412 and Help 5414. Session button is used to access the SPM log-

out option. Logging out returns a user to the main log-in page when desiring to start a new session.

[00535] The Manage SPM button is used to access tools for managing SPM configuration objects. Under the Manage SPM button, other options are available including Sensor Instances, Channels, SPM Configuration and Database Manager. The Sensor Instances button can be used to accesses a listing of all sensor adapters in the SPM installation. The Channels button accesses a listing of all SPM channels and their properties. The SPM Configuration button accesses a listing of SPM configuration parameters and time-out settings. The Database Manager button accesses tools for backing up and restoring SPM databases.

[00536] The Policy button is used to access tools for managing Viper Configuration files. Under the Policy button, other options are available including SPM Policy Wizard and Viper Config Editor. The SPM Policy Wizard is a set of data input forms that step the user through the construction of an SPM policy. These tools are available to System Integrators and Expert Users. The Viper Config Editor represent tools for viewing, editing, loading, compiling and exporting Viper Configuration files and managing SPM policies. System Integrators can start and stop Viper processes from within this window.

[00537] The Reporting button is used to access reporting and monitoring tools. The Reporting button includes other options, such as View Logs, Events and Dashboard. The View Logs is for accessing a listing of log files within a window for viewing them. The Events represent a listing of SPM channels and tools for

viewing messages that have been sent through each channel. The Dashboard includes tools which allow the user to select one or more channels and monitor information being transmitted on those channels.

[00538] Accounts button is used to access tools for setting up users and managing their permissions and roles. The Accounts button includes other options including Corporate Licensing, Manage Roles, Mange Users and User Roles. The Manage Roles include each defined user role that consists of a collection of permissions for viewing different types of information and executing different tasks in SPM. This tool allows users with the appropriate permissions to create and modify user roles. Manage Users is a tool that allows users with administrative permission to assign roles to users and add new SPM users. User Roles provides an overview of the roles and permissions assigned to each user.

[00539] The Help button is used to access various help tools including Open Help Window, Contact Us and About. Open Help Window launches an HTML version of the SPM Software User Guide in a separate browser window. Contact Us provides an e-mail link to ViaLogy Technical Support and other contact information. About provides version information about the SPM software.

[00540] Further, to enhance navigation for a user, Left/right arrows 5416 are provided at the upper left of the Real Time Dashboard toggle display of Main Menu. Also, Up/down arrows 5418 are provided at the bottom right of the Real Time Dashboard toggle display of SPML compiler log information.

[00541] Managing SPM

[00542] SPM can be installed locally on a computer using the installation CD or

network installed on a central Linux server. Also, SPM can be installed and configured on DB2 client and server. Further, managing SPM includes managing sensor adapter instances.

[00543] Minimum Operating System Requirements include the following: Red Hat Enterprise Linux 4 (RHEL4 ES); Red Hat Enterprise Linux 5 (RHEL5); or other comparable operating systems.

[00544] Minimum Hardware Requirements includes the following: 1.5 GHz Linux server; 4 gigabytes memory/RAM; 80GB storage/HDD or other comparable systems.

[00545] Minimum Software Requirements includes the following: autorun Linux executable and OpenLDAP installed on the Linux server. If the application 'autorun' is not available in the operating system software, SPM can be installed manually using the 'Manual Install' instructions below. Alternatively, 'autorun' can be installed to enable a CD-ROM to install SPM automatically.

[00546] OpenLDAP installed on the Linux server can be verified. To determine whether OpenLDAP is installed on the system the following can be performed:

(1) From the Linux command line, enter exactly as written:

which sldap

rpm –qa | grep “ldap”

Note that the pipe symbol is preceded and followed by a space. In the lines returned by the system the user should find:

openldap-servers 2xxxxx

where xxxxx can be any numerals. This message indicates that openldap is

installed on the server.

[00547] The user can define all the hardware resources available for SPM to deploy its software agents. The default hardware resource is the machine the software is deployed on, i.e., the user's machine. Users are able to edit a machine's hardware and network profile.

[00548] If a machine is behind a firewall the user must define how routing is done and what the public interface to the fire-walled machine will be. The user can also define hardware gateways to "declare" their existence to SPM. Most of these gateways are transparent to proxy functionality but once "declared" to SPM, the gateways can also be monitored for health and status information. This will add value to SPM as a tool for monitoring edge gateways and alerting people when those gateways are in failure modes.

[00549] This form allows users to define the hardware and network profiles of each machine that has an SPM Agent Daemon running on it. These are config objects in the system which can be used to help the user design an agent deployment scheme to maximize performance, minimize network bandwidth, ensure security, and device resource utilization. The pull-down is a comprehensive list of device objects. Devices can be machines (physical servers), gateways, and network infrastructure.

[00550] Installing SPM Using the SPM Installation CD

[00551] Automatic installation can be peformed with autorun. If autorun is already installed, the SPM CD-ROM Installer should automatically initiate the autorun processes. To determine whether or not autorun is installed on the

system, use the command:

which autorun

[00552] If autorun is not installed, use the following command to install the autorun package from the Red Hat Network at http://rhn.redhat.com:

Red Hat Enterprise Linux 4 (RHEL4):

up2date autorun

Red Hat Enterprise Linux 5 (RHEL5):

yum install autorun

[00553] Once autorun is installed, ensure that it is running by using the following command.

ps aux | grep -i autorun

[00554] If autorun is not running, autorun can be run by using the following command.

autorun &

[00555] Once autorun is running, the CD-ROM Disc can be inserted into the CD-ROM drive to initiate an install program that begins copying the installer to a /tmp folder. Once the install program has finished copying, the install program will run the binary installer.

[00556] When the binary installer begins, options are presented for the type of installation procedure: 1) New; 2) Update/Restore; 3) Quit; #?

[00557] Installation procedure should be chosen accordingly. If not desiring to install SPM from the autorun installer, "Quit" can be chose to abort the installation, and then manually close the window.

[00558] Manual installation from the CD-ROM

[00559] To install SPM manually, the installation can be run via a GUI/Desktop icon or from the command line. Installing via a GUI/Desktop icon includes opening the SPM CDROM icon on the desktop; double clicking on 'spm-install.sh' and selecting the Run in Terminal option. The installation program copies the installer binary from the CDROM to the /tmp folder. This takes several minutes. Further, the SPM Installation Wizard presents three options. Entering the number of the option allows the installation to complete.

[00560] Installing from the command line inside the CDROM parent directory includes accessing the CD and running the install script:

```
# cd /media/cdrom
# .spm-install.sh
```

Then, proceed as above.

[00561] Installing from the command line outside the CDROM parent directory includes the following. If installing from a directory outside the CDROM parent directory, the path should be appended to the CDROM:

```
# /media/cdrom/spm-install.sh /media/cdrom
```

Then, proceed as above.

[00562] Network Installation of SPM

[00563] When having access to the installer binary on a central server, SPM can be installed remotely from a Windows PC onto the central server (e.g., a Linux server). Before beginning the installation of SPM using the spminstaller-1.1-linux-installer.bin script, it should be confirmed that the system meets the

requirements outlined above and that the requisite software and information are available. In addition, the user needs root permissions on the Linux server to make this installation.

[00564] Additional Requirements for Network Installation includes the following: the correct host name and/or IP address of the Linux server; Secure Copy Protocol (SCP) client software; and Secure Shell (SSH) client software. An example of a free SCP client is WinSCP. If needed, can download it from http://winscp.net. An example of a free SSH client is PuTTY. PuTTy can be download from http://www.chiark.greenend.org.uk.

[00565] FIG. 60 shows an example user interface panel for using a WinSCP Client. If not already done, assure that the /etc/hosts file has the desired server name linked to the server IP address. The SPM install uses the host name to configure the JMS path and will do so automatically if the host definition is already present. An example includes the following:

```
# Do not remove the following line, or various programs
# that require network functionality will fail.
127.000.000.001     localhost.localdomain     localhost
XXX.XXX.XXX.XXX     Viperservername
```

[00566] Copying 'spminstaller-1.1-linux-installer.bin' to the Linux Server

[00567] Using the SCP client shown in FIG. 60, log in to the remote Linux server as root using the server's IP address or hostname. This requires the proper root password. If a warning appears regarding the server's host key, click Yes or No, but not Cancel. Then, copy the spminstaller-1.1-linux-installer.bin file

from the SPM Installation CD (or a directory on a mapped network drive) to any directory on the Linux server.

[00568] FIG. 61 shows an example PuTTY SSH Client Login interface 5600. Installing SPM on the Linux Server includes the following interactions with the login interface 5600. A user can launch PuTTY and accept the default SSH setting. The user enters the hostname or IP address and clicks on Open. If a host key warning appears, the user can click either Yes or No, but not Cancel. To access the Linux server as root, the root password is needed. To navigate to the directory into the spminstaller and spm-uninstall scripts are copied, use the following command:

```
# cd /tmp
```

[00569] The permissions on the spm-install script file are changed using the switches exactly as shown. Until the permissions are changed, no one will be able to execute the install script. Changing the permissions enables the user and only the user to execute the install script. Changing the permission can be performed using the following command:

```
# chmod u+x spminstaller-1.1-linux-installer.bin
```

[00570] The install binary can be run from within the same directory by using the following command:

```
# ./spminstaller-1.1-linux-installer.bin
```

[00571] The install program displays the following on the console:

```
Welcome to the SPM Installation Wizard!
Please choose the index (Number) for the type of SPM Installation
```

1) New

2) Update/Restore

3) Quit

#?

[00572] Selecting New creates a new SPM database and installs the SPM software. This destroys the existing database which cannot be recovered. If desiring to preserve the existing database, the Manage SPM > Database Manager > Database Backup function must be used.

[00573] Update/Restore backs up the existing SPM database, updates the SPM software, and restores the database. If attempting to reinstall SPM over an existing installation, the install script informs the user (after several minutes) that an SPM installation exists. The user must uninstall SPM before proceeding. See the Uninstall instructions below.

[00574] It may take several minutes for the installation to complete. When the installation has finished, reboot the Linux server by using the following command

reboot

[00575] Rebooting may take as long as 15 minutes. Also, the PuTTY session may time out during the installation.

[00576] To check when the server is back up, right-click on the PuTTY title bar and select Restart Session and see if the command prompt has returned. When the server is back up, add the SPM license key file 'spm.license' to the directory /usr/local/spm/license/. To listen for SNMP traps in case a Viper server goes down, edit the processmanager.spm file to change the host name (or host IP

address) of the server that will function as the SNMP, or NMS (Network Management System), server. For more information, see the SNMP and Trap Glossary topics.

[00577] During setup, the SPM installation program will temporarily assign to the SNMP message server the same host name as the server on which SPM is being installed. (The installation program automatically determines this name during installation.) Assume that the hostnames for SPM server and the SNMP server are “mySPM” and “myTRAP”, respectively. The pertinent lines in processmanager.spm at first will be similar to:

```
snmp_message_server SnmpMsgServer {
    host "mySPM":162
    channel ViperEquipmentAlarm
        message_type SnmpEquipAlarmType
}
```

[00578] Change the snmp_message_server statement to read:

```
host "myTRAP":162
channel ViperEquipmentAlarm
    message_type SnmpEquipAlarmType
}
```

[00579] The port number on the SNMP server may be different than “162”. To see the full text of a typical processmanager.spm file, consult Process Manager Configuration File. To make the changes take effect, stop all SPM processes and restart them:

service spmAll stop

service spmAll start

[00580] A user can log into the SPM console using the web browser. A Secure Socket Layer connection is needed:

https://<hostname>/

[00581] When a user first log into the SPM console, admin is used for both username and password. The user can enter the password exactly as written. Navigate to Policy > Viper Config Editor. The user can select a Viper configuration file from the tree and click Edit to confirm that the spm configuration has taken place. If the configuration has taken place, an SPML file will appear in the editing field below.

[00582] Uninstalling SPM on the Linux Server

[00583] To uninstall SPM, follow the steps described above to launch PuTTY and access the Linux server. Then navigate to /usr/local/<spm-install-dir>/uninstaller. This is where the installation program creates the uninstall script. Change the permissions as root on the spm-uninstall script file using the switches exactly as shown. Until the permissions are changed, no one will be able to execute the uninstall script. Changing the permissions enables root and only root to execute the uninstall script.

chmod u+x spm-uninstall.sh

[00584] To run the uninstall script from within the same directory, use the following command:

./spm-uninstall.sh

[00585] The uninstall program displays the following on the console:

Welcome to the SPM Un-Installation Wizard!

Please choose the index (Number) for the type of SPM Uninstall

1) Complete

2) Backup

3) Quit

[00586] Complete is used completely removes SPM. Backup backs up the existing SPM database so that it can be restored during reinstallation. The backup option used when making maintenance upgrades of the SPM software.

[00587] Database Management in Distributed Systems

[00588] SPM uses the IBM DB2 Database. Using the DB2 Express-C Version 9.1 server for Linux 32-bit, the configuration files for SPM as well as the data, alerts, and other information produced can be stored, backups made, and retrieval performed to and from the database.

[00589] DB2 is composed of a DB2 server and a DB2 client component. Whereas the server is pre-configured with SPM and automatically installed when installing SPM, the DB2 client for managing distributed databases should be downloaded.

[00590] Basic installation, configuration, and some essential functions of the DB2 client are described. For a more thorough description of the client or for command line usage, refer to:

http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp.

[00591] Installing the DB2 Client

[00592] The DB2 Client is available for free from IBM's web-site: http://www-306.ibm.com/software/data/db2/express/download.html. After accessing the site, the user can scroll down, and click on Download next to the DB2 Client under the DB2 Development and Administration Tools section. This site requires all users to sign in with IBM IDs and passwords which can be used site-wide. An IBM ID is needed to continue. If an IBM ID is not available, one can be obtained at the IBM registration site (http://tinyurl.com/2llb46).

[00593] After signing in, the user is taken to the DB2 9 Client web-page, where the user must choose the proper version of the client the use wants to use. In this example the client is downloaded for the Windows platform by selecting "DB2 9 client for Windows on 32-bit AMD and Intel systems (x86). Clicking on Continue at the bottom of the page continues the process.

[00594] The license agreement with IBM is confirmed as to the proper use of the DB2 9 Client for Windows. Next, the user is taken to the download page. The user can check the desired software, and click Download now at the bottom of the page. The zipped folder is almost 300MB, so the download may take several hours. The user may wish to download the file overnight to insure that the transmission is not interrupted by other use of the Internet.

[00595] After downloading the software package (db2_v9_nt32_client.zip), the user can highlight the file and select Extract All from the right-click menu. This launches the Extraction Wizard. The user can click Next to continue. In the Extraction Wizard dialog, the user can accept the default location for the unzipped files (or browse to a different location), ignore the Password option and

click Next. When the extraction is complete, the user can accept the defaults and click Finish. If the user has unzipped the package in the same folder where the zipped package resides, the Explorer window may need to be refreshed to make the unzipped folder (db2_v9_nt32_client) appear. The use can navigate down to the ~\Client\image folder and start the client installation by running the setup.exe file. This action launches a web page in the user's default browser entitled DB2 Setup Launchpad. From the menu options, the user can select Install a Product > Install New. The DB2 Setup Wizard - DB2 Client - DB2COPY1 dialog now appears. The user should follow the instructions in the Wizard. For the most part, the user can accept the default options including selecting the typical installation type, since it includes the database administration tools. Also, the user can accept the default Install DB2 Client and save my setting in a response file. If the user does not already have an instance of the DB2 Client installed, the user can accept the default location. If an instance already exists, the user will be prompted to choose whether to work with the existing instance or to create a new one. In the latter case, the user may want to create a new location for this installation.

[00596] In the next dialog, the user can clear the Enable operating system security check box before clicking Next. The Operating System Security option allows users to access the database implicitly using their network login security – i.e., the same login procedure used when accessing a networked computer also provides access to local and to remote databases – for example, Windows Authentication. Implicit login may pose problems if a company's network

administration policy calls for users to periodically change their network passwords, inasmuch as the renewal of database passwords and the network passwords can become desynchronized. Disabling Operating System Security enforces explicit login by each user to the database system, and also provides a better audit trail, since authentication takes place on per user basis rather than on a per machine basis.

[00597] The user can create a FireFox user profile if desired. If the user anticipates that multiple users will use the machine with the DB2 Client installation, each should have his own browser profile. If the user selects this option, the user will be prompted to create a user profile the next time the user launches Firefox as shown in FIG. 62. FIG. 62 shows an example Browser User Profile Selection panel 5700.

[00598] If the user checks the Don't ask at startup option, the dialog in FIG. 62 will not appear the next time the user launches FireFox. The user can re-access this dialog by selecting Start > Run... from the Windows main menu and entering firefox.exe -ProfileManager. Finish set up. A shortcut will be generated in the Programs folder under the Start menu.

[00599] Configuring the DB2 Client

[00600] The one-time configuration of the DB2 client to add the SPM DB2 database is described. To configure the DB2 Administration Tools, select Start > Programs > IBMDB2->DB2COPY1 > General Administration Tools >Control Center. This accesses the Control Center View window. Check the Advanced View button. Click OK. To configure the client to communicate with the

database server on SPM, right-click All systems in the left pane, and choose Add. The user must know the name or IP address of the SPM server in order to complete this form. The user will not be able to use the Discover button to browse for this information as shown in FIG. 63.

[00601] FIG. 63 shows an example panel for Adding an SPM Node to the System. The System name/Host name represents the name or IP address of SPM server. The user can use a name only if the name can be resolved on the same network; otherwise, the user must use an IP address. Node name represents some descriptive name of the SPM being added (not to exceed eight characters). Operating system represents the OS of the target server, such as Linux. Protocol represents protocol to use. The TCP/IP default should be accepted. Comment represents optional description of SPM node (not to exceed 30 characters).

[00602] Click OK to submit the information. The particulars of the SPM Server that the user has just added will appear in the All Systems tree. To associate with the particular database on the SPM, the user now has to add an instance of Database under this system.

[00603] FIG. 63 shows an example user interface panel for associating a DB2 database instance with the SPM Node. Expand the node (+) at the SPM Server. Click on the Instances folder and choose Add from the right-click menu. Enter db2inst2 in the Instance Name field and any valid name not exceeding eight characters in the Instance Node Name field. Operating System is Linux, and Protocol is TCP/IP. The hostname is the name or IP address of SPM, as used

above. Leave the Service name field empty. Enter 3700 for the Port Number. Do not enable TCP/IP SOCKS security. Click OK.

[00604] FIG. 64 shows an example Control Center Explorer with New Database Folder. If the user now expand the Instances node in the Control Center explorer, the user will find the new instance db2inst2. Click on the Databases folder, and add the database by selecting Add from the right-click menu. Enter SPM for the database name and supply a unique alias. Accept the default Authentication type Value in server's DBM configuration. Click OK. The database with the name SPM will now be listed under Databases at the bottom level of the All Systems tree in the Control Center explorer.

[00605] FIG. 65 shows an example Newly Added SPM Database 6000. Expanding the SPM node of the tree displays a listing of the database objects.

[00606] Connecting to and Managing the DB2 Database

[00607] Once the DB2 client has been configured to the SPM DB2 server, the user can connect to it and perform Back-Up and Restore. Every time the client is closed, it will be disconnected from the database. Thus, connecting to the SPM database must be treated as a routine procedure.

[00608] Connecting the DB2 Client to the SPM DB2 Server

[00609] In the Control Center, expand the All Databases folder. Right click on the SPM database symbol and choose the Connect option. The user will be prompted for the User ID and Password. These can be obtained from the SysAdmin. Enter them, then click OK.

[00610] Backing up the SPM DB2 Database

[00611] After connecting to the database, a Backup can be performed. Database backup/restore assumes jboss, DB2, and SPM are installed on a single machine. Highlight the SPM database symbol and select the Backup option from the right-click menu. This launches the Back up Wizard. For the most part the user can accept the default values. Click Next to confirm the details of the database. Typically, the user will want to back up the database to a File System. To select the media where the user will store the backup, click Add and navigate to the desired backup location. This is generally a machine on the network file system other than the SPM server. Note that it may take a few seconds to locate the file system in the Path Browser. Once the user has located the proper directory, the user can click OK to close the Path Browser.

[00612] In the Backup Wizard click Next. Select the Include Log Files In Back Up Image option. (See under 'Availability'; most users would also choose to compress the image, see 'Compression'.) Click Next. Leave the Performance and Schedule functions as is, clicking Next in both cases. Under the Summary window, the user can show the command that will be used for the back-up, and copy this for future command line use of the back-up procedure. Otherwise, just click Finish to invoke the back-up after a few seconds.

[00613] A DB2 Message window will show the status of the back up. If any errors occur, please refer to IBM for help: http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp.

[00614] For routine database maintenance, the user may find the built-in SPM Backup utility more convenient. The user can access this by choosing Main

Menu > Manage SM > Database Manager >Database Backup.

[00615] Restoring the SPM DB2 database

[00616] After the user has connected to the database, the user can now perform the Restore function. Highlight the SPM database symbol and select the Restore option from the right-click menu. This launches the Restore Data Wizard. For the most part the user can accept the default values. The user can click Next to confirm the details of your database. Typically, the user can choose to restore the entire database. Thus, the user should click Next when asked What would you like to restore? The user can select from a list or manually select the database image you are going to restore. The user can use the > button to make that the database to restore. The user can click Next to continue. Also, the user should click Next under Container options. In the Roll forward after restoring options, the user can choose Restore the database and roll forward as follows and then choose Roll forward to the end of logs. The user then clicks Next to continue.

[00617] Under the Final State of the Database options, the user can choose Complete the restore and return to the active state. The user then clicks Next to continue. The user can accept the Options, Performance and Schedule function defaults by clicking Next in all three windows. Under the Summary window, the user can show the command that will be used for the restore, and copy this for future command line use of the Restore procedure. Otherwise, the user can just click Finish to invoke the restore after a few seconds. A DB2 Message window will show the status of the back up. If any errors occur, the user can refer to IBM

for help: http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp.

[00618] For routine database maintenance the user may find the built-in SPM Restore utility more convenient. The user can access this by choosing Main Menu > Manage SM > Database Manager >Database Restore.

[00619] Sensor Adapter Instances

[00620] The Manage SPM > Sensor Instances web page lists all the sensor adapter instances that have been defined in the SPM installation, their channels, properties and parameters. FIG. 66 shows an example operation panel 6100 for Managing Sensor Instances Operations. The Operations panel 6100 provides access to basic sensor adapter information, such as Reload, Delete, Save and Save as. Reload is used to reload the properties of the currently selected instance, overwriting any unsaved edits. Changes to a sensor adapter instance must be made by directly editing the Viper Configuration file (see Policy Management below). Delete is used to delete the currently selected sensor adapter instance. Save is used to save any edits to the currently selected sensor adapter instance. Save as is used to save a copy of the currently selected instance under the name entered in the text field.

[00621] In some implementations, the user can create, edit and delete sensor instances. For a given sensor instance, all dependencies that are linked to it are displayed including a list of all config objects that have a relational link to a given sensor instance. When the user clicks on a sensor instance, the GUI also shows the Viper (if any at that moment) that manages it and lists the policies that depend on its existence to fire (if any at the moment) in addition to the Instance

Properties.

[00622] Sensor Instances Panel

[00623] This panel contains tables which lists all the Sensor Adapter instances in the SPM system. The templates upon which the instances are based, the channels defined for the various message servers and the defined message types. These tables are populated with information taken from the SPM database. To view the properties of a sensor adapter instance, the user can highlight a record in the table shown in FIG. 67. Settings and parameters for that instance are displayed in the Instance Properties panel to the right as shown in FIG. 68 and FIG. 69.

[00624] Instances Listing

[00625] FIG. 67 shows an example Sensor Adapter Instances Panel 6200. The Instances table in FIG. 67 includes information for each Sensor Adapter instance. Further information is displayed in the Instance Properties panel. Instance Name is a user-supplied name (usually assigned by the System Integrator when creating the instance). A given instance can be located using the search tools at the bottom of the table. The same search tools are available for all SPM tables. Template Name is a template on which the Sensor Adapter (SA) is built.

[00626] Instance Properties Panel

[00627] Most sensor adapters share a large number of Common Properties with other sensor adapters in addition to their own Custom Properties. FIG. 68 shows example Sensor Adapter Common Properties. Virtually all sensor

adapters share the following Common Properties: Instance Name and Template Name; Channels; and Properties.

[00628] The Channels include Data Channels, Communications Alarm Channels, Input Channels, Alert Channels and Filter. Data Channels represent channels that send information to a sensor adapter. Communications Alarm Channels represent channels that send information about possible malfunctions in the communications links to or from a sensor. Input Channels represent channels that carry information that is used by a sensor adapter in evaluating a policy. Alert Channels represent channels that carry messages from a sensor adapter. Filter represents the value used to distinguish which kind of reading the sensor is making. For gas sensors, the filter value would indicate which kind of cartridge is being used.

[00629] Properties include SimulatorMode, Enabled, IPAddress; Port; PollingInterval and ReceiveTimeOut. SimulatorMode toggles the sensor adapter in and out of Simulator Mode. In Simulator Mode, the sensor adapter simulates the triggering of a policy. Enabled toggles the adapter on and off. IPAddress represents IPAddress of the sensor or of the gateway linking to the sensor. Port represents the port used will vary according to sensor. PollingInterval represents the polling interval in seconds. ReceiveTimeOut represents sensor adaptors expecting a continuous data stream from a sensor (e.g., a motion dedector). These sensor adapters generate a communications alarm if no message is received within the specified interval. This parameter is not used for sensors such as real-time PCR analyzers that deliver discrete, one-time information.

[00630] Custom Properties

[00631] FIG. 69 is a panel 6400 showing example Sensor Adapter Custom Properties. Most sensor adaptors have a set of properties that are specific to the sensor type.

[00632] FIG. 70 is a panel 6500 showing example templates. Sensor adapter templates are themselves template instances based on the generic adapters. Template names are also user-supplied (usually assigned by the System Integrator when creating the templates for the SPM project). Search tools for locating a given template are available at the bottom of the table.

[00633] Channels Listing

[00634] The Channels table shown in FIG. 70 lists all Channels defined in the current SPM installation and their properties. Selecting a record in the table displays the channel's properties in the Channel Definition panel as shown in FIG. 66. This information is also contained in columns in the table that can be viewed by using the horizontal scroll-bar.

[00635] Channel Name represents unique name of the channel. This can be any name that the System Integrator wishes to assign to the channel. As a mnemonic the name may include message server name, message server type, sensor genre (chemical, biological, radiation, etc.), message type (data, alert, alarm, etc.), and so forth, although such information is included only to make the name easily recognized. There is no mandatory format for channel naming. A full channel description is provided by the information in the other columns of the table.

[00636] FIG. 71 shows an example Sensor Adapter Instance Channel Listing. Channel Type represents classification of channel indication which type of message server it employs, such as Local, SOAP (Simple Object Access Protocol), JMS or TCP. Message Type represents the names of the generic and sensor-specific message data structures declared in the message types section in the first part of the Viper Configuration file. Channel Role represents the function of the channel, such as data, alert, alarm or comm alarm, etc. Message Server represents unique name of the message server. This is either the local message server (for local messages) or the name of the physical server and the message server type (JMS, SOAP or TCP). Payload Type represents the message's SDO type: SML, CAP1_1 or ICD. Protocol Type is typically the same as the message server type (JMS, SOAP or TCP). Sensor Name represents a distinguished name for the sensor. Mapper displays the name of the type_structure_map used by the sensor adaptor to map the sensor message's properties to the SPM message_type properties. The Mapper is most commonly shown in the case of sensors that use the MODBUS TCP protocol. Is Persistent specifies whether the message is held in the database.

[00637] FIG. 72 shows example Channel Property Definitions.

[00638] FIG. 73 shows an example list of message types. The Message Types listed in FIG. 73 includes all message type declarations available in SPM. A given Viper configuration file typically includes a subset of the available message types. Selecting a record in the Message Types tables displays the message properties in the right-hand panel as shown in FIG. 73. Message Type is a user-

supplied name, Message_types are instances based on the type template. The properties associated with a given message_type can be seen by inspecting the Viper configuration file.

[00639] FIG. 74 shows example instance message properties.

[00640] Connecting SPM to Other SPM Systems

[00641] In order to control where different parts of the sensor data are processed it is necessary to establish communications channels between servers. In one case a remote SPM server or SPM Edge server might process and filter the data through its sensor adaptor(s) before transferring the data to a central SPM for policy screening and viewing. In another case, an event might be triggered on the central SPM server which then must communicate a command to a remote sensor connected to a remote SPM server. This kind of communication is effected using the SOAP messaging protocol.

[00642] The central and remote servers communicate via a SOAP message server on the remote servers and a SOAP Channel Receiver or a remote_sensor_controller on the central machine. In both cases, the communication can be set up by editing the appropriate Viper Server configuration files on the two SPM Servers.

[00643] To edit the configuration files, log in to the SPM Console and select Policy > Viper Config Editor. Choose the appropriate Viper configuration file and click Edit. The configuration file in spml format will appear in the edit field below. The user can save a copy of this file under a different name, so that the user can revert to the original if needed. Consult the descriptions which follow to see what

to modify in the configuration file for each of the communicating servers depending on direction of communication. After completing the edits, the user can click Compile to test the edits for syntax errors. The user should fix any errors until the new edits have been validated, then save the file.

[00644] Sending Data from a Remote SPM via a SOAP Messaging Server

[00645] FIG. 75 is a diagram 7000 showing an example of communicating from a remote SPM server. The diagram 7000 includes a remote SPM 7010 (hostname, IP address) and a central SPM 7020 (host name, IP address, port number of SOAP Channel Receiver Sensor Adapter Instance).

[00646] The remote SPM server 7010 that receives data from a sensor 7030 has its own Viper configuration file. This SPM has to forward the sensor data on to a central SPM 7020 using a channel to the SOAP Messaging Server on the central server. Thus, in the “Message Servers” section of the configuration file of the remote SPM 7010 it is necessary to create a soap_message_server instance, specifying the hostname (or IP address) of the central SPM server to which the data will be sent, as well as a port number of the SOAP channel receiver there. (See below for setting up the channel receiver at the central SPM). The SOAP message server also needs to know what data channels to forward and the message_type associated with each channel. The following shows an example SOAP message that assumes a QTL sensor adapter.

```
// ##### Message Servers
.
soap_message_server SoapMsgServer }
        host "209.78.110.130":4040
```

```
// IP address (or hostname) of the central SPM, as well as port number
// of the SOAP channel receiver sensor adaptor instance on the central SPM
        channel QTLDataSoapChannel message_type QTLData
.
.
}
// ##### End Message Servers
```

[00647] In the same remote Viper configuration file, it is necessary to add the name of the SOAP message server instance under message_servers in the "Viper server" section:

```
//########## Viper server:
viper_server NameofViperServer {
        soap_server_port 4000
        message_servers
                LocalMsgServer
                SoapMsgServer
sensor_adapters
        QTL
        .                    .                    .
}
```

[00648] Receiving Data from a Remote SPM via a SOAP Messaging Server

[00649] In order for the central SPM to receive the data through this SOAP channel, a SOAP Channel Receiver sensor adapter instance must be set up in the central SPM's Viper configuration file. This instance defines the port number that was being used above to identify where to send the data from the remote SPM.

```
//### SoapChannelReceiverSA:
sensor_adapter_template SoapChannelReceiverSA_template {
```

```
        message_types
                comm_alarm_message CommAlarmMessage
        protocol SoapChannelReceiver
        properties
                Enabled                 integer
                soapServerPort          integer
                soapServerAcceptTO      integer
}
        sensor_adapter_instance SoapChannelReceiverSA {
        template SoapChannelReceiverSA_template
        channels
                comm_alarm_channels
                LocalMsgServer::LocalCommAlarmChannel
        properties
                Enabled = 1
                soapServerPort = 4040
                soapServerAcceptTO = 100
}
//### END SoapChannelReceiverSA.
```

[00650] Once the SoapChannelReceiver sensor adaptor on the central SPM has received the data from the remote SPM, it must determine which internal channel the data should be sent on. To do this, it identifies the name of the sensor adapter that was used to generate the data on the remote machine by searching through its own instances of the “Relay” sensor. The first part of the name of the “Relay” sensor adaptor instance must match the name of the sensor adaptor used on the remote SPM. In the example shown here, the name of the remote sensor adaptor is 'QTLBiosensor2000_SA'. Thus the 'Relay' sensor adaptor instance to match on the central SPM is called

'QTLBiosensor2000_SA_Relay'. This instance in turn specifies which internal channel to use when forwarding the data. Typically, a channel to the Java Message Service is used for this purpose.

```
//########## Sensor adapters (central SPM config file):
.
//### QTL Relays:
// The following is a template
sensor_adapter_template QTLBiosensor2000_SA_Relay_template {
        message_types
                data_message QTLData
        protocol Relay
}
// End of template
// Beginning of instance based on template
sensor_adapter_instance QTLBiosensor2000_SA_Relay {
        template QTLBiosensor2000_SA_Relay_template
// matches the name of the sensor adapter used on the remote
// SPM to generate the data coming in through the SOAP channel receiver
        channels
                data_channels
                        JmsMsgServer::Bio.Data.QTL.Biosensor_2000
}
// End of instance
.
.
//### END QTL Relays.
```

[00651] Controlling a Sensor Adapter on a Remote SPM

[00652] Control and command from a central SPM server to a sensor connected to a remote SPM server must go through a SOAP server port on the

remote SPM. In this case, a sensor adapter called a "RemoteSensorController"resides on the central SPM. It directs the command and control to its remote location. The command and control logic is set up inside a policy instance on the central SPM. The policy instance is configured to point to the remote Viper server as well as specify which sensor adaptor on the remote SPM is to be controlled.

```
//########## Sensor adapters:
//### RemoteSensorController
// The following is a template
sensor_adapter_template RemoteSensorControllerTemplate {
        protocol RemoteSensorController
        properties
                Enabled         integer
                PollingInterval  double
                PingInterval     integer
}
// End of template
// Beginning of instance based on template
sensor_adapter_instance RemoteSensorController {
        template RemoteSensorControllerTemplate
        properties
                Enabled         = 1
                PollingInterval = 1.0
                PingInterval    = 100
}
// End of instance
//### END RemoteSensorController.
//########## END Sensor adapters.
```

[00653] FIG. 76 is a diagram showing an example of Sending Controls and

Commands from a Central SPM. In this example, a policy on the central SPM 7120 controls the remote sensor adapter. An action_template instance controls an Inova digital messaging display, which is specified under the central SPM policy instance as the “Inova” sensor proxy on the remote server, ViperServer2. The remote ViperServer2 instance is also included under the remote_viper_servers section of the central SPM’s configuration file.

```
// Start of condition and action templates
condition_template QTL_NEG_cond_template {
        (channel_reference(input_channel->TestResult) == "Negative")
}
action_template QTL_NEG_action_template {
        publish
                alert : "Sample " + channel_reference(input_channel->SampleId) +
                " tested Negative in " +
                channel_reference(input_channel->AssayType) + " assay."
                reference_channels(input_channel)
                output_channels(alert_channel1, alert_channel2, alert_channel3)
        control
                commandMessage : "Sample " + channel_reference(input_channel-
>SampleId) +
                " tested Negative in " +
                 channel_reference(input_channel->AssayType) + " assay."
                sensors(Display)
}
// End of condition and action template
// Start of policy template
policy_template QTL_NEG_template {
        condition_template QTL_NEG_cond_template
        action_template QTL_NEG_action_template
```

```
}
// End of policy template
// Start of policy instance
policy_instance QTL_NEG {
        template QTL_NEG_template
        input_channels
                JmsMsgServer::Bio.Data.QTL.Biosensor_2000
// Channel bindings tell SPM where to look for input data and where to send
alerts
        channel_bindings
                input_channel : JmsMsgServer::Bio.Data.QTL.Biosensor_2000
                alert_channel1: JmsMsgServer::Bio.Alert.Technician
                alert_channel2: JmsMsgServer::Bio.Alert.LabManager
                alert_channel3: JmsMsgServer::Bio.Alert.FireDepartment
        sensor_bindings
                Display : ViperServer2::Inova
}
// End of policy instance
```

[00654] Configuration of the central SPM's viper_server declaration, including specifying the hostname or IP address of the remote SPM and the SOAP server port to use to connect with the remote 'ViperServer2':

```
//########### Viper server:
viper_server ViperServer1 {
        soap_server_port 5000
        remote_viper_servers
                ViperServer2 : host "MassSpecDC" soap_server_port 6010
        message_servers
                JmsMsgServer
                LocalMsgServer
```

```
        sensor_adapters
                Inova
                RemoteSensorController
        policies
                QTL_NEG
                QTL_POS
}
//########## END Viper servers.
```

[00655] Connecting Sensors to SPM

[00656] A sensor adaptor (or 'sensor proxy') is provided for the sensors or devices that are to be integrated into SPM. Each sensor adapter is written to handle the specific data that is produced by the sensor. (Some devices do not require a specially written sensor adaptor, but can be integrated using a generic Modbus protocol instead.)

[00657] Emulating Sensors

[00658] User can emulate the sensors within the SPM framework. The user can define the key properties for a sensor in order to simulate actual input from a sensor and to generate test alert messages.

[00659] FIG. 77 shows an example Sensor Emulation. To emulate a sensor event, the following are performed. The Simulation Type can be specificed from the pull-down list at the upper right or by selecting the appropriate tab at the bottom of the form. Some Simulation Types allow the user to modify the alert or message text for the current emulation.

[00660] The Application Filter is used to specify the mode of data collection or sensory input. A demonstration SPM installation may contain some of the

following applications. Each customer installation incorporates completely different combinations of sensors. Chem represents chemical sensors and alerts, for example. Mpex represents panoramic or 360 degree viewing. Coast Guard can represent a sensor associate with the coast guard. Rad represents radiation. Copr02, Corp03, Copr01, etc can represent an entity specific sensor. MAR can represent Mobile Access Router. Bio represents biological sensors. BMS represents Building Management System. WPR represents Wind Profiler Radar. QuickSet represents video camera control system. Compass represents digital compass.

[00661] The Channel Type Filter can be used to specify the output channel type. Also, the specific output channel can be selected from the Parameters field and the Simulate Event can be pressed to execute the event emulation. An alert dialog will allow the user to verify that the user has specified the desired parameters for the emulation. FIG. 78 shows an example of sensor emulation alert. The user can further verify the emulation of the event from within the SPM Console by selecting and specifying the Application and Channel Name. In the example shown in FIG. 78, the Application would be “Rad” and the Channel Type selection would be “All Types” or “Data.”

[00662] FIG. 79 is a panel showing an example alert messages.

[00663] Channels

[00664] FIG. 80 shows an example operation panel. The Operations panel can reload the properties of the selected channel instance, overwriting any unsaved Channel Definition edits. For example, a Delete edit deletes the currently

selected channel instance. A Save edit can save any edits to the currently selected channel instance. A Save As edit can save a copy of the currently selected channel instance under the name entered in the text field.

[00665] SPM Configuration

[00666] FIG. 81 shows an example Manage SPM Configuration form. The Manage SPM Configuration form allows authorized users to specify SOAP Service and Compile time-outs and supply the Google Map Key needed to enable Google Maps. All other fields are read-only.

[00667] Java Message Server Parameters are fields that are read-only and are set by the install program. The Mail Server Parameters are fields that are read-only and are set by the install program. Except for the time-outs, SOAP End Points are fields that are set by the install program. Both time-out values can be reset by the user from within the GUI. GMAP Key is a name of GMAP key for the web server. It is the installer's responsibility to acquire a valid GMAP key for the customer's installation at http://www.google.com/apis/maps/signup.html.

[00668] To acquire a GMAP Key: Access Google at the URL given above. In addition, sign up for a GMAP Key using the IP address of the server where SPM is installed: https://spmServerIPAddress. Also, Generate the key. Copy and paste the key from the Google site into the GMAP Key text field.

[00669] Database Manager

[00670] FIG. 82 shows an example Database Manager. The Database Manager provides tools for executing basic database housekeeping functions from the GUI. The Operations panel contains basic database management tools,

such as New, Reload, Delete, Save and Create. For example, New creates a new empty database instance. Reload is used to reload the properties of the currently selected instance, overwriting any unsaved edits. Delete is used to delete the currently selected sensor adapter instance. Save is used to save any edits to the currently selected sensor adapter instance. Create is used to create a database with the name entered in the text field.

[00671] Database Backup

[00672] FIG. 83 shows example Database Backup Functions. Database backup/restore assumes jboss, DB2, and SPM are installed on a single machine. The user can choose to execute a database backup automatically or manually. For example, Auto can be used to perform automatic backups can be scheduled daily, every two days, weekly or monthly. The user must click Set Schedule to activate the automatic backup. Manual represents selecting the Manual radio button to enable the Backup Now option.

[00673] Database back-ups are distinguished by an identification number that is used internally by DB2 and by a user-readable timestamp. The user should anticipate that each database back-up will require more than 150MB of disk space. After backing up the database, the Database Manager page may need to be reloaded in order to see the backed-up file appear in the List of Backups.

[00674] Database Restore

[00675] FIG. 84 shows example Database Restore Functions. The database can be restored from Last Backup. This option restores the backup that appears at the top of the List of Backups. The database can be restored from selected

file. The user can choose the database to restore by selecting the file from the Database Information table in FIG. 79.

[00676] FIG. 85 shows example database information.

[00677] LDAP Resources

[00678] LDAP data is kept in records (objects with attributes) organized in a tree structure, which can be arbitrarily laid out. The SPM structure normalizes the authentication and authorization information, i.e. stores the permissions in a separate directory from the personal information (name, password, etc.). This design makes for some increase in management complexity, e.g. deletion of a person record may leave dangling permission records that reference that record if a deletion is not properly managed. This factor is more than offset by the advantage of keeping the authorization records in a separate space that can be given separate and specific read/write permissions, as well as separating the authorization data from personal data for which the user may not have modification permissions.

[00679] This web page allows the user to define the LDAP resource available for SPM to manage. SPM supports the Red Hat default LDAP server that ships with the product pre-installed along with the OS. From the LDAP pull-down list the user can select one of the following: Configuration; Template; Connection; and Firewall.

[00680] Database Resources

[00681] The user can define all the database resources available for SPM to manage. For example, INFORMIX IDS 10 and SDAS Oracle Database server.

[00682] Messaging Servers

[00683] Each Viper server contains one or more message servers. Message servers can be of type JMS, SOAP or local. JMS and SOAP message servers have an associated hostname and port number. For JMS, this specifies a JMS server. For SOAP, it is a SOAP end point which exists as a SOAP server running on another Viper instance.

[00684] Local channels allow data to be published on-process from a policy to a sensor proxy or between sensor proxies. In addition, Message servers are the owners of channels.

[00685] The user can define all the messaging server resources available for SPM to manage. For example, SPM supports JBoss Messaging. The system supports a configuration for someone to allocate a new machine resource and say that the messaging server will be located on this device. This would allow one to scale the performance of the messaging server by dedicating an entire machine to it.

[00686] An additional step in creating a messaging server is to define all of the channels that will be managed by that server. Channels names must be unique within a given server. Channel names must follow a well defined convention. A simple form is provided to allow users to create channels and add them to a specific instance of a Messaging server.

[00687] Web services are used to recall a list of messaging server templates that includes JBoss Messaging. In addition, the web services are used to recall the messaging server template details (essentially an SDO describing all of its

attributes, including its channels and each channel's attributes), and create/edit/delete of messaging server instances. In addition, for a given messaging server instance, a list of all dependencies that are linked to the messaging server instance are shown. For example, the user is shown a list of all config objects that have a relational link to a given messaging server instance.

[00688] Config Servers

[00689] The user can define all the Configuration resources available for SPM to manage including the default SPM configuration server. The first version may only have one Configuration server. In addition multiple instances can be supported to provide for high availability and failover capability to all SPM API users whose primary interface into SPM will be the Config server.

[00690] FIG. 86 shows an example panel for Managing a Configuration.

[00691] Viper Servers

[00692] FIG. 87 shows an example panel for assembling a viper server. Using the panel, the user can define all the viper resources available for SPM to manage. The user can drag and drop any of the items shown in the left hand side 8210 to the right hand side 8220 to create a new Viper Server instance.

[00693] The items available to drag and drop inclues Messaging Server Instances; Database Server Instances; Config Server Instances; Policy Instances; and Proxy (Sensor) Instances.

[00694] Deployment

[00695] The users can map SPM Components to machine resources that were previously defined. Example SPM Components include LDAP; Messaging;

DataBase; Viper; and Config serve.

[00696] The Admin Console can provide a DnD interface for mapping SPM Framework components to machine resources. Users should be able to Drag and Drop SPM Framework components from sorted lists into Other Lists that represent machines. Once a machine's software configuration has been defined the user can persist the configuration and instruct SPM to redeploy the entire system. The user should able to click on a machine resource and see it computing resources. If an SPM daemon is running on this machine the user should be able to recall the machine's current resource utilization statistics. The GUI should also provide a visualization for machine resources.

[00697] SPM Machine Resources includes listing of servers.

[00698] Policy Management

[00699] There are two ways to manage conditional publishing and alerting. Simple conditional publishing and alerting is most easily effected at the sensor adapter instance level. More complex policies may require the use of stand-alone policies which comprise individual condition and action instances based on condition and action templates.

[00700] Sensor Adapter Level Conditions

[00701] All sensor adapters can be configured to carry out conditional publishing and alerting without the use of a policy. Sensor-adapter level conditions employ three special keywords as follows:

data_publish_condition

alert_publish_condition

alert_clear_condition

[00702] For each instance of a sensor-adapter level condition, the system creates private instances of three special variables. The first two contain time values, the third is a Boolean.

LAST_TIME_DATA_COND_TRUE

LAST_TIME_ALERT_COND_TRUE

ALERT_ACTIVE

[00703] The scope of each instance of each of these variables is restricted to the sensor adapter instance.

```
sensor_adapter_instance Midas_Condition_Test {
    template Midas_Template
    channels
        data_channels
            LocalMsgServer::Chem.Data.Honeywell.Gas.Chlorine
            JmsMsgServer::Chem.Data.Honeywell.Gas.Chlorine
        alert_channels
            JmsMsgServer::Chem.Alert.Honeywell.Gas.Chlorine1
        comm_alarm_channels
            LocalMsgServer::CommAlarms
            LocalMsgServer::CommAlarmsForICD
            JmsMsgServer::SPM.Alert.Sensor.Comm.Alarms
    data_publish_condition
        Concentration > 5.0
    alert_publish_condition
        (Concentration > 10.0) &&
        (TIMESTAMP_DIFF_SECS(CURRENT_TIME(),LAST_TIME_
        ALERT_COND_TRUE) > 30.0)
    alert_clear_condition
```

```
            ALERT_ACTIVE && (Concentration < 10.0)
        properties
            Enabled = 1
                etc
            SimulatorProperty = "Concentration"
            PollingInterval = 1.0
            IPAddress = "64.210.18.9"
            Port = 502
            MaxConnectFailures = 0
                etc
            Location.Location_description = "ViaLogy"
                etc
            geoLocation.longitude = -118.158805
                etc
}
```

[00704] Each sensor adapter instance that employs conditional messaging and alerting must contain the three conditonal expressions: data_publish_condition, alert_publish_condition, and alert_clear_condition. The syntax for these expression items is analogous to that for filter (i.e., a keyword followed by an expression that must evaluate to true/false).

[00705] Each time such a sensor adapter receives data from its sensor, these statements will be evaluated. When data_publish_condition evaluates to true, the sensor adapter's instance of the variable LAST_TIME_DATA_COND_TRUE is set to the current time and the sensor adapter's data_message is published on the sensor adapter's data_channels. When alert_publish_condition evaluates to true, the sensor adapter's instance of the variable LAST_TIME_ALERT_COND_TRUE is set to the current time, the local instance

of variable ALERT_ACTIVE is set to true and the sensor adapter's alert_message is published on the sensor adapter's alert_channels. Finally, when alert_clear_condition evaluates to true, the variable ALERT_ACTIVE is set to false and the sensor adapter's alert_clear_message is published on the sensor adapter's alert_channels.

[00706] The data_publish_condition statement instructs the system to disregard sensor readings unless the concentration is > 5.0. The alert_publish_condition statement instructs the system to send an alert message when the concentration exceeds 10.0, but only if an alert has not been sent in the last 30 seconds (notice how the current time and the variable LAST_TIME_ALERT_COND_TRUE are being used). In addition, if the concentration stays above 10.0 for a long time, the sensor adapter will continue to send alerts every 30 seconds. The alert_clear_message statement means that if ALERT_ACTIVE has been set to true and then the concentration drops below 10.0, the sensor adapter should publish the alert_clear_message.

[00707] The alert_message and alert_clear_message can include static values. If the implementation requires dynamic values, a policy.

[00708] Sensor Adapter Template

[00709] The alert_message and alert_clear_message must be defined in the sensor_adapter_template. The pertinent part of the sensor_adapter_template definition is shown below:

```
sensor_adapter_template Midas_Template {
        message_types
                data_message        MidasMessage
```

```
            alert_message       MidasAlertMessage
            alert_clear_message MidasAlertClearMessage
            comm_alarm_message  CommAlarmMessage
        protocol Midas
        properties
            SensorCategory  string [] = { "DataReader" }
            <other properties follow>
}
```

[00710] Sensor Adapter Template

[00711] The alert_channels must be defined at the sensor adapter instance level as shown above in the channels statement at the beginning of this topic.

[00712] Alert and Alert Clear Messages

[00713] The alert and alert clear messages that are referenced in the sensor_adapter_template definition would look like the following:

```
type MidasAlertMessage {
        Timestamp          string
        Timezone           string
        UUID               string
        SensorName         string
        SensorType         string
        alert              string = "High chlorine level alert."
        AlertLevel         string = "RED"
        carrierChannel     string
        Location           LocationType
        geoLocation        GeoLocationType
}
type MidasAlertClearMessage {
        Timestamp          string
        Timezone           string
```

```
        UUID            string
        SensorName      string
        SensorType      string
        alert           string = "Chlorine has dropped back to safe levels."
        AlertLevel      string = "GREEN"
        carrierChannel  string
        Location        LocationType
        geoLocation     GeoLocationType
}
```

[00714] Stand-Alone Policies

[00715] The stand-alone policy is one of the basic building blocks in an SPM application. A policy consists of a condition/action pair. Typically, the condition contains expressions that evaluate data taken in through one or more input channels. If the specified condition is met, the action takes place. Most commonly an action involves sending alerts via a number of output channels. The output channels can then be linked to communication systems or serve as input into other policies. An action can also send a command directly to a sensor adapter that in turn communicates with its sensor to perform a specific task such as taking a sample or changing a camera orientation.

[00716] SPM provides a collection of policy templates and instances for a number of different sensors and applications. A policy template is a generic, non-specific pairing of a condition template and an action template, often associated with a particular sensor, which can be useful in a number of different scenarios. In a template the thresholds are typically not specified, nor are the input and output channels. In a policy instance derived from the template, on the other

hand, all properties and parameters are specified and the input and output channels are specified as well.

[00717] One can thus take an existing template, customize it by adding specific parameters and channels for one's particular application, and then save the edited version as a new policy instance.

[00718] SPM Policy Wizard

[00719] FIG. 88 shows an example Policy Wizard that provides step-by-step guidance in creating a new SPM policy or modifying a copy of an existing one. The system copies information entered into the text fields on the preliminary tabs over to the Final tab where it is aggregated for review. The system does not automatically transfer changes entered from the Final tab back to the fields in the other tabs. However, the information as displayed in the Final tab is what is used to create the new custom policy.

[00720] Set Up

[00721] The policy editor provides tools for creating a new policy. For example, NewPolicyName is used to enter the policy name in the text field. No white space or control characters are allowed in the name. For Policy Type, the first two options are used to create or clone a policy that is initiated by a trap. A trap is a message that is sent out by a sensor without being polled. In the case of a Simple Alert, the policy would be initiated by a trap; the Simple All Clear trap message would indicate a state opposite that of the Simple Alert trap.

[00722] The Policy Type includes a simple “Alert” Policy that can be used to enter a custom alert message in the Alerts tab. The policy type also includes a

simple "All Clear" as above.

[00723] If a user selects either of the Simple options, the Trigger tab will be disabled. The third and fourth options can be used to create a policy triggered by a threshold. With these options SPM will poll the sensor at specified intervals to see if a reading exceeds (or falls below) a specified threshold. In the case of high-low thresholds, the policy will be triggered when the reading is outside the specified range.

[00724] High Threshold represents the case where the observed value exceeds the specified threshold. Low Threshold is where the observed value falls below the specified threshold. Both High and Low Thresholds includes the situation where the observed value no longer lies within the threshold range. Select a Viper Server represents a listing of all running Vipers and any remote Vipers declared in the viper_server statement at the end of the spm config file on which the policy will reside.

[00725] Triggers

[00726] FIG. 89 shows example SPM Policy Wizard Triggers. A policy can be triggered by a property value in one or more channels or by a trap. A sensor adapter can be selected from the list. There are two type of policies: those initiated by SNMP traps and those triggered by observed threshold values. The type of Sensor Adapter chosen here determines whether the Pick an Alert Trigger pulldown on the Alert tab is enabled.

[00727] Typical examples of trap messages are those involving equipment status:

Trap Name Trap description

communicationEstablished Informational: Communication with the Sensor restored.

communicationLost Severe: Communication with the Sensor has been lost.

powerRestored Informational: Utility power has been restored.

[00728] Trap messages are handled by the SNMP Trap Receiver sensor adapter.

[00729] Dynamic Properties include properties that display a range of values rather than simply a status. Triggers supply high and low threshold values. Active Policies include a listing of currently active policies in the selected SPM Viper. Policies must be uniquely named.

[00730] FIG. 90 shows an example SPM Policy Wizard Alerts. The fields in the Alerts tab in FIG. 90 are used to set messages for trap policies. The user can pick an alert trigger. The Wizard allows the user to create a policy with only one trigger channel. In some implementations, multiple triggers from the GUI can be used. Multiple triggers can be specified by editing the policy instance in the Viper configuration file.

[00731] The user can create his/her alert message to define the reference_constant for the Alert. This will have a name similar to high_ammonia_alert_message and will be included in the policy_instance > constants statement.

[00732] The user can create his/her all-clear message that defines the reference_constant for the All Clear alert. This will have a name similar to high_ammonia_alert_off_message and will be included in the policy_instance >

constants statement.

[00733] Publish

[00734] FIG. 91 shows an example Publish form that contains a number of alternatives for publishing or displaying alerts. Call a number can be used with VOIP messaging devices like Voxeo. Multiple phone numbers can be entered here if separated by commas. The user can enter the number without spaces or hyphens. SPM does not verify the validity of phone number formats. The user can send an E-Mail. One or more valid E-mail addreses here separated by commas. SPM verifies the validity of E-mail address formats. Digital Display (text) can be provided. The policy can send text output to a variety of messaging displays including the SPM console on you computer. The user can Display a Web Page by entering a valid URL. In addition, SMS (Carrier) can be sent using this with text messages. Multiple phone numbers can be entered here if separated by commas. The number is entered without spaces or hyphens. The user can also Pick a Message Channel. A single message channel can be specified using the Wizard. Multiple message channels can be specified only by editing the policy instance in the Viper configuration file.

[00735] Actions

[00736] FIG. 92 shows an example SPM policy wizard showing an action tab. The check boxes in the Actions tab allow the user to add or modify a number of additional policy actions. These policy action options vary according to the customer's SPM system and are typically self-explanatory.

[00737] Final

[00738] FIG. 93 shows an example SPM policy wizzar with a final tab. The Final tab displays a synopsis of the entries from the other tabs in the Policy Wizard. A policy should be saved before activating.

[00739] Viper Configuration File Editor

[00740] FIG. 94 shows an example Viper Configuration File Editor 8900. The Viper Configuration File editor 8900 is a full-featured text editor that the user can use to edit configuration files from within the SPM application.

[00741] Viper Configs Tree

[00742] The user can perform the following actions with Viper Configuration Editor tools: List, Edit, Delete, Start and Stop. List action can be used to populate the Viper Configs tree with the Viper configuration files (.spm files) currently existing in the SPM system. The Edit action can display an editable version of the selected Viper configuration file in the text edit field. Delete action can delete the selected Viper configuration file. Start action starts the execution of the .spm file. The Start action is available only to users with viperServer start permission. The Stop action stops the execution of the .spm file. The Stop action is available only to users with viperServer stop permission.

[00743] SPML Operators

[00744] FIG. 94 shows example SPML operators. Many of the operators and delimiters used in SPML are common to most programming languages. However, a few have usages that are peculiar to the SPML language.

[00745] Thus, their definition and usage are presented here for the user's convenience. FIGS. 95 and 96 show a list of SPM operators. The operands for

operators, as shown in FIG. 95 may be numbers, strings or Booleans (T/F). The delimiters in SPML generally observe the same usage as in C++.

[00746] FIGS. 97 shows a list of reserved SPML keywords. A number of keywords such as if/then/else/fi, boolean and true/false have no SPML-specific meaning but observe conventional program language usage. These conventional keywords are included for the sake of completeness.

[00747] The SPML keywords are case sensitive. With the exception of several obvious acronyms, the SPML keywords are written in lower case. Elements in compound keywords are always joined using the underscore character.

[00748] FIG. 98 shows example SPML functions. The function names in FIG. 98 and their arguments are case sensitive.

[00749] FIG. 99 shows an example File Editor Tool. The lower panel in the Viper Configuration page contains the EditArea© file editor. The title bar area of the editor provides a number of conventional text editing functions most of which do not warrant separate discussion. The editor 'About' button accesses a useful synopsis of editing functions.

[00750] As mentioned above, an existing configuration file can be loaded into the editor from the file tree. The Configuration file editor should be used only by persons with System Integrator or Expert User permissions.

[00751] The functions of the file editor include Clear; New; Save and Compile. Clear is used to remove the currently loaded SPML file from the editor. New is used to create a new empty file into which you can input/paste an SPML file. Save is used to save most edits to a compiled file. The Save function is not

enabled until the file has been successfully compiled. The Compile function is used to check the current SPML file for syntax errors and loads the compiled file into the SPM database. Compilation overwrites the existing SPML file. When a file is compiled, the compiler coverts the SPML statements into C++ configuration objects. The comments are discarded.

[00752] When the compiled SPML file is retrieved and displayed, a reverse compilation is carried out. The compiler displays the statements in the compiled file in the order which is most appropriate for the functioning of SPM. As a result, the order of objects in the SPML could change from the original input.

[00753] Comments are not retained because reordering the statements changes the correspondence of comment line numbers to the intended objects.

[00754] Still, the user should make ample use of comments when constructing the file in an external editor and save the file externally for future reference before compiling.

[00755] The Export function is used to open the SPML file in a text editor of your choice from which you can save the file; this functionality is available only when using Firefox.

[00756] FIG. 100 shows an example panel for naming a new viper configuration.

[00757] FIG. 101 shows example tools for a main menu reporting panel. As shown n FIG. 101, SPM includes tools for viewing and searching log files, reviewing records of past events, and monitoring data and events in real time on sensor adapter reporting channels and policy output channels.

[00758] SPM generates four kinds of logs as shown below:

1 – a file containing all log messages from the Viper Server

2 – a file containing just error and warning messages from the Viper Server

3 – log messages describing interactions with the console

4 – a real-time JMS stream log

[00759] The first two log files are stored on the SPM Server in a storage location such as: usr/local/spm/bin/serverlogs.

[00760] The log messages describing interactions with the SPM Console are available through the Console.

[00761] View Logs

[00762] View logs are used to search the Admin Console log messages. There are two varieties of Admin console log messages:

spm_adminconsole_boot.log includes all log information

spm_adminconsole.log includes only errors and warnings

The log file records can be searched by time or by a specified string in the record. This is a search tool and not a filter.

[00763] Options

[00764] Examples of options for view logs include Get Logs (which inlcudeds Num, Date/Time, Log Gile and SPM Component), Back Up Logs, Show Logs, Clear All Logs, and Auto Refresh. Get Logs is an option used to populate Logs table that contains Num, Date/Time, Log Gile and SPM Component. The Num option is used to reverse the sort order by clicking on column heading. Clicking

on a cell displays the log file in a field in FIG. 102. FIG. 102 shows an example reporting panel for viewing logs. The Date/Time option is used to display the log file in a field in FIG. 102 by clicking on a cell. The Log File is used to display log file in a separate browser window. The SPM Component option is used to display the log file in a field by clicking on a cell in FIG. 102.The Back Up Logs option can be used to back up the logs to a used-specified location. The Show Logs can be used to displays the selected log file in the viewing field. The user can view the file in a new browser window simply by clicking on the file name in the Log File column. The Clear All Logs option allows the user to delete the existing log files and start new log file series. The Auto Refresh option can be used to update the Logs table whenever a new log file is created.

[00766] FIG. 103 shows example color codes for network nodes. To visualize the SPM network, FIG. 103 represents different nodes in various colors.

[00767] Visualize SPM Network

[00768] Pressing 'Visualize the SPM Network' generates a graphical representation of the existing SPM network as shown in FIG. 104. The color key in FIG. 104 provides an explanation of the nodes.

[00769] Events

[00770] The primary types of Events are trigger events, i.e. sensor readings that trigger a policy within SPM, and communications alarm events. FIG. 105 shows an example of event messages generated by an SPM policy as it evaluates data coming from a sensor via a sensor adapter.

[00771] Event Reporting – Operations

[00772] FIG. 106 shows example Events Message Retrieval and Event Cleanup panels. The tools in the Events Message Retrieval and Event Cleanup panels allows the user to (1) view stored messages for a selected channel or (2) delete messages for a selected channel during a specified time span. The user can specify how many records (lines) are to appear on a page and use the navigation keys to view the messages one page at a time.FIG. 107 shows example communications alarm event messages. Sensor and communications link status can also be a source of event messages. FIG. 107 shows messages generated when SPM polls several sensors and determines that their communications links have been interrupted and restored. If the user attempt to retrieve messages from a channel that has not sent messages to the database, SPM will return a Database Write/Read exception alert like that shown in FIG. 108. FIG. 108 shows an example SOAP message alert.FIG. 109 shows an example channel list status. The Channel List Status panel in FIG. 109 lets the user view the currently defined channels by Channel Type and/or Message Type. The channels are those defined in the Viper configuration file(s) under the channel keyword. (See Viper Configuration File > Message Server Instance Definitions.) T he messages are those defined by the type keyword in the beginning of the configuration file(s). T he “Channel Type” and “Message” pull-down lists function as independent selection filters.Filter results are displayed in the Channel List table. This table is too wide to fit all of its columns into a printed page width. The table columns are discussed below. **Channel Name** is a descriptive mnemonic of arbitrary format. For ease of reference the channel

name typically contains some or most of the following information. **Message server name** that represents some instance of a JMS, SOAP, TCP or local server, such as JmsMsgServer. **Sensor adapter generic type** can be BMS, Bio, Chem or Rad. **Message category** can be Video, Alert, Data, Alarm or CommAlarm. The channel name also includes **Message recipient** (e.g. LabManager) or **Data source** (e.g. a sensor property or data field such as Frames.BufferPlayback or QTL.Biosensor_2000).

[00776] Another column in FIG. 104 shows the **Channel Type** that represents the message server type (JMS, SOAP, TCP, HTTP or LOCAL). The **Channel Role** can be Video, Alert, Data, Alarm or CommAlarm. The **Message Type can include the** name of the message as declared using the *type* keyword (e.g. GeneXpertMessage, QTLMessage, etc.) The **Message Server** names are assigned by the System Integrator. Each Viper configuration file in an SPM system has its own set of uniquely named message servers. Generally the message server name indicates the message server type. The **Message Protocol** includes the type of XML schema used in the message. Pptions are *SML* (default), *CAP1_1* and *SEIWG ICD*. **Is Persistent** represents local messages and video messages that are not persistent (i.e., held in the local database). Most other messages are persistent. Because of their size, streaming video messages are held in local buffer only for periods typically not exceeding one minute. Video loops and clips of any length desired can be stored on a dedicated server such as Broadware NVR.

[00777] Dashboard – SPM Real Time Dashboard

[00778] The Dashboard provides tools for monitoring channels and network video recorders in real time and archived videos. Channel types include data, alert, alarm and video channels. Search for Data Channels (Advanced Tools).

[00779] FIG. 110 shows example advanced search filters and functions. FIG. 106 shows the Show Search Tools button that accesses the full set of search filters and functions as shown in FIG. 110.

[00780] FIG. 111 shows an example real time dashboard. From the dashboard in FIG. 111, the user can view whatever is being monitored on a given channel by selecting the channel and clicking Monitor. Typical examples of Monitor displays are the BMS video camera shown in FIG. 112 and the gas detector in FIG. 114.

[00781] Options for monitoring can include: manage channel, manage device, and monitor. The monitor option displays information from the monitoring device. For example, if the sensor is a video camera, the Monitor may be similar to that shown in FIG. 113. If the sensor is a biological detector, the Monitor might display something similar to that of FIG. 115.

[00782] The controls shown in 113 are typical for video monitors. For example, '(hz)' is used to input video frame display rate. The collection rate is specified in the Viper Configuration file and is set by default to 4 hz. A local user can set the display rate to 4 hz or less. If the display rate is set to a value greater than 4 hz in the SPM Console, the setting in the Configuration file will override the console setting. Because of bandwidth considerations, display rates for remote users should be slower, (e.g. 2 second intervals [0.50 hz] or longer). The Start/Stop

button toggles the acquisition of the video. The Map button accesses a Google map using the location coordinates. Typically these will be the GeoLocation coordinates for the sensor. The Pulldown Menu on the bottom lists all the video channels in the current SPM plus accesses the video buffer in the database.

[00783] JMS Messaging Service

[00784] FIG. 116 shows an example JMS Log Viewer that displays the status of the traffic through the JMS Messaging Server.

[00785] Referring back to FIG. 114, the chart shows the concentration of a gas versus time. The controls shown in FIG. 114 are typical for most SPM monitor displays. The Stop/Start button is used to toggle the operation of the compass. The Map button is used to access a Google map using the location coordinates. Typically these will be the GeoLocation coordinates for the sensor.

[00786] At the same time the gas concentration is being charted and sent to a site that needs to know specifics about the gas levels, a simple text alert can be sent to another site such as a fire department (see FIG. 115). The most recent alert is shown at the top of the monitor window.

[00787] Biological Sensor Alerts and Messages

[00788] The QTL Biosensor Sensor Adapter generates a main alert with full location and device information as shown in FIG. 115 and also creates an abbreviated alert (see FIG. 113). FIG. 115 shows example QTL Biosensor full alert. FIG. 118 shows example QTL BioSensor Abbreviated Alert. In both cases the physical location of the sensor can be viewed via Google Maps by using the Map function.

[00789] FIG. 119 shows an example Cepheid SmartCycler Monitor with Data Table.

[00790] Dashboard – NVR Cameras

[00791] The appropriate icon can be selected to access a locally-networked camera or an externally-networked camera. Local networks also include external servers linked via VPN.

[00792] FIG. 120 shows example network options for network video. Name represents a user-supplied camera name. Names may contain white space or control characters. Type represents the compression technology used. Other options include Model; W x H; NVR HostName; Exec Mode; NVR Port; Public URI and Public Port.

[00793] Dashboard – NVR Archives

[00794] The NVR Archives tab contains an additional Archive Type filter for listing video archives. Archive Type can be used to list all the archives whose names begin with the selected text string. Properties include Table; Media; Resource Name; Archive Name; Clip Name; Status; Size; Start Time; Expire Time; Path; NVR HostName; NVR Port; Source Type; Public URI; Public Port; Media; and Type.There are three Broadware archive types. In the Regular type, the archive recording terminates after a pre-set duration and is stored. In the Loop type, the archive recording continuously records until stopped. The loop archive reuses the same space on a first-in/first-out basis after every completion of the loop. In the Clip type, the source of the archive is extracted from one of the two other types and is stored.Account ManagementCorporate LicensingWith

regard to SPM licensing, SPM components are categorized as Policy Engines, Sensor Adaptors, and Core Components. The Policy Engines includes Viper and Viper Edge. The Sensor Adaptors can include Chemical, Biological, Building Management Systems and Radiation Detection. The Core Components includes Config Servers, Named Users, DB2, LDAP, JBoss Application Servers and JBoss MQ. Different components may be subject to different licensing protocols.FIG. 121 shows example operations for account management. Operations include upgrade, erase, upload, and show full path. Upload can be used to upload a .spm file to the database and recalculates resource allocation. Show Full Path shows full filepath of selected file in an Alert dialog as shown in FIG. 122 to allow the user to confirm that correct file has been uploaded.

[00800] Manage Roles

[00801] FIG. 123 shows example user roles and permissions. SPM can be configured so that it can be accessed by any number of users (see FIG. 121). Each user may be assigned one or more user roles, and the roles may also be assigned differing numbers of permissions.

[00802] Most of the permissions are pertinent only when changes to the SPM configuration can be made from the GUI rather than by directly editing the Viper configuration file. For example, when adding new policies or policy instances, or new sensor adapters and sensor adapter instances from the GUI, the ability to allow or disallow certain users to create, edit or delete policies, sensor adapters and other configuration objects may be important.

[00803] The Manage Roles function allows users to create and edit user roles.

The SPM Admin user is automatically granted overall authority, while other roles have differing sets of permissions. For example, a power user may have the ability to create and delete policy, condition and action templates, while an ordinary user would be allowed only read or list access to these templates.

[00804] FIG. 124 shows an example listing of SPM roles.

[00805] FIGS. 125 and 126 are tables showing example Permission Rankings for Config Objects and Predefined User Roles. The tables in FIGS. 125 and 126 list the Config Objects, their permission rankings and the permissions that have been assigned to the predefined roles of System Integrator/System Administrator (Admin), Expert User (Expert) and End User (User). The highest config object permission level for a given predefined role is indicated by the level's initial.

[00806] Specifying a given permission does not automatically enable all lesser permissions to the right: you must manually include the lesser permissions when creating a role. The 'all' permission disables permission checking within the SPM system. Thus, no prohibitions are in effect.

[00807] FIG. 127 shows an attempt to access an unallowed config object. The New operation creates an empty Permission for Role form in the Roles and Permissions panel. Reload restores the existing permissions to a role that is being edited, but which has not yet been Saved. Delete deletes the selected SPM role. The permission level may allow a user to create a role, but not to delete the same role. A role cannot be deleted if it is currently assigned to a user. That role must be removed from any User Profile that includes the role desired to delete.

[00808] Save saves any changes in the permission made to the selected role. Create allows you to create a new role after (1) supplying a role name in the adjacent text field and (2) adding permissions in the Permissions for Role form. The user cannot Create a role with an empty permissions listing.

[00809] Creating a New Role

[00810] To create a new role, the follows can be performed. The user can click New in the Operations panel of the Manage Roles form. This creates an empty Permissions for Role form in the Roles and Permissions panel. The name for the new role can be entered in the Operations text entry field. The user can click the All Permissions tab in the SPM Roles panel. In the Config Objects column, the user can click to select/deselect an object. The user can select multiple objects. The common Shift-Click and Ctrl-Click options are not available. All selects all Config Objects in the list. The user can click on individual objects to deselect them. None deselects all Config Objects in the list. The user can drag the selected object(s) into the Permissions for Role panel and Click-Drop them. The pointer must change into a standard arrow pointer and be inside the table before you can Click-Drop the objects. Every user must have either the user.read or the all.all permission. Otherwise the user will not be able to log in. The user can click Create. The system will confirm that the new role has been created. When the user opens the All Roles panel the new Role will appear in the listing.

[00811] Modifying an Existing Role

[00812] To change permissions in an existing role, the user can open the All Roles panel and select the Role as shown in FIG. 124. The name of the Role will

appear in the Permissions for Role text field. The user can add permissions as described above. The user can delete permissions. For example, 'All' selects all Config Objects in the list. The user can select individual objects to deselect them. None deselects all Config Objects in the list. Delete deletes the selected object(s). Clear deletes all objects. The user can click Save in the Operations panel to commit the changes to the selected role. The system will confirm the edits.

[00813] FIGS. 128 and 129 show panels for selecting permission to add to a user role. If a user attempts to access a config object but lacks the proper permission, the system will generate an alert similar to that shown in FIG. 127.

[00814] Using the SPM Roles and Role and Permissions Search Tools. Both panels in this form (FIGS. 128 and 129) contain identical sets of search tools. Use is used to enter the case-sensitive target search string here. As is used to choose an option including Substring and Match. Substring returns all records containing the search string. Match returns only the record with a whole word match to the search string. On column is a pull-down list contains entries for each column in the table. Check button executes the search, listing the found records in the table. SPM Roles locate the role(s) with the specified string or whole-word match. Roles and Permission locates the config objects with the specified string or whole-word match. After executing a search in this panel, the user can review which Roles contain a given Permission simply by clicking through the listing in the SPM Roles panel. X button restores the table to the pre-search condition.

[00815] Manage Users

[00816] The SPM users form allows the user to create new users, manage eisting ones and assign them roles. The SPM users table contains a listing of current SPM users and their roles as shown in FIG. 130.

[00817] Operations

[00818] New is used to open an empty SPM User Profile form as shown in FIG. 126. Reload is used to reload the selected user profile for edit, discarding ay unsaved edits in the currently open SPM User Profile form. Delete is used to delete the selected user profile. Save is used to save a new user profile or changes to an existing user. Save As is used to save an edited version of an existing user profile. The text field can be used to allow the user to save a copy of a selected user profile under a different name.

[00819] Adding/Editing a User

[00820] Most of the fields in FIG. 131, SPM User Profile form, are self-explanatory. All fields except Roles are directly editable. For example, UserID should be entered by observing usual username conventions, such as no spaces or control characters. The First Name/Last Name are self-explanatory. Adding Roles to this field are described further below. Password/Confirm Password should be a strong password. For example, no dictionary words should be used, and passwords should contain a combination of characters and numbers.

[00821] Adding a Role to a User Profile

[00822] To add one or more roles to the Roles field, the following can be performed. In the SPM Users table, the user can click the All Roles tab as

shown in FIG. 132. The desired role(s) can be selected from the Available Roles column. The user can click Add to include them in the User Profile. FIG. 132 shows adding a role to a user profile.

[00823] The selected role(s) are now included in the user profile. FIG. 133 shows a role added to a user profile. The user can click Save to add the new or edited user profile to the SPM database. This automatically copies the user profile UserID to the text field in the Operations panel.

[00824] Changing User Roles on a User Profile

[00825] To change user roles in a user profile, the following can be performed. Access the SPM Users table in FIG. 134. The user is selected from the list. In the SPM Users table, the user can click the All Roles tab. One or more Roles can be selected from the Available Roles table and do one of the following: (1) Click Add to add the new role(s) to the currently assigned roles; (2) Click Replace to swap the selected roles with the currently assigned one(s); or (3) Click Clear User Roles to remove the current assignments, then Add the new role(s).

[00826] User Roles

[00827] The User Roles form provides a synopsis of user names, roles and permissions in one convenient location. Table of Users in the SPM System populates the fields shown in the SPM User Profile form. The User Profile form can be expanded to allow entry of the user's company affiliation, phone number, E-mail address and Skype username, fore example.

[00828] FIG. 135 shows a panel for SPM Roles and Permissions. The panel in FIG. 135 includes two subpanels: (1) All Roles that list all roles defined in the

SPM; and (2) All Permissions that list of all the permissions assigned to the various roles. The user can use the search tool to search on the entries of any of the columns. For example, a search on the conditionTemplate configuration object yields a listing of all the roles with any type of permission for that configuration object.

[00829] 'Role/Permissions for this User' is used to allow the user to select a username in the main table and in response, display all the roles and permissions for that user as shown in FIG. 136.

[00830] Alien Sensor Adapter, RFID

[00831] Overview

[00832] The ALR-9800 is a multi protocol, fixed RFID Reader from Alien Technology Corporation. Running in Autonomous Mode, the reader constantly reads tags and can operate on its own. SPM is set up using AutoMode to listen for notification messages from the reader containing any tag data that it has read. Using this mode of operation, many readers can be configured to continuously read tags, and a single SPM can listen for and process data from multiple readers over the network. In this particular Sensor Adaptor, SPM reads in the Tag List from the reader at regular intervals (the PollingPeriod). Once the tags are read by SPM they are erased from the reader, so that one tag only gets registered once.

[00833] Aside from tracking the tags, the sensor adaptor allows for detection of removal of tags, or the separation of two paired tags from each other. In those scenarios, SPM is looking for changes in states of a particular tag. For example,

one second the tag is there, the next it is gone. However, to avoid false alarms, as RFID reading may not be flawless and miss a tag every now and then, the state changes are reviewed over a certain time span (WindowTimeout) which includes a number of polling periods, and not from just one polling event to the next.

[00834] Connection

[00835] You can integrate the ALR-9800 RFID Reader with SPM via the reader's TCP/IP interface using telnet. To implement this connection in SPM, you need to know the IP address of the reader. The connection to the reader is made via a serial communications port. Connect a serial cable to the reader's RS-232 port and to the COM port on the host computer. Use a serial communications program such as HyperTerminal with the appropriate settings to communicate with the reader. The settings include the following:

Baud Rate 115200

Data Bits 8

Parity None

Stop Bits 1

Flow Control None

[00836] Next, power up the reader. Towards the end of the boot-up procedure the following text will be displayed, including the IP address and port number (shown in bold). These numbers are to be used in the configuration of the sensor adapter instance responsible for a particular ALR-9800 RFID Reader.

Boot> Booting Alien RFID Reader

Boot> Boot Level 1 (Console Communication) : Success

Boot> Boot Level 2 (Reader Communication) : Success

Boot> Boot Level 3 (Command Set) : Success

Boot> Boot Level 4 (Tag Manager) : Success

Boot> Boot Level 5 (Initializing Network Interface) : Success

Boot> Boot Level 6 (Network : Success – IP Address is 10.9.8.10)

Boot> Boot Level 7 (Telnet Interface) : Success – Port 23 Ready

[00837] Message Types

[00838] The AlienRFID_Template allows for the generation of three message types. Two of them are the common data message and communication alarm message. The third type is actually an alert message, as in this case, the sensor adapter has a built-in policy which triggers an event when one tag of a tag pair has become separated from the other one.

[00839] The Alien RFID includes two message types that are peculiar to Alien RFID. AlienRFIDMessage contains properties relating to the RFID tag reader or reading event. This data is generated every time a tag is being observed. Most are already described below. AntennaID and ProtocolID are read off the reader and tell the user what antenna and protocol was used by the reader. (This protocol has nothing to do with SPM protocols.) The alert in this structure is not used and should be ignored. AlienRFID_OwnedTagGoneMessage contains properties belonging to instances of the tags themselves and which are used in an alert when an “owned” tag has been detected, but its owner is not present. The alert contains the TagIDs and TagDescriptions of the coupled tags, as well

as which tag is missing and the location of the other tag.

[00840] AlienRFIDMessage

[00841] These are the currently specified elements in the AlienRFIDMessage data structure. The following is an example data structure.

```
type AlienRFIDMessage {
// generic fields:
        Timestamp string
        Timezone string
        UUID string
// end of generic fields.
        TagID string // hexadecimal digits
        TagDescription string
        ReadCount string
        DiscoveryTime string
        LastObservedTime string
        WindowStartTime string
        CouplingType string
        CoupledWith string
        AntennaID string
        ProtocolID string
        alert string
}
```

[00842] FIG. 137 is a table showing descriptions of message data structure properties.

[00843] AlienRFID_OwnedTagGoneMessage

[00844] The elements in the following message data structure are used in alerts when an “owned” tag passes the reader.

```
type AlienRFID_OwnedTagGoneMessage {
```

```
// generic fields:
        Timestamp string
        Timezone string
UUIDstring
// end of generic fields.
        OwnerTagID string
        OwnerTagDescription string
        OwnedTagID string
        OwnedTagDescription string
        OwnedJustLeft boolean // always true !
        OwnerIsPresent boolean // true or false
}
```

[00845] FIG. 138 shows an example data structure for AlienRFID_OwnedTagGoneMessage.

[00846] Sensor Adapter Template and Instance

```
// The following messages are generated by the Alien RFID Sensor
local_message_server myLocalMsgServer {
        channel LocalCommAlarmChannel message_type
CommAlarmMessageType
        channel AlienRFIDLocal.Data message_type AlienRFIDMessage
        channel AlienRFIDLocal.Alert message_type
AlienRFID_OwnedTagGoneMessage
}
// The Alien sensor_adapter_template:
sensor_adapter_template AlienRFID_Template {
        message_types
                data_message AlienRFIDMessage
                alert_message AlienRFID_OwnedTagGoneMessage
                comm_alarm_message CommAlarmMessageType
        protocol AlienRFID
```

```
        properties
                Enabled integer
                SimulatorMode integer
                IPAddress string
                IPPort integer
                PollingPeriod integer // in seconds
                WindowTimeout double // in seconds
                DropTimeout double // in seconds
                LoginUserName string
                LoginPassword string
                ConfigCommands string
                PollCommands string
                CoupledTags string
                TagDescriptions string
}
//an instance based on AlienRFID_Template:
//
sensor_adapter_instance myAlienRFID {
        template AlienRFID_Template
        channels
                data_channels
                        myLocalMsgServer::BMS.Data.Alien.RFID
                alert_channels
                myLocalMsgServer::BMS.Alert.Alien.RFID.OwnedTagGone
        comm_alarm_channels
                myLocalMsgServer::CommAlarms
                myLocalMsgServer::CommAlarmsForJBC2S
                myLocalMsgServer::CommAlarmsForTASS
                myJmsMsgServer::SPM.Alert.Sensor.Comm.Alarms
properties
        Enabled = 1
```

```
        SimulatorMode = 0
        IPAddress = "64.210.18.179"
        IPPort = 23 // default port for Telnet
        PollingPeriod = 1 // seconds
        WindowTimeout = 2.0 // seconds
        DropTimeout = 30.0 // seconds
        LoginUserName = "alien"
        LoginPassword = "password"
        ConfigCommands = "AutoModeReset;AutoMode=ON;PersistTime=-1"
        PollCommands = "get TagList"
// define owner-owned couples and descriptions
        CoupledTags = "1-2;1-3"
        TagDescriptions = "1:Bob;2:Bob`s suitcase;3:Bob`s laptop;4:John"
}
```

[00847] FIG. 139 shows example Alien Sensor Adapter (RFID) properties as implemented in the sensor adapter template.

[00848] Behavior

[00849] In most applications the policies associated with the AlienRFID reader are implemented by logic contained in the reader itself.

[00850] Error Handling

[00851] The Alien Sensor Adapter (RFID) service will respond to error conditions. The Alien Sensor Adapter (RFID) will recognize any non-successful call and send out its own alert on an alert_channel with appropriate error as its message.

[00852] Arecont Video (IP)

[00853] The Arecont Video sensor adapter properties include the property name, Enabled. The property, Enabled represents whether the sensor proxy is

enabled or not. The value of Enabled property is 1.

[00854] The following represents an example of the Arecont Video (IP) in SPML.

[00855] An example of the Arecont Video (IP) in SPML:

```
sensor_adapter_template <Whatever>_template {
        message_types
                alert_message AlertMessageType -and|or
                data_message <Whatever>Data -and|or
        comm_alarm_message CommAlarmMessageType
protocol <Whatever>
properties
        Enabled                 integer -and|or-
        IPAddress               string -and|or-
        HTTPGetUrl              string -and|or
        numberToDial            string -and|or
        tokenID                 string -and|or
        messageToSend           string -and|or
        recvTimeout             integer -and|or
        port                    integer
}
sensor_adapter_instance <Name> {
        template <Whatever>_template
        channels
                input_channels
                        LocalMsgServer::Chem.Alert.FireDepartment -and|or-
                        LocalMsgServer::<some.sortof.channel>
                alert_channels
                        LocalMsgServer::<Whatever>Alert
        properties
                Enabled = 1
```

HTTPGetUrl = "<someURL>"
numberToDial = "555123456"
tokenID =
"341b34d433ed0b4db7bef7fd57a6786ecb04a4ac8940879
ac9c211021cd0e9422678b5c5d5d08fa668714697"
messageToSend = "Your message here"
recvTimeout = <some # of secs>
}

[00856] The Arecont Video (IP) can send other messages, such as the one below.

```
type Alert|AlarmType {
        TTimestamp string
        alert string
        AlertLevel string
        referenceChannels string
        buildingLocation BuildingLocationType
        etc
}
```

[00857] Behavior

[00858] An example behavior of the Arecont Video (IP) is given below.

[00859] Publish policy:

```
policy_template SOME_CONDITIONACTION_template {
        condition
                …
        action
                …
                publish
                        alert : constant Some_Conditionaction_alert_message
                        alertLevel : "RED"
```

referenceChannels : "RefChannel1~
RefChannel2~
RefChannel3"
reference channels (something_channel)
on channels (alertChannel1),
}

[00860] Control Policy:

policy_template SOME_CONDITIONACTION_template {
condition
...
action
...
control
messageToSend : constant
Some_Conditionaction_alert_message
sensors(whatever)
}

[00861] Error Handling

[00862] The service will respond to with the appropriate error messages. For example, The Arecont Video (IP) will recognize any non-successful call and send out its own alert on an alert_channel with the above error as its message.

[00863] Barix Barionet Sensor Adapter

[00864] The Barix Barionet is a network-enabled, programmable automation controller for industrial automation and facility management systems. The Barionet programmable telecontroller device and its accessories communicate in Modbus with each other, to outside systems they can be connected via standard interfaces (SNMP, CGI, HTTP, Modbus/TCP).

[00865] Properties

[00866] FIG. 140 shows example specified Barix Barionet Sensor Adapter properties.

[00867] The following shows an example of the Barix Barionet Sensor Adapter in SPML.

```
sensor_adapter_template Barionet_Template {
        message_types
                comm_alarm_message CommAlarmMessage
        protocol Barionet
        properties
                Enabled integer
                SimulatorMode integer
                IPAddress string
                PollingRate double
                Relay string
                Value string}
sensor_adapter_instance <Name> {
        template <Whatever>_template
        channels
                input_channels
                        LocalMsgServer::Chem.Alert.FireDepartment -and|or-
                        LocalMsgServer::<some.sortof.channel>
                alert_channels
                        LocalMsgServer::<Whatever>Alert
                properties
                        Enabled = 1
                        HTTPGetUrl = "<someURL>"
                        numberToDial = "555123456"
                        tokenID =
```

"341b34d433ed0b4db7bef7fd57a6786ecb04a4ac8940879
ac9c211021cd0e9422678b5c5d5d08fa668714697"
messageToSend = "Your message here"
recvTimeout = <some # of secs>
}

[00868] The Barix Barionet Sensor Adapter can send other messages, including the following:

type Alert|AlarmType {
TTimestamp string
alert string
AlertLevel string
referenceChannels string
buildingLocation BuildingLocationType
etc
}

[00869] Behavior

[00870] The typical behavior of the Barix Barionet Sensor Adapter are described below.

[00871] Publish policy:

policy_template SOME_CONDITIONACTION_template {
condition
…
action
…
publish
alert : constant Some_Conditionaction_alert_message
alertLevel : "RED"
referenceChannels : "RefChannel1~
RefChannel2~

RefChannel3"
reference channels (something_channel)
on channels (alertChannel1),
}

[00872] Control Policy:

```
policy_template SOME_CONDITIONACTION_template {
    condition
        …
    action
        …
        control
            messageToSend : constant
Some_Conditionaction_alert_message
            sensors(whatever)
}
```

[00873] Error Handling

[00874] The serve will respond with appropriate error messages. The Barix Barionet Sensor Adapter will recognize any non-successful call and send out its own alert on an alert_channel with the above error as its message.

[00875] Canberra (Nuclear)

[00876] Properties

[00877] FIG. 141 shows example properties for the Canberra (Nuclear) Sensor Adapter. An example of the Canberra (Nuclear) SA in SPML is shown below.

```
sensor_adapter_template CanberraSensorTemplate {
    message_types
        data_message CanberraMessageType
        comm_alarm_message CommAlarmMessageType
    protocol Canberra
```

```
        properties
                Enabled                 integer
                IPAddress               string
                Port                    integer
                PollingRate             double
                ReceiveTimeOut          double
                geoLocation             GeoLocationType
}
sensor_adapter_instance CanberraSensor {
        template CanberraSensorTemplate
        channels
                data_channels
                        ??
                        ??
                alert_channels
                        comm_alarm_channels
localMessages::StandardCommAlarm
        properties
                Enabled = 1
                IPAddress = "64.210.18.24"
                PollingRate = 1.0
                ReceiveTimeOut = 10.0
}
```

[00878] The Canberra sensor typically generates a message containing the following information.

```
type CanberraMessageType {
        Timestamp                       string
        ObservationNumber               integer
        FilteredDoseRate                double
        UnfilteredDoseRate              double
```

```
        MissionDose                    double
        PeakRate                       double
        Temperature                    double
        location                       LocationType
        geoLocation                    GeoLocationType
}
```

[00879] Behavior

[00880] Example typical behavior of the Canberra is provided below.

[00881] Publish policy:

```
policy_template SOME_CONDITIONACTION_template {
        condition
                …
        action
                …
                publish
                        alert : constant Some_Conditionaction_alert_message
                        alertLevel : "RED"
                        referenceChannels :     "RefChannel1~
                                                RefChannel2~
                                                RefChannel3"
                        reference channels (something_channel)
                on channels (alertChannel1),
}
```

[00882] Control Policy:

```
policy_template SOME_CONDITIONACTION_template {
        condition
                …
        action
                …
                control
```

messageToSend : constant
Some_Conditionaction_alert_message
sensors(whatever)
}

[00883] Error Handling

[00884] The Canberra (Nuclear) service will respond to events with appropriate errors. The Canberra (Nuclear) Sensor Adapter will recognize any non-successful call and send out its own alert on an alert_channel with the error as its message.

[00885] Cepheid GeneXpert Sensor Adapter (Bio)

[00886] Properties

[00887] The specified Cepheid GeneXpert Sensor Adapter (Bio)properties can include the Enabled property that describes whether the sensor proxy is enabled or not. The value of the Enabled property is 1.

[00888] An example of the Cepheid GeneXpert Sensor Adapter (Bio) in SPML is shown below.

```
sensor_adapter_template <Whatever>_template {
        message_types
                alert_message AlertMessageType -and|or
                data_message <Whatever>Data -and|or
                comm_alarm_message CommAlarmMessageType
protocol <Whatever>
properties
        Enabled                 integer -and|or-
        IPAddress               string -and|or-
        HTTPGetUrl              string -and|or
        numberToDial            string -and|or
```

```
        tokenID                 string -and|or\
        messageToSend           string -and|or
        recvTimeout             integer -and|or
        port                integer
}
sensor_adapter_instance <Name> {
        template <Whatever>_template
        channels
                input_channels
                        LocalMsgServer::Chem.Alert.FireDepartment -and|or-
                        LocalMsgServer::<some.sortof.channel>
                alert_channels
                        LocalMsgServer::<Whatever>Alert
        properties
                Enabled = 1
                HTTPGetUrl = "<someURL>"
                numberToDial = "555123456"
                tokenID =
"341b34d433ed0b4db7bef7fd57a6786ecb04a4ac8940879
                ac9c211021cd0e9422678b5c5d5d08fa668714697"
                messageToSend = "Your message here"
                recvTimeout = <some # of secs>
}
```

[00889] The Cepheid GeneXpert Sensor Adapter (Bio) can send other messages, including the follow.

```
type Alert|AlarmType {
        TTimestamp string
        alert string
        AlertLevel string
        referenceChannels string
```

```
        buildingLocation BuildingLocationType
        etc
}
```

[00890] Behavior

[00891] The typical behavior of the Cepheid GeneXpert Sensor Adapter (Bio) is provided below as examples.

[00892] Publish policy:

```
policy_template SOME_CONDITIONACTION_template {
        condition
                …
        action
                …
                publish
                        alert : constant Some_Conditionaction_alert_message
                        alertLevel : "RED"
                        referenceChannels : "RefChannel1~
                                            RefChannel2~
                                            RefChannel3"
                        reference channels (something_channel)
                        on channels (alertChannel1),
}
```

[00893] Control Policy:

```
policy_template SOME_CONDITIONACTION_template {
        condition
                …
        action
                …
                control
                        messageToSend : constant
```

Some_Conditionaction_alert_message
sensors(whatever)
}

[00894] Error Handling

[00895] The Cepheid GeneXpert Sensor Adapter (Bio) will respond to events with appropriate errors. The Cepheid GeneXpert Sensor Adapter (Bio) will recognize any non-successful call and send out its own alert on an alert_channel with the error as its message.

[00896] Cepheid Smart Cycler Sensor Adapter (Bio)

[00897] The example properties for Cepheid Smart Cycler Sensor Adapter (Bio) include the Enabled property that describes whether the sensor proxy is enabled or not. The value of the Enabled property is 1 when enabled.

[00898] An example of the Cepheid Smart Cycler Sensor Adapter (Bio) in SPML is shown below.

```
sensor_adapter_template <Whatever>_template {
    message_types
        alert_message AlertMessageType -and|or
        data_message <Whatever>Data -and|or
        comm_alarm_message CommAlarmMessageType
    protocol <Whatever>
    properties
        Enabled                 integer -and|or-
        IPAddress               string -and|or-
        HTTPGetUrl              string -and|or
        numberToDial            string -and|or
        tokenID                 string -and|or
        messageToSend           string -and|or
```

```
                recvTimeout                                integer -and|or
                port                                       integer
    }
    sensor_adapter_instance <Name> {
        template <Whatever>_template
        channels
                input_channels
                        LocalMsgServer::Chem.Alert.FireDepartment -and|or-
                        LocalMsgServer::<some.sortof.channel>
                alert_channels
                        LocalMsgServer::<Whatever>Alert
        properties
                Enabled = 1
                HTTPGetUrl = "<someURL>"
                numberToDial = "555123456"
                tokenID =
"341b34d433ed0b4db7bef7fd57a6786ecb04a4ac8940879
                ac9c211021cd0e9422678b5c5d5d08fa668714697"
                messageToSend = "Your message here"
                recvTimeout = <some # of secs>
    }
```

[00899] The Cepheid Smart Cycler Sensor Adapter (Bio) can send other messages, including the following:

```
    type Alert|AlarmType {
        TTimestamp string
        alert string
        AlertLevel string
        referenceChannels string
        buildingLocation BuildingLocationType
        etc
```

}

[00900] Behavior

[00901] The typical behavior of the Cepheid Smart Cycler Sensor Adapter (Bio) is to etc etc etc

[00902] Examples given below.

[00903] Publish policy:

```
policy_template SOME_CONDITIONACTION_template {
        condition
                …
        action
                …
                publish
                        alert : constant Some_Conditionaction_alert_message
                        alertLevel : "RED"
                        referenceChannels : "RefChannel1~
                                            RefChannel2~
                                            RefChannel3"
                        reference channels (something_channel)
                on channels (alertChannel1),
}
```

[00904] Control Policy:

```
policy_template SOME_CONDITIONACTION_template {
        condition
                …
        action
                …
                control
                        messageToSend : constant
Some_Conditionaction_alert_message
```

sensors(whatever)

}

[00905] Error Handling

[00906] The Cepheid Smart Cycler Sensor Adapter (Bio) service can respond to events with appropriate errors. For example, The Cepheid Smart Cycler Sensor Adapter (Bio) can recognize any non-successful call and send out its own alert on an alert_channel with error as its message.

[00907] Dragon Force Sensor Adapter

[00908] Dragon Force is a GPS based location tracking and communication system that is intended to replace GPS phones, two-way radios, handheld data recorders and incident report pads with a single, handheld PDA device. It is manufactured by Drakontas, LLC.

[00909] Properties

[00910] Example properties specified for Dragon Force Sensor Adapter are described in FIG. 142.

[00911] An example of the Dragon Force Sensor Adapter in SPML is shown below.

```
sensor_adapter_template DragonForce_SA_template {
        message_types
                comm_alarm_message CommAlarmMessageType
        protocol DragonForce
        properties
                Enabled integer
                IPAddress string
                port integer
}
```

```
sensor_adapter_instance DragonForce_SA {
        template DragonForce_SA_template
        channels
                input_channels
                        LocalMsgServer::LocalQTLDataChannel -and|or- \\(example)
                        LocalMsgServer::LocalQTLmCMSDataChannel -and|or-
\\(example)
                        LocalMsgServer::<some.sortof.channel>
                comm_alarm_channels
                        LocalMsgServer::LocalCommAlarmChannel
properties
        Enabled             = 1
        IPAddress           =“11.22.33.212”
        Port                nnnn
```

[00912] Behavior

[00913] Example behaviors of the Dragon Force Sensor Adapter are provided below.

[00914] Publish policy:

```
policy_template SOME_CONDITIONACTION_template {
        condition
                …
        action
                …
                publish
                        alert :  constant Some_Conditionaction_alert_message
                        alertLevel : "RED"
                        referenceChannels : "RefChannel1~
                                             RefChannel2~
                                             RefChannel3"
                        reference channels (something_channel)
```

on channels (alertChannel1),

}

[00915] Control Policy:

policy_template SOME_CONDITIONACTION_template {

condition

...

action

...

control

messageToSend : constant Some_Conditionaction_alert_message

sensors(whatever)

}

[00916] Error Handling

[00917] The Dragon Force Sensor Adapter server will respond to events with errors. The Dragon Force Sensor Adapter can recognize any non-successful call and send out its own alert on an alert_channel with error as its message.\

[00918] GeneXpert Adapter (Bio)

[00919] GeneXpert is an example real-time polymerase chain reaction (RT-PCR) system manufactured by Cepheid. The GeneXpert System is a 1 to 4-site, random access instrument integrating real-time amplification and detection. The system provides PCR test results from a raw sample in 30 minutes or less.

[00920] Connection

[00921] The connections used can be IP-enabled.

[00922] Message Types

[00923] The GeneXpert sensor adapter generates a message of type

GeneXpertMessage. In addition to some standard identification fields, the message incorporates information contained in the following five example message data structures. GeneXpertSampleInfo is a high-level sample identication ID. GeneXpertLabInfo includes laboratory demographic information. GeneXpertAssayInfo includes session and detailed assay product information. GeneXpertAssayResults is a message type that contains information for the sample testing as a whole and includes individual assay results for the sample taken from the GeneXpertIndividualAssayResult message type.

[00924] An example GeneXpertSampleInfo is shown below.

```
type GeneXpertSampleInfo {
        Sample_ID                   tring
        Flow_Rate                   double
        Collection_time             string
        Collection_time_timezone    string
        Location                    LocationType
        geoLocation                 GeoLocationType
}
```

[00925] FIG. 143 shows example message types for GeneXpertSampleInfo.

[00926] The following shows an example of GeneXpertAssayInfo.

GeneXpertAssayInfo

```
type GeneXpertAssayInfo {
        Application_Name string
        File_export_time string
        File_export_time_timezone string
        Report_User_Name string
        Test_Type string
        Assay string
```

Assay_Version string
Assay_Type string
Operator string
Notes string
Start_time string
Start_time_timezone string
End_time string
End_time_timezone string
Reagent_Lot_ID string
Expiration_Date string
Cartridge_SN string
Module_Name string
Module_SN string
Instrument_SN string
SW_Version string
}

[00927] FIG. 144 shows an example GeneXpertAssayInfo Message type.

[00928] GeneXpertIndividualAssayResult

[00929] One instance of this structure is populated for each different assay carried out on the sample as shown below.

type GeneXpertIndividualAssayResult {

DataGroupInUse	integer
Ct	double
EndPt	double
Analyte_Result	string
Probe_Check_Result	string
Curve_Fit	string

}

[00930] FIG. 145 shows example GeneXpertIndivisualAssayResult Message

types. Example GeneXpertAssayResults are shown below.

```
type GeneXpertAssayResults {
        Status string
        Error_Status string
        Test_Result string
        BgGeneXpert IndividualAssayResult
        FAMGeneXpert IndividualAssayResult
        H5SubGeneXpert IndividualAssayResult
        H7SubGeneXpert IndividualAssayResult
        ICGeneXpert IndividualAssayResult
        LIZGeneXpert IndividualAssayResult
        pMat01GeneXpert IndividualAssayResult
        pMat02GeneXpert IndividualAssayResult
        pX01GeneXpert IndividualAssayResult
        pX02GeneXpert IndividualAssayResult
        SPCGeneXpert IndividualAssayResult
}
```

[00931] FIG. 146 shows example GeneXpertAssayResults Data Structure. An example GeneXpertMessage is shown below.

```
type GeneXpertMessage {
        Timestamp                   string
        Timezone                    string
        UUID                        string
        Headline                    string
        Sample_info                 GeneXpertSampleInfo
        Lab_info                    GeneXpertLabInfo
        Assay_info                  GeneXpertAssayInfo
        Assay_results               GeneXpertAssayResults
        }
```

[00932] FIG. 147 shows an example GeneXpertMessage Data Structure. An

example Sensor Adapter Template and Instance is shown below.

```
//The GeneXpert sensor adaptor template in SPML:
sensor_adapter_template GeneXpert_Template {
        message_types
                data_message GeneXpertMessage
                comm_alarm_message CommAlarmMessage
        protocol GeneXpert
        properties
                Enabled                 integer
                SimulatorMode           integer
                PollingRate             double
                //ExportPath            string // yet to be implemented
                Location LocationType
                geoLocation             GeoLocationType
}
```

[00933] FIG. 148 shows example GeneXpertMessage Template properties. An example GeneXpertMessage Template and Instance is shown below.

```
// an instance based on GeneXpert_Template:
sensor_adapter_instance myGeneXpert {
        template GeneXpert_Template
        channels
                data_channels
                        JmsMsgServer::Bio.Data.Cepheid.GeneXpert
                comm_alarm_channels
                        LocalMsgServer::CommAlarms
                        LocalMsgServer::CommAlarmsForJBC2S
                        LocalMsgServer::CommAlarmsForTASS
                        JmsMsgServer::SPM.Alert.Sensor.Comm.Alarms
properties
        Enabled = 1
```

```
        SimulatorMode = 0
        PollingRate = 5.0
        //ExportPath = "C:\\GeneXpert\\Export\\"
        Location.Location_description = "ViaLogy"
        Location.Location_street_address = "2400 Lincoln Ave"
        Location.Location_city = "Altadena"
        Location.Location_state = "CA"
        Location.Location_zip_code = "91001"
        Location.Location_country = "USA"
        Location.Location_building = "BTC"
        Location.Location_floor = "1"
        Location.Location_room = "DevLab"
        geoLocation.longitude = -118.159705
        geoLocation.latitude = 34.186906
        geoLocation.altitude = 0.0
}
```

[00934] Control Policy

[00935] An example control policy template is shown below.

```
policy_template SOME_CONDITIONACTION_template {
        condition
                ...
        action
                ...
                control
                        messageToSend : constant
Some_Conditionaction_alert_message
                        sensors(whatever)
}
```

[00936] Error Handling

[00937] The GeneXpert sensor adaptor can respond to errors with appropriate

error messages.

[00938] GPS (GeoLocation)

[00939] Properties

[00940] Example GPS (GeoLocation) properties include 'Enabled' property that describes whether the sensor proxy is enabled or not. The value assigned to 'Enabled' is 1.

[00941] An example of the GPS (GeoLocation) in SPML is shown below.

```
sensor_adapter_template <Whatever>_template {
        message_types
                alert_message AlertMessageType -and|or
                data_message <Whatever>Data -and|or
                comm_alarm_message CommAlarmMessageType
        protocol <Whatever>
        properties
                Enabled                         integer -and|or-
                IPAddress                       string -and|or-
                HTTPGetUrl                      string -and|or
                numberToDial                    string -and|or
                tokenID                         string -and|or
                messageToSend                   string -and|or
                recvTimeout                     integer -and|or
                port                            integer
}
sensor_adapter_instance <Name> {
        template <Whatever>_template
        channels
                input_channels
                        LocalMsgServer::Chem.Alert.FireDepartment -and|or-
                        LocalMsgServer::<some.sortof.channel>
```

```
            alert_channels
                LocalMsgServer::<Whatever>Alert
        properties
            Enabled = 1
            HTTPGetUrl = “<someURL>”
            numberToDial = “555123456”
            tokenID =
            “341b34d433ed0b4db7bef7fd57a6786ecb04a4ac8940879
            ac9c211021cd0e9422678b5c5d5d08fa668714697”
            messageToSend = “Your message here”
            recvTimeout = <some # of secs>
}
```

[00942] GPS (GeoLocation) can send other messages including the one shown below.

```
type Alert|AlarmType {
        TTimestamp string
        alert string
        AlertLevel string
        referenceChannels string
        buildingLocation BuildingLocationType
        etc
}
```

[00943] Example behaviors of the GPS (GeoLocation) is shown below.

[00944] Publish Policy:

```
policy_template SOME_CONDITIONACTION_template {
        condition
            …
        action
            …
            publish
```

```
                    alert : constant Some_Conditionaction_alert_message
                    alertLevel : "RED"
                    referenceChannels : "RefChannel1~
                                        RefChannel2~
                                        RefChannel3"
            reference channels (something_channel)
        on channels (alertChannel1),
}
```

[00945] Control Policy:

```
policy_template SOME_CONDITIONACTION_template {
        condition
                ...
        action
                ...
                control
                        messageToSend : constant
Some_Conditionaction_alert_message
                        sensors(whatever)
}
```

[00946] Error Handling

[00947] The GPS (GeoLocation) service can respond to events with appropriate errors. For example, the GPS (GeoLocation) can recognize any non-successful call and send out its own alert on an alert_channel with the appropriate error as its message.

[00948] Hach Sensor Adapter (Bio)

[00949] Hach manufactures a ranges of chemical sensors, including turbidimeters, gas monitors, pH meters, water quality tests, etc. SPM can support these sensors.

[00950] Properties

[00951] Example properties of the specified Hach Sensor Adapter (Bio) include the 'Enabled' property that describes whether the sensor proxy is enabled or not. The value assigned to the 'Enabled' property is 1.

[00952] An example of the Hach Sensor Adapter (Bio) in SPML is shown below.

```
sensor_adapter_template hach_template {
        message_types
                alert_message AlertMessageType -and|or
                data_message <Whatever>Data -and|or
                comm_alarm_message CommAlarmMessageType
        protocol <Whatever>
        properties
                Enabled                 integer -and|or-
                IPAddress               string -and|or-
                HTTPGetUrl              string -and|or
                numberToDial            string -and|or
                tokenID                 string -and|or
                messageToSend           string -and|or
                recvTimeout             integer -and|or
                port                    integer
}
sensor_adapter_instance <Name> {
        template <Whatever>_template
        channels
        input_channels
                LocalMsgServer::Chem.Alert.FireDepartment -and|or-
                LocalMsgServer::<some.sortof.channel>
        alert_channels
```

```
        LocalMsgServer::<Whatever>Alert
    properties
        Enabled = 1
        HTTPGetUrl = "<someURL>"
        numberToDial = "555123456"
        tokenID =
        "341b34d433ed0b4db7bef7fd57a6786ecb04a4ac8940879
        ac9c211021cd0e9422678b5c5d5d08fa668714697"
        messageToSend = "Your message here"
        recvTimeout = <some # of secs>
}
```

[00953] Messages generated by Hach Sensor Adapters include some of the following message fields:

```
type hachMessageType {
    UUID string
    Timestamp                   string
    Timezone                    string
    Sample_Timestamp            string
    Sample_Timezone             string
    Chlorine                    double|
    Conductivity                double|
    TOC                         double|
    Turbidity                   double|
    pH                          double
    Location                    LocationType
    geoLocation                 GeoLocationType
}
```

[00954] Behavior

[00955] Example behavior of the Hach Sensor Adapter (Bio) are shown below.

[00956] Publish policy:

```
policy_template SOME_CONDITIONACTION_template {
        condition
                …
        action
                …
                publish
                        alert : constant Some_Conditionaction_alert_message
                        alertLevel : "RED"
                        referenceChannels : "RefChannel1~
                                            RefChannel2~
                                            RefChannel3"
                        reference channels (something_channel)
                on channels (alertChannel1),
}
```

[00957] Control Policy:

```
policy_template SOME_CONDITIONACTION_template {
        condition
                …
        action
                …
                control
                        messageToSend : constant
Some_Conditionaction_alert_message
                        sensors(whatever)
}
```

[00958] Error Handling

[00959] The Hach Sensor Adapter (Bio) service can respond to events with appropriate errors. The Hach Sensor Adapter (Bio) can recognize any non-successful call and send out its own alert on an alert_channel with apprpriate

error as its message.

[00960] Honeywell-Midas Sensor Adapter (Chem)

[00961] The Honeywell-Midas gas detector is an extractive gas sampling system that draws a sample locally or from a remote point to a sensor cartridge that is located inside the detector's chassis. A number of toxic, flammable and oxygen gas sensor cartridges are available that enable detection of different gases encountered in industrial settings. Individual cartridges are assigned can detect from one to four gases.

[00962] FIG. 149 shows example gases detected by the Honeywell-Midas sensor adapter.

[00963] Connection

[00964] The connections used by the Honeywell-Midas sensor adapter include Modbus/TCP outputs for signal and service connectivity.

[00965] Message Types

[00966] An example message type for this sensor adapter is shown below.

```
type MidasMessage {
        Timestamp string
        Timezone string
        UUID string
        NameOfDetectedGas string
        Temperature_Celsius integer
        Flowrate_cc_per_min integer
        Concentration double
        buildingLocation BuildingLocationType
        alternateLocation AlternateLocationType
        geoLocation GeoLocationType
```

```
}
```

[00967] Sensor Adapter Template and Instance

[00968] An example sensor adapter template and instance is shown below.

```
// The Honeywell Midas sensor adapter template
//
sensor_adapter_template Midas_Template {
        message_types
                data_message MidasMessage
                  omm._alarm_message CommAlarmMessage
        protocol Midas
        properties
                Enabled integer
                SimulatorMode integer
                IPAddress string
                port integer
                PollingRate double
                buildingLocation BuildingLocationType
                geoLocation GeoLocationType
}
// An instance using the Midas_Template
//
sensor_adapter_instance myMidas {
        template Midas_Template
        channels
                data_channels
                        myLocalMsgServer::Chem.Data.Honeywell.Gas.Chlorine
                        myJmsMsgServer::Chem.Data.Honeywell.Gas.Chlorine
                  omm._alarm_channels
                        myLocalMsgServer::CommAlarms
                        myLocalMsgServer::CommAlarmsForTASS
```

```
                    myLocalMsgServer::CommAlarmsForJBC2S
                    myJmsMsgServer::SPM.Alert.Sensor.Comm.Alarms
        properties
            Enabled = 1
            SimulatorMode = 0
            IPAddress = "64.210.nnn.nnn"
            port = 502
            PollingRate = 1.0
            buildingLocation.Location_building = "myBldgName"
            buildingLocation.Location_room = "myLab"
            buildingLocation.Location_floor = "myFloor"
            geoLocation.longitude = -118.159705
            geoLocation.latitude = 34.186906
            geoLocation.altitude = 0.0
}
```

[00969] Example properties of the Honeywell-Midas Sensor Adapter (Chem) include the 'Enabled' property that describes whether the sensor proxy is enabled or not. The value assigned to the 'Enabled' property is 1.

[00970] FIG. 150 shows example properties of the Honeywell-Midas Sensor Adapter.

[00971] Behavior

[00972] The typical behavior of the Honeywell-Midas gas desc is to etc etc etc

[00973] Examples given below.

[00974] Publish policy:

```
policy_template SOME_CONDITIONACTION_template {
    condition
        …
    action
```

```
                …
                publish
                        alert : constant Some_Conditionaction_alert_message
                        alertLevel : "RED"
                        referenceChannels : "RefChannel1~
                                                RefChannel2~
                                                RefChannel3"
                        reference channels (something_channel)
                on channels (alertChannel1),
}
```

[00975] Control Policy:

```
policy_template SOME_CONDITIONACTION_template {
        condition
                …
        action
                …
                control
                        messageToSend : constant
Some_Conditionaction_alert_message
                        sensors(whatever)
}
```

[00976] Error Handling

[00977] The Honeywell-Midas Sensor Adapter (Chem) service can respond to events with appropriate errors: The Honeywell-Midas Sensor Adapter (Chem) can recognize any non-successful call and send out its own alert on an alert_channel with the appropriate error as its message.

[00978] ICD-Compatible Sensors

[00979] Many sensors are employed in installations where the ICD message

schema has been implemented. Examples of these are TASS and JBC2S. The ICD_Template sensor adapter template which conforms to the ICD-message schema invokes the SimpleICD C++ protocol, which make use of the message types listed below.

[00980] Message Types

[00981] The ICD schema message types consists of a number of high level message types that are built up from lower level message types. These lower level message types may in turn include other message types or they may simply consist of one or more fields (lowest-level properties). A message type such as DetectionType may employ a boolean switch which determines in a given instance whether its data will be included in the higher message type.

[00982] Example top-Level Message Type Hierarchies are show below.

DeviceDetectionReport
    DeviceDetectionRecordType
        DeviceIdentificationType
            <fields>
        DetectionType <boolean>
            <fields>
        EventAssessmentType
            <fields>
CameraStatusReport
    DeviceIdentificationType
    CameraStatusType
        <fields>
    LocationType1
        LocationTypeType
            GeodeticLocationType

<fields>
<fields>
DetailsType
<fields>
DeviceStatusReport
DeviceIdentificationType
DeviceStatusType
LocationType1
LocationTypeType
GeodeticLocationType
<fields>
PlatformStatusReportBase
PlatformIdentificationType
PlatformStatusType
LocationType1
LocationTypeType
GeodeticLocationType
<fields>

[00983] Message Type Declarations

[00984] Most of the fields include samples of specific values for the sake of illustration.

type PlatformStatusType {
DeviceState string = "OK"
CommunicationState string = "OK"
UpdateTime string <Zone="GMT"> = "8:00 PM"
}
type PlatformIdentificationType {
DeviceName string = "SPM-001"
DeviceCategory string = "C2 Node"
DeviceType string = "Sensor Policy Engine"
}

```
type DeviceIdentificationType {
        DeviceName string = "DEV-001"
        DeviceCategory string = "Sensor"
        DeviceType string = "Generic sensor"
}
type GeodeticLocationType {
        Latitude string <Units="Decimal Degrees"> = ""
        Longitude string <Units="Decimal Degrees"> = ""
        Altitude string <Units="Meters" 'Reference'="MSL"> = ""
}
type LocationTypeType {
        GeodeticLocation GeodeticLocationType <Datum="WGS84">
}
type LocationType1 {
        LocationType LocationTypeType
        UpdateTime string <Zone="GMT"> = "Teatime"
}
type PlatformStatusReportBase {
        PlatformIdentification PlatformIdentificationType
        Status PlatformStatusType
        Location LocationType1
}
type DeviceStatusType {
        DeviceState string = "OK"
        CommunicationState string = "OK"
UpdateTime string <Zone="GMT"> = "Lunchtime"
}
type DetailsType {
        Range string <Units="Meters"> = "1000"
        ElevationAngle string <Units="Degrees"> = "45"
        Azimuth string <Units="Degrees"> = "120"
```

```
        FieldOfView string <Units="Degrees"> = "50"
}
type CameraStatusType {
        DeviceState string = "OK"
        CommunicationState string = "OK"
        UpdateTime string <Zone="GMT"> = "Closing time"
}
type CameraStatusReport {
        DeviceIdentification DeviceIdentificationType
        Status CameraStatusType
        Location LocationType1
        Details DetailsType
}
type DeviceStatusReport {
        DeviceIdentification DeviceIdentificationType
        Status DeviceStatusType
        Location LocationType1
}
type DetectionType {
        ID string = "SPM001"
        DetectionEvent string = "Biological Agent"
        Details string = "Evacuation order issued"
        UpdateTime string <Zone="GMT"> = "Midnight"
}
type EventAssessmentType {
        Assessment string = "Positive"
        Annotation string = "Evacuation order issued"
}
type DeviceDetectionRecordType {
        DeviceIdentification DeviceIdentificationType
        Detection DetectionType <Assessed="true">
```

```
        EventAssessment EventAssessmentType
}
type DeviceDetectionReport {
        DeviceDetectionRecord DeviceDetectionRecordType
}
```

[00985] Sensor Adapter Template and Instance

```
// The ICD sensor adapter template
sensor_adapter_template ICD_Template {
        message_types
                comm_alarm_message CommAlarmMessage
        protocol SimpleICD
        properties
                Enabled integer
                SimulatorMode integer
                IPAddress string
                port integer
                PollingRate double
                ReceiveTimeOut double
                geoLocation GeoLocationType
}
// An instance based on the ICD_Template
sensor_adapter_instance JBC2S {
        template ICD_Template
        channels
                input_channels
                        LocalMsgServer::AlertsForJBC2S
                        LocalMsgServer::CommAlarmsForJBC2S
                        LocalMsgServer::QTLPolicyAlerts
                comm_alarm_channels
                        LocalMsgServer::CommAlarms
                        LocalMsgServer::CommAlarmsForTASS
```

```
                    JmsMsgServer::SPM.Alert.Sensor.Comm.Alarms
        properties
                Enabled = 1
                SimulatorMode = 0
                IPAddress = "64.210.18.81"
                port = 15005
                PollingRate = 1.0
                ReceiveTimeOut = 120.0
                geoLocation.longitude = -118.159705
                geoLocation.latitude = 34.186906
                geoLocation.altitude = 0.0
}
```

[00986] FIG. 151 shows example ICD-compatible message properties. These are the currently specified lowest-level properties in the ICD-compatible message types. Most properties appear in more than one message type.

[00987] Inova

[00988] Inova is a manufacturer of digital signage and alerting software. An SPM policy can publish or broadcast a message to an Inova system which will typically present the message as a marquee in the recipient's computer monitor or other display.

[00989] Connection

[00990] Inova devices are IP-enabled.

[00991] Message Types

[00992] The Innova sensor adapter employs the generic CommAlarmMessage message type.

[00993] An example Sensor Adapter Template and Instance is shown below.

```
// An example of the Inova sensor adapter in SPML:
sensor_adapter_template Inova_Template {
        message_types
                comm_alarm_message CommAlarmMessageType
        protocol Inova
        properties
        Enabled                                 integer
        IPAddress                               string
        port                                    integer
        PollingRate                             double
        userName                                string
        password                                string
        defaultMessage                          string
        commandDuration                         integer
        defaultColor                            string
        commandMessage                          string
        currentMessage                          integer
        commandColor                            string
        commandRefreshTime                      string
        commandFontSize                         string
        commandDisplayMethod                    string
        useDefaultMessage                       integer
}
// an instance based on Inova_Template
//
sensor_adapter_instance myInova {
        template Inova_Template
        channels
                comm_alarm_channels
                        LocalMsgServer::CommAlarms
                        JMSMsgServer::SPM.Alert.Sensor.Comm.Alarms
```

```
        properties
                Enabled = 1
                IPAddress = "64.210.18.104"
                port = 23
                PollingRate = 1.0
                userName = "admin"
                password = "1n0v@"
                defaultMessage = "Welcome to SPM Dev Lab -- ViaLogy,
                California, USA"
                commandDuration = 60
                defaultColor = "green"
                commandMessage = ""
                currentMessage = 100
                commandColor = "red"
                commandRefreshTime = "5"
                commandFontSize = "14"
                commandDisplayMethod = "ribbon_left"
                useDefaultMessage = 1
}}
```

[00994] FIG. 152 shows example properties for the Inova display.

[00995] Behavior

[00996] Examples of behavior of the Inova sensor adapter are shown below.

[00997] Publish policy:

```
policy_template SOME_CONDITIONACTION_template {
        condition
                …
        action
                …
                publish
                        alert : constant Some_Conditionaction_alert_message
```

```
                    alertLevel : "RED"
                    referenceChannels : "RefChannel1~
                                        RefChannel2~
                                        RefChannel3"
                    reference channels (something_channel)
            on channels (alertChannel1),
}
```

[00998] Control Policy:

```
policy_template SOME_CONDITIONACTION_template {
    condition
        …
    action
        …
        control
            messageToSend : constant
Some_Conditionaction_alert_message
            sensors(whatever)
}
```

[00999] Error Handling

[001000] The Inova server can respond to events with appropriate errors. For example, the Inova sensor adapter can recognize any non-successful call and send out its own alert on an alert_channel with the error as its message.

[001001] LCD (Chemical)

[001002] Lightweight Chemical Detector (LCD) is a lightweight detector of chemical warfare agents that can be hand-carried or attached to a soldier's uniform. Manufactured by Smiths Technologies, the LCD is designed to detect a variety of chemical agents at or below attack concentrations.

[001003] Connection

[001004] Connections for LCD include USB or RS485.

[001005] Message Types

[001006] The LCD sensor adapter allows for the generation of two message types. One is the common alarm message. The other is an LCD-specific data message with fields for a number of chemical warfare agents and sensor status readings. The prefixes “g” rand “h” refer generically to nerve agents and hematolytic or blistering agents, respectively.

[001007] LCDMessage

[001008] An example LCD message is shown below.

```
type LCDMessage {
        systemId integer
        gBar integer
        hBar integer
        gAgentType integer
        hAgentType integer
        sysStatus1 integer
        sysStatus2 integer
        gDose double
        hDose double
        gHighestAgentType integer
        hHighestAgentType integer
        seconds integer
        minutes integer
        hours integer
        dayOfMonth integer
        month integer
        year integer
        vcc integer
```

inletFanCurrent integer
recircFanCurrent integer
waterIntake integer
pressure integer
temperature integer
batteryVoltage integer
positiveHT integer
negativeHT integer
gNoise integer
hNoise integer
positiveDacOffset integer
positiveDacScalar integer
negativeDacOffset integer
negativeDacScalar integer
dspParamCR integer
coronaControlModes integer
dspParamPC integer
dspParamDP integer
dspParamDN integer
dspParamGP integer
dspParamGN integer
coronaNumber integer
coronaFiringCount integer
operatingCycleCount integer
lowVxMode integer
cycleTime integer
gMobilityCal integer
hMobilityCal integer
negativeSpectrumData integer []
positiveSpectrumData integer []
Timestamp string

```
        textBlock string
}
```

[001009] FIGS. 153 and 154 show example properties for a lightweight chemical detector message data structure.

```
Sensor Adaptor Template and Instance
// The LCD sensort adapter template
//
sensor_adapter_template LCD_Template {
        message_types
                data_message LCDMessage
//alert_message LCDMessage
                comm_alarm_message CommAlarmMessage
        protocol Lcd32e
        properties
                Enabled integer
                SimulatorMode integer
                IPAddress string
                port integer
                PollingRate double
                ReceiveTimeOut double
                geoLocation GeoLocationType
}
// An instance based on LCD_Template
//
sensor_adapter_instance myLCD {
template LCD_Template
        channels
                data_channels
LocalMsgServer::Chem.Data.LCD3.Gas
                //alert_channels
```

```
//LocalMsgServer::LocalLCDAlertChannel
        comm_alarm_channels
            myLocalMsgServer::CommAlarms
            myLocalMsgServer::CommAlarmsForTASS
            myLocalMsgServer::CommAlarmsForJBC2S
            myJmsMsgServer::SPM.Alert.Sensor.Comm.Alarms
    properties
        Enabled = 1
        SimulatorMode = 0 //using real data
        IPAddress = "64.210.18.24"
        port = 2106
        PollingRate = 1.0
        ReceiveTimeOut = 5.0
        geoLocation.longitude = -118.159705
        geoLocation.latitude = 34.186906
        geoLocation.altitude = 0.0
}
```

[001010] Lenel Sensor Adapter (Access Control)

[001011] Properties

[001012] Examples of Lenel Sensor Adapter (Access Control) properties include the 'Enabled' property that describes whether the sensor proxy is enabled or not. The value assigned to the 'Enabled' property is 1.

[001013] An example of the Lenel Sensor Adapter (Access Control in SPML is shown below.

```
sensor_adapter_template <Whatever>_template {
    message_types
        alert_message AlertMessageType -and|or
        data_message <Whatever>Data -and|or
        comm_alarm_message CommAlarmMessageType
```

```
        protocol <Whatever>
        properties
                Enabled                         integer -and|or-
                IPAddress                       string -and|or-
                HTTPGetUrl                      string -and|or
                numberToDial                    string -and|or
                tokenID                         string -and|or
                messageToSend                   string -and|or
                recvTimeout                     integer -and|or\
                port                            integer
}
sensor_adapter_instance <Name> {
        template <Whatever>_template
        channels
                input_channels
                        LocalMsgServer::Chem.Alert.FireDepartment -and|or-
                        LocalMsgServer::<some.sortof.channel>
                alert_channels
                        LocalMsgServer::<Whatever>Alert
        properties
                Enabled = 1
                HTTPGetUrl = "<someURL>"
                numberToDial = "555123456"
                tokenID =
                "341b34d433ed0b4db7bef7fd57a6786ecb04a4ac8940879
                ac9c211021cd0e9422678b5c5d5d08fa668714697"
                messageToSend = "Your message here"
                recvTimeout = <some # of secs>
}
```

[001014] The Lenel Sensor Adapter (Access Control) can send other messages including the following example.

```
type Alert|AlarmType {
        TTimestamp                  string
        alert                       string
        AlertLevel                  string
        referenceChannels           string
        buildingLocation            BuildingLocationType
        etc
}
```

[001015] Behavior

[001016] The typical behavior of the Lenel Sensor Adapter (Access Control) is shown below using examples.

[001017] Publish policy:

```
policy_template SOME_CONDITIONACTION_template {
        condition
                …
        action
                …
                publish
                        alert : constant Some_Conditionaction_alert_message
                        alertLevel : "RED"
                        referenceChannels : "RefChannel1~
                                            RefChannel2~
                                            RefChannel3"
                        reference channels (something_channel)
                on channels (alertChannel1),
}
```

[001018] Control Policy:

```
policy_template SOME_CONDITIONACTION_template {
        condition
```

```
            ...
        action
            ...
            control
                messageToSend : constant
Some_Conditionaction_alert_message
                sensors(whatever)
}
```

[001019] Error Handling

[001020] The Lenel Sensor Adapter (Access Control) service can respond to events with appropriate errors. For example, The Lenel Sensor Adapter (Access Control) can recognize any non-successful call and send out its own alert on an alert_channel with the appropriate error as its message.

[001021] Magnetic Compass Sensor Adapter

[001022] Examples of Magnetic Compass Sensor Adapter properties include the 'Enabled' property that indicates whether the sensor proxy is enabled or not. The value assigned to the 'Enabled' value is 1.

[001023] An example of the Magnetic Compass Sensor Adapter in SPML is shown below.

```
sensor_adapter_template <Whatever>_template {
        message_types
            alert_message AlertMessageType -and|ordata_
            message <Whatever>Data -and|or
            comm_alarm_message CommAlarmMessageType
        protocol <Whatever>
        properties
            Enabled                integer -and|or-
```

```
        IPAddress               string -and|or-
        HTTPGetUrl              string -and|or
        numberToDial            string -and|or
        tokenID                 string -and|or
        messageToSend           string -and|or
        recvTimeout             integer -and|or
        port                    integer
}
sensor_adapter_instance <Name> {
    template <Whatever>_template
    channels
        input_channels
            LocalMsgServer::Chem.Alert.FireDepartment -and|or-
            LocalMsgServer::<some.sortof.channel>
        alert_channels
            LocalMsgServer::<Whatever>Alert
    properties
        Enabled = 1
        HTTPGetUrl = "<someURL>"
        numberToDial = "555123456"
        tokenID =
        "341b34d433ed0b4db7bef7fd57a6786ecb04a4ac8940879
        ac9c211021cd0e9422678b5c5d5d08fa668714697"
        messageToSend = "Your message here"
        recvTimeout = <some # of secs>
}
```

[001024] The Magnetic Compass Sensor Adapter can send other messages including the following.

```
type Alert|AlarmType {
    TTimestamp string
```

alert string
AlertLevel string
referenceChannels string
buildingLocation BuildingLocationType
etc
}

[001025] Behavior

[001026] The typical behavior of the Magnetic Compass Sensor Adapter is to etc etc etc

[001027] Examples given below.

[001028] Publish policy:

```
policy_template SOME_CONDITIONACTION_template {
        condition
                …
        action
                …
                publish
                        alert : constant Some_Conditionaction_alert_message
                        alertLevel : "RED"
                        referenceChannels : "RefChannel1~
                                                RefChannel2~
                                                RefChannel3"
                        reference channels (something_channel)
                on channels (alertChannel1),
}
```

[001029] Control Policy:

```
policy_template SOME_CONDITIONACTION_template {
        condition
                …
```

```
        action
                ...
                control
                        messageToSend : constant
Some_Conditionaction_alert_message
                        sensors(whatever)
}
```

[001030] Error Handling

[001031] The Magnetic Compass Sensor Adapter service can respond to events with appropriate errors. For example, The Magnetic Compass Sensor Adapter will recognize any non-successful call and send out its own alert on an alert_channel with the appropriate error as its message.

[001032] Modbus

[001033] Connection

[001034] Message Types

[001035] The Modbus sensor adapter has one message type that is peculiar to itself. An example is shown below.

```
type ModbusRegisterMessage {
        Timestamp string
        RegisterValues Long [] // what are these?
        Location LocationType
        geoLocation GeoLocationType
}
```

[001036] FIG. 155 shows example properties of the ModbusRegisterMessage Data Structure. An example Sensor Adapter Template and Instance is shown below.

```
// The following message are generated by the Modbus Sensor adapter
```

```
message_types
        data_message ModbusRegisterMessage
        comm_alarm_message CommAlarmMessage
        protocol MODBUS
        properties
                Enabled integer
                SimulatorMode integer
                IPAddress string
                Port integer
                PollingRate double
                MBAddress integer // Modbus RTU only.
                ModBusType string
                MBStartAddress integer []
                MBCount integer
                MBFunction string
                MBSlave integer
}
// An instance based on Modbus_Template
sensor_adapter_instance ITrans {
        template Modbus_Template
        channels
                comm_alarm_channels
                        LocalMsgServer::CommAlarms
                        LocalMsgServer::CommAlarmsForJBC2S
                        LocalMsgServer::CommAlarmsForTASS
                        JmsMsgServer::SPM.Alert.Sensor.Comm.Alarms
        properties
                Enabled = 1
                SimulatorMode = 0
                IPAddress = “64.210.18.21”
                Port = 2101
```

```
        PollingRate = 1.0
        MBAddress = 1
        ModBusType = “MODBUS_RTU”
        MBStartAddress = { 100 200 }
        MBCount = 6
        MBFunction = “READ_HOLDING_REGISTERS”
        MBSlave = 1
}
```

[001037] FIG. 156 shows example properties of the Modbus sensor adapter with typical values.

[001038] Ortec (Radiation)

[001039] Connection

[001040] Messages

[001041] The Ortec Sensor Adapter messages include the following fields.

```
type OrtecMessage {
        Timestamp string
        gammaStr string
        gammaCount double
        neutronStr string
        neutronCount double
        doseStr string
        doseAmount double
        numNuclidesFound integer
        foundNuclides string []
        foundNuclidesList string
        location LocationType
        geoLocation GeoLocationType
}
```

[001042] FIG. 157 shows example properties of Ortec Sensor Adapter

Message Data Structure.

[001043] An example of Sensor Adapter Template and Instance is shown below.

```
// The Ortec Sensor Adapter Template in SPML:
//
sensor_adapter_template Ortec_Template {
        message_types
                data_message OrtecMessage
                comm_alarm_message CommAlarmMessage
        protocol Ortec
        properties
                Enabled integer
                SimulatorMode integer
                Polling Rate double
                Enabled Integer
                Location LocationType
                geoLocation GeoLocationType
}
//An instance based on Ortec_Template
sensor_adapter_instance myOrtec {
        template Ortec_Template
        channels
                data_channels
                        MyLocalMsgServer::Rad.ORTEC.DetectiveEx.Data
                comm_alarm_channels
                        MyLocalMsgServer::CommAlarms
                        MyLocalMsgServer::CommAlarmsForJBC2S
                        MyLocalMsgServer::CommAlarmsForTASS
                        MyJmsMsgServer::SPM.Alert.Sensor.Comm.Alarms
        properties
```

Enabled = 1
SimulatorMode = 0
PollingRate = 2.0
Location.Location_description = "myCompany"
Location.Location_street_address = "myStreetAddress"
Location.Location_city = "myTown"
Location.Location_state = "CA"
Location.Location_zip_code = "99999"
Location.Location_country = "USA"
Location.Location_building = "myBldgName"
Location.Location_floor = "myFloorNum"
Location.Location_room = "myLab"
geoLocation.longitude = -117.247602
geoLocation.latitude = 32.707571
geoLocation.altitude = 100.00
}

[001044] FIG. 158 shows example properties of Ortec (Radiation).

[001045] Behavior

[001046] The Ortec Detective-EX Radioisotope Identifier is used to detect gamma rays and neutron emissions and distinguish those that are due to the presence of man-made radioactive material from those due to background causes. The detection reports the gamma ray dose rate, neutron count rate and identifies and classifies which nuclides (radioisotopes) are responsible for the emissions. Information provided by the detector includes:

- General help messages such as:
  - Count rate consistent with background
  - Elevated radiation field

- Possible beta emitter
- Medical positron emitter
- Possible nuclear material

(The general messages are reported in the SA field.)

• Primary identification message of the form: Found <Class>(#), where <Class> includes the following classifications and (#) refers to the number of nuclides (radioisotopes) identified in that class. Classes include but are not limited to:

- Medical
- Industrial
- NORM (Naturally Occurring Radioactive Material)
- Bremsstrahlung (secondary radiation due to beta decay)

[001047] The Class is reported in the SPM sensor adapter as the foundNuclidesList property, the number of nuclides as the NumNuclides property, and the names of the individual nuclides in the foundNuclides array.

[001048] Publish policy:

```
policy_template SOME_CONDITIONACTION_template {
    condition
        …
    action
        …
        publish
            alert : constant Some_Conditionaction_alert_message
            alertLevel : "RED"
            reference channels (something_channel)
        on channels (alertChannel1),
}
```

[001049] Control Policy:

```
policy_template SOME_CONDITIONACTION_template {
        condition
                ...
        action
                ...
                control
                        messageToSend : constant
Some_Conditionaction_alert_message
                        sensors(whatever)
}
```

[001050] Error Handling

[001051] The Ortec (Radiation) SA service can respond to events with appropriate errors. For example, the Ortec (Raditaiotn) SA can recognize any non-successful call and send out its own alert on an alert_channel with the appropriate message.

[001052] Seismic Sensor Adapter

[001053] Properties

[001054] Exampler properties of the Seismic Sensor Adapter include the 'Enabled' property that indicate whether the sensor proxy is enabled or not. The value assigned to the 'Enabled' property is 1.

[001055] An example of the Seismic Sensor Adapter is SPML is shown below.

```
sensor_adapter_template <Whatever>_template {
        message_types
                alert_message AlertMessageType -and|or
                data_message <Whatever>Data -and|or
```

```
                comm_alarm_message CommAlarmMessageType
        protocol <Whatever>
        properties
                Enabled integer -and|or-
                IPAddress string -and|or-
                HTTPGetUrl string -and|ornumberToDial
                string -and|ortokenID
                string -and|ormessageToSend
                string -and|orrecvTimeout
                integer -and|orport
                integer
}
sensor_adapter_instance <Name> {
        template <Whatever>_template
        channels
                input_channels
                        LocalMsgServer::Chem.Alert.FireDepartment -and|or-
                        LocalMsgServer::<some.sortof.channel>
                alert_channels
                        LocalMsgServer::<Whatever>Alert
        properties
                Enabled = 1
                HTTPGetUrl = "<someURL>"
                numberToDial = "555123456"
                tokenID =
                "341b34d433ed0b4db7bef7fd57a6786ecb04a4ac8940879
                ac9c211021cd0e9422678b5c5d5d08fa668714697"
                messageToSend = "Your message here"
                recvTimeout = <some # of secs>
}
```

[001056] The Seismic Sensor Adapter can send other messages including

the one shown below.

```
type Alert|AlarmType {
        TTimestamp string
        alert string
        AlertLevel string
        referenceChannels string
        buildingLocation BuildingLocationType
        etc
}
```

[001057] Behavior

[001058] Example behaviors of the Seismic Sensor Adapter are shown below.

[001059] Publish policy:

```
policy_template SOME_CONDITIONACTION_template {
        condition
                …
        action
                …
                publish
                        alert : constant Some_Conditionaction_alert_message
                        alertLevel : "RED"
                        referenceChannels : "RefChannel1~
                                            RefChannel2~
                                            RefChannel3"
                        reference channels (something_channel)
                on channels (alertChannel1),
}
```

[001060] Control Policy:

```
policy_template SOME_CONDITIONACTION_template {
```

```
        condition
                ...
        action
                ...
                control
                        messageToSend : constant
Some_Conditionaction_alert_message
                        sensors(whatever)
}
```

[001061] Error Handling

[001062] The Seismic Sensor Adapter server can respond to events with appropriate errors. For example, the Seismic Sensor Adapter can recognize any non-successful call and send out its own alert on an alert_channel with the appropriate error as its message.

[001063] Symbol Sensor Adapter (RFID)

[001064] Symbol Sensor Adapter is an RFID device. Example properties of the Symbol Sensor Adapter (RFID) include the 'Enabled' property that indicate whether the sensor proxy is enabled or not. The value assigned to the 'Enabled' property is 1.

[001065] An example of the Symbol Sensor Adapter (RFID) in SPML is shown below.

```
sensor_adapter_template <Whatever>_template {
        message_types
                alert_message AlertMessageType -and|or
                data_message <Whatever>Data -and|or
                comm_alarm_message CommAlarmMessageType
        protocol <Whatever>
```

```
        properties
                Enabled integer -and|or-
                IPAddress string -and|or-
                HTTPGetUrl string -and|ornumberToDial
                string -and|ortokenID
                string -and|ormessageToSend
                string -and|orrecvTimeout
                integer -and|orport
                integer
}
sensor_adapter_instance <Name> {
        template <Whatever>_template
        channels
                input_channels
                        LocalMsgServer::Chem.Alert.FireDepartment -and|or-
                        LocalMsgServer::<some.sortof.channel>
                alert_channels
                        LocalMsgServer::<Whatever>Alert
        properties
                Enabled = 1
                HTTPGetUrl = "<someURL>"
                numberToDial = "555123456"
                tokenID =
                "341b34d433ed0b4db7bef7fd57a6786ecb04a4ac8940879
                ac9c211021cd0e9422678b5c5d5d08fa668714697"
                messageToSend = "Your message here"
                recvTimeout = <some # of secs>
}
```

[001066] The Symbol Sensor Adapter (RFID) can send other messages including the following.

[001067] Publish policy:

```
policy_template SOME_CONDITIONACTION_template {
        condition
                …
        action
                …
                publish
                        alert : constant Some_Conditionaction_alert_message
                        alertLevel : "RED"
                        referenceChannels : "RefChannel1~
                                                RefChannel2~
                                                RefChannel3"
                        reference channels (something_channel)
                on channels (alertChannel1),
}
```

[001068] Control Policy:

```
policy_template SOME_CONDITIONACTION_template {
        condition
                …
        action
                …
                control
                        messageToSend : constant
Some_Conditionaction_alert_message
                        sensors(whatever)
}
```

[001069] Error Handling

[001070] The Symbol Sensor Adapter (RFID) can respond to events with appropriate errors. For example, the Symbol Sensor Adapter (RFID can

recognize any non-successful call and send out its own alert on an alert_channel with the appropriate message.

[001071] ThermoElectron Sensor Adapter (Chem)

[001072] Properties

[001073] Examples of the ThermoElectron Sensor Adapter (Chem) properties include the 'Enabled' property that indicates whether the sensor proxy is enabled or not. The value assigned to the 'Enabled' property is 1.

[001074] An example of the ThermoElectron Sensor Adapter (Chem) in SPML is shown below.

```
sensor_adapter_template <Whatever>_template {
        message_types
                alert_message AlertMessageType -and|or
                data_message <Whatever>Data -and|or
                comm_alarm_message CommAlarmMessageType
        protocol <Whatever>
        properties
                Enabled integer -and|or-
                IPAddress string -and|or-
                HTTPGetUrl string -and|ornumberToDial
                string -and|ortokenID
                string -and|ormessageToSend
                string -and|orrecvTimeout
                integer -and|orport
                integer
}
sensor_adapter_instance <Name> {
        template <Whatever>_template
        channels
```

```
            input_channels
                LocalMsgServer::Chem.Alert.FireDepartment -and|or-
                LocalMsgServer::<some.sortof.channel>
            alert_channels
                LocalMsgServer::<Whatever>Alert
        properties
            Enabled = 1
            Table
            HTTPGetUrl = "<someURL>"
            numberToDial = "555123456"
            tokenID =
            "341b34d433ed0b4db7bef7fd57a6786ecb04a4ac8940879
            ac9c211021cd0e9422678b5c5d5d08fa668714697"
            messageToSend = "Your message here"
            recvTimeout = <some # of secs>
```

[001075] }

[001076] The ThermoElectron Sensor Adapter (Chem) can send other messages including the one below.

```
type Alert|AlarmType {
    TTimestamp string
    alert string
    AlertLevel string
    referenceChannels string
    buildingLocation BuildingLocationType
    etc
}
```

[001077] Behavior

[001078] Example behaviors of the ThermoElectron Sensor Adapter (Chem) are provided below.

[001079] Publish policy:

```
policy_template SOME_CONDITIONACTION_template {
        condition
                …
        action
                …
                publish
                        alert : constant Some_Conditionaction_alert_message
                        alertLevel : "RED"
                        referenceChannels : "RefChannel1~
                                                RefChannel2~
                                                RefChannel3"
                        reference channels (something_channel)
                        on channels (alertChannel1),
}
```

[001080] Control Policy:

```
policy_template SOME_CONDITIONACTION_template {
        condition
                …
        action
                …
                control
                        messageToSend : constant
Some_Conditionaction_alert_message
                        sensors(whatever)
}
```

[001081] Error Handling

[001082] The ThermooElectron Sensor Adapter (Chem) service can respond to evnetns with appropriate errors messages. For example, the ThermoElectron Sensor Adapter (Chem) can recognize any non-successful call and send out its

own alert on an alert_channel with the appropriate error as its message.

[001083] ITrans Sensor Adapter

[001084] The ITrans Sensor Adapter accepts data from the sensor itself. It contains connectivity specifications (i.e IP address,etc) needed to connect to the physical sensor. By contrast, note that the ITrans Channel Sensor Adapter does not have connectivity specifications, but does contain parameters that allow the system to discriminate between the readings for the different gases.

[001085] In the ITrans Sensor Adapter, the NameOfDetectedGas property contains readings for multiple gases, and is thus referred to a “multi-headed” SA. The “multi-headed” SA must be defined before the “single-headed” SAs, which later will extract readings for each individual gas.

[001086] Properties

[001087] FIG. 159 shows examples of ITrans Sensor Adapter properties. An example of sensor adapter template and instance is shown below.

```
// The Itrans Sensor adapter in SPML:
sensor_adapter_template ItransSensorTemplate {
        message_types
                data_message ChemMessageType
                comm_alarm_message CommAlarmMessageType
        protocol ITrans
        properties
                Enabled                 integer
                SimulatorMode           integer
                IPAddress               string
                port                    integer
                PollingRate             double
```

```
            geoLocation                    GeoLocationType
            NameOfDetectedGas [no value should be entered]
    }
    sensor_adapter_instance ItransSensor {
        template ItransSensorTemplate
        channels
            comm_alarm_channels
                LocalMsgServer::LocalCommAlarmChannel
        properties
            Enabled = 1
            SimulatorMode = 0
            IPAddress = "64.210.18.21"
            Port = 2101
            PollingRate = 1.0
            geoLocation.longitude = -118.159705
            geoLocation.latitude = 34.186906
            geoLocation.altitude = 0.0
            NameOfDetectedGas = vector
    }
```

[001088] Behavior

[001089] The ITrans Sensor Adapter accepts data from a specified physical sensor. The NameOfDetectedGas property is an array containing readings for multiple gases. Instances of the ITransChannel SA filter out values for the separate gases.

[001090] Error Handling

[001091] The ITrans Sensor Adapter service can respond to events with appropriate error messages. For example, the ITrans Sensor Adapter adapter can recognize any non-successful call and send out its own alert on an

alert_channel with the above error as its message.

[001092] QTL Biosensor 2000

[001093] The QTL Biosensor 2000 is a hand-held biosensor that tests for anthrax, ricin, botulinum, staphylococcal enterotoxin B (SEB), and other pathogens.

[001094] Properties

[001095] FIG. 160 shows examples of the specified QTL Biosensor 2000 properties. An example of the QTL Biosensor 2000 sensor adapter in SPML is shown below.

```
sensor_adapter_template QTLBiosensor2000_SA_template {
        message_types
                data_message QTLData
                comm_alarm_message CommAlarmMessageType
        protocol QTLBiosensor
        properties
                Enabled integer
                IPAddress string
                port integer
                PollingRate double
                SimulatorMode integer
                AssayType string
                HazMat string
                Location LocationType
                geoLocation GeoLocationType
}
sensor_adapter_instance QTLBiosensor2000_SA {
        template QTLBiosensor2000_SA_template
        channels
```

```
        data_channels
                LocalMsgServer::LocalQTLDataChannel,
                LocalMsgServer::QTLDataSoapChannel
        comm_alarm_channels
                LocalMsgServer::LocalCommAlarmChannel
        properties
                Enabled = 1
                IPAddress = "11.22.33.212"
                port = 4000
                PollingRate = 1.0
                SimulatorMode = 0
                AssayType = "Anthrax"
                HazMat = "HAZMAT 323"
                Location.Location_description = "MyInstitution"
                Location.Location_street_address = "MyStreet"
                Location.Location_city = "MyTown"
                Location.Location_state = "MyState"
                Location.Location_zip_code = "12345"
                Location.Location_country = "USA"
                Location.Location_building = "MyBldg"
                Location.Location_floor = "1"
                Location.Location_room = "MyLab"
                geoLocation.longitude = -75.0000
                geoLocation.latitude = 40.0000
                geoLocation.altitude = 0.0
}
```

[001096] Behavior

[001097] The typical behavior of the QTL Biosensor 2000 sensor adapter is to read in a Boolean value from the sensor which indicates the presence or non-detection of one of the featured pathogens. Examples are given below. The

results of the assay are published on an Inova display.

[001098] Publish and Control policy:

```
policy_template QTL_POS_template {
condition
        (channel_reference(QTL_biosensor2000_TestResult) == "Positive")
action
        publish
                alert : "ALERT: " ||
                channel_reference(QTL_biosensor2000_AssayType) ||
                " found in sample " ||
                channel_reference(QTL_biosensor2000_SampleId) ||
                ". Detected by " ||
                channel_reference(QTL_biosensor2000_HazMat) || " in " ||
                channel_reference(QTL_biosensor2000_City) || ", " ||
                channel_reference(QTL_biosensor2000_State) || "."
                reference channels(QTL_biosensor2000_data_channel)
        on channels (Bio_alert_technician_channel,
                Bio_alert_lab_manager_channel,
                Bio_alert_fire_department_channel),
control
        commandMessage : "ALERT: " ||
channel_reference(QTL_biosensor2000_AssayType) ||
        " found in sample " ||
        channel_reference(QTL_biosensor2000_SampleId) ||
        ". Detected by " ||
        channel_reference(QTL_biosensor2000_HazMat) || " in " ||
        channel_reference(QTL_biosensor2000_City) || ", " ||
        channel_reference(QTL_biosensor2000_State) || "."
sensors(Inova)
}
```

[001099] Error Handling

[001100] The QTL Bioserver 2000 service can respond to events with appropriate error messages. For example, the QTL Bioserver 2000 sensor adapter can recognize any non-successful call and send out its own alert on an alert_channel with the appropriate error as its message.

[001101] QTL Biosensor 2200

[001102] The QTL Biosensor 2200 incorporates an RFID that indicates which pathogen is being tested for. When fully implemented with SPM, the SA will use the RFID information to consult a lookup table and establish the assay type. The QML mCMS sensor is automated and can run 5 assays on multiple samples (anthrax, botulinum, SEB and pos and neg controls)

[001103] Properties

[001104] Examples of the specified QTL Biosensor 2200 properties include the 'Enabled' property that indicates whether the sensor proxy is enabled or not. The value assigned to the 'Enabled' property is 1.

[001105] An example of the QTL Biosensor 2200 sensor adapter in SPML is shown below.

```
sensor_adapter_template <Whatever>_template {
        message_types
                alert_message AlertMessageType -and|or
                data_message <Whatever>Data -and|or
                comm_alarm_message CommAlarmMessageType
        protocol <Whatever>
        properties
                Enabled integer -and|or-
```

```
                IPAddress string -and|or-
                HTTPGetUrl string -and|ornumberToDial
                string -and|ortokenID
                string -and|ormessageToSend
                string -and|orrecvTimeout
                integer -and|orport
                integer
}
sensor_adapter_instance <Name> {
        template <Whatever>_template
        channels
                input_channels
                        LocalMsgServer::Chem.Alert.FireDepartment -and|or-
                        LocalMsgServer::<some.sortof.channel>
                alert_channels
                        LocalMsgServer::<Whatever>Alert
        properties
                Enabled = 1
                HTTPGetUrl = "<someURL>"
                numberToDial = "555123456"
                tokenID =
                "341b34d433ed0b4db7bef7fd57a6786ecb04a4ac8940879
                ac9c211021cd0e9422678b5c5d5d08fa668714697"
                messageToSend = "Your message here"
                recvTimeout = <some # of secs>
}
```

[001106] QTL Biosensor 2200 SA can send other messages including the one described below.

```
type Alert|AlarmType {
        TTimestamp string
```

alert string
AlertLevel string
referenceChannels string
buildingLocation BuildingLocationType
etc
}

[001107] Behavior

[001108] An example of the behavior of the QTL Biosensor 2200 sensor adapter is shown below.

[001109] Publish policy:

```
policy_template SOME_CONDITIONACTION_template {
        condition
                …
        action
                …
                publish
                        alert : constant Some_Conditionaction_alert_message
                        alertLevel : "RED"
                        referenceChannels : "RefChannel1~
                                                RefChannel2~
                                                RefChannel3"
                        reference channels (something_channel)
                on channels (alertChannel1),
}
```

[001110] Control Policy:

```
policy_template SOME_CONDITIONACTION_template {
        condition
                …
        action
```

...
control
messageToSend : constant
Some_Conditionaction_alert_message
sensors(whatever)
}

[001111] Error Handling

[001112] The QTL Biosensor 2200 service can respond to events with corresponding error messages. For example, the QTL Biosensor 2200 sensor adapter can recognize any non-successful call and send out its own alert on an alert_channel with the corresponding error as its message.

[001113] Net Guardian

[001114] SNMP stands for Simple Network Management Protocol which is used by network management systems to monitor network-attached devices for conditions that warrant administrative attention. The notable events in the devices are noted by Traps, which cause messages to be sent to the managing system. SPM contains different templates reflecting each of this roles – the management role and event trapping.

[001115] The Net Guardian device is used to monitor serial devices and acts as a gateway between the serial devices and TCP/IP enabled networks. SPM contains two template for SNMP implmentation – one for the Manager that monitors the devices and the other for the monitored devices.

[001116] Properties

[001117] FIG. 161 shows examples of the specified SNMP Net Guardian properties. An example of the SNMP Net Guardian Manager template in SPML

is shown below.

```
sensor_adapter_template SnmpManagerTemplate {
        message_types
                alert_message SnmpTrapMessageType
                data_message SnmpTrapMessageType
                comm_alarm_message CommAlarmMessageType
        protocol SnmpManager
        properties
                Enabled integer
                TrapPort integer
                PollingRate double
                ReadCommunityString string
                WriteCommunityString string
                NotificationCommunityString string
}
sensor_adapter_template SnmpTrapRxTemplate {
        message_types
                data_message SnmpTrapMessageType
                alert_message SnmpTrapMessageType
        protocol SnmpTrapReceiver
        properties
                SPModel string
                Enabled integer
                geoLocation GeoLocationType
sensor_adapter_instance SmnpManager {
        template SnmpManagerTemplate
        channels
                comm _alarm_channels
                        MyLocalMessageServer::comm_alarm_channels
        properties
                TrapPort = 162
```

```
                    PollingRate = 1.0
                    Enabled = 0
                    ReadCommunityString = "public"
                    WriteCommunityString = "public"
                    NotificationCommunityString = "public"}
sensor_adapter_instance SmnpTrapReceiver {
        template SnmpTrapRxTemplate
        channels
                data_channels localMessages::StandardCommAlarm
                comm_alarm_channels localMessages::StandardCommAlarm
        properties
                SPModel = "SmnpManager"
                Enabled = 0
                geoLocation.longitude = -71.047122
                geoLocation.latitude = 42.31856
                geoLocation.altitude = 0.0
}
```

[001118] Behavior

[001119] Examples of the typical behavior of the SNMP Net Guardian are shown below.

[001120] Publish policy:

```
policy_template SOME_CONDITIONACTION_template {
        condition
                …
        action
                …
                publish
                        alert : constant Some_Conditionaction_alert_message
                        alertLevel : "RED"
                        referenceChannels : "RefChannel1~
```

```
                                        RefChannel2~
                                        RefChannel3"
                    reference channels (something_channel)
                on channels (alertChannel1),
}
```

[001121] Control Policy:

```
policy_template SOME_CONDITIONACTION_template {
    condition
        …
    action
        …
        control
            messageToSend : constant
Some_Conditionaction_alert_message
            sensors(whatever)
}
```

[001122] Error Handling

[001123] The SNMP Net Guardian service can respond to events with appropriate error messages. For example, the SNMP Net Guardian can recognize any non-successful call and send out its own alert on an alert_channel with the the appropriate error as its message.

[001124] Voxeo

[001125] The Voxeo sensor adapter provides a method of delivering text to speech message delivery over the phone. The system can send a text message or alert to a specified phone number and Voxeo will call the number and deliver the specified message. This functionality can be hosted by Voxeo using a VoxeoXML.start token router. The Voxeo sensor adapter works by constructing

a URL consisting of the appropriate information (Voxeo hosting service, phone number, token id, and message) and calling an HTTP Get command. The Voxeo service will respond back with a success or failure message. This is then handled by the Voxeo sensor adapter.

[001126] Properties

[001127] FIG. 1622shows examples of the specified Voxeo properties. An example of the Voxeo sensor adapter template in SPML is shown below.

```
sensor_adapter_template Voxeo_template {
        message_types
                alert_message AlertMessageType
        protocol Voxeo
        properties
                Enabled integer
                HTTPGetUrl string
                numberToDial string
                tokenID string
                messageToSend string
                recvTimeout integer
}
sensor_adapter_instance Voxeo {
        template Voxeo_template
        channels
                input_channels
                        LocalMsgServer::Chem.Alert.FireDepartment
                alert_channels
                        LocalMsgServer::VoxeoAlert
        properties
                Enabled = 1
                HTTPGetUrl = "session.voxeo.net/VoiceXML.start"
```

numberToDial = "6262966436"
tokenID =
"341b34d433ed0b4db7bef7fd57a6786ecb04a4ac8940879
ac9c211021cd0e9422678b5c5d5d08fa668714697"
messageToSend = "Testing Hello"
recvTimeout = 180
}

[001128] The Voxeo sensor adapter can receive data on the command and input channel. The Voxeo sensor adapter can make one phone call per data message received based upon the set properties of the data message. In addition to messages containing the properties specified above, Voxeo can also receive the AlertMessageType as described below.

type AlertMessageType {
Timestamp string
alert string
AlertLevel string
referenceChannels string
buildingLocation BuildingLocationType
alternateLocation AlternateLocationType
geoLocation GeoLocationType
}

[001129] Upon receiving the AlertMessageType message the sensor adapter calls the last number specified and relay the alert string as the new message.

[001130] Behavior

[001131] The typical behavior of the Voxeo sensor adapter is to subscribe to an alert_channel. If another sensor adapter publishes an alert message to this channel, Voxeo receives the alert message and send a call with the given

message.

[001132] The Voxeo sensor adapter makes one call per message received. If no updated phone number is given, it will dial the last number used. Voxeo can receive messages that contain the Voxeo properties specified in the Properties section, or AlertMessageType. Given an AlertMessageType it will call the last configured number with the message in the alert field. A policy can be used on an input channel to publish this alert information on a given channel subscribed to by Voxeo, or use a control to directly send the message to Voxeo.

[001133] Examples are given below.

[001134] Publish policy:

```
policy_template TOXIC_FUME_template {
        condition
                …
        action
                …
                publish
                        alert : constant Toxic_fume_alert_message
                        alertLevel : "RED"
                        referenceChannels : "Chem.Data.ITrans.Gas.Flammables~
                                        Chem.Video.Frames.EmissionsLabEast~
                                        Chem.Video.Frames.EmissionsLabWest"
                        reference channels (ch4_channel)
                on channels (alertChannel1),
}
```

[001135] Control Policy:

```
policy_template TOXIC_FUME_template {
        condition
                …
```

```
        action
                ...
                control
                        messageToSend : constant Toxic_fume_alert_message
                        sensors(Voxeo)
}
```

[001136] Error Handling

[001137] The Voxeo service can respond to an HTTP Get request with the example errors shows in FIG. 163. For example, The Voxeo sensor adapter will recognize any non-successful call and send out its own alert on an alert_channel with the above error as its message. The HTTP receive timeout should be set to an appropriate amount of time for the Voxeo service to respond with either a success or failure. Voicemail response can be spotty. Sometimes voicemail is recognized as no answer, other times it is recognized as a success.

[001138] Viper Configuration File

[001139] The following Viper configuration file ("Example_ViperServe1") is included here to illustrate SPML language syntax. Each Viper configuration requires its own SPML file. This file takes the form of a series of data structures which are presented in the order shown below.

- Generic Location Data Structures – used by all or most sensor adapters
- Generic Message Data Structures – used by all or most sensor adapters
- Sensor-specific Message Data Structures – used by the current sensor adapter; typically these will include a fair amount of generic data (e.g. location data, timestamps, etc.) and a highly variable amount of sensor-specific data

• Type Maps – type maps establish equivalences between different names or different fields used for the same thing in two data structures or different names; for example, in one structure a variable may be called “NameOfDetectedGas” while in another is is written “nameofdetectedgas”.

• Value Maps –

• Message Server Instance Definitions – these structures define channel names for each of the message servers and the type of message that is carried in each channel.

• Sensor Adapter Templates and Instances – a Viper configuration file contains a sensor adapter template for each type of sensor in the system, and one or more sensor adapter instances based on each template.

• Policy Templates and Instances – the policy instances are specific for the Viper configuration. Most configuration files contain policies. However, some sensors such as building access and RFID sensors contain their own policies, which may obviate the need for a high-level policy in the SPM file.

• Viper Server Instance – this structure specifies the SOAP server port, the names of the defined message server instances, register setting defaults, sensor adapter instances and, generally, policy instances.

• Process Manager – this structure specifies the Process Manager controlling the current Viper Server. A Process Manager may control one or more Viper Servers.

• Configuration Server – this structure specifies the Configuration Server controlling the current the current Process Manager. A Configuration Server may

control one or Process Managers.

[001140] Generic Location Data Structures

[001141] The location data structures are used by all sensor adaptors. They are often listed under the "Miscellaneous Subtypes" heading at the beginning of the file. AlternateLocationType is generally used to specify the location where a sample was taken, as distinct from the location of the sensor itself. See example below.

```
type AlternateLocationType {
        Alternate_location_x double // decimal longtitude (not deg, min, sec)
        Alternate_location_y double // decimal latitude
        Alternate_location_z double // elevation in meters
}
```

[001142] BuildingLocationType requires no comment. An example is shown below.

```
type BuildingLocationType {
        Location_building string
        Location_floor string
        Location_room string
}
```

[001143] GeoLocationType is generally used to specify the location of the sensor. An example is shown below.

```
type GeoLocationType {
        longitude double // decimal longtitude
        latitude double // decimal latitude
        altitude double // elevation in meters
}
```

[001144] LocationType is generally used to specify the location of the

sensor. An example is shown below.

```
type LocationType {
        Location_description string
        Location_street_address string
        Location_city string
        Location_state string
        Location_zip_code string
        Location_country string
        Location_building string
        Location_floor string
        Location_room string
}
```

[001145] Generic Message Data Structures

[001146] AlertMessageType is used by all sensor adaptors. It is listed under the "Miscellaneous Message Types" heading. An example is shown below.

```
type AlertMessageType {
        Timestamp string //Milliseconds since 1/1/1970 00:00:00 GMT
        alert string
        AlertLevel string
        buildingLocation BuildingLocationType
        alternateLocation AlternateLocationType
        geoLocation GeoLocationType
}
```

[001147] CommAlarmMessageType is used by all sensor adaptors. It is listed under the "Miscellaneous Message Types" heading. The alarm is sent when some problem with the communication link to the sensor is detected. An example is shown below.

```
type CommAlarmMessageType {
```

Timestamp string //Milliseconds since 1/1/1970 00:00:00 GMT
Description string
alert string = "COMM ALERT!!!"
AlertLevel string
SensorName string
SensorType string
buildingLocation BuildingLocationType
alternateLocation AlternateLocationType
geoLocation GeoLocationType
}

[001148] TimestampAndUUIDType is used by the SPM GUI to identify messages with a timestamp and Universally Unique Identifier. An example is shown below.

type TimestampAndUUIDType {
Timestamp string
UUID string

[001149] Sensor-specific Message Data Structures

[001150] The next items in a Viper configuration file are sensor-specific message data structures. The configuration file typically has a difference message type for each type of sensor in the system. Examples of message data structures used by a chemical sensor and a video camera are shown below.

type GasData {
Timestamp string
NameofdetectedGas string
Concentration double
buildingLocation BuildingLocationType
alternateLocation AlternateLocationType
geoLocation GeoLocationType

```
}
type VideoMessageType {
        Timestamp string
        Timezone string
        VideoFrameTimestamp string
        VideoFrame string
        buildingLocation BuildingLocationType
        alternateLocation AlternateLocationType
        geoLocation GeoLocationType
}
```

[001151] Type Maps

[001152] Type maps establish equivalences between different names used for the same thing in two data structures. For example, in one structure a variable may be called “NameOfDetectedGas” while in another the variable is written “Nameofdetectedgas” or the list in one structure may use abbreviations while the other one does not. An example is shown below.

```
type_map ChemMessageType_to_GasData_map {
        Timestamp: Timestamp
        NameOfDetectedGas: NameofdetectedGas
        Concentration: Conc
        buildingLocation.Location_building: buildingLocation.Location_bldg
        buildingLocation.Location_floor: buildingLocation.Location_flr
        buildingLocation.Location_room: buildingLocation.Location_rm
        alternateLocation.Alternate_location_x: alternateLocation.Alternate_loc_x
        alternateLocation.Alternate_location_y: alternateLocation.Alternate_loc_y
        alternateLocation.Alternate_location_z: alternateLocation.Alternate_loc_z
        geoLocation.longitude: geoLocation.longitude
        geoLocation.latitude: geoLocation.latitude
        geoLocation.altitude: geoLocation.altitude
```

}

[001153] Value Maps

[001154] Value Maps are used to map date from binary format to ASCII format.

[001155] Message Server Instance Definitions

[001156] A Local Message Server instance exists on each computer that processes data from one or more sensors. The actual processing of the data is done by the Sensor Adapter software instances on the local computer. A Local Message Server sends the processed data directly and immediately to the server where the SPM Policies reside and where they will be executed. Local channels allow data to be published on-process from a policy to a sensor adapter or between sensor adapters. If the only consumer of the messages is the local Viper server, then the message should be sent via the Local Message Server.

```
local_message_server LocalMsgServer {
channel LocalCommAlarmChannel message_type CommAlarmMessageType
}
```

[001157] If there are consumers for the message outside the Viper Server (e.g., the SPM GUI), the message must be routed through a JMS Message Server. When creating a channel on a Message Server other than the Local Message Server, itvis necessary to specify that the message be persistent so that a copy of the message will be held in the database.

```
jms_message_server JmsMsgServer {
        host "vialinux2":1099 //IP address may be used in place of hostname
        channel Chem.Alert.FireDepartment
                message_type AlertMessageType
```

```
            persistent
        channel Chem.Data.ITrans.Gas.CarbonMonoxide
            message_type GasData
            message_protocol SML //protocol specification may be optional
            message_type_map ChemMessageType_to_GasData_map //type
map may be optional
            persistent
        . . .
}
```

[001158] If a message has to cross a firewall, or if it is sent from one Viper Server to another without going the the Java Message Server, it must be routed through a SOAP Message Server. It is possible to control the distribution of a message more tightly using a SOAP Message Server than a TCP Message Server.

```
soap_message_server SoapMsgServer {
        host "vialinux2":4444 //IP address may be used in place of hostname
        channel SoapChannel1
            message_type AlertMessageType
            persistent
}
```

[001159] If SPM is connecting to a network that does not support SOAP messaging, it may be necessary to use a TCP Message Server. An example is provided below.

```
soap_message_server TcpMsgServer { //soap??
        host "vialinux2":1099 //IP address may be used in place of hostname
        channel TcpChannel1
            message_type AlertMessageType
            persistent
```

}

[001160] Sensor Adapter Templates and Instances

[001161] A Viper configuration file contains a sensor adapter template for each type of sensor in the system, and a sensor adapter instance based on that template for each sensor instance.

[001162] In this example, the SPM system is set up to monitor readings from an Itrans chemical sensor and broadcast possible warnings through an Inova alerting system based on a policy that will be defined later in the configuration file. The Inova alerting template and instance is shown first, followed by those for the Itrans sensor.

[001163] An example Inova template and instance is shown below.

```
//### Inova
sensor_adapter_template InovaSensorTemplate {
        message_types
                comm_alarm_message CommAlarmMessageType
        protocol Inova
        properties
                Enabled integer
                IPAddress string
                port integer
                PollingRate double
                userName string
                password string
                defaultMessage string
                commandDuration integer
                defaultColor string
                commandMessage string
                currentMessage integer
```

```
            commandColor string
            commandRefreshTime string
            commandFontSize string
            commandDisplayMethod string
            useDefaultMessage integer
}
sensor_adapter_instance Inova {
    template InovaSensorTemplate // refers to template declared immediately
above
    channels
        comm_alarm_channels
            LocalMsgServer::LocalCommAlarmChannel //only one
channel is defined
    properties
        Enabled = 1
        IPAddress = "64.210.18.104" // address of Inova display
        port = 23
        PollingRate = 1.0 // time in seconds to check if status has changed
        userName = "admin"
        password = "1n0v@"
        defaultMessage = "SPM - ViaLogy, California, USA"
        commandDuration = 60
        defaultColor = "green"
        commandMessage = ""
        currentMessage = 100
        commandColor = "red"
        commandRefreshTime = "5"
        commandFontSize = "14"
        commandDisplayMethod = "ribbon_left"
        useDefaultMessage = 1
}
```

//### END Inova.

[001164] The Itrans chemical sensor is a multi-head sensor which can detect concentrations of both carbon monoxide and methane gases and which forwards this information through two channels, one for each gas. The SPM sensor adapter consist of two parts: the overall sensor adapter template for the data stream as a whole, and a second subtemplate for the data for each component gas being monitored. There is a single instance for the whole stream and two instances of the subtemplate (one for each monitored gas). Each subtemplate instance contains a filter (shown in bold type) which looks for the data for a particular gas.

```
//### ITrans:
sensor_adapter_template ItransSensorTemplate { //main template
        message_types
                data_message ChemMessageType
                comm_alarm_message CommAlarmMessageType
        protocol ITrans //name of C++ package for this sensor adapter template
                        //the precise name of the package is supplied by ViaLogy
        properties
                Enabled integer
                SimulatorMode integer
                IPAddress string
                port integer
                PollingRate double
                geoLocation GeoLocationType
                NameOfDetectedGas string
}
sensor_adapter_template ItransChannelSensorTemplate { //subtemplate
        message_types
```

```
                data_message ChemMessageType
                comm_alarm_message CommAlarmMessageType
        protocol ITransChannel //name of C++ package for this sensor adapter
template
        properties
                Enabled integer
                SPModel string
                geoLocation GeoLocationType
}
// communication channels are defined at the instance level
sensor_adapter_instance ItransSensor { //instance of main Itrans template
template ItransSensorTemplate // refers to main template declared above
channels //these channels will be used later by the Policies
        comm_alarm_channels
                LocalMsgServer::LocalCommAlarmChannel
properties
        Enabled = 1
        SimulatorMode = 0
        IPAddress = "64.210.18.21"
        port = 2101
        PollingRate = 1.0
        geoLocation.longitude = -118.159705
        geoLocation.latitude = 34.186906
        geoLocation.altitude = 0.0 // not reported
        NameOfDetectedGas = "Nitrogen" // placeholder
}
sensor_adapter_instance ItransChan0Sensor { //CO reading
        template ItransChannelSensorTemplate // refers to subtemplate declared
above
        channels
                data_channels //these channels will be used later by the Policies
```

```
                    JmsMsgServer::Chem.Data.ITrans.Gas.CarbonMonoxide //
persistent message
                comm_alarm_channels
                    LocalMsgServer::LocalCommAlarmChannel // transient
message
filter NameOfDetectedGas == "CO_Carbon_Monoxide"
properties
        Enabled = 1
        SPModel = "ItransSensor"
        geoLocation.longitude = -118.159705
        geoLocation.latitude = 34.186906
        geoLocation.altitude = 0.0
}
sensor_adapter_instance ItransChan1Se
        template ItransChannelSensorTemplate // refers to subtemplate declared
above
        channels //these channels will be used later by the Policies
                data_channels
                    JmsMsgServer::Chem.Data.ITrans.Gas.Flammables
                comm_alarm_channels
                    LocalMsgServer::LocalCommAlarmChannel
filter NameOfDetectedGas == "CH4_Methane"
        properties
                Enabled = 1
                SPModel = "ItransSensor"
                geoLocation.longitude = -118.159705
                geoLocation.latitude = 34.186906
                geoLocation.altitude = 0.0
}
//### END ITrans.
//########## END sensor adapters.
```

[001165] Policy Templates and Instances

[001166] Specific policy instances must be created for each Viper configuration. Before a policy instance can be created, appropriate condition and action templates must be declared and from these a policy template ican be created simply by including the condition and action templates in a policy template (or class) declaration. The policy instances are then based on the policy class.

```
/### ITrans Policies:
// The first policy deals with the case when the gas(es) exceed their thresholds
// The first condition is that either gas conc. is above its threshold
// This is determined by comparing gas readings to threshold values
condition_template TOXIC_FUME_cond_template {
        channel_reference(co_channel->Concentration) >
        constant_reference(Toxic_fume_threshold_co) |
        channel_reference(ch4_channel->Concentration) >
        constant_reference(Toxic_fume_threshold_ch4)
}
// The first action checks to see if the alert register is true or false.
// If the alert has already been posted, there is no need to post it again.
action_template TOXIC_FUME_action_template {
        if
                ~register_reference(ALERT_POSTED_REGISTER) // check if false
        then
                set register_reference(ALERT_POSTED_REGISTER) : true
//change to true
                set register_reference(ALERT_POSTED_TIME_REG) :
                        register_reference(CurrentTime) //set alert timestamp
publish
        // Add example of stand-alone function call.
```

```
        // Add example of function call in an expression.
        alert : constant_reference(Toxic_fume_alert_message) //standalone call
        AlertLevel : "RED"
        buildingLocation : channel_reference(co_channel->buildingLocation)
        alternateLocation : channel_reference(co_channel->alternateLocation)
        geoLocation : channel_reference(co_channel->geoLocation)
                reference_channels(co_channel->Concentration,
                        ch4_channel->Concentration,
                        camera_channel1,
                        camera_channel2)
                output_channels(alertChannel1)
        control
                commandMessage :
constant_reference(Toxic_fume_alert_message)
                sensors(Inova)
else // An alert has already been sent, but see if it has been > n sec
        if
                TIMESTAMP_GAP_SEC(register_reference(CurrentTime),
                        register_reference(ALERT_POSTED_TIME_REG)) >
                        constant_reference(interval_for_alert_reissue)
        then
                set register_reference(ALERT_POSTED_TIME_REG) :
                        register_reference(CurrentTime)
                publish //publish concatenated alert message
                        alert : "Alert still outstanding: " ||
                                constant_reference(Toxic_fume_alert_message) //call
in expression
                        AlertLevel : "RED"
                        buildingLocation : channel_reference(co_channel-
>buildingLocation)
                        // include alternate location if sample site not same a sensor
```

```
site
                    alternateLocation : channel_reference(co_channel-
>alternateLocation)
                    geoLocation : channel_reference(co_channel->geoLocation)
                        reference_channels(co_channel->Concentration,
                        ch4_channel->Concentration,
                        camera_channel1,
                        camera_channel2)
                    output_channels(alertChannel1)
                control // command Inova display to show concatenated message
                        commandMessage : "Alert still outstanding: " ||
                    constant_reference(Toxic_fume_alert_message)
                    sensors(Inova)
                fi
        fi
}
// create policy template from condition and action templates
// for first case when gas concentration is excessive
policy_template TOXIC_FUME_template {
        condition_template TOXIC_FUME_cond_template
        action_template TOXIC_FUME_action_template
}
// Create a policy instance based on the above policy template by
// supplying specific values for parameters and by setting bindings
policy_instance TOXIC_FUME_instance {
        template TOXIC_FUME_template // refer to newly created template
        input_channels // The trigger keyword is an instruction that any new
message
                // on the channel will cause the policy to be reevaluated. If values
                // for the Depth and Timespan parameters are not specified,
                // defaults depth = 1 (event) and timespan = 0 (seconds) are used
```

```
        JmsMsgServer::Chem.Data.ITrans.Gas.CarbonMonoxide
        JmsMsgServer::Chem.Data.ITrans.Gas.Flammables trigger
constants
        Toxic_fume_threshold_co : 5.0 // percent
        Toxic_fume_threshold_ch4 : 5.0 // percent
        interval_for_alert_reissue : 30 // seconds
        Toxic_fume_alert_message : "ALERT: Explosive gas leak detected."
channel_bindings
        co_channel : JmsMsgServer::Chem.Data.ITrans.Gas.CarbonMonoxide
        ch4_channel : JmsMsgServer::Chem.Data.ITrans.Gas.Flammables
        camera_channel1 :
        JmsMsgServer::Chem.Video.Frames.EmissionsLabEast
        camera_channel2 : JmsMsgServer::Chem.Video.Frames.NightVision
        alertChannel1 : JmsMsgServer::Chem.Alert.FireDepartment
register_bindings
        ALERT_POSTED_REGISTER :
        Example_ViperServer1::TOXIC_FUME_ALERT_POSTED_REGISTER
        ALERT_POSTED_TIME_REG :
        Example_ViperServer1::ALERT_POSTED_TIME_REG
                CurrentTime: Example_ViperServer1::CurrentTime
sensor_bindings
        Inova : Example_ViperServer1::Inova
}
// A second policy deals with the case when the gas(es) drop below their
thresholds
// Constants are encapsulated within each condition and action template.
// Although the value of the constant Toxic_fume_threshold_co is the
// same as No_Toxic_fume_threshold_co, the constants must be defined
// separately in each policy.
//
condition_template NO_TOXIC_FUME_cond_template {
```

```
        channel_reference(co_channel->Concentration) <=
                constant_reference(No_Toxic_fume_threshold_co) &
        channel_reference(ch4_channel->Concentration) <=
                constant_reference(No_Toxic_fume_threshold_ch4) &
        register_reference(ALERT_POSTED_REGISTER)
}
action_template NO_TOXIC_FUME_action_template {
        set register_reference(ALERT_POSTED_REGISTER) : false
        publish
                alert : constant_reference(No_Toxic_fume_alert_message)
                AlertLevel : "GREEN"
                reference_channels
                        (co_channel->Concentration,
                ch4_channel->Concentration,
                camera_channel1, camera_channel2)
        output_channels(alertChannel1)
control
        commandMessage : constant_reference(No_Toxic_fume_alert_message)
        sensors(Inova)
}
// create policy template from condition and action templates
// for second case when gas concentration has dropped below threshold
policy_template NO_TOXIC_FUME_template {
        condition_template NO_TOXIC_FUME_cond_template
        action_template NO_TOXIC_FUME_action_template
}
// create policy instance from second case policy template
policy_instance NO_TOXIC_FUME_instance {
        template NO_TOXIC_FUME_template
        input_channels
                JmsMsgServer::Chem.Data.ITrans.Gas.Flammables trigger depth
```

```
10 timespan 1000
        constants
                No_Toxic_fume_threshold_co : 5.0
                No_Toxic_fume_threshold_ch4 : 5.0
                No_Toxic_fume_alert_message :
                        "ALERT: Explosive gas levels returned to safe levels."
        channel_bindings
                co_channel :
                JmsMsgServer::Chem.Data.ITrans.Gas.CarbonMonoxide
                ch4_channel : JmsMsgServer::Chem.Data.ITrans.Gas.Flammables
                camera_channel1 :
                JmsMsgServer::Chem.Video.Frames.EmissionsLabEast
                camera_channel2 :
                JmsMsgServer::Chem.Video.Frames.NightVision
                alertChannel1 : JmsMsgServer::Chem.Alert.FireDepartment
        register_bindings
                ALERT_POSTED_REGISTER :
        Example_ViperServer1::TOXIC_FUME_ALERT_POSTED_REGISTER
        sensor_bindings
                Inova : Example_ViperServer1::Inova
}
//### END ITrans Policies.
```

[001167] Viper Server Instance

[001168] Once the user has defined all the sensor adapter and policies instances, it is possible to define the ViperServer instance. This structure specifies the SOAP server port, the names of remote Viper instances, if any, the names of the defined message server instances, register setting defaults, sensor adapter instances and policy instances.

```
viper_server Example_ViperServer1 {
```

```
        soap_server_port 4000
        remote_viper_servers
                ViperServer2 : host "vialinux4" soap_server_port 4040
                ViperServer3 : host "vialinux5" soap_server_port 4050
        message_servers
                LocalMsgServer
                JmsMsgServer
                SoapMsgServer
                TcpMsgServer
        registers
                TOXIC_FUME_ALERT_POSTED_REGISTER : boolean = false
                ALERT_POSTED_TIME_REG : string = "0000000000"
                MOST_RECENT_MSG_REG : TimestampAndUUIDType (100)
        sensor_adapters
        //@@@ Multi-headed sensor adapter must
        //precede single-headed sensor adapters.
                Inova
                ItransSensor
                ItransChan0Sensor
                ItransChan1Sensor
        policies
                TOXIC_FUME_instance
                NO_TOXIC_FUME_instance
}
```

[001169] Process Manager

[001170] An SPM installation can include one or more Process Managers. Each Process Manager is typically installed on its own physical server and controls one or more Viper Servers. In a heterogeneous SPM installation, i.e. one which comprises both Linux and Windows computers, at

```
process_manager ProcessMgr1 {
        host "vialinux2"
        soap_server_port 4110
        viper_servers
        Example_ViperServer1 : soap_server_port 4000
}
```

[001171] Configuration Server

[001172] The Configuration Server is used to control one or more Process Managers. The Configuration Server can reside in its own physical server or it can be installed in the same physical server as one of the Process Managers.

```
configuration_server TheConfigServer {
        soap_server_port 4444
        process_managers
                ProcessMgr1
}
```

[001173] SPML Keywords

[001174] The following are examples of Reserved SPML Keywords.

[001175] An action_template is the template or class declaration of an action. A policy_template contains a condition_template and an action_template. A typical action_template includes: set statements, publish statements a control statements. The template statements may contain default parameter values. Specific values are set in a policy_instance.

[001176] An action_template may contain an alert_channels listing which is a list of default alert channels in case an alert is to be sent out on multiple channels. Not all action_templates contain an alert_channels listing. The alert_publish_condition is used together with the system variable. The

alert_message is one of the four basic message_types in SPM. The others are data_message, comm_alarm_message. and equipment_alarm_message. asn_message_server. The boolean keyword is used conventionally in SPML. A common use is to set the state of the resister during Viper configuration. For example:

registers

TOXIC_FUME_ALERT_POSTED_REGISTER : boolean = false

[001177] CAP1_1 is An XML-based message_protocol. One of the three types of SDO used within SPM. See also SML and ICD100. A message_protocol is different from the sensor adapter protocol that refers to a sensor-specific C++ routine included in the SPM file.

[001178] A channel is a pipe used to communicate data between sensors, sensor adapters and policies. Each channel is associated with a message_type. The channel keyword (singular) is used when defining channel instances. Each channel definition has it own channel statement. A full channel definition would assume the form:

channel ChannelName
message_type myMessageType
message_protocol SML|ICD|CAP1_1 //SML is default
message_value_map myTypeValueMap
message_structure_map myTypeStructureMap
persistent|<null> //non-persistent is default

[001179] The items in a channel definition must appear in the order shown above. The channel definition typically includes message_value_map and message_structure_map statements only for sensor adapters that are handling

data from Modbus format sensors. If persistent is not specified, the default is non-persistent.

[001180] The channels keyword (plural) is used in sensor_adapter_instance definitions to specify which message servers are used to retrieve the stated data fields or to handle certain types of messages. A typical channels usage in a sensor_adapter_instance definition would be:

```
sensor_adapter_instance myCamera {
    template ArecontCameraTemplate
    channels
        data_channels
            JmsMsgServer::Chem.Video.Frames.NightVision
        comm_alarm_channels
            LocalMsgServer::LocalCommAlarmChannel
```

[001181] There are seven types of channels in SPM:

- alert_channels – used for general alerts
- comm_alarm_channels – used to carry alerts when the communication to a sensor has been interrupted or otherwise compromised
- data_channels – used to send information to a sensor adapter
- equipment_alarm_channels – used to carry alerts when there is a malfunction within the sensor itself. The ability to generate an equipment alarm is a property of the sensor alone.
- input_channels – used to convey information to a policy

- output_channels – used to send information from a policy
- reference_channels – information collected via reference channels is accessible through the SPM GUI. Reference channels are used to carry information for display but not for executing policies. Because reference_channel information is used generically, the reference_channels statement appears in the action_template, while the input_channels statement appears in the policy_instance where the channel that triggers the policy is specified.

[001182] channel_bindings

[001183] The channel_bindings keyword is used inside a policy_instance to bind a name to a field in a channel, i.e. a property in a message. A common usage is to create a short, more easily managed alias in place of a long pathname. The basic syntax is:

channel_bindings

myAlias : someMsgServer::ChannelName.FieldName

[001184] For example, the alias co_channel can be used in place of the verbose expression to the right of the equivalence delimiter (:).

channel_bindings

co_channel : JmsMsgServer::Chem.Data.ITrans.Gas.CO

ch4_channel : JmsMsgServer::Chem.Data.ITrans.Gas.CH4

alertChannel1 : JmsMsgServer::Chem.Alert.FDNY

[001185] channel_reference

[001186] The channel_reference keyword is used inside a

condition_template to point to the value of some field in a bound (referenced) channel. Here the CO concentration in the co_channel in the channel_bindings example is compared to some reference threshold value:

```
condition_template myCondTemplate {
    (channel_reference(co_channel->Conc) <=
        constant_reference(threshold_co))
}
```

[001187] The constant_reference keyword tells the system to look for the specified constant in the constants list of the pertinent policy instance.

[001188] comm_alarm_message

[001189] This is one of the generic message_types in SPM that is used for almost all sensor adapters. The fields in messages such as data_message and equipment_alarm_message vary according to the sensor adapter. Message properties and fields are defined in the Miscellaneous Message Type declarations at the beginning of the SPM file.

[001190] condition_template

[001191] The class declaration of a condition. A policy_template comprises a condition_template and an action_template. A typical condition_template consists of a conditional statement that returns a boolean value if some combination of conditions is met (or not met) in whole or in part. The action template declares one or more actions. A policy_instance is created by defining values in its condition instance and action instance.

[001192] configuration_server

[001193] The Configuration Server is a program that controls one or more Process Managers, which in turn control one or more Viper Servers. The Configuration Server can be located on the same physical machine as one of the Process Managers, or it can exist in standalone configuration.

[001194] The configuration_server keyword is used at the very end of the Viper Configuration SPML file to specify the soap_server_port and the process_managers controlled by the Configuration Server.

```
configuration_server myConfigServer {
    soap_server_port 4444
    process_managers
        myProcessMgr1
}
```

[001195] constants

[001196] The constants in a Viper Configuration file are typically defined in a policy_instance and are accessed using the constant_reference keyword discussed below. Constants can be any data type: integer, long integer, double, or string.

[001197] constant_reference

[001198] The constant_reference keyword is used inside a condition_template to retrieve the value of a constant. The values of constants are typically defined in a policy_instance. In the following example the value of the Conc field in the referenced co_channel is compared to a referenced constant:

```
condition_template myCondTemplate {
```

((channel_reference(co_channel->Conc) >

constant_reference(co_threshold)))

}

[001199] control

[001200] The control keyword appears in an action_template (and optionally in an action_instance) and typically commands one or more sensors to execute an action or display or convey a message. Parallel statements in an action_template are set and publish. A typical implementation of a control in an action_template would be:

action_template myActionTemplate {

set <some value in a reference register>

publish <some message(s) to some channel(s)>

control

commandMessage:constant_reference(someMsgIDNo)

sensors(someSensor)

}

[001201] data_channels

[001202] See channels.

[001203] data_message

[001204] See message types.

[001205] data_publish_condition

[001206] All sensor adapters are capable of When data_publish_condition evaluates to true, the system variable LAST_TIME_DATA_COND_TRUE is set

to the current time and the sensor adapter's data_message is published on the sensor adapter's data_channels. For example, if a sensor adapter instance contains the following lines:

data_publish_condition

[001207] Concentration > 2.5

[001208] Evaluates to true only concentration readings > 2.5 are observed that the time of the reading will be stored in the LAST_TIME_DATA_COND_TRUE variable. This statement can be used in lieu of a policy.

[001209] depth

[001210] Used to set the number of reading or polling events above or below a threshold needed to trigger an alarm. The value is an integer.

[001211] double

[001212] Double is an SPML primitive data type. Double is a 64-bit double precision floating point number. It is used conventionally.

[001213] else

[001214] Conditions in SPM are created using the if/then/else/fi keywords. They are used conventionally.

[001215] enum

[001216] The enum statement in SPM is used conventionally.

[001217] equipment_alarm_channels

[001218] See channels.

[001219] equipment_alarm_message

[001220] See message_types.

[001221] event_server

[001222] All messages are sent to the EventServer. The EventServer is responsible for writing persistent messages to the database. The ViperServer process does not directly interact with the database when messages are made persistent.

[001223] event_servers

[001224] The event_servers keyword is included in the viper_server definition statement where it lists one or more EventServers with which the Viper Server interacts. EventServers are used in the order listed. The first one is the EventServer of choice. If something goes wrong with that EventServer, the next one on the list will be used.

[001225] Each EventServer requires a definition statement similar to the following:

[001226] event_servers

[001227] EventServer1 : host "vialinux8" tcp_server_port 5556

[001228] This statement specifies that events will be stored on the server vialinux8 and the TCP/IP connection made via port 5556.

[001229] false

[001230] See boolean. It is used conventionally.

[001231] fi

[001232] The fi keyword is used to mark the end of an if/then/else/fi statement.

[001233] field_bindings

[001234] The field_bindings keyword can be used in a policy_instance to bind a name to a field or to create an alias. In this example “City” and “State” are bound to the corresponding fields of the Location subtype.

[001235] field_bindings

[001236] City : Location.Location_city

[001237] filter

[001238] The filter keyword instructs a sensor adapter to check what kind of a measurement the sensor is making. For example, a sensor might be capable of reading concentrations of any number of gases depending on the cartridge being used. The filter statement is always used in conjunction with the SPModel property. The following instructs the sensor adapter to check a register for the value it holds. If the value is “1”, the sensor is using a carbon monoxide cartridge.

```
        State : Location.Location_state
sensor_adapter_instance ITrans_CO {
    template ITransSingleHead_Template
    channels
        data_channels
            JmsMsgServer::Chem.Data.ITrans.Gas.CarbonMonoxide
            LocalMsgServer::LocalCommAlarmChannel
    filter RegisterValues[2]{1:8} == 1
    properties
        Enabled = 1
```

```
        SPModel = "ITrans"
}
```

[001239] host

Used in jms_message_server, soap_message_server, and tcp_message_server definitions to specify the physical server used to send and receive messages. The full definition includes the IP address (or host name) and the port number.

[001240] ICD100

[001241] Interface Control Document – in SPML this refers to the SEIWG ICD message schema. See SDO Glossary topic and `message_type` keyword entry.

[001242] if/then/else/fi

[001243] Used to create conditional statements and expressions. They are used conventionally.

[001244] input_channels

[001245] See **Error! Reference source not found.**.

[001246] Integer

[001247] An SPML primitive data type. It is used conventionally.

[001248] jms_message_server

[001249] Keyword used in SPML to define a JMS message server instance. See

[001250] `message_servers`. The JMS message server statement is similar in form to that of the tcp_message_server.

[001251] local_message_server

One of the four possible message servers in an SPM system. In SPM, a message server is a software application, not a physical piece of hardware. The other message servers are the

[001252] `soap_message_server`, the

[001253] `jms_message_server` and the

[001254] `tcp_message_server`. Unlike the other message server types, the Local message server statement does not contain a host address or port number (it is on the same physical machine as the Viper), and it also does not include a properties statement.

[001255] Long

[001256] The long integer data type in SPML. It is used conventionally.

[001257] message_protocol

[001258] The message_protocol specifies the kind of XML schema the message is based on. The specification is one of the parameters in a **Error! Reference source not found.** definition. Current message_protocol options that make up the SPM

[001259] `SDO` are

[001260] `SML|ICD100|`**Error! Reference source not found.**. The default is SML.

[001261] message_servers

[001262] There are fours types of message server in SPM.

- Local – a Local Message Server instance exists on each computer that processes data from one or more sensors. The actual processing of the data

is done by the Sensor Adapter software instances on the local computer. A Local Message Server sends the processed data directly and immediately to the SPM Policies where they will be executed. A Local Message Server can only dispatch messages immediately and within its network firewall.

- JMS – if the consumer of a message is outside the Viper Server, the message must be routed through a JMS Message Server. An example of an outside consumer would be the SPM GUI.
- SOAP – if the message has to go from one Viper Server to another without going through a JMS Message Server, it must be routed through a SOAP Message Server.
- TCP – some IP-enabled systems do not subscribe to JMS or SOAP. SPM supports a TCP Message Server in order to communicate with such systems.

[001263] The message_servers keyword is used in the viper_server definition statement to list the message server instances in the Viper Server instance.

```
viper_server ViperServer {
    soap_server_port 4000
    message_servers
        JmsMsgServer
        LocalMsgServer
    registers
        TOXIC_FUME_ALERT_POSTED_REGISTER : boolean = false
    sensor_adapters
```

Camera1

...

policies

BioAgents_POS

BioAgents_NEG

...

}

[001264] message_structure_map

[001265] This keyword is used in **Error! Reference source not found.** definition statements when data must be converted from one format to another. The conversion is effected by using a previously defined

[001266] `type_structure_map`.

channel Chem.Data.ITrans.Gas.CarbonMonoxide

message_type ITransMessage

message_value_map ITrans_value_map

message_structure_map ModbusTCP_to_ITrans_map

[001267] message_type

[001268] This keyword (singular) is used to include a required parameter in a **Error! Reference source not found.** statement.

[001269] message_types

There are four types of messages in SPM.

- data_message – produced by a sensor adapter
- alert_message – produced by a sensor adapter

- comm_alarm_message – produced by a sensor adapter
- equipment_alarm_message – produced only by a sensor

[001270] message_value_map

This keyword is used in **Error! Reference source not found.** definition statements when data must be converted from binary to ASCII format. The conversion is effected by using a previously defined type_value_map. See the entry for type_structure_map below.

[001271] output_channels

[001272] See **Error! Reference source not found.**.

[001273] Persistent

[001274] A persistent message is stored in the database. Most messages are defined as persistent. Video messages are an exception since, owing to their size, flagging them as persistent would quickly fill the ordinary database. When a separate video server is in place, video messages can be safely made persistent.

[001275] Policies

[001276] A policy is a condition/action pair. The policies keyword is used in the viper_server declaration to list the policy instances that will be activated in the configuration. For an example, see the

[001277] `message_servers` entry.

[001278] policy_instance

[001279] An instantiation of a `policy_template`; i.e., specific values for conditions and actions are set, usually by reference by means of a series of bindings. The viper_server declaration contains a list of the policy instances that

will be activated on the current configuration. A policy_instance definition starts by referencing a policy_template using the template keyword:

```
policy_instance TOXIC_FUME_instance {
    template TOXIC_FUME_template
    input_channels
        JmsMsgServer::Chem.Data.SomeMfg.Gas.CarbonMonoxide trigger
        . . .
    constants
        Toxic_fume_threshold_co : 5.0
        . . .
    channel_bindings
        co_channel :
            JmsMsgServer::Chem.Data.SomeMfg.Gas.CarbonMonoxide
        . . .
    register_bindings
        ALERT_POSTED_REGISTER :
            ViperServer::TOXIC_FUME_ALERT_POSTED_REGISTER
    sensor_bindings
        Sensor : ViperServer::SomeSensorName
}
```

[001280] policy_template

[001281] A policy template comprises a condition template and an action template:

```
policy_template TOXIC_FUME_template {
    condition_template TOXIC_FUME_cond_template
    action_template TOXIC_FUME_action_template
}
```

[001282] When testing, use the TRUE_condition_template to force the policy always to execute no matter what the observed value for the tracking variable is:

```
condition_template TRUE_cond_template {
    true
}
```

Thus:

```
policy_template TOXIC_FUME_template {
    condition_template TRUE_cond_template
    action_template TOXIC_FUME_action_template
```

[001283] process_manager

[001284] Used when defining a Process Manager instance in the ProcessManager.spm file. This file resides in the /usr/local/spm/resources folder of the SPM server.

[001285] The process_manager has the following optional properties:

Table of ThermoElectronMessage Properties

Property name	Description	Value
ViperHeartBeatInterval	double, the ProcessManager will check the ViperServer's status every	5.0 (sec)

	ViperHeartBeatInterval	
MaxTriesToStartViper	long, see below	3
MaxTimeToStartViper	long, the ProcessManager will try up to 'MaxTriesToStartViper' times in 'MaxTimeToStartViper' seconds to start a viper before it gives up	5 (sec)

[001286] process_managers

[001287] Used in a configuration_server definition to list the process managers controlled by the Configuration Server. The inclusion of the list has been made optional so that the CompilerConfigServer definition does not need to list any process_managers.

[001288] Properties

[001289] The properties keyword in a sensor_adapter_template introduces a listing in which the data types of the properties are declared or default values are set. In a sensor_adapter_instance the keyword introduces a listing in which property values particular to that instance are set. If a particular property is not listed in the instance, its default value from the template will be used.

[001290] Protocol

[001291] Each sensor_adapter_template declaration must include a protocol statement. The Viper Server code contains a list of hard-coded sensor adapter protocols that vary for each SPM implementation and customer.

```
sensor_adapter_template myCameraTemplate {
    message_types
        data_message      VideoMessage
        comm_alarm_message CommAlarmMessage
    protocol myProtocol
    properties
        Enabled      integer
        . . .
        geoLocation   GeoLocationType
}
```

[001292] When the Viper is called upon to run a sensor_adapter_instance, it will employ the hard-coded protocol declared in the pertinent sensor_adapter_template.

[001293] Publish

[001294] The publish statement appears in the **Error! Reference source not found.**, and instructs the application which messages to generate when the action is triggered. Other statements in an action_template are `Set` and `control`.

[001295] reference_channels

[001296] See **Error! Reference source not found.**.

[001297] Registers

[001298] The registers keyword appears in the viper_server configuration statement and introduces a list of registers. A register typically indicates a state

(true|false), or contains a string or some numerical value.

[001299] register_bindings

[001300] A register binding creates an alias for the register. Register bindings are made in the policy_instance statement and bind some alias to the register value defined in a viper_server statement. For example, in this viper_server definition the initial state of an alert is set to 'off':

```
viper_server myViperServer{
. . .
    TOXIC_ALERT_REG : boolean = false
}
```

One can instruct a policy to refer to that alert status by means of the register_bindings statement when defining the policy_instance:

```
policy_instance TOXIC_ALERT {
    . . .
    register_bindings
        TOX_ALERT_STATUS : myViperServer::TOXIC_ALERT_REG
    . . .
}
```

[001301] register_reference

[001302] A register_reference points to the value of a register. In this example, the condition is met if the concentration of either of two gases exceeds the threshold values set in the policy_instance constants statement, but only if an

alert has not been posted already:

```
condition_template TOXIC_cond_template {
    (((channel_reference(co_channel->Conc) >
      constant_reference(Toxic_threshold_co)) |
     (channel_reference(ch4_channel->Conc) >
      constant_reference(Toxic_threshold_ch4))) &
     (~ register_reference(ALERT_POSTED_REGISTER)))
}
```

[001303] remote_viper_servers

[001304] Used in a viper_server instance definition to list other Viper Server instances located on other servers. The definition includes host IP address or hostname and soap_server_port number.

[001305] SDO

[001306] Service Data Object – see SDO Glossary entry. An SPM SDO may be in the form of one of three

[001307] `message_protocol`s,

[001308] `SML` (default), `ICD100` or **Error! Reference source not found.**.

[001309] Sensors

[001310] Used with the control keyword to denote a listing of sensors in an action_template.

[001311] sensor_adapters

[001312] The sensor_adapters keyword is used in the viper_server configuration statement of an SPM file to denote a listing of

sensor_adapter_instances.

[001313] sensor_adapter_instance

[001314] A sensor_adapter_instance is an instantiation of a sensor_adapter_template, and defines property values for the instance. Data and message channels are also defined in the instance.

[001315] sensor_adapter_template

[001316] The sensor_adapter_template declares the message_types, sensor adapter

[001317] `Protocol`, and properties associated with a given sensor adapter.

[001318] sensor_bindings

[001319] The sensor_bindings keyword is used inside a policy_instance to tell SPM what sensor to use to carry out an action. The bound sensor must be included in the sensor_adapters statement in the pertinent viper_server definition.

[001320] For example: in an action_template we can use the control statement to declare what action SPM should carry out when a condition is met. In this example, the action would be to publish a commandMessage which has been defined as a constant. The message is to be displayed on the messaging device called InovaDisplay.

control

commandMessage : constant_reference(Toxic_fume_alert_message)

sensors(InovaDisplay)

[001321] In order for SPM to know which sensor is the InovaDisplay device,

we include a sensor_bindings statement in the policy_instance to bind the InovaDisplay name to a specific Inova device.

```
policy_instance TOXIC_FUME_instance {
    template TOXIC_FUME_template
        . . .
    sensor_bindings
        InovaDisplay : myViperServer::Inova
}
```

[001322] Custom Properties

[001323] The specific display device is handled by a sensor_adapter_instance called Inova. All sensors in the pertinent Viper are listed in the viper_server instance definition toward the end of the file.

```
viper_server myViperServer {
    soap_server_port 4000
        . . .
    sensor_adapters
        Inova
}
```

[001324] Custom Properties

SPM now knows that the display device called InovaDisplay in the policy_instance is one of the sensor_adapters listed in the viper_server definition, more specificially, the Inova sensor adapter, and it will look at the parameters in

the Inova sensor_adapter_instance to determine how the message should be displayed.

[001325] Set

[001326] The set statement is used in an **Error! Reference source not found.** to change register values when the action is executed. Other statements in an action_template are `Publish` and `control`.

[001327] SML

[001328] Sensor Markup Language – see the SML Glossary topic.

[001329] snmp_message_server

[001330] Keyword used in the processmanager.spm file to define the physical server which handles SNMP traps.

[001331] soap_message_server

[001332] Keyword used in SPML to declare a SOAP message server. The SOAP message server statement is similar in form to that of the

[001333] `tcp_message_server`.

[001334] soap_server_port

[001335] The definition of a SOAP end point requires a host name and port number in the form:

host “IPAddress|hostname”:NNNN

[001336] String

[001337] The string data type in SPML. It is used conventionally.

[001338] tcp_message_server

[001339] Keyword used in SPML to declare a TCP message server. The

message server statement is used to define channels for that server. It can also include an optional properties list. Except for the local message server, all message server statements must contain a host and port definition.

```
tcp_message_server TcpMsgServer_ICD {
    host "10.32.63.27":7000
    properties
        PollingInterval = 1.0
        MaxConnectFailures = 3
        MaxSendFailures = 0
    channel ICDMaster
        message_type PlatformStatusReport
        message_protocol ICD100
    channel DSR
        message_type DeviceStatusReport
        message_protocol ICD100
        message_structure_map SPM_DSR_to_ICD_DSR_map
}
```

[001340] Custom Properties

[001341] tcp_server_port

[001342] The definition of a TCP end point requires a host name and port number in the form:

host “IPAddress|hostname”:NNNN template

[001343] The template keyword is used in sensor adapter or policy instance

definitions to reference the pertinent sensor adapter or policy template.

[001344] Then

[001345] See

[001346] `if/then/else/fi`.

[001347] Timespan

[001348] Sets the range of time of reading or polling events with values above a threshold needed to

[001349] `Trigger` an alarm. Unit is milliseconds.

[001350] Trigger

[001351] Used when defining the input_channels in a policy_instance to specify which channel is supposed to function as a trigger for sending an alert or alarm message or other action. (A policy instance can also access information in other channels that would not trigger an action. Such information may be, for example, for display only. In such cases the trigger property would not be assigned to the informational channels.) The trigger flag can also be used together with a

[001352] `Timespan` and/or **Error! Reference source not found.** to set conditions for invoking the trigger.

[001353] True

[001354] See **Error! Reference source not found.**. It has the conventional meaning.

[001355] Type

[001356] Keyword used to define message types.

[001357] type_structure_map

[001358] There are two kinds of maps, namely "structure maps" and "value maps." The purpose of a structure map is to convert between two message formats. For example, from a SensorML-like format to **Error! Reference source not found.** or ICD100. The purpose of a value map is to change the presentation of the data pulled from sensors. This is typically used for display purposes and is particularly useful for converting binary format to ASCII.

[001359] The following type_structure_map is for an ITrans sensor. Output from the Modbus sensor can be managed using a generic Modbus sensor adapter together with a structure map and a value map.

```
type_structure_map itrans_type_map {
    RegisterValues[0]{9:15} : InstrumentType
    RegisterValues[1]       : Concentration
    RegisterValues[2]{1:8}  : NameOfDetectedGas
    RegisterValues[5]{1:2}  : Alarm
    RegisterValues[5]{5:6}  : EquipmentFault
}
```

[00369] The Modbus sensor adapter reads the registers and places the results in the "RegisterValues" array which is a part of the raw message. The map above specifies how to map this raw message into the refined message (which in this case has been defined earlier as "ITransMessage"). The first entry is an instruction to take the value in the register array (index 0) for bits 9-15 and place it in a final field named "InstrumentType." The second entry instructs to

take the value in the register array (index 1) and place it in a final field named "Concentration," and so forth.

```
type ITransMessage {
    Timestamp          string
    . . .
    InstrumentType     string
    NameOfDetectedGas  string
    Concentration      double
    Alarm              string
    EquipmentFault     string
    . . .
    geoLocation        GeoLocationType
}
```

[001360] While this makes everything a little more intuitive, we are still dealing with binary data. For display purposes, ASCII is preferable. The type_value_map provides the means for translating to a readable format.

[001361] type_value_map

[001362] Type value maps are used to change the presentation of data that is pulled from sensors. This is typically done for display purposes and is particularly useful for converting binary formats to ASCII. The type_value_map keyword invokes the logic of a case statement.

```
type_value_map itrans_value_map {
    RegisterValues[0]{9:15}  == 768   :"BBIR (broad band IR)"
    RegisterValues[0]{9:15}  == 1024  :"TOX (toxic)"
```

```
    . . .
    RegisterValues[2]{9:16}  == 0     :  1
    RegisterValues[2]{9:16}  == 256   :  10
    RegisterValues[2]{9:16}  == 512   :  100
    . . .
    RegisterValues[2]{9:16}  == 1280  :  100000
    RegisterValues[1]              : / RegisterValues[2]{9:16}
    RegisterValues[2]{1:8}   == 1     : "CO_Carbon_Monoxide"
    RegisterValues[2]{1:8}   == 2     : "H2S_Hydrogen_Sulfide"
    . . .
    RegisterValues[2]{1:8}   == 22    : "LEL (Combustible_Gases)"
    RegisterValues[2]{1:8}   == 0     : "Unknown"
    RegisterValues[5]{1:2}   == 0     : "Clear"
    RegisterValues[5]{1:2}   == 1     : "Low"
    RegisterValues[5]{1:2}   == 2     : "High"
    . . .
}
```

[001363] The first few entries map to the “InstrumentType” field in the final message. The first two entries mean: Look at bits 9-15 in the first entry in the register array. If it evaluates to “true”, take the most significant byte and divide it by 256. If the result is equal to “3” (768/256 = 3), the value to place in “InstrumentType” is the third option “BBIR (broad band IR)”. If the result is “4”, the “InstrumentType” is “TOX (toxic)”. It is occasionally necessary to perform simple arithmetical operations on numeric values. Value maps also allow for this kind of conversion. The following lines are used to indicate the order of magnitude of the concentration stored in RegisterValues[1].

```
    RegisterValues[2]{9:16}  == 0     :  1
    RegisterValues[2]{9:16}  == 256   :  10
```

```
    ...
    RegisterValues[2]{9:16}  == 1280  :  100000
    RegisterValues[1]              : / RegisterValues[2]{9:16}
}
```

[001364] Custom Properties

[001365] A similar procedure is followed to assign the values for the type of gas and the sensor status by looking at the appropriate register values.

[001366] viper_server

[001367] The keyword is used when defining a Viper Server instance. The instance definition contains the following components:

```
viper_server ViperServer {

    soap_server_port nnnn
        ...
    remote_viper_servers
        ...

    message_servers
        ...

    Registers
        ...

    sensor_adapters
        ...

    Policies
        ...
}
```

[001368] viper_servers

Used when including one or more Viper Server instances in a Process Manager.

```
process_manager ProcessMgr {
    host "vialinux2"
    soap_server_port 4110
    viper_servers
        Example_ViperServer1 : soap_server_port 4000
}
```

[001369] SPM system can provide massively scaleable sensor applications that drive QOS-based network growth. SPM enales convergence of sensor - video - voice to drive growth and bandwidth demand faster.

[001370] SPM is a generalized network-centric policy framework for delivering sensor, video and GIS interoperability. SPM is sensor-centric focused, standards-based, and delivers high QOS for sensor processing with no false alarms, no event drop-outs. SPM includes a sensor virtualization engine in a “product-format” that integrates multiple sensors from multiple vendors providing data in multiple formats. Extensive and complete policy framework is provided for addressing multiple verical requirements. SPM can addresses real-time, mission critical requirements and guaranteed timeliness in sensor event processing.

[001371] Scalability

[001372] SPM is scalable with respect to various factors. For example, the SPM is scalbe with respect to Sensor types, number of Sensors; Policies; Policy

complexity; Concurrent Policies; Information products; and Information syndication.

[001373] Example of Sensor and Sensor Systems

[001374] SPM provides Sensor Supporting Legacy protocols for protocols such as Modbus, lonworks, Baknet etc. These protocols typically use the serial to IP gateway to touch the network. Sensors can be connected via serial interfaces, such as RS232, RS485, and UART devices, Dry contacts, Relays, etc. Sensors supporting IP protocols include Http, Telnet, SSH, SOAP, CAP1.1, XMLoIP, SNMP or an IP socket. SPM can include Sensor Systems with ICD (Interface Control Document) XML schema, such as PTZ systems, DOD Sensor systems etc. that have certification and compliance requirements. SPM can include Sensor Systems that allow access to memory structures that typically do not have an APIs and require some work for integration at a low level.

[001375] Cisco® IPICS leverages the Base code from Cisco IPCC Express that allows HTTP Post. SPM application can uses Java API towards SPM and HTTP post on the IPICS 2.0. The message comes from SPM and is passed on to the IPICS 2.0 by the demo application for notification to individual user. IPICS policies may not be programmatically triggered. Also, SPM can include a supported interface for SPM-IPICS integration.

[001376] SPM Overview - Features

[001377] SPM is an all-in-one data and alert tracking system that provides centralized access to all relevant data from disparate sensors, cameras, and communication systems. SPM also provides automated retrieval, intelligent

processing and analysis of data. SPM works on condition that are action based policies. Also, SPM provides database storage and on-demand retrieval of data and alerts. Web-based GUI can be implemented to provide worldwide access to data and alerts.

[001378] SPM Intergration

[001379] SPM can provide IP standards-based solution. Also, SPM provides two way integration with communication systems, sensors and cameras. Database integration are also provide. FIG. 164 shows an example SPM event traceability.

[001380] SPM Components

[001381] Componets of SPM includes the Viper Policy Engine, Sensor Policy Client API, Configuration Server, Sensor Adapters and Administration Console, such as a graphical user interface (GUI).

[001382] VIPER Policy Engine

[001383] The policy engine is a Very High Density Policy Encoder controls the connections between the various sensors and their respective adapters as well as associated policies. The policy engine controls input and output channels. Also, the policy engine can publish/subscribe paradigm. For example, each data producer can publish to a topic and the consumers can subscribe to it. The policy engine also contains policies, such as the Condition->Action Pair. In addition, the policy engine provides the algebra for dealing with any combination of user-specified policies and sensor attributes. Each policy engine includes one or more of the same or different sensors. One or more

policy engines can exist in same SPM system.

[001384] VIPER Policy Engine Controlled by VIPER Server Configuration File

[001385] The server configuration file can be edited using the Administration Console, implemented as a GUI, by expert users. The server configuration file sets the configuration for Sensor Adapters, point to input and output channels used.specifies registers and specifies policy templates and specific instances.

[001386] The following is an example of VIPER Policy Engine – VIPER Server Configuration file –Input and Output Channels Used.

```
jms_message_server vialinux2JmsServer {

        host "vialinux2":1099

        //######### All the channels hosted by the message server (with the corresponding message types):
        channel Bio.Data.Cepheid.GeneXpert.Temp type GeneXpertMessageType persistent
        channel Bio.Data.Cepheid.GeneXpert type GeneXpertMessageType persistent
        channel Bio.Data.Cepheid.SmartCycler.Temp type SmartCyclerMessageType persistent
        channel Bio.Data.Cepheid.SmartCycler type SmartCyclerMessageType persistent
        channel Bio.Data.QTL.Biosensor_2000 type QTLData persistent
        channel Bio.Alert.Monitors.Directives type AlertMessageType persistent
        channel Bio.Alert.Technician type AlertMessageType persistent
        channel Bio.Alert.LabManager type AlertMessageType persistent
        channel Bio.Alert.CoastGuard type AlertMessageType persistent
        channel Bio.Alert.PublicHealthAuthority type AlertMessageType persistent
        channel Bio.Alert.FBI type AlertMessageType persistent
        channel Bio.Alert.FDA type AlertMessageType persistent
        channel Bio.Alert.USDA type AlertMessageType persistent
        channel Bio.Alert.Veterinarian type AlertMessageType persistent
        channel Bio.Alert.FireDepartment type AlertMessageType persistent
        //######### END All the channels hosted by the message server (with the corresponding message types).

} // END message server vialinux2JmsServer
```

[001387] The following is an example of VIPER Policy Engine –VIPER Server Configuration file –Configuration for Sensor Adapter Template.

```
// Template for QTL Biosensor 2000 sensor adapter that is running on the local server:
sensor_adapter_template QTLBiosensor2000SA_template {

    message_types
            data_message QTLData
            comm_alarm_message CommAlarmMessageType

    protocol QTLBioSensor

    properties
            Enabled                                 integer
            IPAddress                               string
            port                                    integer
            PollingRate                             double
            GeoLocation                             GeoLocationType
}
```

[001388] The following is an example of VIPER Policy Engine – VIPER Server Configuration file – Configuration for Sensor Adapter Instance.

```
// QTL Biosensor 2000 sensor adapter that is running on the local server:
sensor_adapter_instance QTLBiosensor2000SA {

    template QTLBiosensor2000SA_template

    channels
            data_channels
                    Bio.Data.QTL.Biosensor_2000
            comm_alarm_channels
                    MyLocalMsgServer::LocalCommAlarmChannel

    properties
            Enabled                                         : 1
            IPAddress                                : "64.210.18.24"
            port                                            : 2107
            PollingRate                              : 1.0
            geoLocation.longitude   : -118.282000
            geoLocation.latitude    : 34.158000
            geoLocation.altitude    : 0.0
}
```

[001389] The following is an example of VIPER Policy Engine – VIPER Server Configuration file – Sensor Adapter – Data Elements.

```
type QTLData {
    Timestamp                   string
    Timezone                    string
    Signal                      double
    Threshold                   double
    CalibrationFactor           double
    AssayType                   string
    ManufacturingDate           string
    SerialNo                    string
    SampleId                    string
    Instrument_SN               string
    TestResult                  string
    HazMat                      string
    Location                    LocationType
    geoLocation                 GeoLocationType
}
```

LocationType and GeoLocationType are similarly defined in this file

[001390] The following is an example of VIPER Policy Engine – VIPER Server Configuration file – Policies – Condition/Action Template.

```
policy_template QTL_POS_template {

    condition
            (channel(QTL_biosensor2000_TestResult input) == "Positive")

    action
            publish
                    alert : "ALERT: " || channel(QTL_biosensor2000_AssayType input) || " found in sample " ||
                                        channel(QTL_biosensor2000_SampleId input) || ".  Detected by " ||
                                        channel(QTL_biosensor2000_HazMat input) || " in " ||
                                        channel(QTL_biosensor2000_City input) || ", " ||
                                        channel(QTL_biosensor2000_State input) || "."
                    reference channels(QTL_biosensor2000_data_channel)
                    on channels {Bio_alert_technician_channel, Bio_alert_lab_manager_channel, Bio_alert_fire_department_channel},
            control
                    commandMessage : "ALERT: " || channel(QTL_biosensor2000_AssayType input) || " found in sample " ||
                                                                channel(QTL_biosensor2000_SampleId input) || ".  Detected by " ||
                                                                channel(QTL_biosensor2000_HazMat input) || " in " ||
                                                                channel(QTL_biosensor2000_City input) || ", " ||
                                                                channel(QTL_biosensor2000_State input) || "."
                    sensors(Inova)
}
```

[001391] The following is an example of VIPER Policy Engine – VIPER

Server Configuration file – Policies – Instance pointing to template.

```
policy QTL_POS {

    template QTL_POS_template

    channel bindings {
            QTL_biosensor2000_data_channel : Bio.Data.QTL.Biosensor_2000
            QTL_biosensor2000_TestResult : Bio.Data.QTL.Biosensor_2000->TestResult
            QTL_biosensor2000_AssayType : Bio.Data.QTL.Biosensor_2000->AssayType
            QTL_biosensor2000_SampleId : Bio.Data.QTL.Biosensor_2000->SampleId
            QTL_biosensor2000_HazMat : Bio.Data.QTL.Biosensor_2000->HazMat
            QTL_biosensor2000_City : Bio.Data.QTL.Biosensor_2000->Location.Location_city
            QTL_biosensor2000_State : Bio.Data.QTL.Biosensor_2000->Location.Location_state
            Bio_alert_technician_channel : Bio.Alert.Technician
            Bio_alert_lab_manager_channel : Bio.Alert.LabManager
            Bio_alert_fire_department_channel : Bio.Alert.FireDepartment
    }

    sensor bindings {
            Inova : Bio_ViperServer::Inova
    }
}
```

[001392] VIPER Policy Engine provides real-time dynamic routing of events, data and normalized information from the Sensor Adapters to the policy Actions (or multiple destinations). The policy engine can implement condition-based execution of user-define sensor policies and policy assertions. The policy engine can provide efficient algebra for encoding combinations of user-specified policies and sensor attributes. In addition, the policy engine can encode policies based on single sensor datum, multiple sensors of the same type from the same vendor, multiple sensors of the same type from different vendors, or multiple different types of sensors from multiple different vendors. In addition, the policy engine can provide a run-time repository for all policies persisted within the network. The policy engine can operate as a primary sensor information

syndication resource by guaranteeing delivery of sensor data to any user or policy that requires it. Alos, all communication within SPM can take place through topics with the exception of command and control information. Each policy encoded in VIPER can have a global unique identification (GUID). Further, The VIPER policy engine can provide regular heartbeats to the Administration API.

[001393] Sensor Policy Client API

[001394] The Java API can allow the clients to configure and access the functionality of the VIPER Software. The Java API can be delivered as a single jar file, which contains the Java classes. MLAPI provide a Java interface for carrying out the various operations. For example, MLAPI can provide functionality to create, delete, modify and retrieve (1) Policy Alternatives; (2) Policy Assertions; (3) Policy Actions; (4) Sensors; (5) Topics; and (6) Users.

[001395] Also, MLAPI can be capable of provisioning new VIPER and SP instances. MLAPI can be capable of reporting on VIPER and SP instances in the current deployment. MLAPI can provide a heartbeat monitor for VIPER policy engine. Further, MLAPI can allow subscription to topics.

[001396] Sensor Adapters

[001397] Sensor Adapters receive and convert data from the physical sensors into a common SPML. The SPML is sent to the standard VIPER protocol. The Sensor Adapters can utilize specialized interface libraries to communicate with each sensor type and vendor model. The Sensor Adapters can include intelligent algorithms for conversion of complicated data into easily

understandable data for downstream representation and policy evaluation.

[001398] The Sensor Adapters (SAs) can encapsulate all complexity associated with the specifics of integrating sensors from different vendors. The SA can acquire samples from a network attached sensor at a prescribed frequency, apply low level signal processing logic and syndicate the raw data and events to any subscriber of that topic. The SA's interface libraries can normalize sensor data in alignment with SensorML standards. The SAs can support the following types of processing of sensor measurements: (1) Data gating; and (2) Rescaling. The SAs can support the geo-location of the measured data.

[001399] A Sensor Adapter can have the capability to autonomously manage an individual sensor, a sensor group or a sensor array. The Sensor Adapter can provide a complete encapsulation around the APIs provided by the sensor manufacturer to manage the sensor and unify the interface for executing such functionality across all sensor types. Each Sensor Adapter and each sensor that it manages can have a globally unique identifier (GUID). The Sensor Adapter can expose an identical interface to the VIPER policy engine, the Admin Agent Mediator and to the Administration API.

[001400] An SA API can be used to manage Fault Management (as applicable), Configuration Management and Security Management. The Sensor Adapters can accommodate both polled sensors and asynchronous event based sensors. Further, the Sensor Adapters can provide a heartbeat to the Administration API.

[001401] IP-based Generic Sensor Adapter (GSA)

[001402] An IP-based GSA can b eintroduced to facilitate integration of, emerging IP-based OEM Sensor gateways, standalone sensor gateways and hardware protocol conversion modules, within the Client Policy Framework. A gateway may already be performing some of the SA signal processing, normalization, and/or aggregation functionality. The GSA input may include standardized sensor outputs (e.g., TML format). The GSA is sensor-independent API that delivers all the SP functionality (syndication, policy assertions, etc.) to remotely attached edgegateway enabled sensors. The GSA can provide standards for gateway development to support policy based architectures.

[001403] Sensor Adapters – Communication Failures

[001404] For Sensor to Sensor Adapter, different protocols may be used depending on the sensor. If data needs to be sent to a different SPM, SOAP can be used between adapter and SPM. After the adapter, data formats and protocols are the same, and no longer dependent on the sensor. Heartbeats are being transferred from adapter to main SPM. Main SPM acknowledges heartbeat back to the adapter. Thyen the Adapter send a communication alarm about missing heartbeat acknowledgement. The adapter is responsible for obtaining its own IP address, and telling the main SPM where it is. The main SPM can then reestablish the various connections.

[001405] Administration Console

[001406] The Administration Console is a GUI-based tool and shall have the capability to test all functions of the MLAPI. The Admin Console can display active configurations for SAs and VIPER policy engine. The Admin Console can

allow the user to create, delete, modify, and retrieve: Policy Alternatives; Policy Assertions; Policy Actions; Sensors; Topics; and Users. The Admin Console can allow construction of simple rules for event gating on the SAs. The Admin Console canallow the user to provision or decommission VIPER and SA instances. The Admin Console can display faults and exceptions reported by VIPER and SA. The Admin Console can allow the user to clear historical information on any topic.

[001407] Administration Agent Mediator (AAM) Specifiction

[001408] The AAM can provide the sole interface to the permanent repository of configuration information (e.g., within embedded Informix Database) for all the SPM stack components and processes. The AAM can receive all configuration requests (create, modify, delete) from the Management API. The AAM can have the capability to store events in the database. In order to do so, it registers with the VIPER switch for those events which it needs to store.

[001409] SPM User Types

[001410] A user that uses SPM is assigned one or more roles. Roles define which SPM features a user can access and what functions that user can perform. The assigned roles include the End User (everybody has the user role); Expert User; and System Administrator.

[001411] The End User can access the Administration Console, view real-time data and alerts and retrieve archived data and alerts. The Expert User can perform policy management; integrate new sensor proxies; creates and assigns roles to users; associate users with one or more channels; and manage activity

logs. A System Administrator can perform many functions.

[001412] The Expert User can add/delete/edit individual users; add/delete/edit user groups and members; and set access permissions for individuals/groups.

[001413] Multiple SPM Servers – Use Case

[001414] For example, a Gas Sensor on SPM-Edge 3 can trigger (goes above a certain threshold more than 3 times and stays above threshold for more than 5 minutes) and the following can be implemented by SPM in response. The policy for this trigger requires the following: the SPM-EDGE2 Video from the buffered video; Pan / Tilt and capture the video from SPM-Edge4 Video camera; Send the alarm to the Alarm-System (SPM-EDGE2) connected to the SOC; Stop the HVAC system on SPM-EDGE 3; Create an event on the Main SPM; Allow access to the dashboard on the Main-SPM; and Trigger a communication to a predefined number using the Main SPM.

[001415] Then, almost the same time Glass Break sensor triggers on the SPM-Edge3. The Policy includes the SPM-EDGE2 Video from the buffered video; Pan / Tilt and capture the video from SPM-Edge4 Video camera; Send the alarm to the Alarm-System (SPM-EDGE2) connected to the SOC; Create an event on the Main SPM; Allow access to the dashboard on the Main-SPM; and Trigger a communication to a predefined number using the Main SPM

[001416] SPM – SPM Communications

[001417] When publishing data, each SPM has the option to use JMS channels or SOAP channels. Inter-SPM communication is carried out using

SOAP channels. This has a superior performance to JMS. The data is published in the efficient Service Data Object format.

[001418] Scalability

[001419] The SOAP server on the receiving side implements the reactor design pattern. This means that a new thread is started to service every new request received. This prevents messages from being discarded. Received messages are republished to local or JMS channels and the local queuing mechanism is used to buffer these messages.

[001420] The following are examples of SPML grammars.

```
0 $accept: spm_description "end of file"
1 spm_description: spm_components

2 spm_components: spm_component
3                 | spm_component spm_components

4 spm_component: type
5                 | type_structure_map
6                 | type_value_map
7                 | message_server
8                 | sensor_adapter_template
9                 | sensor_adapter_instance
10                | condition_template
11                | action_template
12                | policy_template
13                | policy_instance
14                | viper_server
15                | process_manager
16                | config_server

17 config_server: "configuration_server" distinguished_name "{"
"soap_server_port" "const_integer" process_manager_list "}"
18                | "configuration_server" error "}"

19 process_manager_list: "process_managers" distinguished_name
20                 | process_manager_list distinguished_name

21 process_manager: "process_manager" distinguished_name "{" "host"
"const_string" "soap_server_port" "const_integer" "viper_servers"
viper_server_list_for_process_manager "}"
22                | "process_manager" error "}"

23 viper_server_list_for_process_manager: distinguished_name ":"
"soap_server_port" "const_integer"
24                                                  |
viper_server_list_for_process_manager distinguished_name ":"
"soap_server_port" "const_integer"

25 message_server: "local_message_server" distinguished_name "{"
msg_server_channel_defs "}"
26                | "jms_message_server" distinguished_name "{"
"host" "const_string" ":" "const_integer" msg_server_channel_defs "}"
27                | "soap_message_server" distinguished_name "{"
"host" "const_string" ":" "const_integer" msg_server_channel_defs "}"
28                | "tcp_message_server" distinguished_name "{"
"host" "const_string" ":" "const_integer" msg_server_channel_defs "}"
```

```
29                  | error distinguished_name "{" "host"
"const_string" ":" "const_integer" msg_server_channel_defs "}"
30                  | "local_message_server" error "{"
msg_server_channel_defs "}"
31                  | "local_message_server" distinguished_name error
msg_server_channel_defs "}"
32                  | "local_message_server" distinguished_name "{"
error "}"
33                  | "local_message_server" distinguished_name "{"
msg_server_channel_defs error
34                  | "local_message_server" error "}"
35                  | "jms_message_server" error "{" "host"
"const_string" ":" "const_integer" msg_server_channel_defs "}"
36                  | "jms_message_server" distinguished_name error
"host" "const_string" ":" "const_integer" msg_server_channel_defs "}"
37                  | "jms_message_server" distinguished_name "{" error
"const_string" ":" "const_integer" msg_server_channel_defs "}"
38                  | "jms_message_server" distinguished_name "{"
"host" error ":" "const_integer" msg_server_channel_defs "}"
39                  | "jms_message_server" distinguished_name "{"
"host" "const_string" error "const_integer" msg_server_channel_defs "}"
40                  | "jms_message_server" distinguished_name "{"
"host" "const_string" ":" error msg_server_channel_defs "}"
41                  | "jms_message_server" distinguished_name "{"
"host" "const_string" ":" "const_integer" error "}"
42                  | "jms_message_server" distinguished_name "{"
"host" "const_string" ":" "const_integer" msg_server_channel_defs error
43                  | "jms_message_server" error "}"
44                  | "soap_message_server" error "{" "host"
"const_string" ":" "const_integer" msg_server_channel_defs "}"
45                  | "soap_message_server" distinguished_name error
"host" "const_string" ":" "const_integer" msg_server_channel_defs "}"
46                  | "soap_message_server" distinguished_name "{"
error "const_string" ":" "const_integer" msg_server_channel_defs "}"
47                  | "soap_message_server" distinguished_name "{"
"host" error ":" "const_integer" msg_server_channel_defs "}"
48                  | "soap_message_server" distinguished_name "{"
"host" "const_string" error "const_integer" msg_server_channel_defs "}"
49                  | "soap_message_server" distinguished_name "{"
"host" "const_string" ":" error msg_server_channel_defs "}"
50                  | "soap_message_server" distinguished_name "{"
"host" "const_string" ":" "const_integer" error "}"
51                  | "soap_message_server" distinguished_name "{"
"host" "const_string" ":" "const_integer" msg_server_channel_defs error
52                  | "soap_message_server" error "}"
53                  | "tcp_message_server" error "{" "host"
```

```
"const_string" ":" "const_integer" msg_server_channel_defs "}"
54                      | "tcp_message_server" distinguished_name error
"host" "const_string" ":" "const_integer" msg_server_channel_defs "}"
55                      | "tcp_message_server" distinguished_name "{" error
"const_string" ":" "const_integer" msg_server_channel_defs "}"
56                      | "tcp_message_server" distinguished_name "{"
"host" error ":" "const_integer" msg_server_channel_defs "}"
57                      | "tcp_message_server" distinguished_name "{"
"host" "const_string" error "const_integer" msg_server_channel_defs "}"
58                      | "tcp_message_server" distinguished_name "{"
"host" "const_string" ":" error msg_server_channel_defs "}"
59                      | "tcp_message_server" distinguished_name "{"
"host" "const_string" ":" "const_integer" error "}"
60                      | "tcp_message_server" distinguished_name "{"
"host" "const_string" ":" "const_integer" msg_server_channel_defs error
61                      | "tcp_message_server" error "}"

62 type: enum_type
63       | structured_type

64 enum_type: "enum" distinguished_name "{" enum_list "}"
65               | "enum" error "}"

66 enum_list: "const_string"
67              | enum_list "const_string"

68 structured_type: "type" distinguished_name "{" type_attributes
"}"
69                      | error distinguished_name "{" type_attributes "}"
70                      | "type" error "{" type_attributes "}"
71                      | "type" distinguished_name type_attributes "}"
72                      | "type" error "}"

73 type_attributes: scalar_type_attribute
74                      | structured_type_attribute
75                      | enum_type
76                      | structured_type
77                      | type_attributes scalar_type_attribute
78                      | type_attributes structured_type_attribute
79                      | type_attributes enum_type
80                      | type_attributes structured_type

81 scalar_type_attribute: "identifier" scalar_type_declaration

82 scalar_type_declaration: scalar_type array_decl
xml_attributes_decl scalar_type_init
```

```
83 scalar_type: "integer"
84              | "long"
85              | "double"
86              | "boolean"
87              | "string"
88              | "enum" "identifier"

89 array_decl: /* empty */
90             | "[" "]"

91 xml_attributes_decl: /* empty */
92                      | xml_attributes_list
93                      | "{" xml_attributes_list_of_lists "}"

94 xml_attributes_list_of_lists: xml_attributes_list
95                              | xml_attributes_list_of_lists
xml_attributes_list

96 xml_attributes_list: "<" xml_attributes ">"

97 xml_attributes: xml_attribute
98                      | xml_attributes xml_attribute

99 xml_attribute: "identifier" "=" "const_string"

100 scalar_type_init: /* empty */
101                     | "=" constant
102                     | "=" "{" constant_list "}"
103                     | ":" constant
104                     | ":" "{" constant_list "}"

105 constant_list: constant
106             | constant_list constant

107 structured_type_attribute: distinguished_name distinguished_name
xml_attributes_decl

108 type_structure_map: "type_structure_map" "identifier" "{"
attribute_structure_mappings "}"
109                             | "type_structure_map" error "}"

110 attribute_structure_mappings: type_structure_map_component ":"
type_structure_map_component
111                                     | attribute_structure_mappings
type_structure_map_component ":" type_structure_map_component
```

```
112 type_structure_map_component: distinguished_name array_index
bitmap

113 array_index: /* empty */
114             | "[" "const_integer" "]"

115 bitmap: /* empty */
116             | "{" "const_integer" "}"
117             | "{" "const_integer" ":" "const_integer" "}"

118 type_value_map: "type_value_map" "identifier" "{"
attribute_value_mappings "}"
119                     | "type_value_map" error "}"

120 attribute_value_mappings: left_type_value_map_component ":"
right_type_value_map_component
121                                 | attribute_value_mappings
left_type_value_map_component ":" right_type_value_map_component

122 left_type_value_map_component: distinguished_name array_index
bitmap left_type_value_map_component_value

123 left_type_value_map_component_value: /* empty */
124                                             | "==" constant

125 right_type_value_map_component:
right_type_value_map_component_operator constant
126                                         |
right_type_value_map_component_operator distinguished_name array_index
bitmap

127 right_type_value_map_component_operator: /* empty */
128                                             | "+"
129                                             | "-"
130                                             | "*"
131                                             | "/"

132 msg_server_channel_defs: msg_server_channel_def
133                                 | msg_server_channel_defs
msg_server_channel_def

134 msg_server_channel_def: "channel" distinguished_name
"message_type" distinguished_name message_protocol_decl
message_value_map_decl message_structure_map_decl is_persistent
```

```
135 message_protocol_decl: /* empty */
136                         | "message_protocol" message_protocol_token

137 message_protocol_token: "SML"
138                         | "SDO"
139                         | "CAP1_1"
140                         | "ICD100"

141 message_value_map_decl: /* empty */
142                         | "message_value_map" "identifier"

143 message_structure_map_decl: /* empty */
144                                 | "message_structure_map" "identifier"

145 is_persistent: /* empty */
146             | "persistent"

147 viper_server: "viper_server" distinguished_name "{"
"soap_server_port" "const_integer" remote_viper_servers message_servers
registers sensor_adapters policies "}"
148             | "viper_server" error "}"

149 remote_viper_servers: /* empty */
150                         | "remote_viper_servers"
remote_viper_servers_list

151 remote_viper_servers_list: distinguished_name ":" "host"
"const_string" "soap_server_port" "const_integer"
152                         | remote_viper_servers_list
distinguished_name ":" "host" "const_string" "soap_server_port"
"const_integer"

153 message_servers: /* empty */
154                     | "message_servers" message_server_list

155 message_server_list: distinguished_name
156                         | message_server_list distinguished_name
157 registers: /* empty */

158             | "registers" register_type_list

159 register_type_list: register_name ":" register_type register_size
160                     | register_type_list register_name ":"
register_type register_size

161 register_name: distinguished_name
```

```
162 register_type: distinguished_name
163                 | scalar_type_declaration

164 register_size: /* empty */
165                 | "(" "const_integer" ")"

166 sensor_adapters: /* empty */
167                 | "sensor_adapters" sensor_adapter_instance_list

168 sensor_adapter_instance_list: distinguished_name
169                                 | sensor_adapter_instance_list
distinguished_name

170 sensor_adapter_instance: "sensor_adapter_instance"
distinguished_name sensor_adapter_instance_description
171                         | "sensor_adapter_instance" error "}"

172 sensor_adapter_instance_description: "{" "template"
distinguished_name sensor_channels filter_description
sensor_adapter_instance_props "}"
173                                         | "{" error distinguished_name
sensor_channels filter_description sensor_adapter_instance_props "}"
174                                         | "{" "template" error
sensor_channels filter_description sensor_adapter_instance_props "}"
175                                         | "{" error "}"

176 sensor_channels: /* empty */
177                 | "channels" sensor_channels_list
178                 | error sensor_channels_list

179 sensor_channels_list: /* empty */
180                         | sensor_channels_list sensor_channels_pair

181 sensor_channels_pair: "data_channels" qualified_channel_list
182                         | "alert_channels" qualified_channel_list
183                         | "input_channels" qualified_channel_list
184                         | "comm_alarm_channels"
qualified_channel_list
185                         | "equipment_alarm_channels"
qualified_channel_list
186                         | error qualified_channel_list
187                         | "data_channels" error
188                         | "alert_channels" error
189                         | "input_channels" error
190                         | "comm_alarm_channels" error
```

```
191                         | "equipment_alarm_channels" error

192 filter_start: "filter"

193 filter_description: /* empty */
194                         | filter_start expression

195 sensor_adapter_instance_props: /* empty */
196                                 | "properties" vdn_value_list

197 vdn_value_list: distinguished_name scalar_type_init
198                         | vdn_value_list distinguished_name
scalar_type_init

199 condition_template: "condition_template" distinguished_name "{"
expression "}"
200                         | "condition_template" error "}"

201 action_template: "action_template" distinguished_name "{"
action_list "}"
202                         | "action_template" error "}"

203 policy_template: "policy_template" distinguished_name "{"
"condition_template" "identifier" "action_template" "identifier" "}"
204                         | "policy_template" error "}"
205 expression: factor
206                         | expression "+" expression
207                         | expression "-" expression
208                         | expression "||" expression
209                         | expression "*" expression
210                         | expression "/" expression
211                         | expression "&&" expression
212                         | expression "**" expression
213                         | expression "!&&" expression
214                         | expression "!||" expression
215                         | expression "^||" expression
216                         | expression "&" expression
217                         | expression "|" expression
218                         | expression "^" expression
219                         | expression "<<" expression
220                         | expression ">>" expression
221                         | expression "==" expression
222                         | expression "!=" expression
223                         | expression ">" expression
224                         | expression ">=" expression
225                         | expression "<" expression
```

```
226                     | expression "<=" expression
227                     | expression "-<" expression
228                     | expression ">-" expression

229 factor: constant
230             | reference_expression
231             | function_call_expression
232             | "-" factor
233             | "!" factor
234             | "~" factor
235             | "(" expression ")"

236 function_call_expression: distinguished_name "(" expression_list
")"

237 expression_list: /* empty */
238                     | expression
239                     | expression_list "," expression

240 constant: const_bool
241             | "const_integer"
242             | "const_double"
243             | "const_string"

244 const_bool: "true"
245             | "false"

246 trigger_qualifier: /* empty */
247                     | "trigger"

248 depth_qualifier: /* empty */
249                     | "depth" "const_integer"

250 timespan_qualifier: /* empty */
251                     | "timespan" "const_integer"

252 reference_expression: channel_ref_expr
253                             | constant_ref_expr
254                             | register_ref_expr
255                             | sensor_ref_expr
256                             | raw_ref_expr

257 channel_ref_expr: "channel_reference" "(" ref_expr_alias
array_index bitmap ")"

258 constant_ref_expr: "constant_reference" "(" ref_expr_alias ")"
```

```
259 register_ref_expr: "register_reference" "(" ref_expr_alias
array_index bitmap ")"

260 sensor_ref_expr: "sensor_reference" "(" ref_expr_alias ")"

261 raw_ref_expr: distinguished_name array_index bitmap

262 ref_expr_alias_list: ref_expr_alias
263                     | ref_expr_alias_list "," ref_expr_alias

264 ref_expr_alias: distinguished_name
265                     | distinguished_name "->" distinguished_name

266 distinguished_name: "identifier"
267                             | distinguished_name "." "identifier"
268                             | distinguished_name "." "const_integer"

269 policies: /* empty */
270             | "policies" policy_instance_list

271 policy_instance_list: distinguished_name
272                             | policy_instance_list distinguished_name

273 policy_instance: "policy_instance" distinguished_name "{"
"template" distinguished_name "input_channels" qualified_channel_list
constants_definition channel_bindings field_bindings register_bindings
sensor_bindings "}"
274                     | error distinguished_name "{" "input_channels"
qualified_channel_list "template" distinguished_name
constants_definition channel_bindings field_bindings register_bindings
sensor_bindings "}"
275                     | "policy_instance" error "}"

276 constants_definition: /* empty */
277                             | "constants" constants_def_list

278 constants_def_list: "identifier" ":" constant
279                     | constants_def_list "identifier" ":" constant

280 channel_bindings: "channel_bindings" bindings

281 field_bindings: /* empty */
282                     | "field_bindings" bindings

283 register_bindings: /* empty */
```

```
284                     | "register_bindings" bindings

285 sensor_bindings: /* empty */
286                     | "sensor_bindings" bindings

287 bindings: binding
288             | bindings binding

289 binding: distinguished_name ":" qualified_name
290             | distinguished_name ":" distinguished_name

291 qualified_name: "identifier" "::" distinguished_name

292 qualified_channel: qualified_name trigger_qualifier
depth_qualifier timespan_qualifier

293 qualified_channel_list: qualified_channel
294                         | qualified_channel_list qualified_channel

295 sensor_adapter_template: "sensor_adapter_template"
distinguished_name "{" sensor_adapter_template_description "}"
296                                 | error distinguished_name "{"
sensor_adapter_template_description "}"
297                                 | "sensor_adapter_template" error "{"
sensor_adapter_template_description "}"
298                                 | "sensor_adapter_template"
distinguished_name error sensor_adapter_template_description "}"
299                                 | "sensor_adapter_template"
distinguished_name "{" error "}"
300                                 | "sensor_adapter_template" error "}"

301 sensor_adapter_template_description: sensor_message_types
"protocol" distinguished_name sensor_adapter_template_props
302 sensor_message_types: /* empty */

303                         | "message_types" sensor_message_type_list
304                         | error sensor_message_type_list

305 sensor_message_type_list: sensor_message_type_pair
306                         | sensor_message_type_list
sensor_message_type_pair

307 sensor_message_type_pair: "data_message" distinguished_name
308                                 | "alert_message" distinguished_name
309                                 | "comm_alarm_message" distinguished_name
310                                 | "equipment_alarm_message"
```

```
distinguished_name
311                                 | "data_message" error
312                                 | "alert_message" error
313                                 | "comm_alarm_message" error
314                                 | "equipment_alarm_message" error

315 sensor_adapter_template_props: /* empty */
316                                 | "properties" type_attributes
317                                 | "properties" error

318 action_list: action
319             | action_list action

320 action: primitive_action
321             | conditional_action

322 primitive_action: publish_action
323                     | control_action
324                     | function_call_expression
325                     | assign_action

326 assign_action: "set" register_ref_expr ":" expression

327 conditional_action: "if" expression "then" action_list "else"
action_list "fi"
328                     | "if" expression "else" action_list "fi"
329                     | "if" expression "then" action_list "fi"

330 publish_action: "publish" publish_map "reference_channels" "("
ref_expr_alias_list ")" "output_channels" "(" ref_expr_alias_list ")"
331                     | "publish" publish_map "," "reference_channels"
"(" ref_expr_alias_list ")" "output_channels" "(" ref_expr_alias_list
")"
332                     | "publish" publish_map "reference_channels" "("
ref_expr_alias_list ")" "," "output_channels" "(" ref_expr_alias_list
")"
333                     | "publish" publish_map "," "reference_channels"
"(" ref_expr_alias_list ")" "," "output_channels" "("
ref_expr_alias_list ")"

334 publish_map: star_named_expression_pair
335                     | named_expression_pair_list

336 star_named_expression_pair: "*" ":" channel_ref_expr

337 named_expression_pair_list: named_expression_pair
```

```
338                                 | named_expression_pair_list
named_expression_pair

339 named_expression_pair: distinguished_name ":" expression
340                               | distinguished_name expression

341 control_action: "control" named_expression_pair_list "sensors"
"(" ref_expr_alias_list ")"
342                       | "control" named_expression_pair_list ","
"sensors" "(" ref_expr_alias_list ")"

KEYWORDS:
action_template
alert_channels
alert_message
boolean
CAP1_1
channel
channel_bindings
channel_reference
channels
comm_alarm_channels
comm_alarm_message
condition_template
configuration_server
control
constant_reference
constants
data_channels
data_message
depth
double
else
enum
equipment_alarm_channels
equipment_alarm_message
false
fi
field_bindings
filter
host
ICD100
if
input_channels
integer
jms_message_server
```

local_message_server
long
message_protocol
message_servers
message_structure_map
message_type
message_types
message_value_map
output_channels
persistent
policies
policy_instance
policy_template
properties
process_manager
process_managers
protocol
publish
reference_channels
register_reference
register_bindings
registers
SDO
sensor_reference
sensor_adapter_instance
sensor_adapter_template
sensor_adapters
sensor_bindings
sensors
set
SML
soap_message_server
soap_server_port
string
tcp_message_server
template
then
timespan
trigger
true
type
type_structure_map
type_value_map
viper_server
viper_servers

```
OPERATORS: // Note that all of these are binary operators except "~"
and "!" which are unary.
relational operators:
==          equal
!=          not equal
>           greater than
>=          greater than or equal
<           less than
<=          less than or equal

logical operators:
!           NOT (unary)
||          OR
&&          AND
!&&         NAND            // !(a && b)
!||         NOR                 // !(a || b)
^||         XOR                 // (a || b) && !(a && b)

bitwise operators
|           bitwise OR
&           bitwise AND
^           bitwise XOR
<<          bitwise shift left
>>          bitwise shift right
~           one`s complement (unary)

other operators:
=           assignment          // Assigns the value of b to a
-<          isPrefix            // true if b is a prefix of a
>-          isSuffix            // true if b is a suffix of a

mathematical operators:
+           addition            // + yields a concatenated string if
either operand is a string
-           subtraction
*           multiplication
/           division
-           negative (unary)
**          power operator

DELIMITERS:
()
{}
[]
:
,
```

```
.       // Used to delimit names in a distinguished name
::      // Scope delimeter, e.g., MsgServerName::ChannelName
->      // Field delimeter, e.g., ChannelName->FieldName

FUNCTIONS:      // Note that function names are case sensitive.

LOWER_CASE(str)
// Converts all letters in str to lower case letters.

UPPER_CASE(str)
// Converts all letters in str to upper case letters.

CONTAINS_ANY_STRING(str1, str2, str3, ...)
// Returns true if str1 contains any of the subsequent strings.

CONTAINS_ALL_STRINGS(str1, str2, str3, ...)
// Returns true if str1 contains all of the subsequent strings.

SQRT(num)
// NEED TO IMPLEMENT!!!
// Returns sqrt of num.

CURRENT_TIME()
// Returns the current time (represented as the number of milliseconds
since 1/1/1970 00:00:00 GMT).

TIMESTAMP_DIFF_SECS(timestamp1, timestamp2)
// Both parameters are strings representing a timestamp (number of
milliseconds since 1/1/1970 00:00:00 GMT).
// Returns timestamp1 - timestamp2 (result is in seconds).

FORMAT_TIMESTAMP(timestamp, format = "%m/%d/%y %H:%M:%S")
// Converts 'timestamp' to a string in the format indicated in the
string 'format'.

COMPOSITE_VIDEO(videoFrame1, videoFrame2, ...)
// Combines all provided video frames (base64 encoded strings) into a
composite image (also a base64 encoded string).

GEO_LOCATION_DISTANCE(geoLoc1, geoLoc2)
// Calculates the distance in miles between geoLocations geoLoc1 and
geoLoc2.
// Employs a function called calculateGeographicDistance() which is in
\ViaServiceDataObject\src\ViaSdoUtil.cpp

FIND_CLOSEST_CAMERA(targetGeoLoc, camera1GeoLoc, camera2GeoLoc)
```

```
// Using the provided geoLocations, calculates the camera which is
closest to the target.
// This should be made generic, e.g.,

FIND_CLOSEST_TO_LOCATION(targetGeoLoc, geoLoc1, geoLoc2, geoLoc3, ...).


FUNCTIONS THAT WE HAVE USED IN HOUSE BUT MAY NOT RELEASE TO THE
CUSTOMER:
// THESE SHOULD NOT BE RELEASED UNTIL FULLY IMPLEMENTED AND
VALIDATED!

CONFIRM_TARGET(geoLocation, targetId, cameraURL)
// INPUT:
//      "geoLocation", GeoLocation
//      "targetId, Integer
//      "cameraURL", String
// OUTPUT:
//      VIA_BSS_common_type(bssConfirmTargetMessageType,
"bssConfirmTargetMessageType")
//      VIA_add_prop(bssConfirmTargetMessageType, "TargetId", Integer)
//      VIA_add_prop(bssConfirmTargetMessageType, "image", String)
// Calls analyzeVideo() in ViaVideoAnalytics project
      // THIS CODE IS NOT THREAD SAFE -- NEEDS TO BE FIXED.

DETECT_TARGET(cameraName, videoFrame)
// Calls detectTarget() in ViaVideoAnalytics project.
// Output is DetectTargetType.

EXTRACT_FACES(videoFrame)
// Calls extractFaces() in ViaVideoAnalytics project.
// Output is faceRecognitionExtractedFacesMessageType.

MATCH_FACE(annotatedImage, referenceImage)
// MAY NOT BE FULLY IMPLEMENTED.
// The "annotatedImage" is an image from an IPCam in which a face has
been extracted:
// Returns true if the face in the annotated image matches the face in
the "referenceImage"
// Calls matchFace() in ViaVideoAnalytics project

FUNCTIONS NOT FULLY IMPLEMENTED (POSSIBLY NOT FOR VERSION 1.0):

GET_CAMERAS_IN_PROXIMITY(x, y, radius)
// THIS FUNCTION HAS NOT BEEN FULLY IMPLEMENTED.
// Comment from the code about what the function is supposed to do:
```

```
lookup all the configured cameras and see which one has a distance
lower than the radius
// When implement, change to (geoLoc, radius)

CHECK_ALERTS_IN_PROXIMITY()
// THIS FUNCTION HAS NOT BEEN FULLY IMPLEMENTED.
// The current code does the following:
// Get all SDOs from "alert_timestamp" register (i.e., this makes use
of temporal data -- based on depth of register).
// Get all SDOs from "alert_loc_x" register (i.e., this makes use of
temporal data -- based on depth of register).
// Get all SDOs from "alert_loc_y" register (i.e., this makes use of
temporal data -- based on depth of register).
// Get all SDOs from "alert_loc_z" register (i.e., this makes use of
temporal data).
// Makes the following function call to get the result: result =
bio_alert_system_1(timeVector, xVector, yVector, zVector)
// Note that bio_alert_system_1 is in an old version of
ViaConfigEvaluateVisitor.cpp

CHECK_ALERTS_SAME_LOCATION()
// THIS FUNCTION HAS NOT BEEN FULLY IMPLEMENTED.
// The current code does the same thing as CHECK_ALERTS_IN_PROXIMITY()
except that it calls bio_alert_system_2(timeVector, xVector, yVector,
zVector)

//########## Misc. Subtypes:
type AlternateLocationType {
        Alternate_location_x        double = 3.1
        Alternate_location_y        double = 4.2
        Alternate_location_z        double = 5.3
}

type BuildingLocationType {
        Location_building           string
        Location_floor              string
        Location_room               string
}

type GeoLocationType {
        longitude     double
        latitude      double
        altitude      double
}

type LocationType {
```

```
        Location_description            string
        Location_street_address         string
        Location_city                           string
        Location_state                          string
        Location_zip_code               string
        Location_country                string
        Location_building               string
        Location_floor                          string
        Location_room                           string
}

type ExampleType1 {
        Prop1 double[] {<attr1 = "value1" attr2 = "value2"> <attr3 =
"value3">} = {1.1 2.3}
        Prop2 double <attr1 = "value1" attr2 = "value2"> = 3.5
}
//########## END Misc. Subtypes.


//########## Misc. Message Types:
type AlertMessageType {
        Timestamp                       string
        alert                           string
        AlertLevel                      string
        referenceChannels  string
        buildingLocation    BuildingLocationType
        alternateLocation   AlternateLocationType
        geoLocation                     GeoLocationType
}

type ChemMessageType {
        Timestamp                       string
        NameOfDetectedGas       string
        Concentration                   double
        buildingLocation    BuildingLocationType
        alternateLocation   AlternateLocationType
        geoLocation             GeoLocationType
}

type CommAlarmMessageType {
        Timestamp               string
        Description             string
        alert                   string = "COMM ALERT!!!"
        AlertLevel              string
        SensorName              string
        SensorType              string
```

```
        buildingLocation    BuildingLocationType
        alternateLocation   AlternateLocationType
        geoLocation             GeoLocationType
}

type GasData {
        Timestamp               string
        NameofdetectedGas string
        Concentration           double
        buildingLocation    BuildingLocationType
        alternateLocation   AlternateLocationType
        geoLocation             GeoLocationType
}

type TimestampAndUUIDType {
        Timestamp               string
        UUID                    string
}

type VideoMessageType {
        Timestamp                   string
        Timezone                    string
        VideoFrameTimestamp         string
        VideoFrame                  string
        buildingLocation    BuildingLocationType
        alternateLocation   AlternateLocationType
        geoLocation                 GeoLocationType
}

type operatorTestOutput {
                result1     integer
                result2     double
                result3     string
                result4     string
                result5     integer
                result6     integer
                result7     integer
                result8     double
                result9     boolean
                result10    boolean
                result11    boolean
                result12    boolean
                result13    boolean
                result14    boolean
                esult15     boolean
                esult16     boolean
```

```
        esult17     boolean
        esult18     boolean
        esult19     boolean
        esult20     boolean
        esult21     boolean
        esult22     boolean
        esult23     boolean
        esult24     boolean
        esult25     boolean
        esult26     boolean
        esult27     boolean
        esult28     boolean
        esult29     boolean
        esult30     boolean
        esult31     boolean
        esult32     boolean
        esult33     boolean
        esult34     boolean
        esult35     boolean
        esult36     boolean
        esult37     integer
        esult38     double
        esult39     long
        esult40     long
        esult41     long
        esult42     long
        esult43     long
        esult44     long
        esult45     double
        esult46     double
        esult47     double

//########## END Misc. Message Types.


//########## Type maps:
type_structure_map ChemMessageType_to_GasData_map {
        imestamp                    :       Timestamp
        NameOfDetectedGas :     NameofdetectedGas
        Concentration               :       Concentration
        buildingLocation.Location_building              :
        buildingLocation.Location_building
        buildingLocation.Location_floor                         :
        buildingLocation.Location_floor
        buildingLocation.Location_room                          :
        buildingLocation.Location_room
```

```
        alternateLocation.Alternate_location_x          :
        alternateLocation[0]
        alternateLocation.Alternate_location_y          :
        alternateLocation[1]
        alternateLocation.Alternate_location_z          :
        alternateLocation[2]
        geoLocation.longitude                                   :
        geoLocation.longitude
        geoLocation.latitude                                    :
        geoLocation.latitude
        geoLocation.altitude                                    :
        geoLocation.altitude
}

type_structure_map junk_map {
        Timestamp[2] {1}            : Timestamp1
        Timestamp {2}               : Timestamp2
        Timestamp {3}               : Timestamp3
        Timestamp {16}              : Timestamp4
        Timestamp {1:2}             : Timestamp5
        Timestamp {1:3}             : Timestamp6
        Timestamp {2:16}    : Timestamp7
        Timestamp {7:9}             : Timestamp8
        Timestamp {15:16}  : Timestamp9
}
type_value_map junk_value_map {
        RegisterValues[0] == 3                  :       "BBIR (broad band
infrared)"
        RegisterValues[0] == 5                  :       3.0
        RegisterValues[2]{9:16} == 0 :     1
        RegisterValues[2]{9:16} == 1 :     10
        RegisterValues[2]{9:16} == 2 :     100
        RegisterValues[1]                       :       /
        RegisterValues[2]{9:16}
}
//########## END Type maps.


//########## Message Servers:
local_message_server LocalMsgServer {
        channel LocalCommAlarmChannel message_type CommAlarmMessageType
}

jms_message_server JmsMsgServer {

        host "vialinux2":1099
```

```
        channel Bio.Alert.USDA message_type operatorTestOutput
        channel Chem.Alert.FireDepartment message_type AlertMessageType
persistent
        channel Chem.Data.ITrans.Gas.CarbonMonoxide
                message_type GasData
                message_protocol SML
                message_structure_map ChemMessageType_to_GasData_map
                persistent
        channel Chem.Data.ITrans.Gas.Flammables
                message_type GasData
                message_protocol SML
                message_structure_map ChemMessageType_to_GasData_map
                persistent
        channel Chem.Video.Frames.EmissionsLabEast message_type
VideoMessageType
        channel Chem.Video.Frames.NightVision message_type
VideoMessageType

}

soap_message_server SoapMsgServer {
        host "vialinux2":4444

        channel SoapChannel1 message_type AlertMessageType persistent

}

tcp_message_server TcpMsgServer {
host "vialinux2":1099
channel SoapChannel1 message_type AlertMessageType persistent
}
//########## END Message Servers.


//########## Sensor adapters:
//### Inova
sensor_adapter_template InovaSensorTemplate {

        message_types
                comm_alarm_message CommAlarmMessageType

        protocol Inova

        properties
                Enabled                                 integer
                IPAddress                       string
```

```
            port                                integer
            PollingRate                         double
            userName                            string
            password                            string
            defaultMessage                      string
            commandDuration                     integer
            defaultColor                        string
            commandMessage                      string
            currentMessage                      integer
            commandColor                        string
            commandRefreshTime                  string
            commandFontSize                     string
            commandDisplayMethod        string
            useDefaultMessage           integer
}

sensor_adapter_instance Inova {

        template InovaSensorTemplate

        channels
                comm_alarm_channels
                        LocalMsgServer::LocalCommAlarmChannel
        properties
                Enabled = 1
                IPAddress = "64.210.18.104"
                port = 23
                PollingRate = 1.0
                userName = "admin"
                password = "1n0v@"
                defaultMessage = "SPM - ViaLogy, California, USA"
                commandDuration = 60
                defaultColor = "green"
                commandMessage = ""
                currentMessage = 100
                commandColor = "red"
                commandRefreshTime = "5"
                commandFontSize = "14"
                commandDisplayMethod = "ribbon_left"
                useDefaultMessage = 1
}
//### END Inova.


//### ITrans:
sensor_adapter_template ItransSensorTemplate {
```

```
        message_types
                data_message ChemMessageType
                comm_alarm_message CommAlarmMessageType

        protocol ITrans

        properties
                Enabled                         integer
                SimulatorMode                   integer
                IPAddress                       string
                port                            integer
                PollingRate                     double
                geoLocation                     GeoLocationType
                NameOfDetectedGas       string
}

sensor_adapter_template ItransChannelSensorTemplate {

        message_types
                data_message ChemMessageType
                comm_alarm_message CommAlarmMessageType

        protocol ITransChannel

        properties
                Enabled         integer
                SPModel         string
                geoLocation             GeoLocationType
}

sensor_adapter_instance ItransSensor {

        template ItransSensorTemplate

        channels
                comm_alarm_channels
                        LocalMsgServer::LocalCommAlarmChannel
        properties
                Enabled = 1
                SimulatorMode = 0
                IPAddress = "64.210.18.21"
                port = 2101
                PollingRate = 1.0
                geoLocation.longitude = -118.159705
                geoLocation.latitude = 34.186906
                geoLocation.altitude = 0.0
```

```
            NameOfDetectedGas = "Nitrogen"
}

sensor_adapter_instance ItransChan0Sensor {

        template ItransChannelSensorTemplate

        channels
                data_channels
                        JmsMsgServer::Chem.Data.ITrans.Gas.CarbonMonoxide
                comm_alarm_channels
                        LocalMsgServer::LocalCommAlarmChannel
        filter NameOfDetectedGas == "CO_Carbon_Monoxide"

        properties
                Enabled       = 1
                SPModel       = "ItransSensor"
                geoLocation.longitude = -118.159705
                geoLocation.latitude = 34.186906
                geoLocation.altitude = 0.0
}

sensor_adapter_instance ItransChan1Sensor {

        template ItransChannelSensorTemplate

        channels
                data_channels
                        JmsMsgServer::Chem.Data.ITrans.Gas.Flammables
                comm_alarm_channels
                        LocalMsgServer::LocalCommAlarmChannel

        filter NameOfDetectedGas == "CH4_Methane"

        properties
                Enabled       = 1
                SPModel       = "ItransSensor"
                geoLocation.longitude = -118.159705
                geoLocation.latitude = 34.186906
                geoLocation.altitude = 0.0
}
//### END ITrans.

//### RemoteSensorController
sensor_adapter_template RemoteSensorControllerTemplate {
        protocol RemoteSensorController
```

```
        properties
                Enabled                                 integer
                PollingRate                     double
                PingCount                       integer
}

sensor_adapter_instance RemoteSensorController {

        template RemoteSensorControllerTemplate

        properties
                Enabled = 1
                PollingRate = 1.0
                PingCount = 100
}
//### END RemoteSensorController.
//########## END sensor adapters.


//########## Policies:
//### ITrans Policies:
condition_template TOXIC_FUME_cond_template {
        channel_reference(co_channel->Concentration) >
constant_reference(Toxic_fume_threshold_co) ||
        channel_reference(ch4_channel->Concentration) >
constant_reference(Toxic_fume_threshold_ch4)
}

action_template TOXIC_FUME_action_template {
        if
                !register_reference(ALERT_POSTED_REGISTER)
        then
                set register_reference(ALERT_POSTED_REGISTER) : true
                set register_reference(ALERT_POSTED_TIME_REG) :
register_reference(CurrentTime)
                publish
                        // Add example of stand-alone function call.
                        // Add example of function call in an expression.
                        alert : constant_reference(Toxic_fume_alert_message)
                        AlertLevel : "RED"
                        referenceChannels :
"Chem.Data.ITrans.Gas.Flammables~Chem.Video.Frames.EmissionsLabEast~Che
m.Video.Frames.EmissionsLabWest"
                        buildingLocation : channel_reference(co_channel-
>buildingLocation)
```

```
                    alternateLocation : channel_reference(co_channel-
>alternateLocation)
                    geoLocation : channel_reference(co_channel-
>geoLocation)
                    reference_channels(co_channel->Concentration,
ch4_channel->Concentration, camera_channel1, camera_channel2)
                    output_channels(alertChannel1)
            control
                    commandMessage :
constant_reference(Toxic_fume_alert_message)
                    sensors(Inova)
        else // An alert has already been sent, but see if it has been >
n sec
            if
                    TIMESTAMP_GAP_SEC(register_reference(CurrentTime),
register_reference(ALERT_POSTED_TIME_REG)) >
constant_reference(interval_for_alert_reissue)
            then
                    set register_reference(ALERT_POSTED_TIME_REG) :
register_reference(CurrentTime)
                    publish
                            alert : "Alert still outstanding: " +
constant_reference(Toxic_fume_alert_message)
                            AlertLevel : "RED"
                            referenceChannels :
"Chem.Data.ITrans.Gas.Flammables~Chem.Video.Frames.EmissionsLabEast~Che
m.Video.Frames.EmissionsLabWest"
                            buildingLocation :
channel_reference(co_channel->buildingLocation)
                            alternateLocation :
channel_reference(co_channel->alternateLocation)
                            geoLocation : channel_reference(co_channel-
>geoLocation)
                            reference_channels(co_channel->Concentration,
ch4_channel->Concentration, camera_channel1, camera_channel2)
                            output_channels(alertChannel1)
                    control
                            commandMessage : "Alert still outstanding: " +
constant_reference(Toxic_fume_alert_message)
                            sensors(Inova)
            fi
        fi
}

policy_template TOXIC_FUME_template {
        condition_template TOXIC_FUME_cond_template
```

```
        action_template TOXIC_FUME_action_template
}

policy_instance TOXIC_FUME_instance {

        template TOXIC_FUME_template

        input_channels
                JmsMsgServer::Chem.Data.ITrans.Gas.CarbonMonoxide
                JmsMsgServer::Chem.Data.ITrans.Gas.Flammables trigger

        constants
                Toxic_fume_threshold_co : 5.0
                Toxic_fume_threshold_ch4 : 5.0
                interval_for_alert_reissue : 30
                Toxic_fume_alert_message : "ALERT: Explosive gas leak
detected."

        channel_bindings
                co_channel :
JmsMsgServer::Chem.Data.ITrans.Gas.CarbonMonoxide
                ch4_channel : JmsMsgServer::Chem.Data.ITrans.Gas.Flammables
                camera_channel1 :
JmsMsgServer::Chem.Video.Frames.EmissionsLabEast
                camera_channel2 :
JmsMsgServer::Chem.Video.Frames.NightVision
                alertChannel1 : JmsMsgServer::Chem.Alert.FireDepartment

        register_bindings
                ALERT_POSTED_REGISTER :
Example_ViperServer1::TOXIC_FUME_ALERT_POSTED_REGISTER
                ALERT_POSTED_TIME_REG :
Example_ViperServer1::ALERT_POSTED_TIME_REG
                CurrentTime : Example_ViperServer1::CurrentTime

        sensor_bindings
                Inova : Example_ViperServer1::Inova
}

condition_template NO_TOXIC_FUME_cond_template {
        channel_reference(co_channel->Concentration) <=
constant_reference(No_Toxic_fume_threshold_co) &&
        channel_reference(ch4_channel->Concentration) <=
constant_reference(No_Toxic_fume_threshold_ch4) &&
        register_reference(ALERT_POSTED_REGISTER)
}
```

```
action_template NO_TOXIC_FUME_action_template {
        set register_reference(ALERT_POSTED_REGISTER) : false
        publish
                alert : constant_reference(No_Toxic_fume_alert_message)
                AlertLevel : "GREEN"
                referenceChannels :
"Chem.Data.ITrans.Gas.Flammables~Chem.Video.Frames.EmissionsLabEast~Che
m.Video.Frames.EmissionsLabWest"
                reference_channels(co_channel->Concentration, ch4_channel-
>Concentration, camera_channel1, camera_channel2)
                output_channels(alertChannel1)
        control
                commandMessage :
constant_reference(No_Toxic_fume_alert_message)
                sensors(Inova)
}

policy_template NO_TOXIC_FUME_template {
        condition_template NO_TOXIC_FUME_cond_template
        action_template NO_TOXIC_FUME_action_template
}

policy_instance NO_TOXIC_FUME_instance {

        template NO_TOXIC_FUME_template

        input_channels
                JmsMsgServer::Chem.Data.ITrans.Gas.Flammables trigger depth
10 timespan 1000
        constants
                No_Toxic_fume_threshold_co : 5.0
                No_Toxic_fume_threshold_ch4 : 5.0
                No_Toxic_fume_alert_message : "ALERT: Explosive gas levels
returned to safe levels."

        channel_bindings
                co_channel :
JmsMsgServer::Chem.Data.ITrans.Gas.CarbonMonoxide
                ch4_channel : JmsMsgServer::Chem.Data.ITrans.Gas.Flammables
                camera_channel1 :
JmsMsgServer::Chem.Video.Frames.EmissionsLabEast
                camera_channel2 :
JmsMsgServer::Chem.Video.Frames.NightVision
                alertChannel1 : JmsMsgServer::Chem.Alert.FireDepartment
```

```
        register_bindings
                ALERT_POSTED_REGISTER :
Example_ViperServer1::TOXIC_FUME_ALERT_POSTED_REGISTER

        sensor_bindings
                Inova : Example_ViperServer1::Inova
}
//### END ITrans Policies.

//### Policy to test operators:
condition_template operatorTest_cond_template {
        true
}

action_template operatorTest_action_template {
        publish
                result1 : 2 + 3
                result2 : 2 + 3.0
                result3 : "hello " + "world"
                result4 : "hello " + 5.1
                result5 : 5 - 7
                result6 : 5 * 2
                result7 : 5 / 2
                result8 : 5 / 2.0
                result9 : 5 == 2.0
                result10 : 5 == 5
                result11 : 5 != 2.0
                result12 : 5 != 5
                result13 : 5 > 2.0
                result14 : 5 > 5
                result15 : 5 >= 2.0
                result16 : 5 >= 5
                result17 : 5 < 2.0
                result18 : 5 < 5
                result19 : 5 <= 2.0
                result20 : 5 <= 5
                result21 : !true
                result22 : !false
                result23 : true || false
                result24 : false || false
                result25 : true && false
                result26 : false && false
                result27 : true !&& true
                result28 : true !&& false
                result29 : true !|| false
                result30 : false !|| false
```

```
                result31 : true ^|| false
                result32 : true ^|| true
                result33 : "he" -< "hello"
                result34 : "hep" -< "hello"
                result35 : "he" >- "hello"
                result36 : "llo" >- "hello"
                result37 : -5
                result38 : 5**2.0
                result39 : 13 & 10
                result40 : 13 | 10
                result41 : 13 ^ 10
                result42 : 13 << 2
                result43 : 13 >> 2
                result44 : ~13
                result45 : channel_reference(input_channel-
>alternateLocation[0])
                result46 : channel_reference(input_channel-
>alternateLocation[1])
                result47 : channel_reference(input_channel-
>alternateLocation[2])
                reference_channels(input_channel)
                output_channels(output_channel)
}

policy_template operatorTest_template {
        condition_template operatorTest_cond_template
        action_template operatorTest_action_template
}

policy_instance operatorTest {
        template operatorTest_template

        input_channels
                JmsMsgServer::Chem.Data.ITrans.Gas.Flammables trigger

        channel_bindings
                input_channel :
JmsMsgServer::Chem.Data.ITrans.Gas.Flammables
                output_channel : JmsMsgServer::Bio.Alert.USDA

}
//### END Policy to test operators.
//########## END Policies.


//########## Viper Server:
```

```
viper_server Example_ViperServer1 {

        soap_server_port 4000

        remote_viper_servers
                ViperServer2 : host "vialinux5" soap_server_port 4040
                ViperServer3 : host "vialinux5" soap_server_port 4050

        message_servers
                LocalMsgServer
                JmsMsgServer
                SoapMsgServer
                TcpMsgServer

        registers
                TOXIC_FUME_ALERT_POSTED_REGISTER : boolean = false
                ALERT_POSTED_TIME_REG : string = "0000000000"
                MOST_RECENT_MSG_REG : TimestampAndUUIDType (100)

        sensor_adapters
//@@@ Multi-headed sensor adapter needs to be before single-headed
sensor adapters.
                Inova
                ItransSensor
                ItransChan0Sensor
                ItransChan1Sensor

        policies
                TOXIC_FUME_instance
                NO_TOXIC_FUME_instance

}
//########## END Viper Server.


//########## Process Manager:
process_manager ProcessMgr1 {

        host "vialinux2"
        soap_server_port 4110

        viper_servers
                Example_ViperServer1 : soap_server_port 4000
}
//########## END Process Manager.
```

```
//########## Config Server:
configuration_server TheConfigServer {

        soap_server_port 4444

        process_managers
                ProcessMgr1
}
//########## END Config Server.


//##### Types
#########################################################################
//##### Location types
type AlternateLocationType {
        Alternate_location_x        double
        Alternate_location_y        double
        Alternate_location_z        double
}

type BuildingLocationType {
        Location_building           string
        Location_floor              string
        Location_room               string
}

type GeoLocationType {
        longitude     double
        latitude      double
        altitude      double
}

type LocationType {
        Location_description            string
        Location_street_address         string
        Location_city                           string
        Location_state                          string
        Location_zip_code               string
        Location_country                string
        Location_building               string
        Location_floor                          string
        Location_room                           string
}
//##### END Location types
```

```
//##### Misc.
type AlertMessage {
        Timestamp               string
        Timezone                string
        UUID                    string
        alert                   string
        AlertLevel              string
        referenceChannels       string
        buildingLocation BuildingLocationType
        alternateLocation AlternateLocationType
        geoLocation GeoLocationType
}

type CommAlarmMessage {
        Timestamp               string
        Timezone                string
        UUID                    string
        Description             string
        AlertLevel              string
        SensorName              string
        SensorType              string
        alert                   string // I added the field 'alert' to
CommAlarmMessage so that the GUI can display
                                        // these messages when
they are published to SPM.Alert.Sensor.Comm.Alarms
                                        // (without VJ/Tong
having to do any recoding).
        buildingLocation BuildingLocationType
        alternateLocation AlternateLocationType
        geoLocation     GeoLocationType
}
//##### END Misc.


//##### AlienRFID
type AlienRFIDMessage {
        Timestamp                       string
        Timezone                        string
        UUID                            string
        TagID                           string // hexadecimal digits
        TagDescription                  string
        ReadCount                       string // The number of times
the tag has been read in the current session
        DiscoveryTime                   string // Time the tag was
first seen. 0 if it was given from the coupling list.
        LastObservedTime                string // Time the tag was last
```

```
seen.
        WindowStartTime                 string // Time the current
session started.
        CouplingType                    string // Not coupled, or
Owner, or Owned.
        CoupledWith                     string // Its owned object,
or its owner or "N/A".
        AntennaID                       string // The antenna ID that
the tag was last read from
        ProtocolID                      string
        alert                           string
}

// Message used for an alarm indicating that an "owned" tag just left:
type AlienRFID_OwnedTagGoneMessage {
        Timestamp                       string
        Timezone                        string
        UUID                            string
        OwnerTagID                      string
        OwnerTagDescription             string
        OwnedTagID                      string
        OwnedTagDescription             string
        OwnedJustLeft                   boolean // Always true!
        OwnerIsPresent                  boolean // true or false.
}
//##### END Alien


//##### Barionet
type BarionetRelayControlMessage {
        Relay string
        Value string
}
//##### END Barionet


//##### GeneXpert
type GeneXpertSampleInfo {
        Sample_ID                       string
        Flow_Rate                       double
        Collection_time                 string
        Collection_time_timezone        string
        Location                        LocationType
        geoLocation                     GeoLocationType
}
```

```
type GeneXpertLabInfo {
        Lab_Name                        string
        Location                        LocationType
        geoLocation                     GeoLocationType
}

type GeneXpertAssayInfo {
        Application_Name                string
        File_export_time                string
        File_export_time_timezone       string
        Report_User_Name                string
        Test_Type                       string
        Assay                           string
        Assay_Version                   string
        Assay_Type                      string
        Operator                        string
        Notes                           string
        Start_time                      string
        Start_time_timezone             string
        End_time                        string
        End_time_timezone               string
        Reagent_Lot_ID                  string
        Expiration_Date                 string
        Cartridge_SN                    string
        Module_Name                     string
        Module_SN                       string
        Instrument_SN                   string
        SW_Version                      string
}

type GeneXpertIndividualAssayResult {
        DataGroupInUse                  integer
        Ct                              double
        EndPt                           double
        Analyte_Result                  string
        Probe_Check_Result              string
        Curve_Fit                       string
}

type GeneXpertAssayResults {
        Status                          string
        Error_Status                    string
        Test_Result                     string
        Bg                              GeneXpertIndividualAssayResult
        FAM                             GeneXpertIndividualAssayResult
        H5Sub                           GeneXpertIndividualAssayResult
```

```
        H7Sub                                   GeneXpertIndividualAssayResult
        IC                                      GeneXpertIndividualAssayResult
        LIZ                                     GeneXpertIndividualAssayResult
        pMat01                                  GeneXpertIndividualAssayResult
        pMat02                                  GeneXpertIndividualAssayResult
        pX01                                    GeneXpertIndividualAssayResult
        pX02                                    GeneXpertIndividualAssayResult
        SPC                                     GeneXpertIndividualAssayResult
}

type GeneXpertMessage {
        Timestamp                               string
        Timezone                                string
        UUID                                    string
        Headline                                string
        Sample_info                             GeneXpertSampleInfo
        Lab_info                                GeneXpertLabInfo
        Assay_info                              GeneXpertAssayInfo
        Assay_results                           GeneXpertAssayResults
}
//##### END GeneXpert


//##### ICD
type PlatformStatusType {
        DeviceState string = "OK"
        CommunicationState string = "OK"
        UpdateTime string <Zone="GMT"> = "8:00 PM"
}

type PlatformIdentificationType {
        DeviceName string = "SPM-001"
        DeviceCategory string = "C2 Node"
        DeviceType string = "Sensor Policy Engine"
}

type DeviceIdentificationType {
        DeviceName string = "DEV-001"
        DeviceCategory string = "Sensor"
        DeviceType string = "Generic sensor"
}

type GeodeticLocationType {
        Latitude string <Units="Decimal Degrees"> = ""
        Longitude string <Units="Decimal Degrees"> = ""
        Altitude string <Units="Meters" 'Reference'="MSL"> = ""
```

```
}

type LocationTypeType {
        GeodeticLocation GeodeticLocationType <Datum="WGS84">
}

type LocationType1 {
        LocationType LocationTypeType
        UpdateTime string <Zone="GMT"> = "Teatime"
}

type PlatformStatusReportBase {
        PlatformIdentification PlatformIdentificationType
        Status PlatformStatusType
        Location LocationType1
}

type DeviceStatusType {
        DeviceState string = "OK"
        CommunicationState string = "OK"
        UpdateTime string <Zone="GMT"> = "Lunchtime"
}

type DetailsType {
        Range string <Units="Meters"> = "1000"
        ElevationAngle string <Units="Degrees"> = "45"
        Azimuth string <Units="Degrees"> = "120"
        FieldOfView string <Units="Degrees"> = "50"
}

type CameraStatusType {
        DeviceState string = "OK"
        CommunicationState string = "OK"
        UpdateTime string <Zone="GMT"> = "Lunchtime"
}

type CameraStatusReport {
        DeviceIdentification DeviceIdentificationType
        Status CameraStatusType
        Location LocationType1
        Details DetailsType
}

type DeviceStatusReport {
        DeviceIdentification DeviceIdentificationType
        Status DeviceStatusType
```

```
        Location LocationType1
}

type DetectionType {
        ID string = "SPM001"
        DetectionEvent string = "Biological Agent"
        Details string = "Evacuation order issued"
        UpdateTime string <Zone="GMT"> = "Midnight"
}

type EventAssessmentType {
        Assessment string = "Positive"
        Annotation string = "Evacuation order issued"
}

type DeviceDetectionRecordType {
        DeviceIdentification DeviceIdentificationType
        Detection DetectionType <Assessed="true">
        EventAssessment EventAssessmentType
}

type DeviceDetectionReport {
        DeviceDetectionRecord DeviceDetectionRecordType
}
//##### END ICD

//##### ITrans
type ITransMessage {
        Timestamp                       string
        Timezone                        string
        UUID                            string
        InstrumentType                  string
        NameofdetectedGas               string // Get rid of this one once GUI is
        fixed
        NameOfDetectedGas               string
        Concentration                   double
        Alarm                           string
        EquipmentFault                  string
        buildingLocation    BuildingLocationType
        alternateLocation   AlternateLocationType
        geoLocation                     GeoLocationType
}

type_structure_map ModbusTCP_to_ITrans_map {
        RegisterValues[0]{9:15} : InstrumentType
        RegisterValues[1] : Concentration
```

```
        RegisterValues[2]{1:8} : NameofdetectedGas // Get rid of this
one once GUI is fixed
        RegisterValues[2]{1:8} : NameofDetectedGas
        RegisterValues[5]{1:2} : Alarm
        RegisterValues[5]{5:6} : EquipmentFault
}

type_value_map ITrans_value_map {
        RegisterValues[0]{9:15} == 768 : "BBIR (broad band
infrared)"
        RegisterValues[0]{9:15} == 1024 : "TOX (toxic)"
        RegisterValues[0]{9:15} == 1280 : "OXY (oxygen)"
        RegisterValues[0]{9:15} == 1536 : "AAW (toxic)"
        RegisterValues[0]{9:15} == 1792 : "CAT (catalytic)"
        RegisterValues[1] : /
        RegisterValues[2]{9:16}
        RegisterValues[2]{1:8} == 1 : "CO_Carbon_Monoxide"
        RegisterValues[2]{1:8} == 2 : "H2S_Hydrogen_Sulfide"
        RegisterValues[2]{1:8} == 3 : "SO2_Sulfur_Dioxide"
        RegisterValues[2]{1:8} == 4 : "NO2_Nitrogen_Dioxide"
        RegisterValues[2]{1:8} == 5 : "Cl2_Chlorine"
        RegisterValues[2]{1:8} == 6 : "ClO2_Chlorine_Dioxide"
        RegisterValues[2]{1:8} == 7 : "HCN_Hydrogen_Cyanide"
        RegisterValues[2]{1:8} == 8 : "PH3_Phosphine"
        RegisterValues[2]{1:8} == 9 : "H2_Hydrogen"
        RegisterValues[2]{1:8} == 12 : "NO_Nitric_Oxide"
        RegisterValues[2]{1:8} == 13 : "NH3_Ammonia"
        RegisterValues[2]{1:8} == 14 : "HCl_Hydrogen_Chloride"
        RegisterValues[2]{1:8} == 20 : "O2_Oxygen"
        RegisterValues[2]{1:8} == 21 : "CH4_Methane"
        RegisterValues[2]{1:8} == 22 :
        "LEL_Lower_Explosive_Limit_(Combustible_Gases)"
        RegisterValues[2]{1:8} == 0 : "Unknown"
        RegisterValues[2]{9:16} == 0 : 1
        RegisterValues[2]{9:16} == 1 : 10
        RegisterValues[2]{9:16} == 2 : 100
        RegisterValues[2]{9:16} == 3 : 1000
        RegisterValues[2]{9:16} == 4 : 10000
        RegisterValues[2]{9:16} == 5 : 100000
        RegisterValues[5]{1:2} == 0 : "Clear"
        RegisterValues[5]{1:2} == 1 : "Low"
        RegisterValues[5]{1:2} == 2 : "High"
        RegisterValues[5]{5:6} == 0 : "Clear"
        RegisterValues[5]{5:6} == 1 : "5 volt fault"
        RegisterValues[5]{5:6} == 2 : "power fault"
}
```

```
//##### END ITrans


//##### LCD
type LCDMessage {
        systemId integer
        gBar integer
        hBar integer
        gAgentType integer
        hAgentType integer
        sysStatus1 integer
        sysStatus2 integer
        gDose double
        hDose double
        gHighestAgentType integer
        hHighestAgentType integer
        seconds integer
        minutes integer
        hours integer
        dayOfMonth integer
        month integer
        year integer
        vcc integer
        inletFanCurrent integer
        recircFanCurrent integer
        waterIntake integer
        pressure integer
        temperature integer
        batteryVoltage integer
        positiveHT integer
        negativeHT integer
        gNoise integer
        hNoise integer
        positiveDacOffset integer
        positiveDacScalar integer
        negativeDacOffset integer
        negativeDacScalar integer
        dspParamCR integer
        coronaControlModes integer
        dspParamPC integer
        dspParamDP integer
        dspParamDN integer
        dspParamGP integer
        dspParamGN integer
        coronaNumber integer
        coronaFiringCount integer
```

```
        operatingCycleCount integer
        lowVxMode integer
        cycleTime integer
        gMobilityCal integer
        hMobilityCal integer
        negativeSpectrumData integer []
        positiveSpectrumData integer []
        Timestamp string
        textBlock string
}
//##### END LCD


//##### Lenel
type LenelAccessGrantedMessage {
        Timestamp                       string
        Timezone                        string
        UUID                            string
        LnlPhoto                        string
        LnlAlarmType                    string
        LnlAlarmSubType                 string
        LnlBanner                       string
        LnlDescription                  string
        LnlFirstName                    string
        LnlLastName                     string
        LnlNumberOfPersons              string
        LnlPanelId                      string
        LnlDeviceId                     string
        Location                        LocationType
        geoLocation                     GeoLocationType
}

type LenelAlarmMessage {
        Timestamp                       string
        Timezone                        string
        UUID                            string
        LnlDeviceId                     string
        LnlAlarmType                    string
        LnlAlarmSubType                 string
        LnlDescription                  string
        LnlPanelId                      string
        Location                        LocationType
        geoLocation                     GeoLocationType
}
//##### END Lenel
```

```
//##### Midas (Honeywell)
type MidasMessage {
        Timestamp                       string
        Timezone                        string
        UUID                            string
        NameOfDetectedGas               string
        Temperature_Celsius             integer
        Flowrate_cc_per_min             integer
        Concentration                   double
        buildingLocation                BuildingLocationType
        alternateLocation               AlternateLocationType
        geoLocation                     GeoLocationType
}
type OLD_MidasMessage {
        Timestamp                       string
        Timezone                        string
        UUID                            string
        NameofdetectedGas               string
        Temperature_Celsius             integer
        Flowrate_cc_per_min             integer
        Concentration                   double
        buildingLocation                BuildingLocationType
        alternateLocation               AlternateLocationType
        geoLocation                     GeoLocationType
}

type_structure_map MidasMessage_to_OLD_MidasMessage_map { // SA now
publishes MidasMessage; GUI is expecting OLD_MidasMessage
        Timestamp : Timestamp
        Timezone : Timezone
        UUID : UUID
        NameOfDetectedGas : NameofdetectedGas
        Temperature_Celsius : Temperature_Celsius
        Flowrate_cc_per_min : Flowrate_cc_per_min
        Concentration : Concentration
        buildingLocation.Location_building :
        buildingLocation.Location_building
        buildingLocation.Location_floor :
        buildingLocation.Location_floor
        buildingLocation.Location_room :
        buildingLocation.Location_room
        alternateLocation.Alternate_location_x :
        alternateLocation.Alternate_location_x
        alternateLocation.Alternate_location_y :
        alternateLocation.Alternate_location_y
```

```
        alternateLocation.Alternate_location_z :
        alternateLocation.Alternate_location_z
        geoLocation.longitude :
        geoLocation.longitude
        geoLocation.latitude :
        geoLocation.latitude
        geoLocation.altitude :
        geoLocation.altitude
}
//##### END Midas (Honeywell)


//##### Modbus
type ModbusRegisterMessage {
        Timestamp                       string
        RegisterValues                  Long []
        Location                        LocationType
        geoLocation                     GeoLocationType
}
//##### END Modbus


//##### Ortec
type OrtecMessage {
        Timestamp                       string
        Timezone                        string
        UUID                            string
        gammaStr                        string
        gammaCount                      double
        neutronStr                      string
        neutronCount                    double
        doseStr                         string
        doseAmount                      double
        numNuclidesFound                integer
        foundNuclides                   string []
        foundNuclidesList               string
        Location                        LocationType
        geoLocation                     GeoLocationType
}
//##### END Ortec


//##### QTL
type QTLMessage {
        Timestamp                       string
        Timezone                        string
```

```
        UUID                    string
        Signal                  double
        Threshold               double
        CalibrationFactor       double
        AssayType               string
        ManufacturingDate       string
        SerialNo                string
        SampleId                string
        Instrument_SN           string
        TestResult              string
        HazMat                  string
        Location                LocationType
        geoLocation             GeoLocationType
}
//##### END QTL


//##### SmartCycler
// For now, SmartCycler message type is identical to the GeneXpert
message type:
type SmartCyclerMessage {
        Timestamp               string
        Timezone                string
        UUID                    string
        Headline                string
        Sample_info             GeneXpertSampleInfo
        Lab_info                GeneXpertLabInfo
        Assay_info              GeneXpertAssayInfo
        Assay_results           GeneXpertAssayResults
}
//##### END SmartCycler


//##### SMC
type SMCMessage {
        Timestamp               string
        Timezone                string
        UUID                    string
        Concentration           double
        Temperature             double
        HighAlarmRelay          double
        LowAlarmRelay           double
        HighAlarmValue          double
        LowAlarmValue           double
        Location                LocationType
        geoLocation             GeoLocationType
```

```
}

type_structure_map ModbusTCP_to_SMC_map {
        RegisterValues[0] : Concentration
        RegisterValues[1] : Temperature
        RegisterValues[2] : HighAlarmRelay
        RegisterValues[3] : LowAlarmRelay
        RegisterValues[4] : HighAlarmValue
        RegisterValues[5] : LowAlarmValue
}

type_value_map SMC_value_map {
        RegisterValues[1] : / 100
}
//##### END SMC


//##### SNMP
type SnmpTrapVariableBindings {
        Description                     string
        Status                          string
}

type SnmpTrapMessage {
        Timestamp                       string
        Timezone                        string
        UUID                            string
        alertId                         integer
        Variable_bindings   SnmpTrapVariableBindings
        buildingLocation    BuildingLocationType
        alternateLocation   AlternateLocationType
        geoLocation                     GeoLocationType
}
//##### END SNMP


//##### ThermoElectron
type ThermoElectronMessage {
        Timestamp                       string
        Timezone                        string
        UUID                            string
        NameOfDetectedGas               string
        Concentration                   double
        buildingLocation                BuildingLocationType
        alternateLocation               AlternateLocationType
        geoLocation                     GeoLocationType
```

```
}

type OLD_ThermoElectronMessage {
        Timestamp                       string
        Timezone                        string
        UUID                            string
        NameofdetectedGas               string
        Concentration                   double
        buildingLocation                BuildingLocationType
        alternateLocation               AlternateLocationType
        geoLocation                     GeoLocationType
}

type_structure_map
ThermoElectronMessage_to_OLD_ThermoElectronMessage_map { // SA now
publishes MidasMessage; GUI is expecting OLD_MidasMessage
        Timestamp : Timestamp
        Timezone : Timezone
        UUID : UUID
        NameOfDetectedGas : NameofdetectedGas
        Concentration : Concentration
        buildingLocation.Location_building :
        buildingLocation.Location_building
        buildingLocation.Location_floor :
        buildingLocation.Location_floor
        buildingLocation.Location_room :
        buildingLocation.Location_room
        alternateLocation.Alternate_location_x :
        alternateLocation.Alternate_location_x
        alternateLocation.Alternate_location_y :
        alternateLocation.Alternate_location_y
        alternateLocation.Alternate_location_z :
        alternateLocation.Alternate_location_z
        geoLocation.longitude :
        geoLocation.longitude
        geoLocation.latitude :
        geoLocation.latitude
        geoLocation.altitude :
        geoLocation.altitude
}
//##### END ThermoElectron


//##### Video
type VideoMessage {
        Timestamp                       string
```

```
        Timezone                        string
        UUID                            string
        VideoFrameTimestamp             string
        VideoFrame                      string
        buildingLocation                BuildingLocationType
        alternateLocation               AlternateLocationType
        geoLocation                     GeoLocationType
}

type OLD_VideoMessage {
        Timestamp                       string
        Timezone                        string
        VideoData                       string
        buildingLocation                BuildingLocationType
        alternateLocation               AlternateLocationType
        geoLocation                     GeoLocationType
}

type_structure_map VideoMessage_to_OLD_VideoMessage_map { // SA now
publishes VideoMessage; GUI is expecting OLD_VideoMessage
        Timestamp :
        Timestamp
        Timezone :
        Timezone
        VideoFrame :
        VideoData
        buildingLocation.Location_building :
        buildingLocation.Location_building
        buildingLocation.Location_floor :
        buildingLocation.Location_floor
        buildingLocation.Location_room :
        buildingLocation.Location_room
        alternateLocation.Alternate_location_x :
        alternateLocation.Alternate_location_x
        alternateLocation.Alternate_location_y :
        alternateLocation.Alternate_location_y
        alternateLocation.Alternate_location_z :
        alternateLocation.Alternate_location_z
        geoLocation.longitude :
        geoLocation.longitude
        geoLocation.latitude :
        geoLocation.latitude
        geoLocation.altitude :
        geoLocation.altitude
}
//##### END Video
```

```
//#########################################################################
#########


//##### Message Servers
#########################################################
local_message_server LocalMsgServer {
        //### Policy alert channels:
        channel AlertsForJBC2S message_type AlertMessage
        channel AlertsForTASS message_type AlertMessage
        channel AlertsForVoxeo message_type AlertMessage
        channel QTLPolicyAlerts message_type AlertMessage

        //### Sensor adapter data channels:
        channel Bio.Data.Cepheid.GeneXpert message_type GeneXpertMessage
        channel Bio.Data.Cepheid.SmartCycler message_type
SmartCyclerMessage
        channel Bio.Data.QTL.Biosensor_2000 message_type QTLMessage
        channel Bio.Data.QTL.Biosensor_2000.2 message_type QTLMessage
        channel BMS.Data.Alien.RFID message_type AlienRFIDMessage
        channel BMS.Data.Barionet.RelayControl message_type
BarionetRelayControlMessage
        channel BMS.Data.Lenel.AccessGranted message_type
LenelAccessGrantedMessage
        channel BMS.Data.Lenel.BrokenWindow message_type
LenelAlarmMessage
        channel Chem.Data.Honeywell.Gas.Chlorine message_type
MidasMessage
        channel Chem.Data.ITrans.Gas.CarbonMonoxide
                message_type ITransMessage
                message_value_map ITrans_value_map
                message_structure_map ModbusTCP_to_ITrans_map
//      channel Chem.Data.ITrans.Gas.Flammables
//              message_type ITransMessage
//               message_value_map ITrans_value_map
//              message_structure_map ModbusTCP_to_ITrans_map
// ### NEED THE FOLLOWING INSTEAD OF THE ONE ABOVE WHEN USING
SIMULATOR
MODE:
        channel Chem.Data.ITrans.Gas.Flammables message_type
ITransMessage
        channel Chem.Data.LCD3.Gas message_type LCDMessage
        channel Chem.Data.SMC.Gas
                message_type SMCMessage
                message_value_map SMC_value_map
                message_structure_map ModbusTCP_to_SMC_map
```

```
        channel Chem.Data.ThermoElectron.Gas.Ammonia message_type
ThermoElectronMessage
        channel NetGuardian.SNMP.Data.Alarms message_type SnmpTrapMessage
        channel Rad.Data.Ortec.DetectiveEx message_type OrtecMessage

        //### Sensor adapter alert channels:
        channel BMS.Alert.Alien.RFID.OwnedTagGone message_type
AlienRFID_OwnedTagGoneMessage

        //### Sensor adapter comm. alarm channels:
        channel CommAlarms message_type CommAlarmMessage
        channel CommAlarmsForJBC2S message_type CommAlarmMessage
        channel CommAlarmsForTASS message_type CommAlarmMessage
}

jms_message_server JmsMsgServer {

        host "vialinux8":1099
//local_message_server JmsMsgServer {
        //### Policy alert channels:

        //### Sensor adapter data channels:
        channel Bio.Data.Cepheid.GeneXpert message_type GeneXpertMessage
persistent
        channel Bio.Data.Cepheid.SmartCycler message_type
SmartCyclerMessage persistent
        channel Bio.Data.QTL.Biosensor_2000 message_type QTLMessage
persistent
        channel BMS.Video.Frames.IPCam.1
                message_type OLD_VideoMessage // Temporary until GUI is
fixed
                message_protocol SML
                message_structure_map VideoMessage_to_OLD_VideoMessage_map
// Temporary until GUI is fixed
        channel Chem.Data.Honeywell.Gas.Chlorine
                message_type OLD_MidasMessage // Temporary until GUI is
fixed
                message_protocol SML
                message_structure_map MidasMessage_to_OLD_MidasMessage_map
// Temporary until GUI is fixed
                persistent
        channel Chem.Data.ITrans.Gas.CarbonMonoxide
                message_type ITransMessage
                message_protocol SML
                message_value_map ITrans_value_map
                message_structure_map ModbusTCP_to_ITrans_map
```

```
                persistent
//      channel Chem.Data.ITrans.Gas.Flammables
//              message_type ITransMessage
//              message_protocol SML
//              message_value_map ITrans_value_map
//              message_structure_map ModbusTCP_to_ITrans_map
//              persistent
// ### NEED THE FOLLOWING INSTEAD OF THE ONE ABOVE WHEN USING
SIMULATOR
MODE:
        channel Chem.Data.ITrans.Gas.Flammables message_type
ITransMessage persistent
        channel Chem.Data.ThermoElectron.Gas.Ammonia
                message_type OLD_ThermoElectronMessage // Temporary until
GUI is fixed
                message_protocol SML
                message_structure_map
ThermoElectronMessage_to_OLD_ThermoElectronMessage_map // Temporary
until GUI is fixed
                persistent

        //### Sensor adapter alert channels:
        channel Bio.Alert.FireDepartment message_type AlertMessage
persistent
        channel Bio.Alert.LabManager message_type AlertMessage persistent
        channel Bio.Alert.Technician message_type AlertMessage persistent
        channel BMS.Video.Frames.BufferPlayback
                message_type OLD_VideoMessage // Temporary until GUI is
fixed
                message_protocol SML
                message_structure_map VideoMessage_to_OLD_VideoMessage_map
// Temporary until GUI is fixed
        channel Chem.Alert.FireDepartment message_type AlertMessage
persistent

        //### Sensor adapter comm. alarm channels:
        channel SPM.Alert.Sensor.Comm.Alarms message_type
CommAlarmMessage persistent

}

soap_message_server SoapMsgServer {

        host "64.210.18.81" : 5000

        channel Bio.Data.QTL.Biosensor_2000 message_type QTLMessage
```

```
}

tcp_message_server TCPMsgServer {

        host "64.210.18.81":15000

        channel Chem.Data.SMC.Gas
                message_type SMCMessage
                message_protocol CAP1_1
                message_value_map SMC_value_map
                message_structure_map ModbusTCP_to_SMC_map
}
//#########################################################################
#########


//##### Sensor Adapters
############################################################
//##### Alien RFID:
sensor_adapter_template AlienRFID_Template {

        message_types
                data_message AlienRFIDMessage
                alert_message AlienRFID_OwnedTagGoneMessage
                comm_alarm_message CommAlarmMessage

protocol AlienRFID

properties
                Enabled                 integer
                SimulatorMode           integer
                IPAddress               string
                IPPort                  integer
                PollingPeriod           integer // in seconds.
                WindowTimeout           double // in seconds.
                DropTimeout             double // in seconds.
                LoginUserName           string
                LoginPassword           string
                ConfigCommands          string
                PollCommands            string
                CoupledTags             string
                TagDescriptions         string
}

sensor_adapter_instance AlienRFID {
```

```
template AlienRFID_Template

channels
        data_channels
                LocalMsgServer::BMS.Data.Alien.RFID
        alert_channels
                LocalMsgServer::BMS.Alert.Alien.RFID.OwnedTagGone
        comm_alarm_channels
                LocalMsgServer::CommAlarms
                LocalMsgServer::CommAlarmsForJBC2S
                LocalMsgServer::CommAlarmsForTASS
                JmsMsgServer::SPM.Alert.Sensor.Comm.Alarms

properties
        Enabled = 1
        SimulatorMode = 0
        IPAddress = "64.210.18.179"
        IPPort = 23 // default port
for Telnet.
        PollingPeriod = 1 // seconds
        WindowTimeout = 2.0 // seconds
        DropTimeout = 30.0 // seconds
        LoginUserName = "alien"
        LoginPassword = "password"
        ConfigCommands =
"AutoModeReset;AutoMode=ON;PersistTime=-1"
        PollCommands = "get TagList"
        //
        // Here: define owner-owned couples:
        // "a-b" means that owner 'a' owns equipment 'b'
        // To define multiple couplings:
        // "a-b;c-d;e-f" ...
        //
        CoupledTags             = "1-6;1-7;1-8;1-9;1-A;2-B;2-
C;2-D;2-E;2-F;3-10;3-11;3-12;3-13;3-14"
        // Tag Descriptions:
        // This strings are passed in messages and alerts. They
should describe
        // what the tag is attached to. They should be short, at
most a few words.
        // The format for this field is:
        // "tagid1:description1;tagid2:description2" ...
        //
        // Caveat: do not use apostrophes (') in the description
strings.
        // Workaround: you can use backquotes (`) instead (the key
```

```
above the tab key).
                TagDescriptions =
"1:Yogesh;2:John;3:Sandeep;6:Router;7:Luggage2;8:Luggage3;9:Luggage4;A:
Luggage5;B:Luggage1;C:Luggage2;D:Luggage3;E:Luggage4;F:Luggage5;10:Lugg
age1;11:Luggage2;12:Luggage3;13:Luggage4;14:Luggage5;"
}
//##### END Alien RFID.


//##### Arecont
sensor_adapter_template ArecontCam_Template {

        message_types
                data_message VideoMessage
                alert_message VideoMessage
                comm_alarm_message CommAlarmMessage

        protocol AreCont

        properties
        Enabled integer
        SimulatorMode integer
        PublishAlert integer
        IPAddress string
        HTTPService string
        cameraResolution string
        x0 integer
        y0 integer
        x1 integer
        y1 integer
        doublescan integer
        quality integer
        acquisitionBufferSize integer
        PollingRate double
        ResizeDivisor integer
        numFailsforCommAlarm integer
        BufferEnabled integer
        SecondsInBuffer integer
        SecondsToPublishVideoAlert integer
        geoLocation GeoLocationType
}

sensor_adapter_instance ArecontCam_01 {

        template ArecontCam_Template
```

```
        channels
                data_channels
                        JmsMsgServer::BMS.Video.Frames.IPCam.1
                alert_channels
                        JmsMsgServer::BMS.Video.Frames.BufferPlayback
                comm_alarm_channels
                        LocalMsgServer::CommAlarms
                        LocalMsgServer::CommAlarmsForJBC2S
                        LocalMsgServer::CommAlarmsForTASS
                        JmsMsgServer::SPM.Alert.Sensor.Comm.Alarms

        properties
                Enabled = 1
                SimulatorMode = 0
                PublishAlert = 0
                IPAddress = "64.210.18.17"
                HTTPService = "image"
                cameraResolution = "FULL"
                x0 = 0
                y0 = 0
// Resolution for 2 MPixel camera is 1600 x 1200.
// Resolution for 3 MPixel camera is 2048 x 1536.
// Resolution for 5 MPixel camera is 2592 x 1936.
                x1 = 2592
                y1 = 1936
                doublescan = 0
                quality = 20
                acquisitionBufferSize = 2000000
                PollingRate = 0.25
                ResizeDivisor = 2
                numFailsforCommAlarm = 3
                BufferEnabled = 1
                SecondsInBuffer = 10
                SecondsToPublishVideoAlert = 20
                geoLocation.longitude = -118.159705
                geoLocation.latitude = 34.186906
                geoLocation.altitude = 0.0
}
//##### END Arecont


//##### Barionet
sensor_adapter_template Barionet_Template {

        message_types
                comm_alarm_message CommAlarmMessage
```

```
        protocol Barionet

        properties
                Enabled integer
                SimulatorMode integer
                IPAddress string
                PollingRate double
                Relay string
                Value string
}

sensor_adapter_instance Barionet {

        template Barionet_Template

        channels
                comm_alarm_channels
                        LocalMsgServer::CommAlarms
                        LocalMsgServer::CommAlarmsForJBC2S
                        LocalMsgServer::CommAlarmsForTASS
                        JmsMsgServer::SPM.Alert.Sensor.Comm.Alarms
        properties
                Enabled = 1
                SimulatorMode = 0
                IPAddress = "64.210.18.51"
                PollingRate = 1.0
                Relay = "1"
                Value = "0"
}
//##### END Barionet


//##### GeneXpert:
sensor_adapter_template GeneXpert_Template {

        message_types
                data_message GeneXpertMessage
                comm_alarm_message CommAlarmMessage

        protocol GeneXpert

        properties
                Enabled                         integer
                SimulatorMode                   integer
                PollingRate                     double
//@@@ Parser cannot handle path below yet due to terminal '\\'; hardcoding
```

```
path in C++ code for now.
//              ExportPath                      string
                Location                        LocationType
                geoLocation                     GeoLocationType
}

sensor_adapter_instance GeneXpert {

        template GeneXpert_Template

        channels
                data_channels
                        JmsMsgServer::Bio.Data.Cepheid.GeneXpert
                comm_alarm_channels
                        LocalMsgServer::CommAlarms
                        LocalMsgServer::CommAlarmsForJBC2S
                        LocalMsgServer::CommAlarmsForTASS
                        JmsMsgServer::SPM.Alert.Sensor.Comm.Alarms

        properties
                Enabled = 1
                SimulatorMode = 0
                PollingRate = 5.0
//@@@ Parser cannot handle the following path yet due to terminal '\\';
hard-coding path in C++ code for now.
//              ExportPath = "C:\\GeneXpert\\Export\\"
                Location.Location_description = "ViaLogy"
                Location.Location_street_address = "2400 Lincoln Ave"
                Location.Location_city = "Altadena"
                Location.Location_state = "CA"
                Location.Location_zip_code = "91001"
                Location.Location_country = "USA"
                Location.Location_building = "BTC"
                Location.Location_floor = "1"
                Location.Location_room = "DevLab"
                geoLocation.longitude = -118.159705
                geoLocation.latitude = 34.186906
                geoLocation.altitude = 0.0
}
//##### END GeneXpert.


//##### HTTP Server
sensor_adapter_template HTTPServer_Template {

        message_types
```

```
            comm_alarm_message CommAlarmMessage

        protocol HTTPServer

        properties
            Enabled                     integer
            port                        integer
            PollingRate                 double
}

sensor_adapter_instance HTTPServer {

        template HTTPServer_Template

        channels
            comm_alarm_channels
                LocalMsgServer::CommAlarms
                JmsMsgServer::SPM.Alert.Sensor.Comm.Alarms

        properties
            Enabled = 1
            port = 15000
            PollingRate = 1.0
}
//##### END HTTP Server


//##### Inova
sensor_adapter_template Inova_Template {

        message_types
            comm_alarm_message CommAlarmMessage

        protocol Inova

        properties
            Enabled                     integer
            IPAddress                   string
            port                        integer
            PollingRate                 double
            userName                    string
            password                    string
            defaultMessage              string
            commandDuration             integer
            defaultColor                string
            commandMessage              string
```

```
            currentMessage              integer
            commandColor                string
            commandRefreshTime          string
            commandFontSize             string
            commandDisplayMethod        string
            useDefaultMessage           integer
}
sensor_adapter_instance Inova {

        template Inova_Template

        channels
            comm_alarm_channels
                LocalMsgServer::CommAlarms
                JmsMsgServer::SPM.Alert.Sensor.Comm.Alarms

        properties
            Enabled = 1
            IPAddress = "64.210.18.104"
            port = 23
            PollingRate = 1.0
            userName = "admin"
            password = "1n0v@"
            defaultMessage = "Welcome to SPM Dev Lab -- ViaLogy,
California, USA"
            commandDuration = 60
            defaultColor = "green"
            commandMessage = ""
            currentMessage = 100
            commandColor = "red"
            commandRefreshTime = "5"
            commandFontSize = "14"
            commandDisplayMethod = "ribbon_left"
            useDefaultMessage = 1
}
//##### END Inova


//##### ITrans
sensor_adapter_template Modbus_Template {

        message_types
            data_message ModbusRegisterMessage
            comm_alarm_message CommAlarmMessage

        protocol MODBUS
```

```
        properties
                Enabled                         integer
                SimulatorMode                   integer
                IPAddress                       string
                Port                            integer
                PollingRate                     double
                MBAddress                       integer // Modbus RTU only.
                ModBusType                      string
                MBStartAddress                  integer []
                MBCount                         integer
                MBFunction                      string
                MBSlave                         integer
}

sensor_adapter_instance ITrans {

        template Modbus_Template

        channels
                comm_alarm_channels
                        LocalMsgServer::CommAlarms
                        LocalMsgServer::CommAlarmsForJBC2S
                        LocalMsgServer::CommAlarmsForTASS
                        JmsMsgServer::SPM.Alert.Sensor.Comm.Alarms

        properties
                Enabled = 1
                SimulatorMode = 0
                IPAddress = "64.210.18.21"
                Port = 2101
                PollingRate = 1.0
                MBAddress = 1
                ModBusType = "MODBUS_RTU"
                MBStartAddress = { 100 200 }
                MBCount = 6
                MBFunction = "READ_HOLDING_REGISTERS"
                MBSlave = 1
}

sensor_adapter_template ITransSingleHead_Template {

        message_types
                data_message ITransMessage

        protocol ITransChannel
```

```
        properties
                Enabled integer
                SPModel string
}

sensor_adapter_instance Itrans_CO {

        template ITransSingleHead_Template

        channels
                data_channels
                        LocalMsgServer::Chem.Data.ITrans.Gas.CarbonMonoxide
                        JmsMsgServer::Chem.Data.ITrans.Gas.CarbonMonoxide

        filter RegisterValues[2]{1:8} == 1

        properties
                Enabled = 1
                SPModel = "ITrans"
}

sensor_adapter_instance Itrans_CH4 {

        template ITransSingleHead_Template

        channels
                data_channels
                        LocalMsgServer::Chem.Data.ITrans.Gas.Flammables
                        JmsMsgServer::Chem.Data.ITrans.Gas.Flammables

        filter RegisterValues[2]{1:8} == 21

        properties
                Enabled = 1
                SPModel = "ITrans"
}

sensor_adapter_template ITrans_OLD_Template {

        message_types
                data_message ITransMessage
                comm_alarm_message CommAlarmMessage

        protocol ITrans
```

```
        properties
                Enabled integer
                SimulatorMode integer // 0 = use real sensor;
1 = use simulator
        SimulatorType integer // 1 = square wave; 2 =
triangle wave
        SimulatorPeriod double // The period of the wave (in
seconds).
        SimulatorHeight double // The maximum height of the
wave.
        SimulatorProperty string // The SDO property in the
published message to which the simulated data are to be applied.
                IPAddress string
                port integer
                PollingRate double
                geoLocation GeoLocationType
                NameOfDetectedGas string
}

sensor_adapter_instance ITrans_OLD {

        template ITrans_OLD_Template

        channels
                comm_alarm_channels
                        LocalMsgServer::CommAlarms

        properties
                Enabled = 1
                SimulatorMode = 1
        SimulatorType = 1
        SimulatorPeriod = 150.0
        SimulatorHeight = 7.0
        SimulatorProperty = "Concentration"
                IPAddress = "64.210.18.21"
                port = 2101
                PollingRate = 1.0
                geoLocation.longitude = -118.159705
                geoLocation.latitude = 34.186906
                geoLocation.altitude = 0.0
                NameOfDetectedGas = "CH4_Methane" // Need this for
simulator mode
}

sensor_adapter_instance ITrans_CO_OLD {
```

```
        template ITransSingleHead_Template

        channels
                data_channels
                        LocalMsgServer::Chem.Data.ITrans.Gas.CarbonMonoxide
                        JmsMsgServer::Chem.Data.ITrans.Gas.CarbonMonoxide

        filter NameOfDetectedGas == "CO_Carbon_Monoxide"

        properties
                Enabled = 1
                SPModel = "ITrans_OLD"
}

sensor_adapter_instance ITrans_CH4_OLD {

        template ITransSingleHead_Template

        channels
                data_channels
                        LocalMsgServer::Chem.Data.ITrans.Gas.Flammables
                        JmsMsgServer::Chem.Data.ITrans.Gas.Flammables

        filter NameOfDetectedGas == "CH4_Methane"

        properties
                Enabled = 1
                SPModel = "ITrans_OLD"
}
//##### END ITrans


//##### JBC2S
sensor_adapter_template ICD_Template {

        message_types
                comm_alarm_message CommAlarmMessage

        protocol SimpleICD

        properties
                Enabled integer
                SimulatorMode integer
                IPAddress string
                port integer
                PollingRate double
```

```
            ReceiveTimeOut double
            geoLocation GeoLocationType
}

sensor_adapter_instance JBC2S {

        template ICD_Template

        channels
            input_channels
                LocalMsgServer::AlertsForJBC2S
                LocalMsgServer::CommAlarmsForJBC2S
                LocalMsgServer::QTLPolicyAlerts
            comm_alarm_channels
                LocalMsgServer::CommAlarms
                LocalMsgServer::CommAlarmsForTASS
                JmsMsgServer::SPM.Alert.Sensor.Comm.Alarms

        properties
            Enabled = 1
            SimulatorMode = 0
//           IPAddress = "192.168.42.80"
//           port = 7080
            IPAddress = "64.210.18.81"
            port = 15005
            PollingRate = 1.0
            ReceiveTimeOut = 120.0
            geoLocation.longitude = -118.159705
            geoLocation.latitude = 34.186906
            geoLocation.altitude = 0.0
}

sensor_adapter_instance TEMPHTTPServer {

        template HTTPServer_Template

        channels
            comm_alarm_channels
                LocalMsgServer::CommAlarms
                JmsMsgServer::SPM.Alert.Sensor.Comm.Alarms

        properties
            Enabled = 1
            port = 15005
            PollingRate = 1.0
}
```

```
//##### END JBC2S


//##### LCD
sensor_adapter_template LCD_Template {

        message_types
                data_message LCDMessage
//              alert_message LCDMessage
                comm_alarm_message CommAlarmMessage

        protocol Lcd32e

        properties
                Enabled integer
                SimulatorMode integer
                IPAddress string
                port integer
                PollingRate double
                ReceiveTimeOut double
                geoLocation GeoLocationType
}

sensor_adapter_instance LCD {

        template LCD_Template

        channels
                data_channels
                        LocalMsgServer::Chem.Data.LCD3.Gas
//              alert_channels
//                      LocalMsgServer::LocalLCDAlertChannel
                comm_alarm_channels
                        LocalMsgServer::CommAlarms
                        LocalMsgServer::CommAlarmsForTASS
                        LocalMsgServer::CommAlarmsForJBC2S
                        JmsMsgServer::SPM.Alert.Sensor.Comm.Alarms

        properties
                Enabled = 1
                SimulatorMode = 0
                IPAddress = "64.210.18.24"
                port = 2106
//              IPAddress = "10.32.63.24"
//              port = 2106
                PollingRate = 1.0
```

```
            ReceiveTimeOut = 5.0
            geoLocation.longitude = -118.159705
            geoLocation.latitude = 34.186906
            geoLocation.altitude = 0.0
}
//##### END LCD


//##### Midas
sensor_adapter_template Midas_Template {

        message_types
            data_message MidasMessage
            comm_alarm_message CommAlarmMessage

        protocol Midas

        properties
            Enabled integer
            SimulatorMode integer
            IPAddress string
            port integer
            PollingRate double
            buildingLocation BuildingLocationType
            geoLocation GeoLocationType
}

sensor_adapter_instance Midas {

        template Midas_Template

        channels
            data_channels
                LocalMsgServer::Chem.Data.Honeywell.Gas.Chlorine
                JmsMsgServer::Chem.Data.Honeywell.Gas.Chlorine
            comm_alarm_channels
                LocalMsgServer::CommAlarms
                LocalMsgServer::CommAlarmsForTASS
                LocalMsgServer::CommAlarmsForJBC2S
                JmsMsgServer::SPM.Alert.Sensor.Comm.Alarms

        properties
            Enabled = 1
            SimulatorMode = 0
            IPAddress = "64.210.18.9"
            port = 502
```

```
            PollingRate = 1.0
            buildingLocation.Location_building = "BTC"
            buildingLocation.Location_room = "DevLab"
            buildingLocation.Location_floor = "1"
            geoLocation.longitude = -118.159705
            geoLocation.latitude = 34.186906
            geoLocation.altitude = 0.0
}
//##### END Midas


//##### Ortec:
sensor_adapter_template Ortec_Template {

        message_types
            data_message OrtecMessage
            comm_alarm_message CommAlarmMessage

        protocol Ortec

        properties
            Enabled                 integer
            SimulatorMode           integer
            PollingRate             double
            Location                LocationType
            geoLocation             GeoLocationType
}
sensor_adapter_instance Ortec {

        template Ortec_Template

        channels
            data_channels
                LocalMsgServer::Rad.Data.Ortec.DetectiveEx
        comm_alarm_channels
            LocalMsgServer::CommAlarms
            LocalMsgServer::CommAlarmsForJBC2S
            LocalMsgServer::CommAlarmsForTASS
            JmsMsgServer::SPM.Alert.Sensor.Comm.Alarms

        properties
            Enabled = 1
            SimulatorMode = 0
            PollingRate = 3.0
            Location.Location_description = "ViaLogy"
            Location.Location_street_address = "2400 Lincoln Ave"
```

```
            Location.Location_city = "Altadena"
            Location.Location_state = "CA"
            Location.Location_zip_code = "91001"
            Location.Location_country = "USA"
            Location.Location_building = "BTC"
            Location.Location_floor = "1"
            Location.Location_room = "DevLab"
            geoLocation.longitude = -118.159705
            geoLocation.latitude = 34.186906
            geoLocation.altitude = 0.0
}
//##### END Ortec.


//##### NetGuardian
sensor_adapter_template SnmpManager_Template {

        message_types
                data_message SnmpTrapMessage
                comm_alarm_message CommAlarmMessage

        protocol SnmpManager

        properties
                Enabled                         integer
                SimulatorMode                   integer
                TrapPort                        integer
                PollingRate                     double
                ReadCommunityString             string
                WriteCommunityString            string
                NotificationCommunityString     string
}

sensor_adapter_instance NetGuardianSnmpManager {

        template SnmpManager_Template

        channels
                comm_alarm_channels
                        LocalMsgServer::CommAlarms

        properties
                Enabled = 1
                SimulatorMode = 0
                TrapPort = 162
                PollingRate = 1.0
```

```
            ReadCommunityString = "public"
            WriteCommunityString = "public"
            NotificationCommunityString = "public"
}

sensor_adapter_template SnmpTrapReceiver_Template {

        message_types
            data_message SnmpTrapMessage

        protocol SnmpTrapReceiver

        properties
            Enabled                 integer
            SimulatorMode           integer
            SPModel                 string
            geoLocation             GeoLocationType
}

sensor_adapter_instance NetGuardianSnmpTrapReceiver {

        template SnmpTrapReceiver_Template

        channels
            data_channels
                LocalMsgServer::NetGuardian.SNMP.Data.Alarms
            comm_alarm_channels
                LocalMsgServer::CommAlarms

        properties
            Enabled = 1
            SimulatorMode = 0
            SPModel = "NetGuardianSnmpManager"
            geoLocation.longitude = -118.159705
            geoLocation.latitude = 34.186906
            geoLocation.altitude = 0.0
}
//##### END NetGuardian


//##### QTL
sensor_adapter_template QTL_Template {

        message_types
            data_message QTLMessage
            comm_alarm_message CommAlarmMessage
```

```
        protocol QTLBioSensor

        properties
                Enabled                         integer
                SimulatorMode                   integer
                IPAddress                       string
                port                            integer
                PollingRate                     double
                AssayType                       string
                HazMat                          string
                Location                        LocationType
                geoLocation                     GeoLocationType
}

sensor_adapter_instance QTL {

        template QTL_Template

        channels
                data_channels
                        LocalMsgServer::Bio.Data.QTL.Biosensor_2000
                        JmsMsgServer::Bio.Data.QTL.Biosensor_2000
                        SoapMsgServer::Bio.Data.QTL.Biosensor_2000
                comm_alarm_channels
                        LocalMsgServer::CommAlarms
                        LocalMsgServer::CommAlarmsForJBC2S
                        LocalMsgServer::CommAlarmsForTASS
                        JmsMsgServer::SPM.Alert.Sensor.Comm.Alarms

        properties
                Enabled = 1
                SimulatorMode = 0
                IPAddress = "64.210.18.24"
                port = 2107
                PollingRate = 1.0
                AssayType = "Anthrax"
                HazMat = "HAZMAT 323"
                Location.Location_description = "ViaLogy"
                Location.Location_street_address = "2400 Lincoln Ave"
                Location.Location_city = "Altadena"
                Location.Location_state = "CA"
                Location.Location_zip_code = "91001"
                Location.Location_country = "USA"
                Location.Location_building = "BTC"
                Location.Location_floor = "1"
                Location.Location_room = "DevLab"
```

```
            geoLocation.longitude = -118.159705
            geoLocation.latitude = 34.186906
            geoLocation.altitude = 0.0
}
//##### END QTL


//##### SMC
sensor_adapter_instance SMC {

        template Modbus_Template

        channels
                data_channels
                        LocalMsgServer::Chem.Data.SMC.Gas
                        TCPMsgServer::Chem.Data.SMC.Gas
                comm_alarm_channels
                        LocalMsgServer::CommAlarms
                        LocalMsgServer::CommAlarmsForJBC2S
                        LocalMsgServer::CommAlarmsForTASS
                        JmsMsgServer::SPM.Alert.Sensor.Comm.Alarms

        properties
                Enabled = 1
                SimulatorMode = 0
                IPAddress = "64.210.18.100"
                Port = 502
                PollingRate = 2.0
                MBAddress = 11
                ModBusType = "MODBUS_TCP"
                MBStartAddress = { 0 }
                MBCount = 11
                MBFunction = "READ_HOLDING_REGISTERS"
                MBSlave = 11
}
//##### END SMC


//##### TASS
sensor_adapter_instance TASS {

        template ICD_Template

        channels
                input_channels
                        LocalMsgServer::AlertsForTASS
```

```
                LocalMsgServer::CommAlarmsForTASS
        comm_alarm_channels
                LocalMsgServer::CommAlarms
                LocalMsgServer::CommAlarmsForJBC2S
                JmsMsgServer::SPM.Alert.Sensor.Comm.Alarms

    properties
        Enabled = 1
        SimulatorMode = 0
        IPAddress = "192.168.42.94"
        port = 4445
        PollingRate = 1.0
        ReceiveTimeOut = 120.0
        geoLocation.longitude = -118.159705
        geoLocation.latitude = 34.186906
        geoLocation.altitude = 0.0
}
//##### END TASS

//##### ThermoElectron
sensor_adapter_template ThermoElectron_Template {

    message_types
        data_message ThermoElectronMessage
        comm_alarm_message CommAlarmMessage

    protocol ThermoElectron

    properties
        Enabled                 integer
        SimulatorMode           integer
        IPAddress               string
        port                    integer
        PollingRate             double
        ReceiveTimeOut          double
        NameOfDetectedGas       string
        geoLocation             GeoLocationType
}
sensor_adapter_instance ThermoElectron {

    template ThermoElectron_Template

    channels
        data_channels
            LocalMsgServer::Chem.Data.ThermoElectron.Gas.Ammonia
            JmsMsgServer::Chem.Data.ThermoElectron.Gas.Ammonia
```

```
                comm_alarm_channels
                        LocalMsgServer::CommAlarms
                        LocalMsgServer::CommAlarmsForJBC2S
                        LocalMsgServer::CommAlarmsForTASS
                        JmsMsgServer::SPM.Alert.Sensor.Comm.Alarms

        properties
                Enabled = 1
                SimulatorMode = 0
//              IPAddress = "64.210.18.20"
//              port = 4000
                IPAddress = "64.210.18.24"
                port = 2108
                PollingRate = 1.0
                ReceiveTimeOut = 10.0
                NameOfDetectedGas = "Ammonia" // Is this necessary? Does
it not come from the sensor?
                geoLocation.longitude = -118.159705
                geoLocation.latitude = 34.186906
                geoLocation.altitude = 0.0
}
//##### END ThermoElectron


//##### Tivella
sensor_adapter_template Tivella_template {

        message_types
                comm_alarm_message CommAlarmMessage

        protocol Tivella

        properties
                Enabled                 integer
                IPAddress               string
                Port                    integer
                PollingRate             double
                HTTPGetURL              string
}
sensor_adapter_instance Tivella {

        template Tivella_template

        channels
                comm_alarm_channels
                        LocalMsgServer::CommAlarms
```

```
                    JmsMsgServer::SPM.Alert.Sensor.Comm.Alarms

        properties
                Enabled = 1
                IPAddress = "64.210.18.94"
                Port = 80
                PollingRate = 1.0
                HTTPGetURL = "http://www.google.com"
}
//##### END Tivella


//##### Voxeo
sensor_adapter_template Voxeo_Template {

        message_types
                alert_message AlertMessage

        protocol Voxeo

        properties
                Enabled integer
                SimulatorMode integer
                IPAddress string
                port integer
                application string
                numberToDial string
                tokenID string
                messageToSend string
                recvTimeout integer
}
sensor_adapter_instance Voxeo {

        template Voxeo_Template

        channels
                input_channels
                        LocalMsgServer::AlertsForVoxeo
                comm_alarm_channels
                        LocalMsgServer::CommAlarms
                        JmsMsgServer::SPM.Alert.Sensor.Comm.Alarms

        properties
                Enabled = 1
                SimulatorMode = 0
                IPAddress = "session.voxeo.net"
```

```
                port = 0
                application = "VoiceXML.start"
                numberToDial = "6262966473"
                tokenID =
"341b34d433ed0b4db7bef7fd57a6786ecb04a4ac8940879ac9c211021cd0e9422678b5
c5d5d08fa668714697"
                messageToSend = "Have a nice day"
                recvTimeout = 180
}
//##### END Voxeo
//#########################################################################
#########


//##### Policies
#################################################################
//##### ITrans
condition_template EXPLOSIVE_GAS_cond_template {
        channel_reference(input_channel->Concentration) >
constant_reference(Explosive_gas_threshold)
}

action_template EXPLOSIVE_GAS_action_template {
        if !register_reference(ALERT_POSTED_REG)
        then

                set register_reference(ALERT_POSTED_REG) : true
                set register_reference(NUM_ALERTS_REG) : 1
                set register_reference(FIRST_ALERT_POSTED_TIME_REG) :
CURRENT_TIME()
                set register_reference(LAST_ALERT_POSTED_TIME_REG) :
                register_reference(FIRST_ALERT_POSTED_TIME_REG)
                set register_reference(ALERT_TEXT_REG) :
                        "ALERT: Flammable gas concentration has exceeded " +
constant_reference(Explosive_gas_threshold) + " " +
constant_reference(Concentration_units) + "."
                publish
                        alert : register_reference(ALERT_TEXT_REG)
                        AlertLevel : "RED"
                        referenceChannels :
"Chem.Data.ITrans.Gas.Flammables~Chem.Video.Frames.EmissionsLabEast~Che
m.Video.Frames.EmissionsLabWest"
                        buildingLocation : channel_reference(input_channel-
>buildingLocation)
                        alternateLocation : channel_reference(input_channel-
>alternateLocation)
```

```
                    geoLocation : channel_reference(input_channel-
>geoLocation)
                    reference_channels(input_channel->Concentration,
camera_channel1)
                    output_channels(alertChannel1)
            control
                    commandMessage : register_reference(ALERT_TEXT_REG)
                    sensors(Inova)
    else
            if TIMESTAMP_DIFF_SECS(CURRENT_TIME(),
register_reference(LAST_ALERT_POSTED_TIME_REG)) >
constant_reference(Repeat_alert_pause_secs)
            then
                    set register_reference(NUM_ALERTS_REG) :
register_reference(NUM_ALERTS_REG) + 1
                    set register_reference(LAST_ALERT_POSTED_TIME_REG) :
CURRENT_TIME()
                    set register_reference(ALERT_TEXT_REG) :
                            "ALERT: Flammable gas concentration has
exceeded " + constant_reference(Explosive_gas_threshold) + " " +
constant_reference(Concentration_units) +
                            " since " +
FORMAT_TIMESTAMP(register_reference(FIRST_ALERT_POSTED_TIME_REG)) + "
(Alert " + register_reference(NUM_ALERTS_REG) + ")."
                    publish
                            alert : register_reference(ALERT_TEXT_REG)
                            AlertLevel : "RED"
                            referenceChannels :
"Chem.Data.ITrans.Gas.Flammables~Chem.Video.Frames.EmissionsLabEast~Che
m.Video.Frames.EmissionsLabWest"
                            buildingLocation :
channel_reference(input_channel->buildingLocation)
                            alternateLocation :
channel_reference(input_channel->alternateLocation)
                            geoLocation : channel_reference(input_channel-
>geoLocation)
                            reference_channels(input_channel->Concentration,
camera_channel1)
                            output_channels(alertChannel1, alertChannel2)
                    control
                            commandMessage :
register_reference(ALERT_TEXT_REG)
                            sensors(Inova)
            fi
    fi
}
```

```
policy_template EXPLOSIVE_GAS_template {
        condition_template EXPLOSIVE_GAS_cond_template
        action_template EXPLOSIVE_GAS_action_template
}

policy_instance EXPLOSIVE_GAS {

        template EXPLOSIVE_GAS_template

        input_channels
                LocalMsgServer::Chem.Data.ITrans.Gas.Flammables trigger
        constants
                Explosive_gas_threshold : 5.0
                Concentration_units : "ppm"
                Repeat_alert_pause_secs : 60

        channel_bindings
                input_channel :
LocalMsgServer::Chem.Data.ITrans.Gas.Flammables
                camera_channel1 : JmsMsgServer::BMS.Video.Frames.IPCam.1
                alertChannel1 : JmsMsgServer::Chem.Alert.FireDepartment
                alertChannel2 : LocalMsgServer::AlertsForVoxeo

        register_bindings
                ALERT_POSTED_REG :
SA_Viper::EXPLOSIVE_GAS_ALERT_POSTED_REG
                FIRST_ALERT_POSTED_TIME_REG :
SA_Viper::EXPLOSIVE_GAS_FIRST_ALERT_POSTED_TIME_REG
                LAST_ALERT_POSTED_TIME_REG :
SA_Viper::EXPLOSIVE_GAS_LAST_ALERT_POSTED_TIME_REG
                ALERT_TEXT_REG : SA_Viper::EXPLOSIVE_GAS_ALERT_TEXT_REG
                NUM_ALERTS_REG :
SA_Viper::EXPLOSIVE_GAS_NUM_ALERTS_REG

        sensor_bindings
                Inova : SA_Viper::Inova
}

condition_template NO_EXPLOSIVE_GAS_cond_template {
        (channel_reference(input_channel->Concentration) <=
constant_reference(Explosive_gas_threshold)) &&
        register_reference(ALERT_POSTED_REG)
}

action_template NO_EXPLOSIVE_GAS_action_template {
```

```
        set register_reference(ALERT_POSTED_REG) : false
        set register_reference(ALERT_TEXT_REG) :
                "Flammable gas concentration has dropped below " +
constant_reference(Explosive_gas_threshold) + " " +
constant_reference(Concentration_units) + "."
        publish
                alert : register_reference(ALERT_TEXT_REG)
                AlertLevel : "GREEN"
                referenceChannels :
"Chem.Data.ITrans.Gas.Flammables~Chem.Video.Frames.EmissionsLabEast~Che
m.Video.Frames.EmissionsLabWest"
                buildingLocation : channel_reference(input_channel-
>buildingLocation)
                alternateLocation : channel_reference(input_channel-
>alternateLocation)
                geoLocation : channel_reference(input_channel->geoLocation)
                reference_channels(input_channel->Concentration,
camera_channel1)
                output_channels(alertChannel1)
        control
                commandMessage : register_reference(ALERT_TEXT_REG)
                sensors(Inova)
        if (register_reference(NUM_ALERTS_REG) > 1)
        then
                publish
                        alert : register_reference(ALERT_TEXT_REG)
                        AlertLevel : "GREEN"
                        referenceChannels :
"Chem.Data.ITrans.Gas.Flammables~Chem.Video.Frames.EmissionsLabEast~Che
m.Video.Frames.EmissionsLabWest"
                        buildingLocation : channel_reference(input_channel-
>buildingLocation)
                        alternateLocation : channel_reference(input_channel-
>alternateLocation)
                        geoLocation : channel_reference(input_channel-
>geoLocation)
                        reference_channels(input_channel->Concentration,
camera_channel1)
                        output_channels(alertChannel2)
        fi
        set register_reference(NUM_ALERTS_REG) : 0
}

policy_template NO_EXPLOSIVE_GAS_template {
        condition_template NO_EXPLOSIVE_GAS_cond_template
        action_template NO_EXPLOSIVE_GAS_action_template
```

```
}

policy_instance NO_EXPLOSIVE_GAS {

        template NO_EXPLOSIVE_GAS_template

        input_channels
                LocalMsgServer::Chem.Data.ITrans.Gas.Flammables trigger

        constants
                Explosive_gas_threshold : 5.0
                Concentration_units : "ppm"

        channel_bindings
                input_channel :
LocalMsgServer::Chem.Data.ITrans.Gas.Flammables
                camera_channel1 : JmsMsgServer::BMS.Video.Frames.IPCam.1
                alertChannel1 : JmsMsgServer::Chem.Alert.FireDepartment
                alertChannel2 : LocalMsgServer::AlertsForVoxeo

        register_bindings
                ALERT_POSTED_REG :
SA_Viper::EXPLOSIVE_GAS_ALERT_POSTED_REG
                ALERT_TEXT_REG : SA_Viper::EXPLOSIVE_GAS_ALERT_TEXT_REG
                NUM_ALERTS_REG :
SA_Viper::EXPLOSIVE_GAS_NUM_ALERTS_REG

        sensor_bindings
                Inova : SA_Viper::Inova
}
//##### END ITrans


//##### QTL
condition_template QTL_POS_cond_template {
        (channel_reference(input_channel->TestResult) == "Positive")
}

action_template QTL_POS_action_template {
        publish
                alert : "ALERT: " + channel_reference(input_channel-
>AssayType) + " found in sample " +
                                channel_reference(input_channel->SampleId) + ".
Detected by " +
                                channel_reference(input_channel->HazMat) + " in
" +
```

```
                        channel_reference(input_channel->City) + ", " +
                        channel_reference(input_channel->State) + "."
            reference_channels(input_channel)
            output_channels(alert_channel1, alert_channel2,
alert_channel3, alert_channel4)
        control
            commandMessage : "ALERT: " +
channel_reference(input_channel->AssayType) + " found in sample " +
                                channel_reference(input_channel-
>SampleId) + ". Detected by " +
                                channel_reference(input_channel-
>HazMat) + " in " +
                                channel_reference(input_channel-
>City) + ", " +
                                channel_reference(input_channel-
>State) + "."
            sensors(Inova)
}

policy_template QTL_POS_template {
        condition_template QTL_POS_cond_template
        action_template QTL_POS_action_template
}

policy_instance QTL_POS {

        template QTL_POS_template

        input_channels
            LocalMsgServer::Bio.Data.QTL.Biosensor_2000 trigger

        channel_bindings
            input_channel : LocalMsgServer::Bio.Data.QTL.Biosensor_2000
            alert_channel1 : JmsMsgServer::Bio.Alert.Technician
            alert_channel2 : JmsMsgServer::Bio.Alert.LabManager
            alert_channel3 : JmsMsgServer::Bio.Alert.FireDepartment
            alert_channel4 : LocalMsgServer::QTLPolicyAlerts

        field_bindings
            City : Location.Location_city
            State : Location.Location_state

        sensor_bindings
            Inova : SA_Viper::Inova
}
//##### END QTL
```

```
//#########################################################################
#########

viper_server SA_Viper {

        soap_server_port 4000

        message_servers
                LocalMsgServer
                JmsMsgServer
                SoapMsgServer
                TCPMsgServer

        registers
                EXPLOSIVE_GAS_ALERT_POSTED_REG : boolean = false
                EXPLOSIVE_GAS_FIRST_ALERT_POSTED_TIME_REG : string =
"0000000000"
                EXPLOSIVE_GAS_LAST_ALERT_POSTED_TIME_REG : string =
"0000000000"
                EXPLOSIVE_GAS_ALERT_TEXT_REG : string = ""
                EXPLOSIVE_GAS_NUM_ALERTS_REG : integer = 0

        sensor_adapters
//@@@ Multi-headed sensor adapter needs to be before single-headed
sensor adapters.
//* Defense:
//*** Chemical:
                LCD
                Midas
                SMC
                ThermoElectron
//*** Biological:
                GeneXpert
                QTL
// Radiological:
                Ortec
// Nuclear:
// Explosive:
//              ITrans
//              Itrans_CH4
//              Itrans_CO
                ITrans_OLD
                ITrans_CH4_OLD
                ITrans_CO_OLD
//*** Misc:
                TASS // SimpleICD
```

```
                JBC2S // SimpleICD
//* CCRE:
                AlienRFID
                Barionet
                NetGuardianSnmpManager
                NetGuardianSnmpTrapReceiver
//* Misc:
                ArecontCam_01
                HTTPServer
                Inova
                Tivella
                Voxeo
        policies
                EXPLOSIVE_GAS
                NO_EXPLOSIVE_GAS
                QTL_POS
} // END SA_Viper
```

[001421] Embodiments of the subject matter and the functional operations of systems described in this specification can be implemented in digital electronic circuitry, or in computer software, firmware, or hardware, including the structures disclosed in this specification and their structural equivalents, or in combinations of one or more of them. Embodiments of the subject matter described in this specification can be implemented as one or more computer program products, i.e., one or more modules of computer program instructions encoded on a tangible program carrier for execution by, or to control the operation of, data processing apparatus. The tangible program carrier can be a computer readable medium. The computer readable medium can be a machine-readable storage device, a machine-readable storage substrate, a memory device, a composition of matter effecting a machine-readable propagated signal, or a combination of one or more of them.

[001422] The term “data processing apparatus” encompasses all apparatus, devices, and machines for processing data, including by way of example a programmable processor, a computer, or multiple processors or computers. The apparatus can include, in addition to hardware, code that creates an execution environment for the computer program in question, e.g., code that constitutes processor firmware, a protocol stack, a database management system, an

operating system, or a combination of one or more of them.

[001423] A computer program (also known as a program, software, software application, script, or code) can be written in any form of programming language, including compiled or interpreted languages, or declarative or procedural languages, and it can be deployed in any form, including as a stand alone program or as a module, component, subroutine, or other unit suitable for use in a computing environment. A computer program does not necessarily correspond to a file in a file system. A program can be stored in a portion of a file that holds other programs or data (e.g., one or more scripts stored in a markup language document), in a single file dedicated to the program in question, or in multiple coordinated files (e.g., files that store one or more modules, sub programs, or portions of code). A computer program can be deployed to be executed on one computer or on multiple computers that are located at one site or distributed across multiple sites and interconnected by a communication network.

[001424] Operations among the processing/storage unit 420, the display 410, the reader, and the network 430 can be performed by one or more programmable processors executing one or more computer programs to perform functions by operating on input data and generating output. In addition, special purpose logic circuitry, e.g., an FPGA (field programmable gate array) or an ASIC (application specific integrated circuit) can be used.

[001425] Processors suitable for the execution of a computer program include, by way of example, both general and special purpose microprocessors, and any one or more processors of any kind of digital computer. Generally, a

processor will receive instructions and data from a read only memory or a random access memory or both. The essential elements of a computer are a processor for performing instructions and one or more memory devices for storing instructions and data. Generally, a computer will also include, or be operatively coupled to receive data from or transfer data to, or both, one or more mass storage devices for storing data, e.g., magnetic, magneto optical disks, or optical disks. However, a computer need not have such devices. Moreover, a computer can be embedded in another device.

[001426] Computer readable media suitable for storing computer program instructions and data include all forms of non volatile memory, media and memory devices, including by way of example semiconductor memory devices, e.g., EPROM, EEPROM, and flash memory devices; magnetic disks, e.g., internal hard disks or removable disks; magneto optical disks; and CD ROM and DVD-ROM disks. The processor and the memory can be supplemented by, or incorporated in, special purpose logic circuitry.

[001427] To provide for interaction with a user, embodiments of the subject matter described in this specification can be implemented on a computer having a display device, e.g., a CRT (cathode ray tube) or LCD (liquid crystal display) monitor, for displaying information to the user and a keyboard and a pointing device, e.g., a mouse or a trackball, by which the user can provide input to the computer. Other kinds of devices can be used to provide for interaction with a user as well; for example, input from the user can be received in any form, including acoustic, speech, or tactile input.

[001428] The processing/storage unit 420 as described in this specification can be implemented in a computing system that includes a back end component, e.g., as a data server, or that includes a middleware component, e.g., an application server, or that includes a front end component, e.g., a client computer having a graphical user interface or a Web browser through which a user can interact with an implementation of the subject matter described is this specification, or any combination of one or more such back end, middleware, or front end components. The computing system can be interconnected to the display 410 and the reader 130 by the network 430 that includes any form or medium of digital data communication, e.g., a communication network. Examples of communication networks include a local area network ("LAN") and a wide area network ("WAN"), e.g., the Internet.

[001429] The computing system can include clients and servers. A client and server are generally remote from each other and typically interact through a communication network. The relationship of client and server arises by virtue of computer programs running on the respective computers and having a client-server relationship to each other.

[001430] While this specification contains many specifics, these should not be construed as limitations on the scope of any invention or of what may be claimed, but rather as descriptions of features that may be specific to particular embodiments of particular inventions. Certain features that are described in this specification in the context of separate embodiments can also be implemented in combination in a single embodiment. Conversely, various features that are

described in the context of a single embodiment can also be implemented in multiple embodiments separately or in any suitable subcombination. Moreover, although features may be described above as acting in certain combinations and even initially claimed as such, one or more features from a claimed combination can in some cases be excised from the combination, and the claimed combination may be directed to a subcombination or variation of a subcombination.

[001431] Similarly, while operations are depicted in the drawings in a particular order, this should not be understood as requiring that such operations be performed in the particular order shown or in sequential order, or that all illustrated operations be performed, to achieve desirable results. In certain circumstances, multitasking and parallel processing may be advantageous. Moreover, the separation of various system components in the embodiments described above should not be understood as requiring such separation in all embodiments.

[001432] Only a few implementations and examples are described and other implementations, enhancements and variations can be made based on what is described and illustrated in this application.

CLAIMS

What is claimed is:

1. A system for implementing sensor policy markup language (SPML) comprising:
   a system for defining sensor primitives;
   a system for defining policy primitives;
   a system for generating sensor conditionals; and
   a system for genating sensor actions

2. A method for processing sensor data based on an interaction of one or more of the following:
   one or more sensor adaptors;
   a policy engine;
   one or more messaging servers; and
   an bject mediator

3. A system for transforming a user defined policy into a sensor policy markup language comprising:
   defining policies in a sequence of SPML expressions.

4. A method for interfacing to sensor, video, GIS, external applications, external models, robotic and actuator systems.

5. A Method as in claim 4, further comprising normalizing sensor, video, GIS, application and model data into a service data object (SDO); and
   using policies to operate on the SDO.

6. A method for converting data on any protocol to SDO for use in policy-based computations comprising:
generating a computational efficient model for protocol mediation; and
generating a computationally efficient representation for SDO.

7. A computer implemented method for efficiently mapping data from any sensor protocol to another standardized protocol or sub-protocol.

8. A method for developing asimple and complex/composite sensor/protocol adaptors to drive a policy engine.

9. A method for distributing protocol mediation over a wired or wireless network

10. A method for hierarchical protocol mediation over a large distributed network.

11. A method for multistage protocol mediation over a network.

12. A method for generating a protocol mediation on demand.

13. A method for protocol mediation over closed and firewalled networks.

14. A method for dynamic distribution of protocol mediation computation over a network with variable quality of service (QOS) behaviors.

15. A method for distrusting policy engine over multiple computational elements over the network.

16. A method for deploying SPM on a server.

17. A method for deploying SPM on multiple servers.

18. A method for deploying SPM in a network applicance – router.

19. A method for deploying SPM in a mobile network appliance – mobile access router.

20. A method for deploying SPM in a network access point

21. A method for deploying SPM in a Application Over Network (AON) blade

22. A method for deploying SPM in a Video switch.

23. A method for transforming video streams into a computationally enabled SDO.

24. A method for deploying SPM in a DSP board.

25. A novel method for deploying SPM in a FPGA board.

26. A method for deploying policies on a network address to alleviate server load.

27. A computerized system for managing sensor data from sensors, comprising:

a plurality of sensor adapter modules that receive digital sensor data from sensors and convert received sensor data in a selected sensor data format;

a sensor adaptor application programming interface (API) in communication with the sensor adapter modules to transmit the sensor data from the sensor adapter modules;

a sensor policy engine comprising sensor management policies and in communication with the sensor adaptor API;

a graphic user interface (GUI) to provide user control tools enabling a user to manage and control sensor adapter and sensor management policies in the sensor policy engine; and

a sensor policy manager API in communication with the GUI, the sensor adaptor API and the sensor policy engine, the sensor policy manager API operable to communicate instructions and messages between the GUI and the sensor policy engine and the sensor adapter API.

28. The system as in claim 27, comprising:

a sensor database to receive and store sensor data from the sensors.

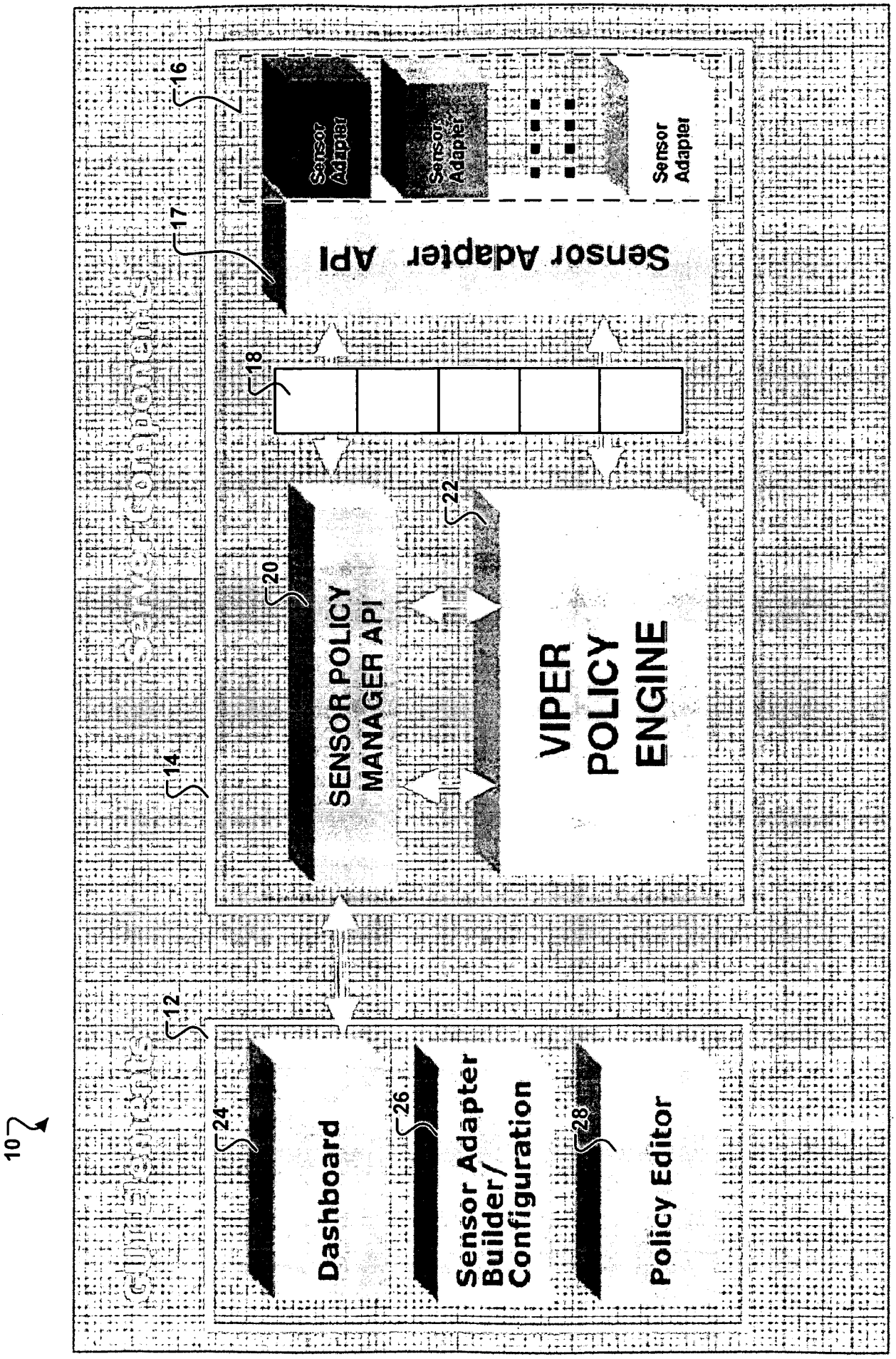

10
12
14
16
17
18
20
22
24
26
28
Sensor Adapter
Sensor Adapter
Sensor Adapter
Sensor Adapter API
SENSOR POLICY MANAGER API
VIPER POLICY ENGINE
Dashboard
Sensor Adapter Builder/ Configuration
Policy Editor

FIG. 1

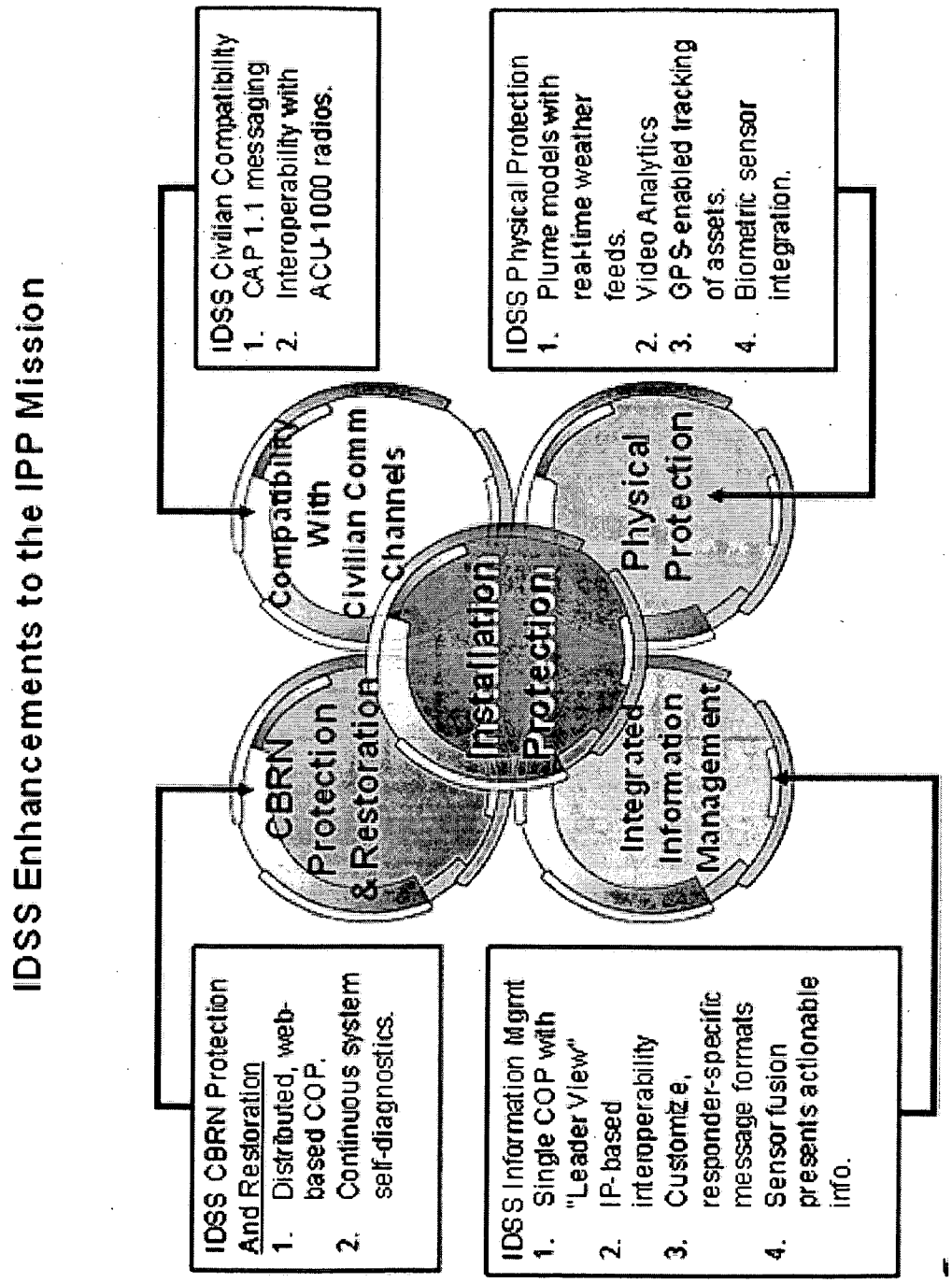

IDSS Enhancements to the IPP Mission
IDSS Civilian Compatibility
1. CAP 1.1 messaging
2. Interoperability with ACU-1000 radios.
IDSS Physical Protection
1. Plume models with real-time weather feeds.
2. Video Analytics
3. GPS-enabled tracking of assets.
4. Biometric sensor integration.
Compatibility With Civilian Comm Channels
CBRN Protection & Restoration
Installation Protection
Physical Protection
Integrated Information Management
IDSS CBRN Protection And Restoration
1. Distributed, web-based COP.
2. Continuous system self-diagnostics.
IDSS Information Mgmt
1. Single COP with "Leader View"
2. IP-based interoperability
3. Customize, responder-specific message formats
4. Sensor fusion presents actionable info.

FIG. 2

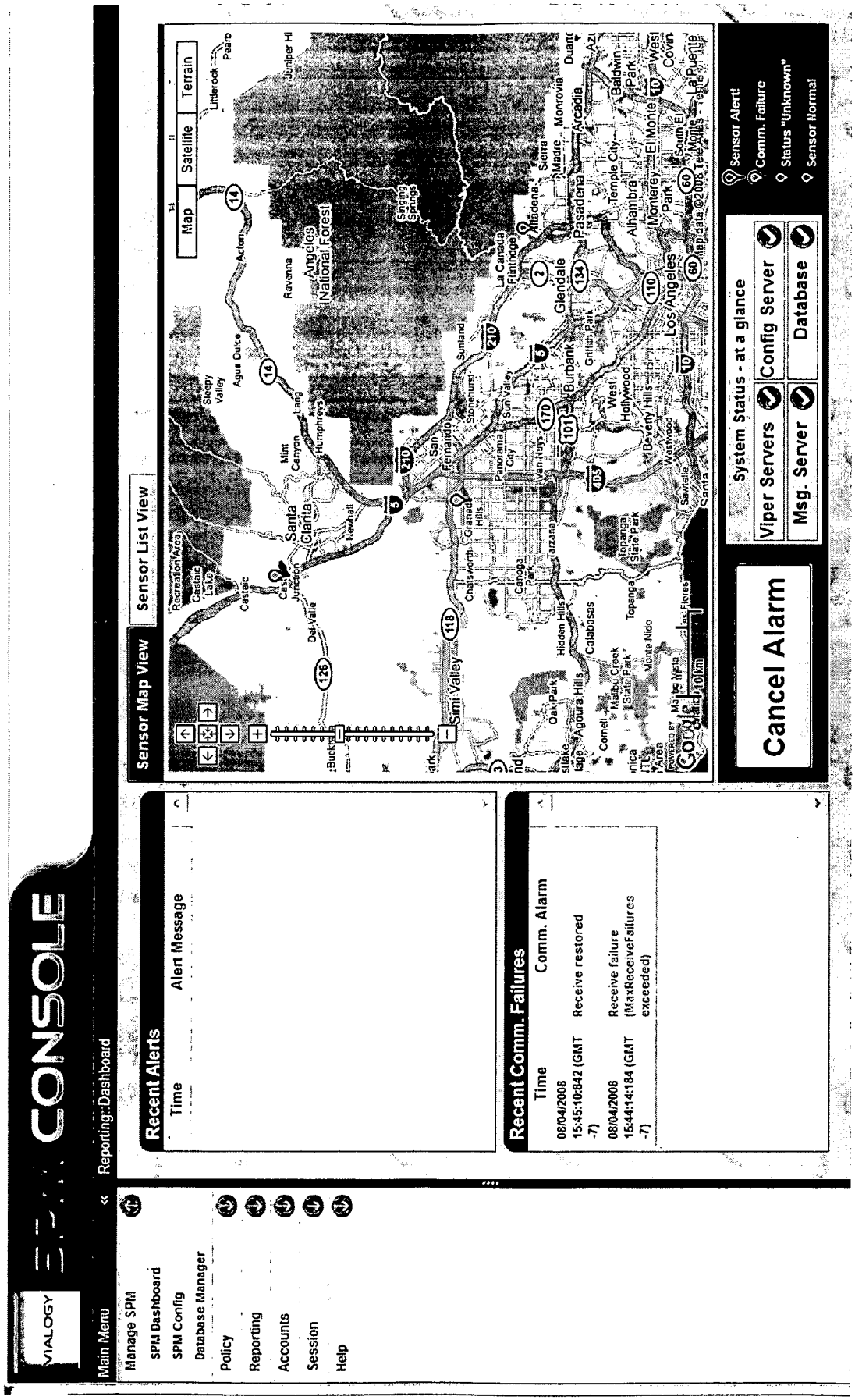

VIALOGY
CONSOLE
Main Menu
Reporting::Dashboard
Manage SPM
SPM Dashboard
SPM Config
Database Manager
Policy
Reporting
Accounts
Session
Help
Sensor Map View
Sensor List View
Map
Satellite
Terrain
Recent Alerts
Time
Alert Message
Recent Comm. Failures
Time
Comm. Alarm
08/04/2008 15:45:10:842 (GMT -7)
Receive restored
08/04/2008 15:44:14:184 (GMT -7)
Receive failure (MaxReceiveFailures exceeded)
Cancel Alarm
System Status - at a glance
Viper Servers
Config Server
Msg. Server
Database
Sensor Alert!
Comm. Failure
Status "Unknown"
Sensor Normal

FIG. 3

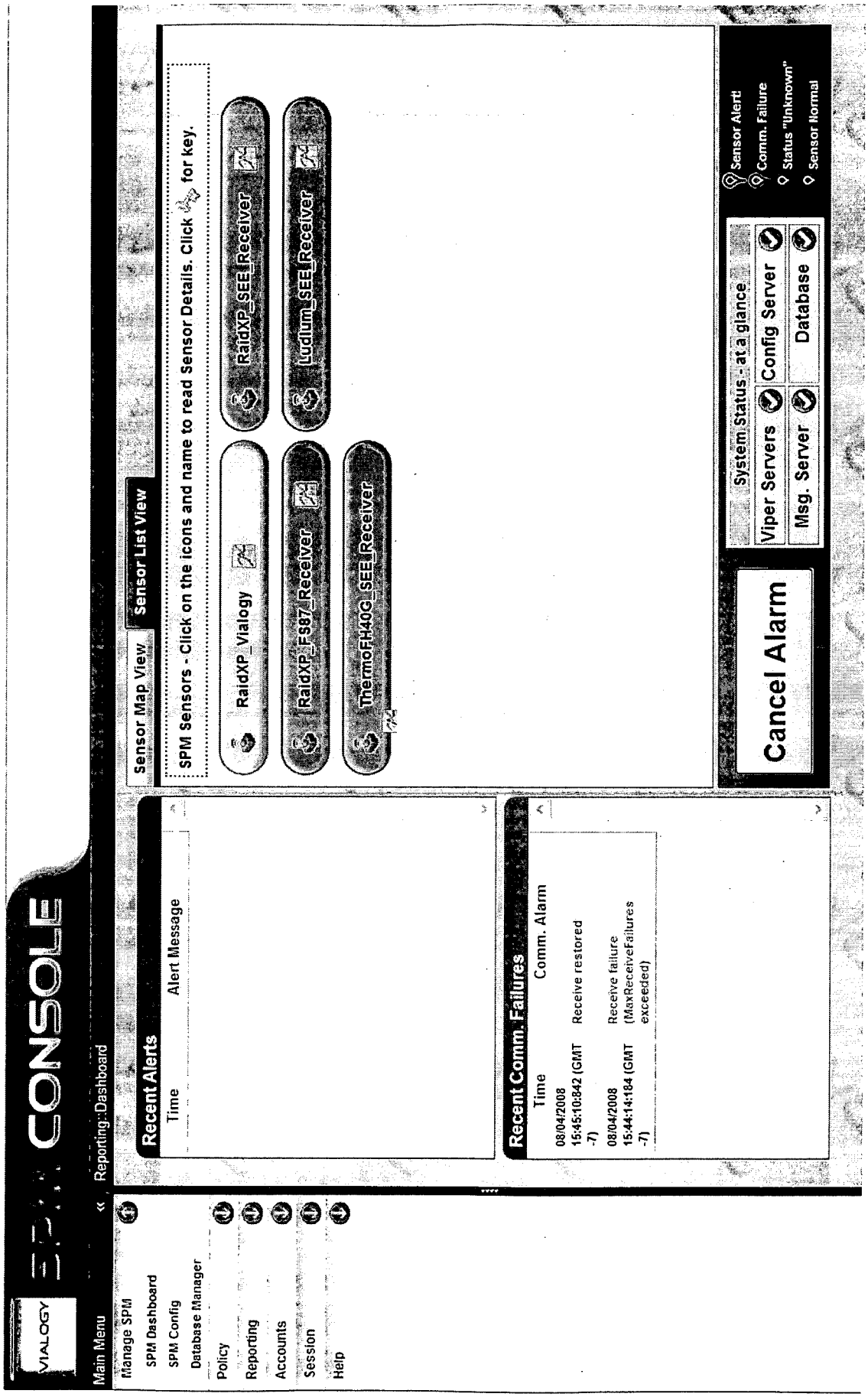

VIALOGY
CONSOLE
Main Menu
Reporting::Dashboard
Manage SPM
SPM Dashboard
SPM Config
Database Manager
Policy
Reporting
Accounts
Session
Help
Recent Alerts
Time
Alert Message
Recent Comm. Failures
Time
Comm. Alarm
08/04/2008 15:45:10:842 (GMT -7)
Receive restored
08/04/2008 15:44:14:184 (GMT -7)
Receive failure (MaxReceiveFailures exceeded)
Sensor Map View
Sensor List View
SPM Sensors - Click on the icons and name to read Sensor Details. Click for key.
RaidXP Vialogy
RaidXP_SEE Receiver
RaidXP_FS87 Receiver
Ludlum_SEE Receiver
ThermoFH40G_SEE Receiver
Cancel Alarm
System Status - at a glance
Viper Servers
Config Server
Msg. Server
Database
Sensor Alert!
Comm. Failure
Status "Unknown"
Sensor Normal

FIG. 4

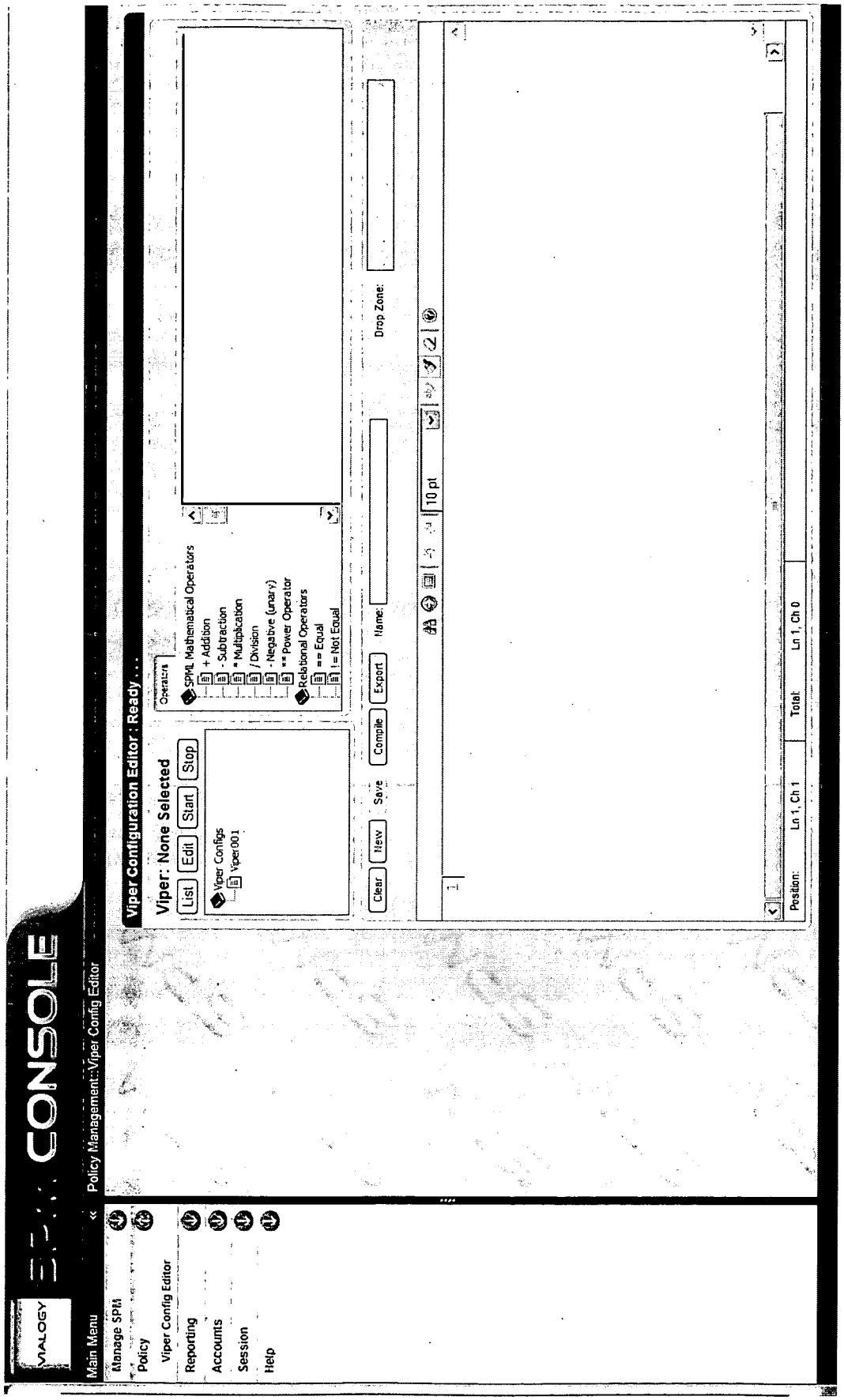

CONSOLE
Main Menu
Policy Management::Viper Config Editor
Manage SPM
Policy
Viper Config Editor
Reporting
Accounts
Session
Help
Viper Configuration Editor : Ready . . .
Viper: None Selected
List
Edit
Start
Stop
Viper Configs
Viper001
Operators
SPML Mathematical Operators
+ Addition
- Subtraction
* Multiplication
/ Division
- Negative (unary)
** Power Operator
Relational Operators
== Equal
!= Not Equal
Clear
New
Save
Compile
Export
Name:
Drop Zone:
10 pt
Position:
Ln 1, Ch 1
Total:
Ln 1, Ch 0

FIG. 5

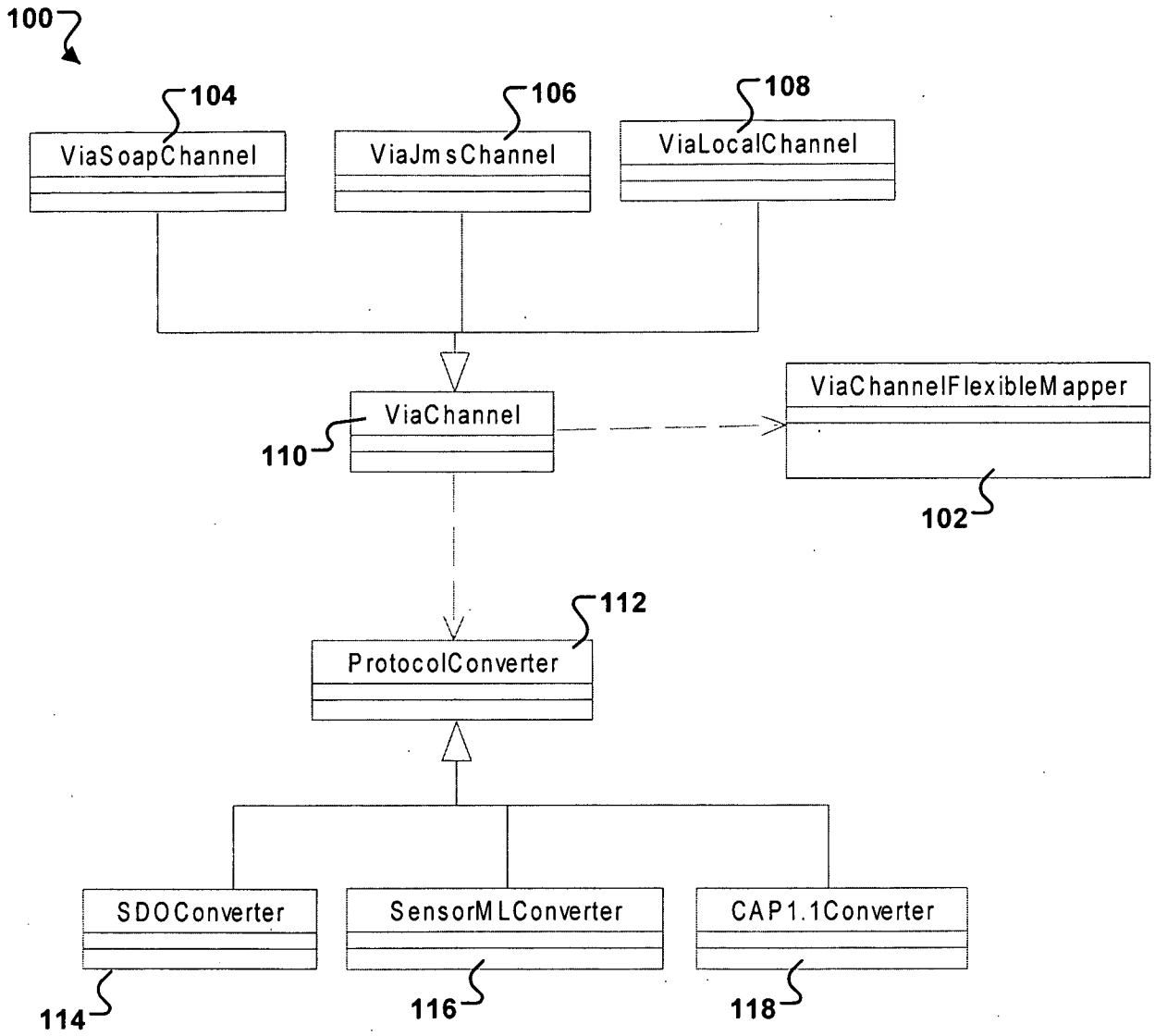

100
104
ViaSoapChannel
106
ViaJmsChannel
108
ViaLocalChannel
ViaChannel
110
ViaChannelFlexibleMapper
102
112
ProtocolConverter
SDOConverter
114
SensorMLConverter
116
CAP1.1Converter
118

FIG. 6

200

```
Data_mapper NC4ToSensorML {

Source “info.event”,                    Destination "Event
Source "info.description,          Destination "Description
Source "msgType",                       Destination "AlertLevel"
Source "sender,                    Destination "Sender”
Source "info.area.geocode.9”,           Destination "geoLocation.latitude
Source "info.area.geocode.10”,     Destination "geoLocation.longitude”
}
```

FIG. 7

300

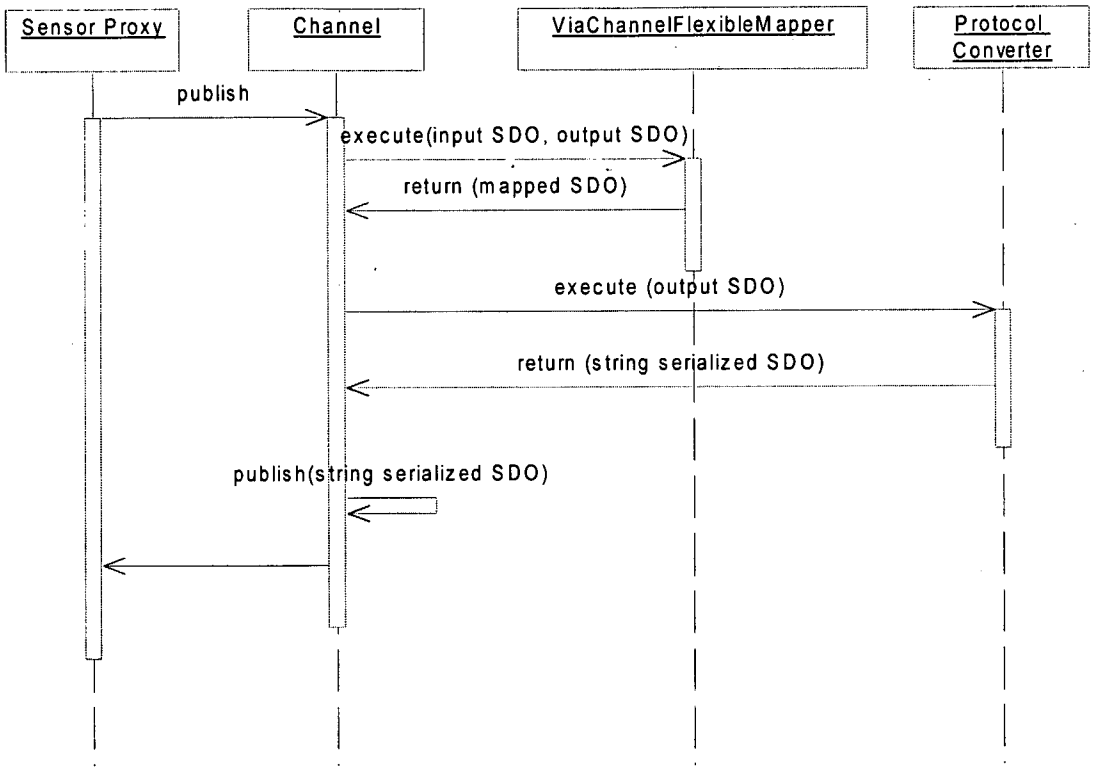

Sensor Proxy
Channel
ViaChannelFlexibleMapper
Protocol Converter
publish
execute(input SDO, output SDO)
return (mapped SDO)
execute (output SDO)
return (string serialized SDO)
publish(string serialized SDO)

FIG. 8

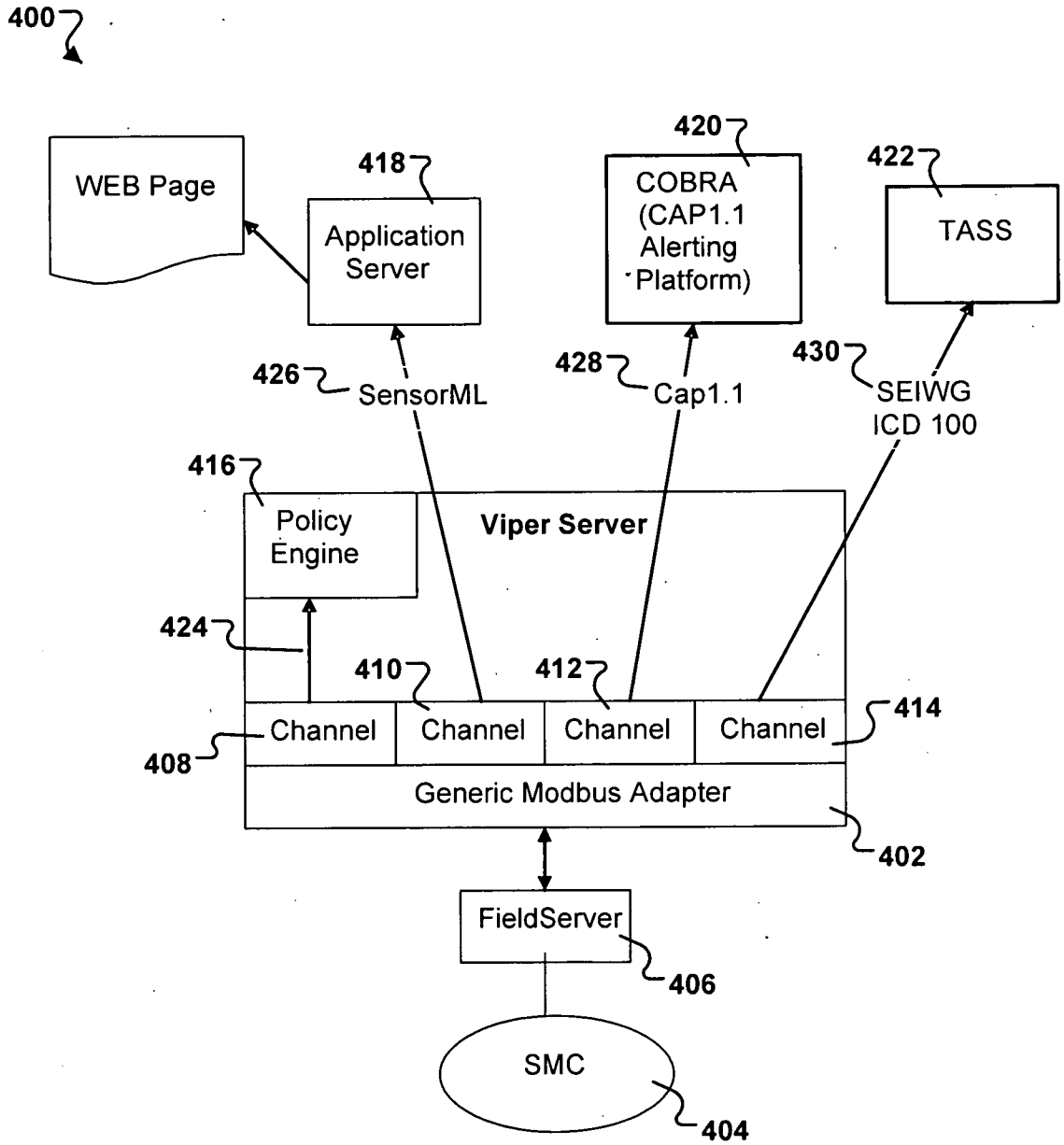

400
WEB Page
418
Application Server
420
COBRA (CAP1.1 Alerting Platform)
422
TASS
426
SensorML
428
Cap1.1
430
SEIWG ICD 100
416
Policy Engine
Viper Server
424
410
412
414
Channel
Channel
Channel
Channel
408
Generic Modbus Adapter
402
FieldServer
406
SMC
404

FIG. 9

500

```
sensor_adapter_instance ModbusSensor {
        template ModbusTemplate
    channels
        data_channels
                    TCPServer::TassChannel
                    TCPServer::CobraChannel
                    JMSServer::SMC.Data.Channel
                    LocalServer::PolicyChannel
    properties
        PollingRate = 10.0
        Enabled = 1
        IPAddress = "64.210.18.100"
        Port = 502
        MBAddress = 11
        ModBusType = "MODBUS_TCP"
        MBFunction = "READ_HOLDING_REGISTERS"
        MBStartAddress = 0
        MBCount = 11
        MBSlave = 11
}
```

FIG. 10

600

610 Property	620 Description
Polling Rate	Rate for sampling data (seconds)
Enabled	Flag for enabling / disabling this sensor adapter.
IPAddress	IP address or hostname of the FieldServer Gateway
Port	Port of FieldServer gateway. 502 is the standard port for Modbus
MBAddress	This is ignored for Modbus TCP. It is required for Modbus RTU.
MBFunction	The name of the Modbus function to be used for the poll
MBStartAddress	Register start address for the poll
MBCount	The number of registers to read.
MBSlave	Modbus slave address.
ModbusType	Valid entries are: MODBUS_TCP and MODBUS_RTU.

FIG. 11

```
type ModbusRegisterData {
        Timestamp               string
        RegisterValues          integer []
        Location                LocationType
        geoLocation             GeoLocationType
}
```

FIG. 12

```
//########## Type maps:
type_map modbus_tcp_to_smc_map {
        RegisterValues[0]          :        Concentration
        RegisterValues[1]          :        Temperature
        RegisterValues[2]          :        HighAlarmRelay
        RegisterValues[3]          :        LowAlarmRelay
        RegisterValues[4]          :        HighAlarmValue
        RegisterValues[5]          :        LowAlarmValue
}
```

FIG. 13

```
channel ModbusChannel
        message_type SMCData
        message_protocol CAP1.1
        message_type_map modbus_tcp_to_smc_map
```

FIG. 14

```
type SMCData {
        Timestamp                string
        Concentration            double
        Temperature              double
        HighAlarmRelay           double
        LowAlarmRelay            double
        HighAlarmValue           double
        LowAlarmValue            double
        Location                 LocationType
        geoLocation              GeoLocationType
}
```

FIG. 15

```
<?xml version = "1.0" encoding = "UTF-8"?>
<SMCData>
    <Timestamp></Timestamp>
    <Concentration>0</Concentration>
    <Temperature>25</Temperature>
    <HighAlarmRelay>0</HighAlarmRelay>
    <LowAlarmRelay>0</LowAlarmRelay>
    <HighAlarmValue>20</HighAlarmValue>
    <LowAlarmValue>60</LowAlarmValue>
    <Location>
        <Location_description></Location_description>
        <Location_street_address></Location_street_address>
        <Location_city></Location_city>
        <Location_state></Location_state>
        <Location_zip_code></Location_zip_code>
        <Location_country></Location_country>
        <Location_building></Location_building>
        <Location_floor></Location_floor>
        <Location_room></Location_room>
    </Location>
    <geoLocation>
        <longitude>0</longitude>
        <latitude>0</latitude>
        <altitude>0</altitude>
    </geoLocation>
</SMCData>
```

FIG. 16

Context Diagram

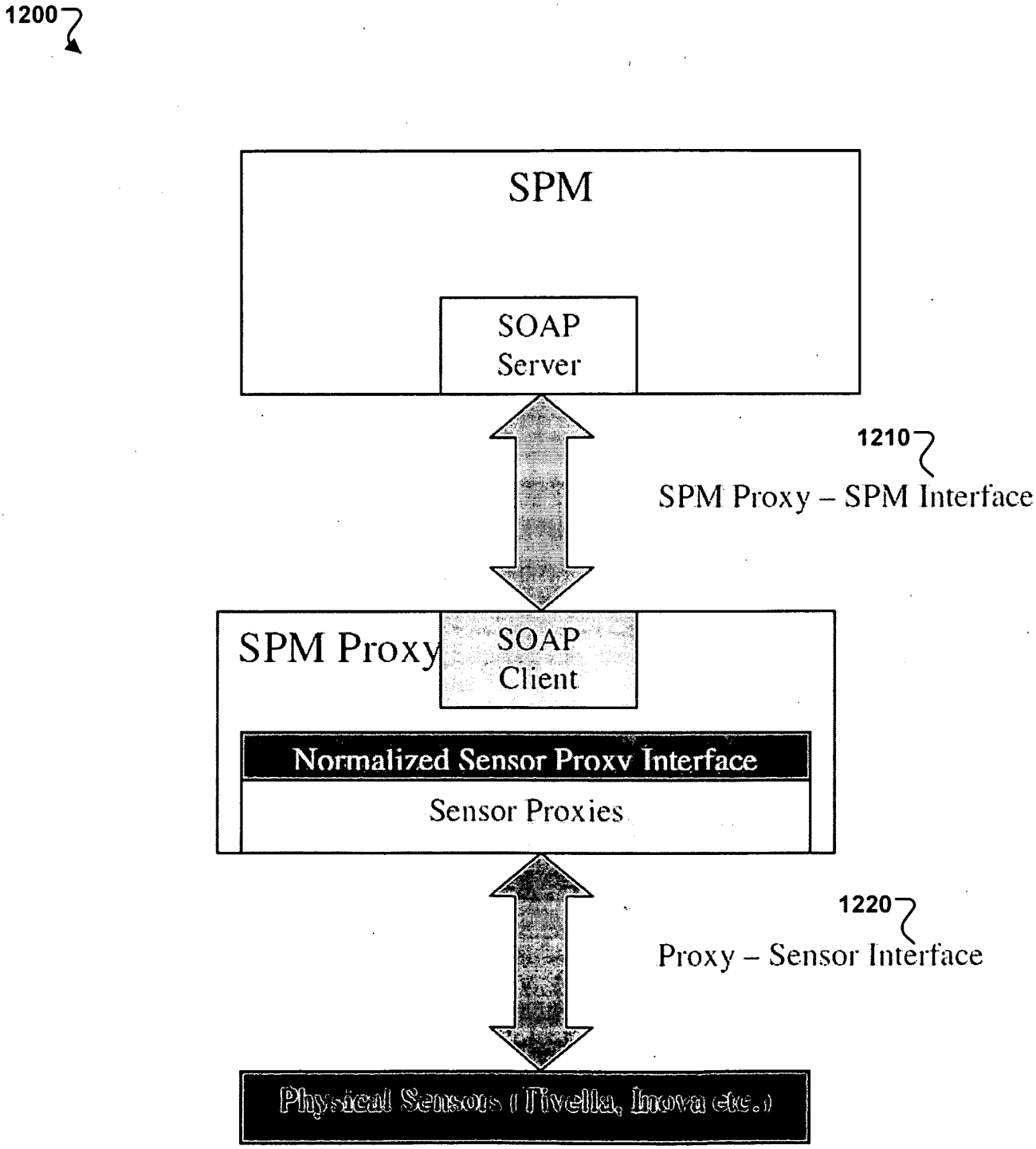


FIG. 17

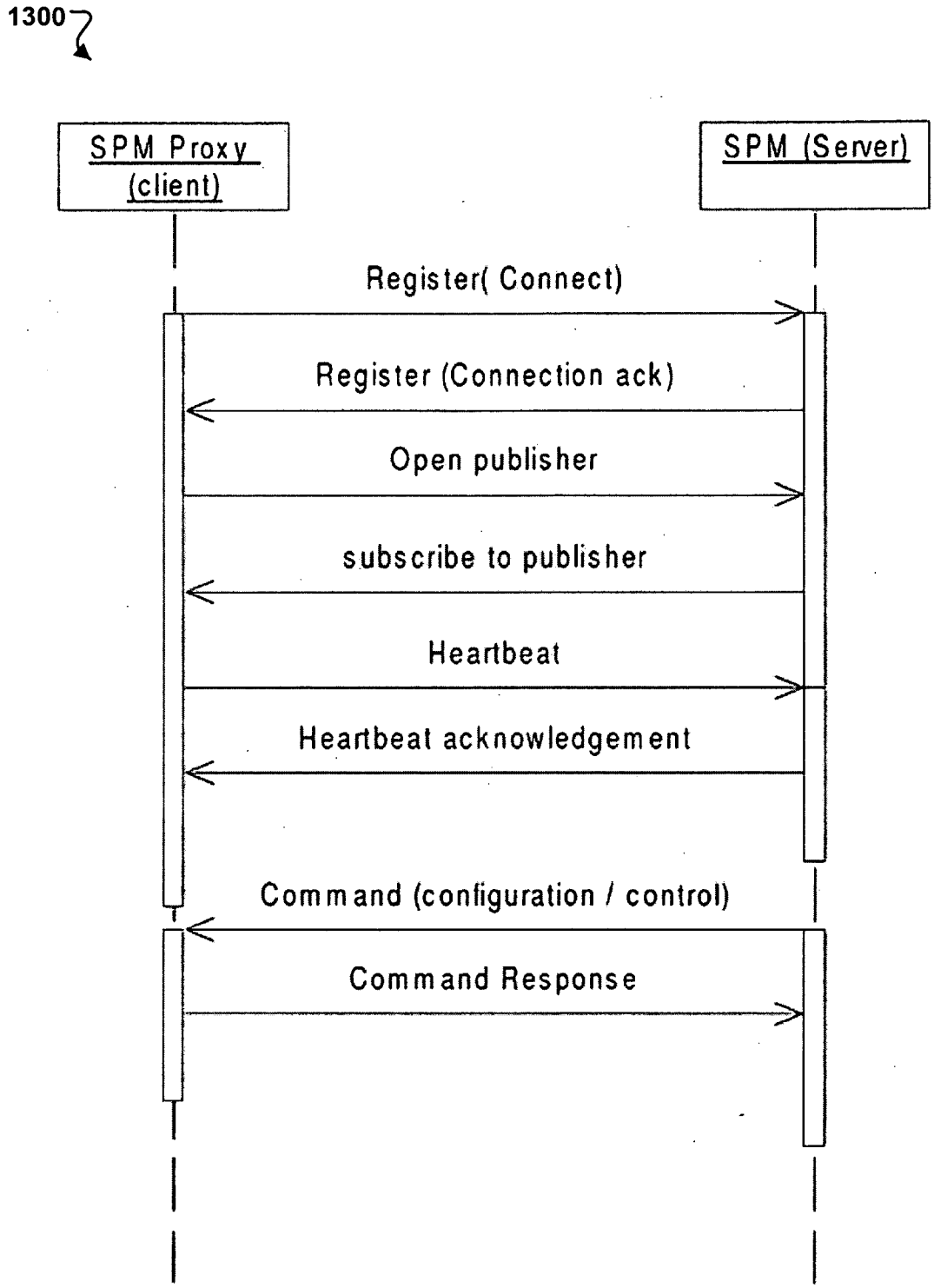

1300
SPM Proxy (client)
SPM (Server)
Register(Connect)
Register (Connection ack)
Open publisher
subscribe to publisher
Heartbeat
Heartbeat acknowledgement
Command (configuration / control)
Command Response

FIG. 18

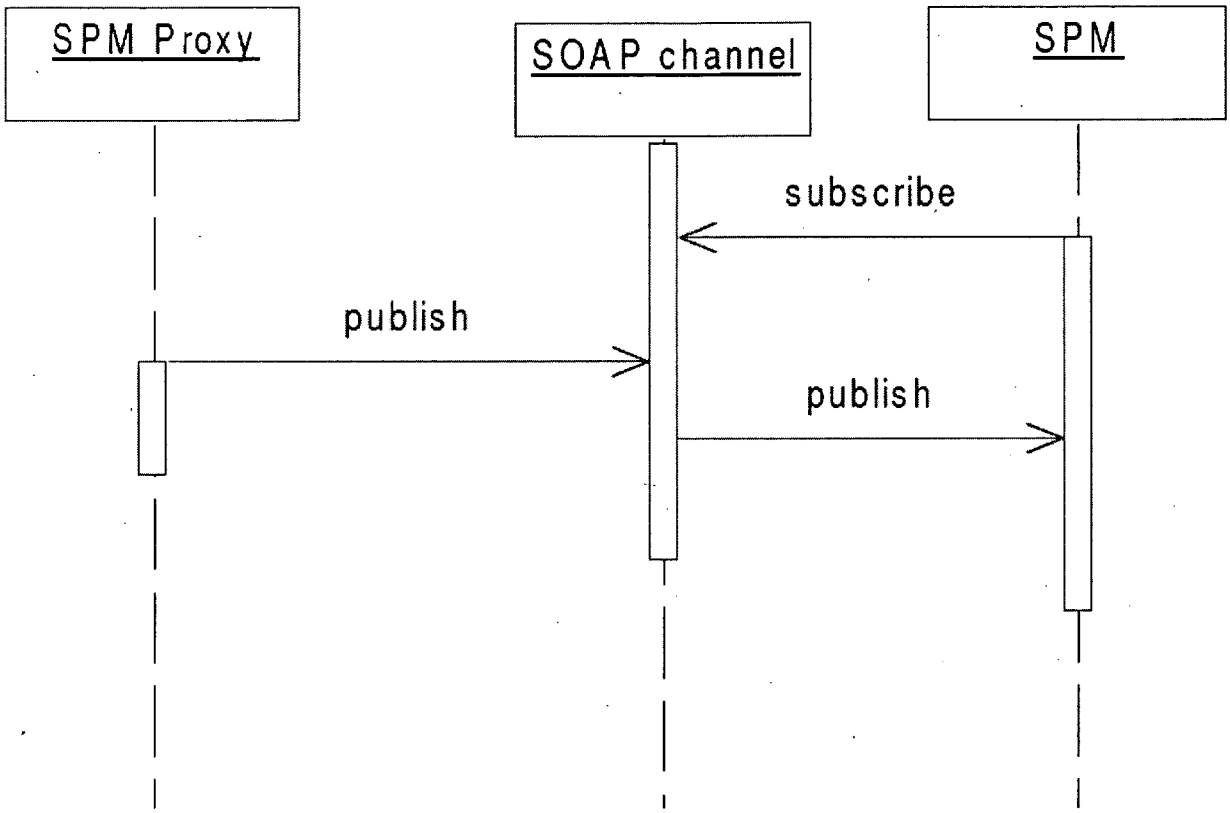


FIG. 19

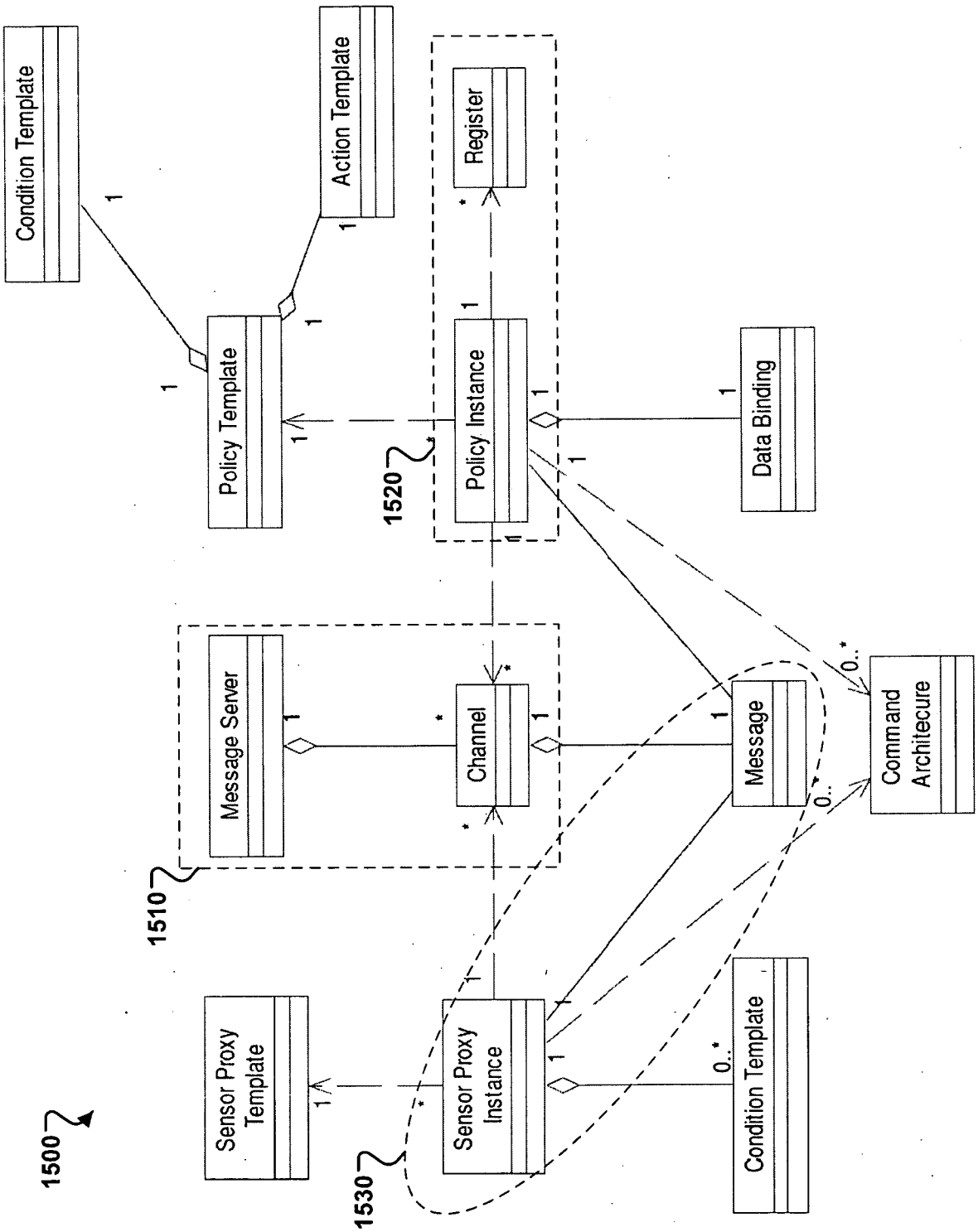

Condition Template
Action Template
Register
Policy Template
Policy Instance
Data Binding
Message Server
Channel
Message
Command Architecure
Sensor Proxy Template
Sensor Proxy Instance
Condition Template
1500
1510
1520
1530
1
*
0..*

FIG. 20

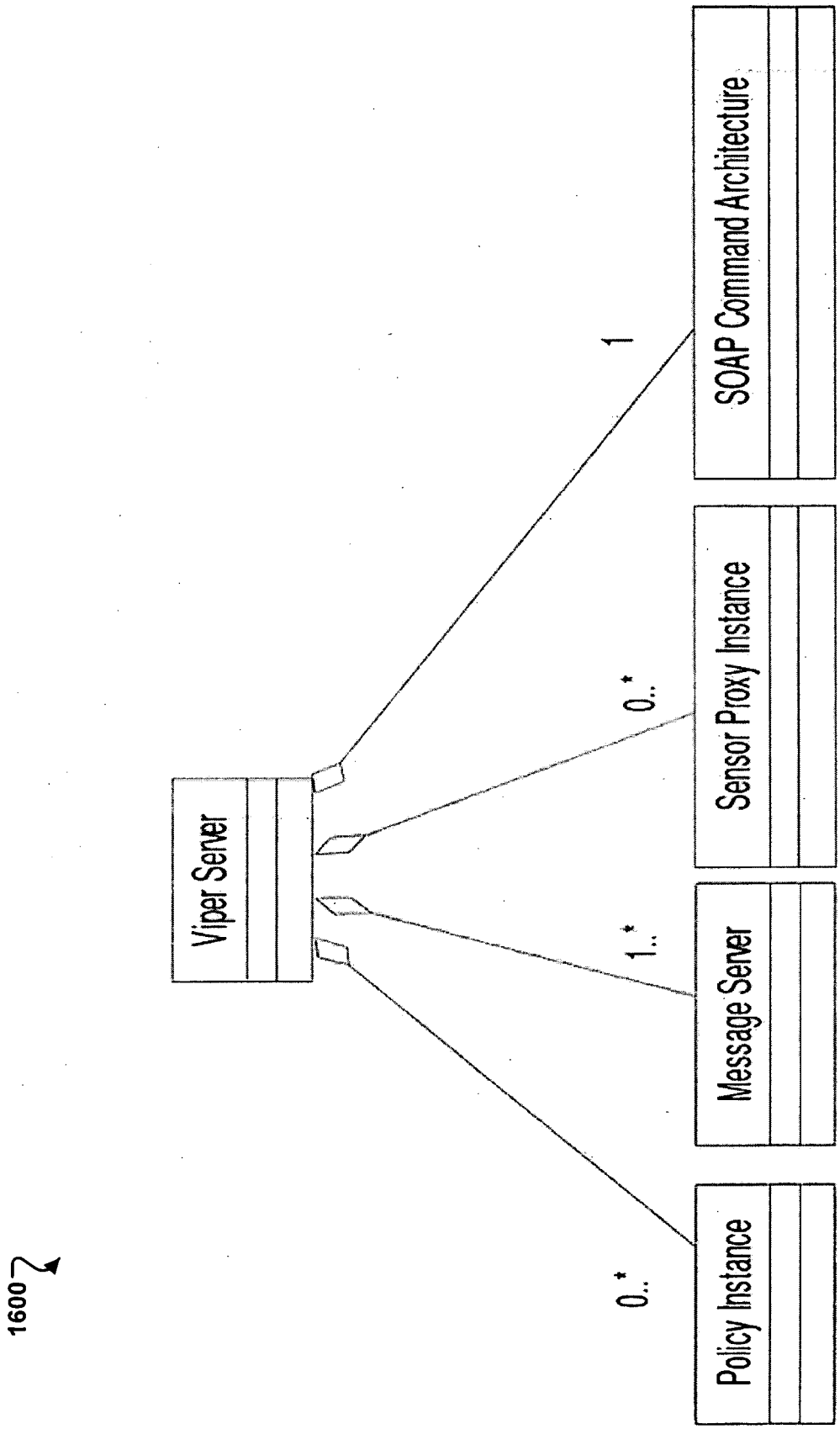

1600
Viper Server
1
SOAP Command Architecture
0..*
Sensor Proxy Instance
1..*
Message Server
0..*
Policy Instance

FIG. 21

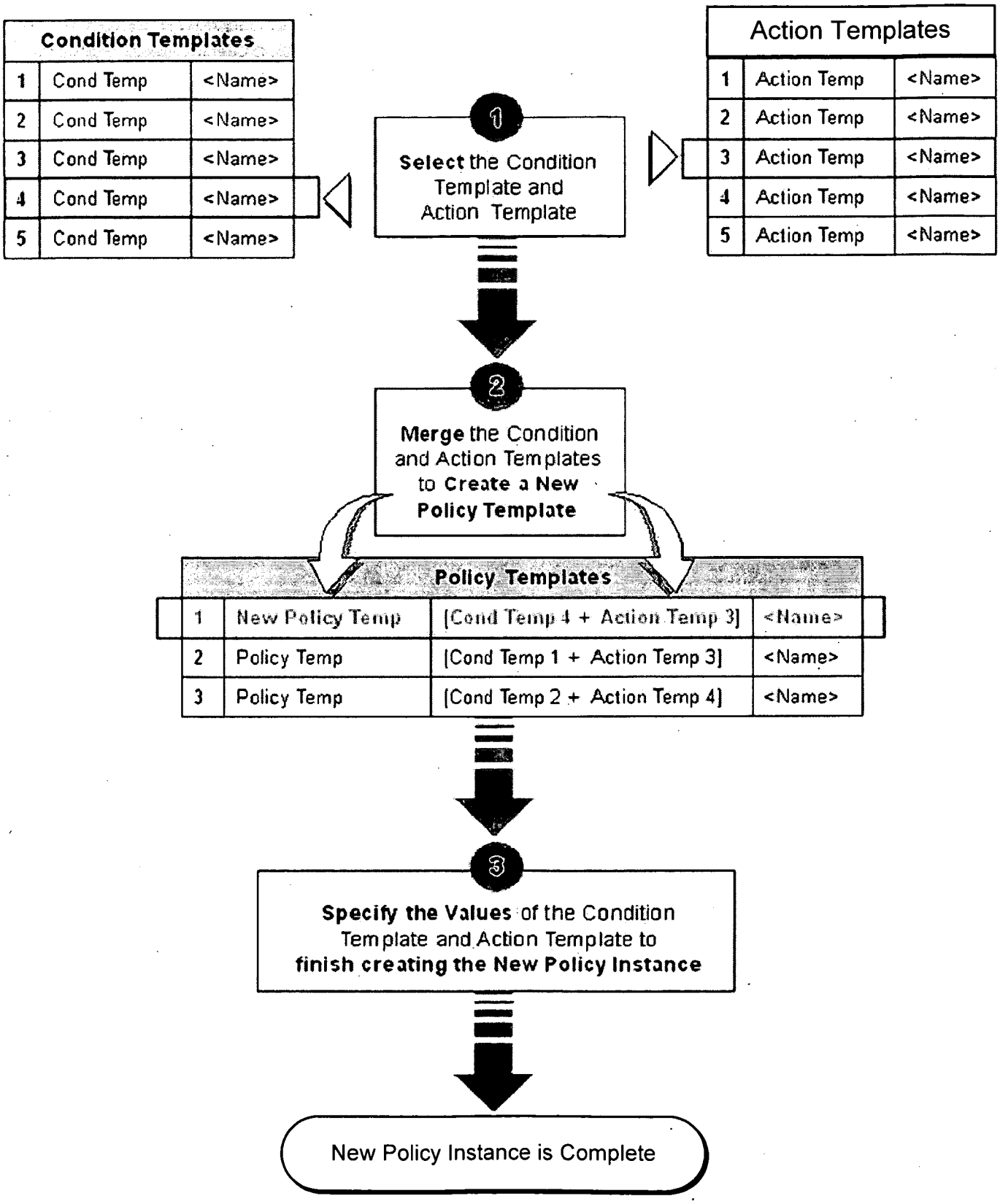

Condition Templates
1 Cond Temp <Name>
2 Cond Temp <Name>
3 Cond Temp <Name>
4 Cond Temp <Name>
5 Cond Temp <Name>
Action Templates
1 Action Temp <Name>
2 Action Temp <Name>
3 Action Temp <Name>
4 Action Temp <Name>
5 Action Temp <Name>
1
Select the Condition Template and Action Template
2
Merge the Condition and Action Templates to Create a New Policy Template
Policy Templates
1 New Policy Temp [Cond Temp 4 + Action Temp 3] <Name>
2 Policy Temp [Cond Temp 1 + Action Temp 3] <Name>
3 Policy Temp [Cond Temp 2 + Action Temp 4] <Name>
3
Specify the Values of the Condition Template and Action Template to finish creating the New Policy Instance
New Policy Instance is Complete

FIG. 22

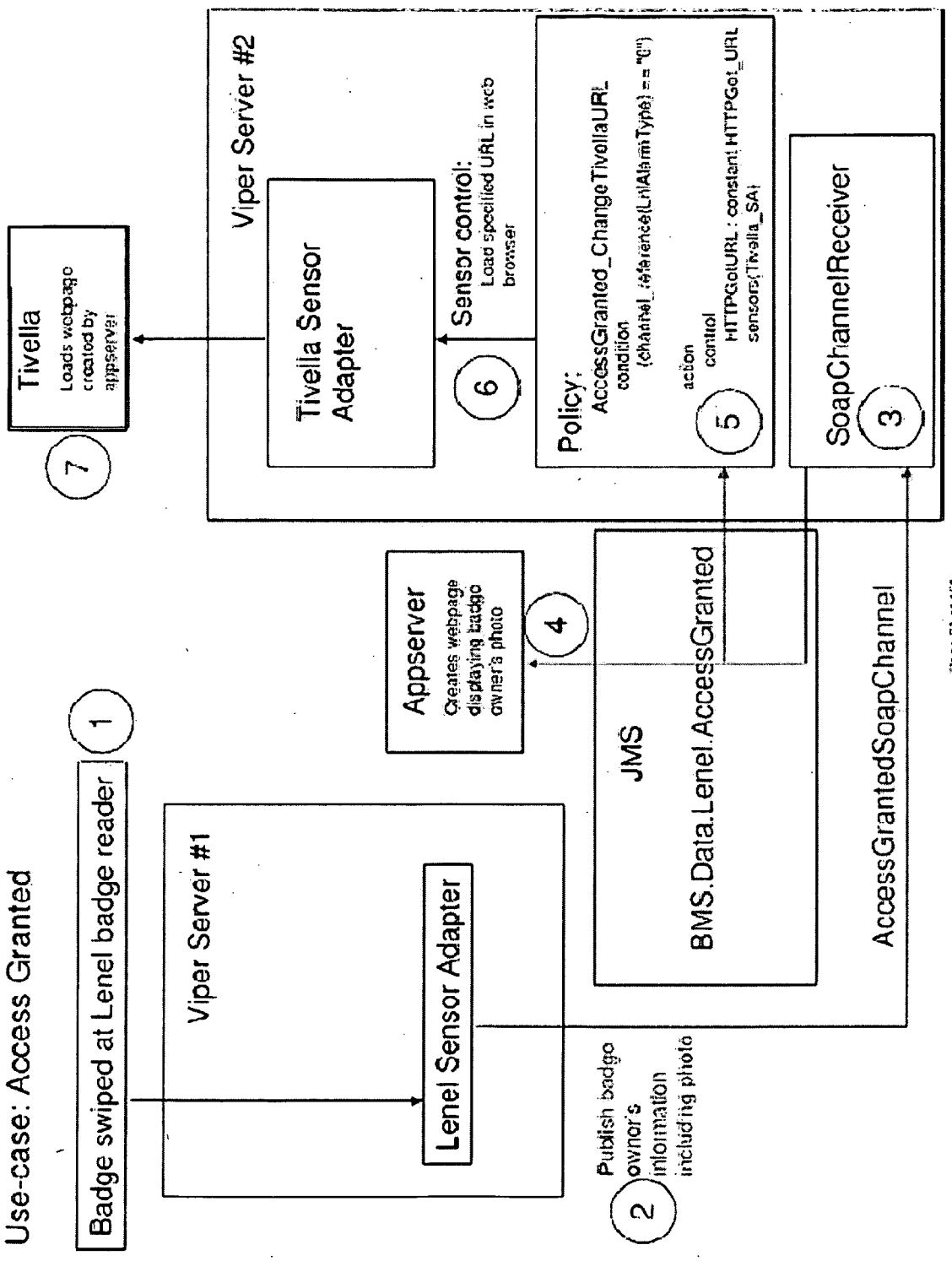

Use-case: Access Granted
Badge swiped at Lenel badge reader
1
Viper Server #1
Lenel Sensor Adapter
2
Publish badge owner's information including photo
JMS
BMS.Data.Lenel.AccessGranted
Appserver
Creates webpage displaying badge owner's photo
4
AccessGrantedSoapChannel
Viper Server #2
Tivella Sensor Adapter
Sensor control:
Load specified URL in web browser
6
Policy:
AccessGranted_ChangeTivellaURL
condition
action
control
HTTPGotURL : constant HTTPGot_URL
sensors/Tivella_SA
5
SoapChannelReceiver
3
Tivella
Loads webpage created by appserver
7

FIG. 23

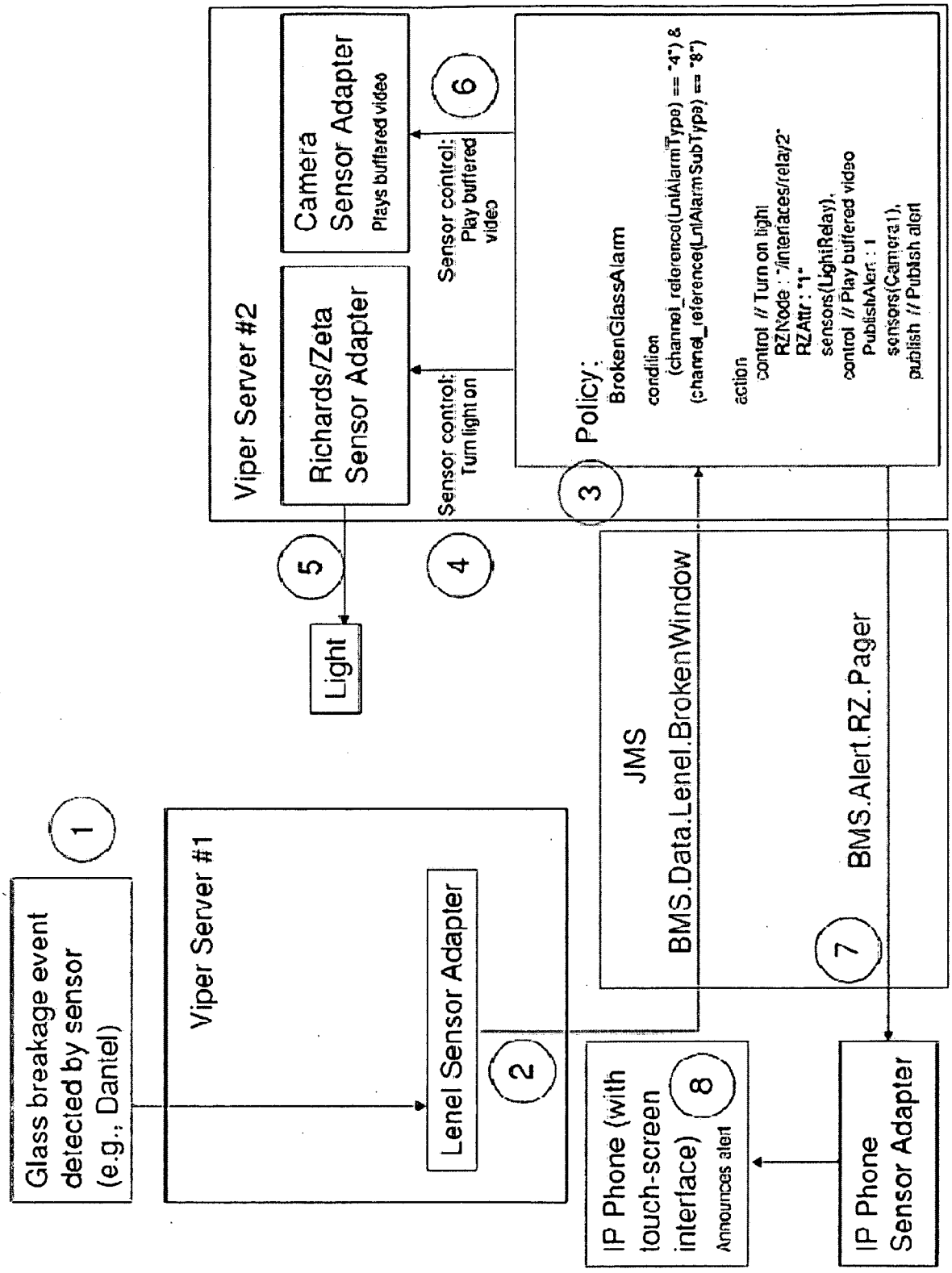

Glass breakage event detected by sensor (e.g., Daniel)
1
Viper Server #1
Lenel Sensor Adapter
2
Viper Server #2
Richards/Zeta Sensor Adapter
Camera Sensor Adapter
Plays buffered video
Sensor control: Turn light on
Sensor control: Play buffered video
6
3
Policy:
BrokenGlassAlarm
condition
(channel_reference(LniAlarmType) == "4") &
(channel_reference(LniAlarmSubType) == "8")
action
control // Turn on light
RZNode : "/interfaces/relay2"
RZAttr : "1"
sensors(LightRelay),
control // Play buffered video
PublishAlert : 1
sensors(Camera1),
publish // Publish alert
4
5
Light
JMS
BMS.Data.Lenel.BrokenWindow
BMS.Alert.RZ.Pager
7
IP Phone (with touch-screen interface)
8
Announces alert
IP Phone Sensor Adapter

FIG. 24

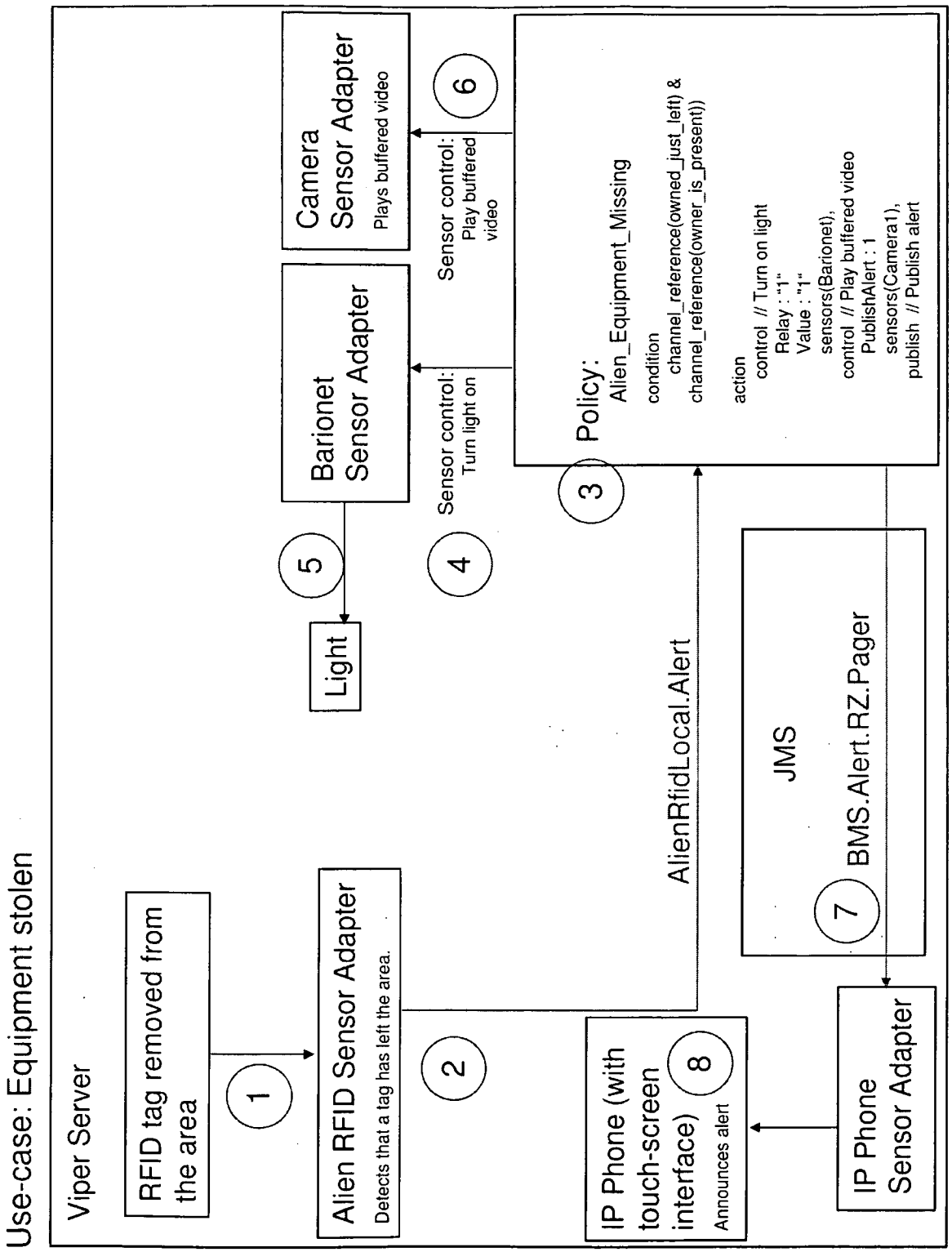

Use-case: Equipment stolen
Viper Server
RFID tag removed from the area
1
Alien RFID Sensor Adapter
Detects that a tag has left the area.
2
IP Phone (with touch-screen interface)
Announces alert
8
IP Phone Sensor Adapter
AlienRfidLocal.Alert
3
Policy:
Alien_Equipment_Missing
condition
channel_reference(owned_just_left) &
channel_reference(owner_is_present))
action
control // Turn on light
Relay : "1"
Value : "1"
sensors(Barionet),
control // Play buffered video
PublishAlert : 1
sensors(Camera1),
publish // Publish alert
4
Sensor control:
Turn light on
5
Light
Barionet Sensor Adapter
6
Sensor control:
Play buffered video
Camera Sensor Adapter
Plays buffered video
JMS
7
BMS.Alert.RZ.Pager

FIG. 25

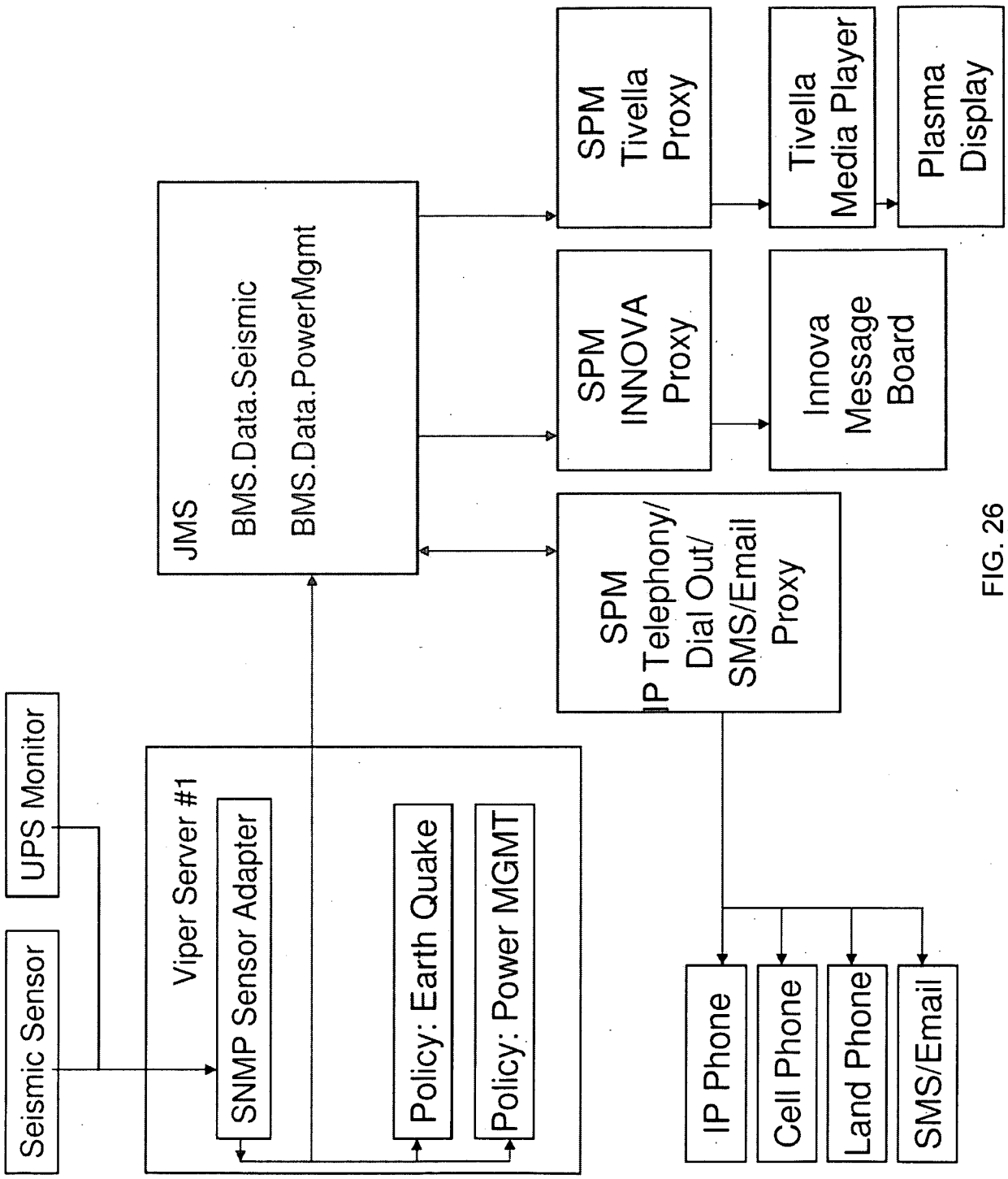

Seismic Sensor
UPS Monitor
Viper Server #1
SNMP Sensor Adapter
Policy: Earth Quake
Policy: Power MGMT
JMS
BMS.Data.Seismic
BMS.Data.PowerMgmt
SPM Tivella Proxy
SPM INNOVA Proxy
SPM IP Telephony/ Dial Out/ SMS/Email Proxy
Tivella Media Player
Plasma Display
Innova Message Board
IP Phone
Cell Phone
Land Phone
SMS/Email

FIG. 26

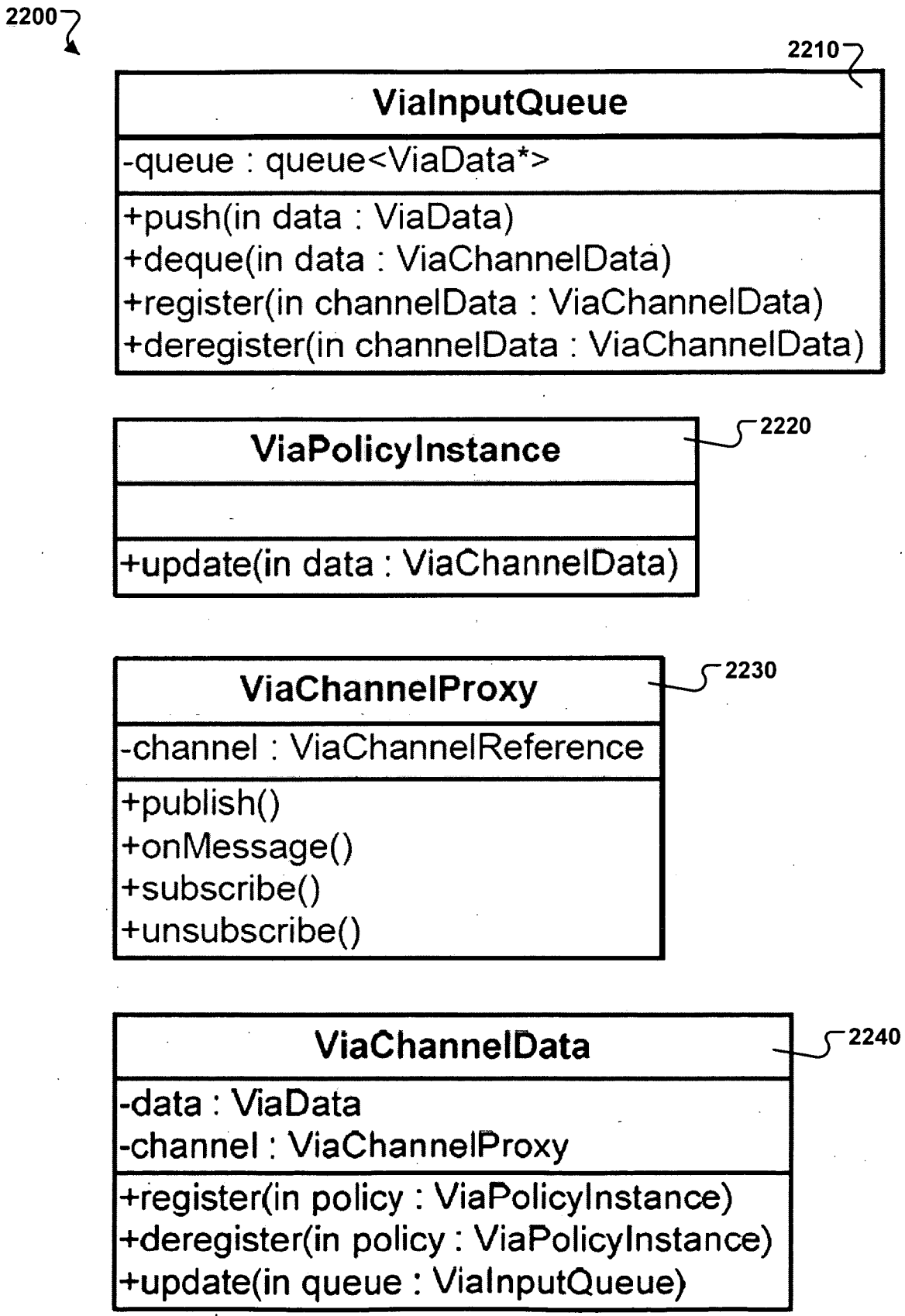

2200
2210
ViaInputQueue
-queue : queue<ViaData*>
+push(in data : ViaData)
+deque(in data : ViaChannelData)
+register(in channelData : ViaChannelData)
+deregister(in channelData : ViaChannelData)
2220
ViaPolicyInstance
+update(in data : ViaChannelData)
2230
ViaChannelProxy
-channel : ViaChannelReference
+publish()
+onMessage()
+subscribe()
+unsubscribe()
2240
ViaChannelData
-data : ViaData
-channel : ViaChannelProxy
+register(in policy : ViaPolicyInstance)
+deregister(in policy : ViaPolicyInstance)
+update(in queue : ViaInputQueue)

FIG. 27

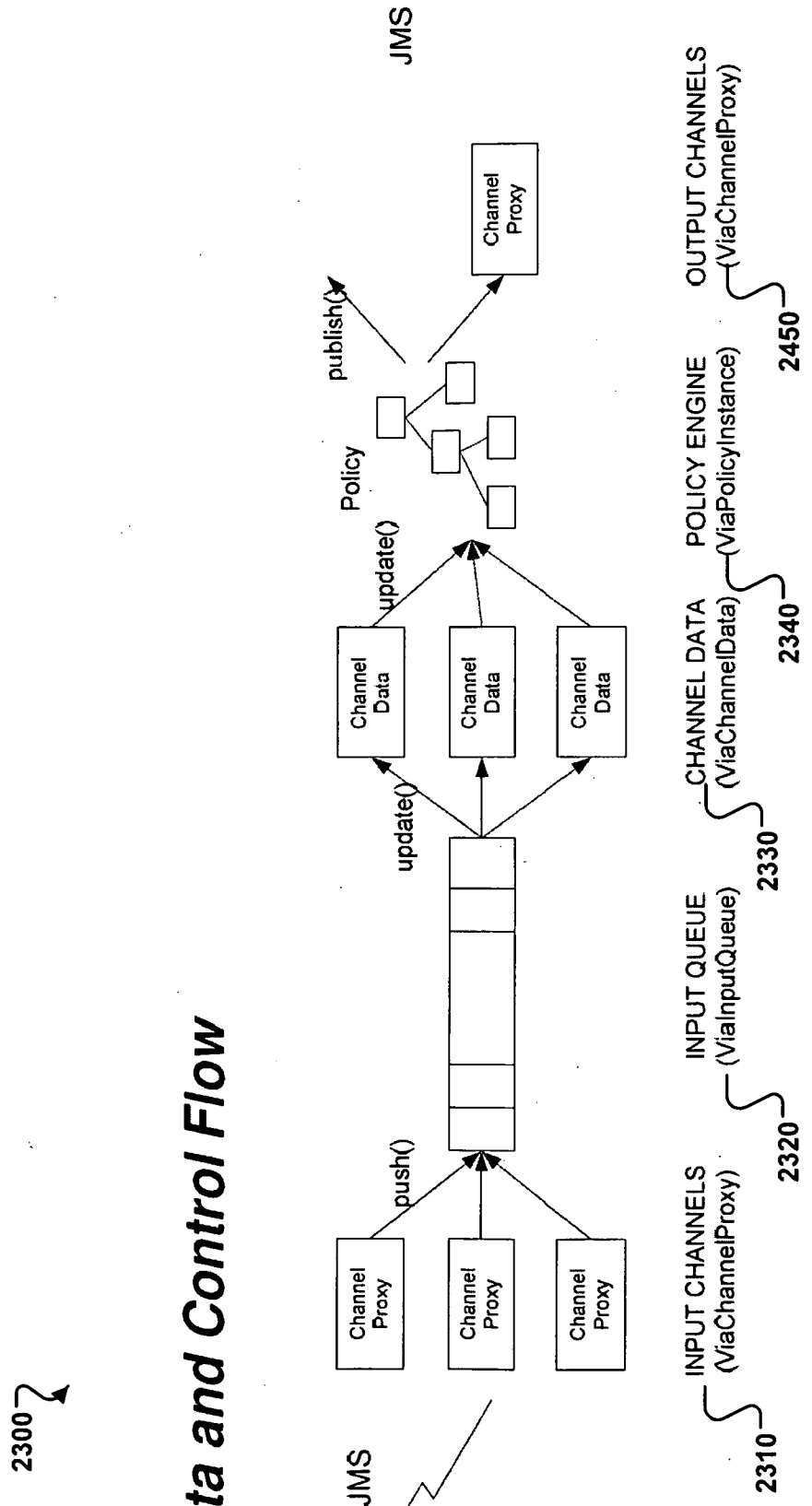

2300
Data and Control Flow
JMS
Channel Proxy
Channel Proxy
Channel Proxy
push()
update()
Channel Data
Channel Data
Channel Data
update()
Policy
publish()
JMS
Channel Proxy
INPUT CHANNELS (ViaChannelProxy)
2310
INPUT QUEUE (ViaInputQueue)
2320
CHANNEL DATA (ViaChannelData)
2330
POLICY ENGINE (ViaPolicyInstance)
2340
OUTPUT CHANNELS (ViaChannelProxy)
2450

FIG. 28

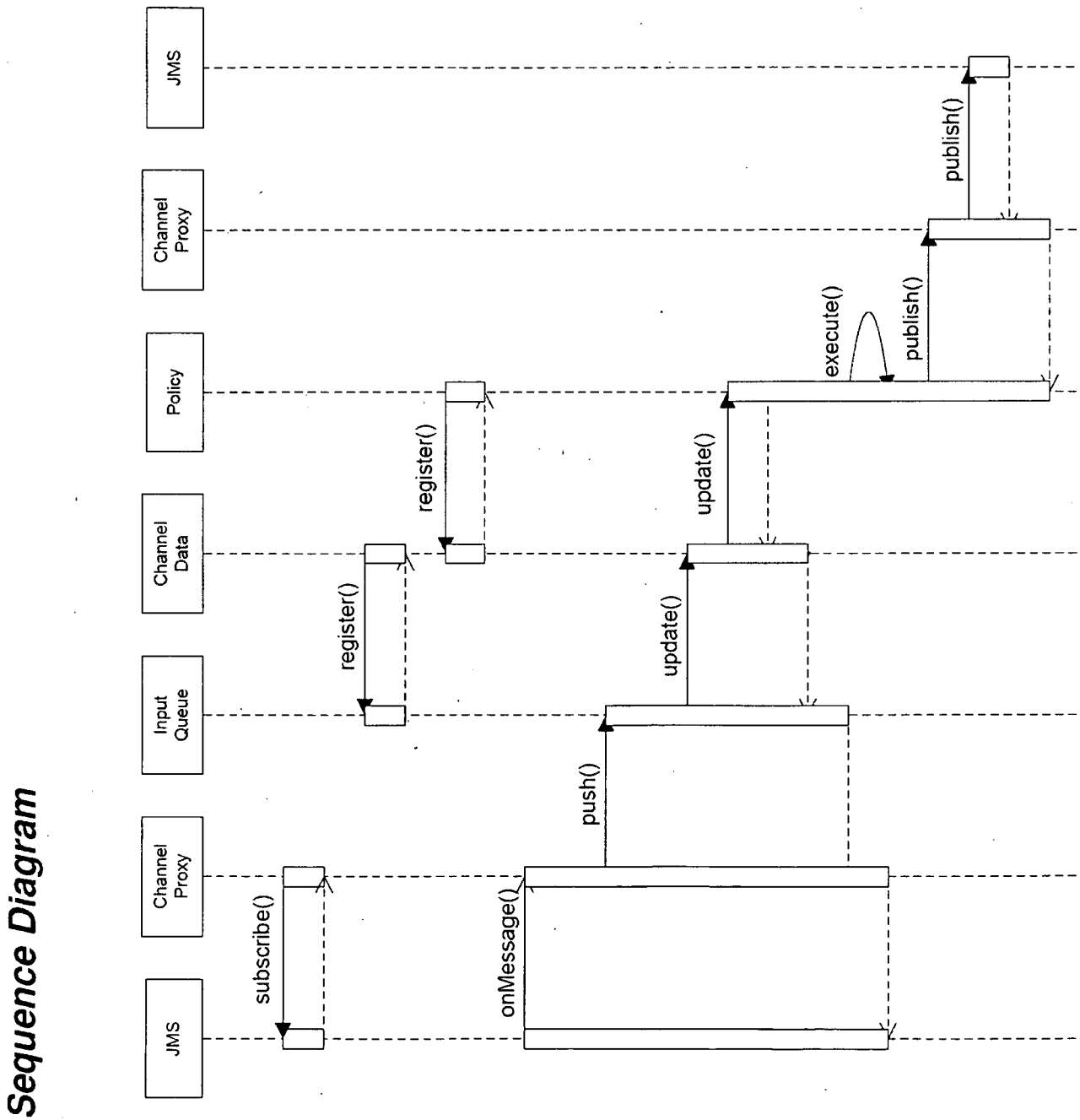

Sequence Diagram
JMS
Channel Proxy
Input Queue
Channel Data
Policy
Channel Proxy
JMS
subscribe()
onMessage()
push()
register()
register()
update()
update()
execute()
publish()
publish()

FIG. 29

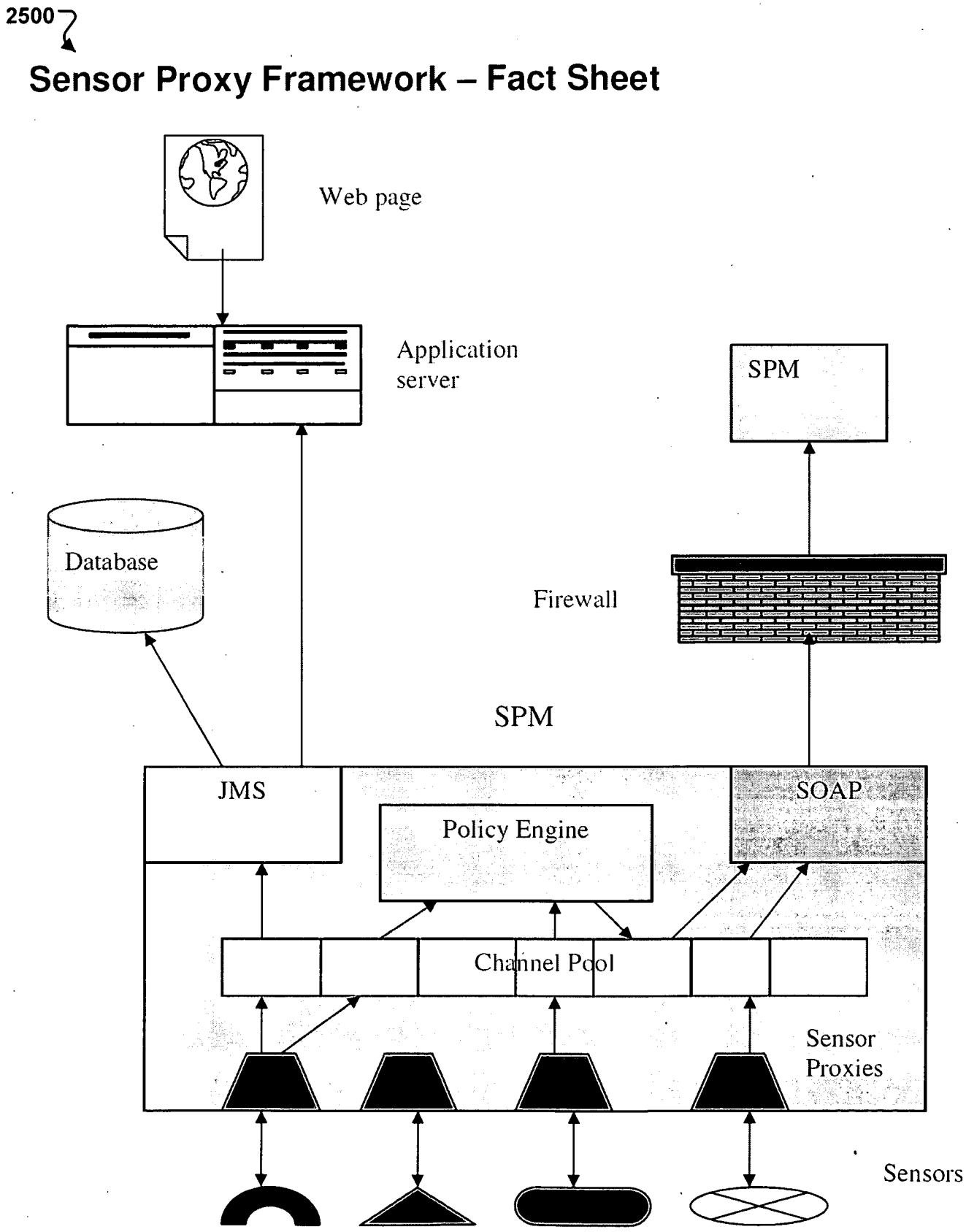

2500
Sensor Proxy Framework – Fact Sheet
Web page
Application server
SPM
Database
Firewall
SPM
JMS
SOAP
Policy Engine
Channel Pool
Sensor Proxies
Sensors

FIG. 30

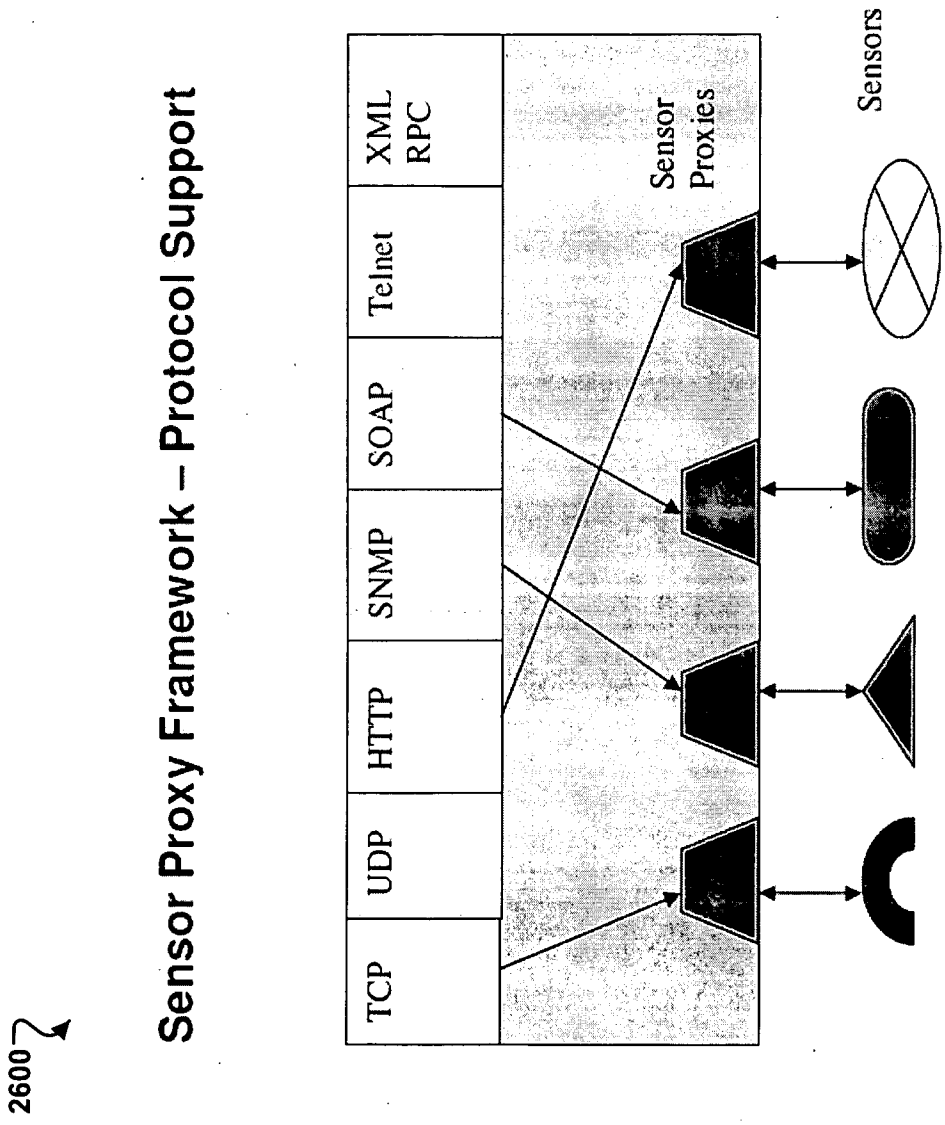

2600
Sensor Proxy Framework – Protocol Support
TCP
UDP
HTTP
SNMP
SOAP
Telnet
XML RPC
Sensor Proxies
Sensors

FIG. 31

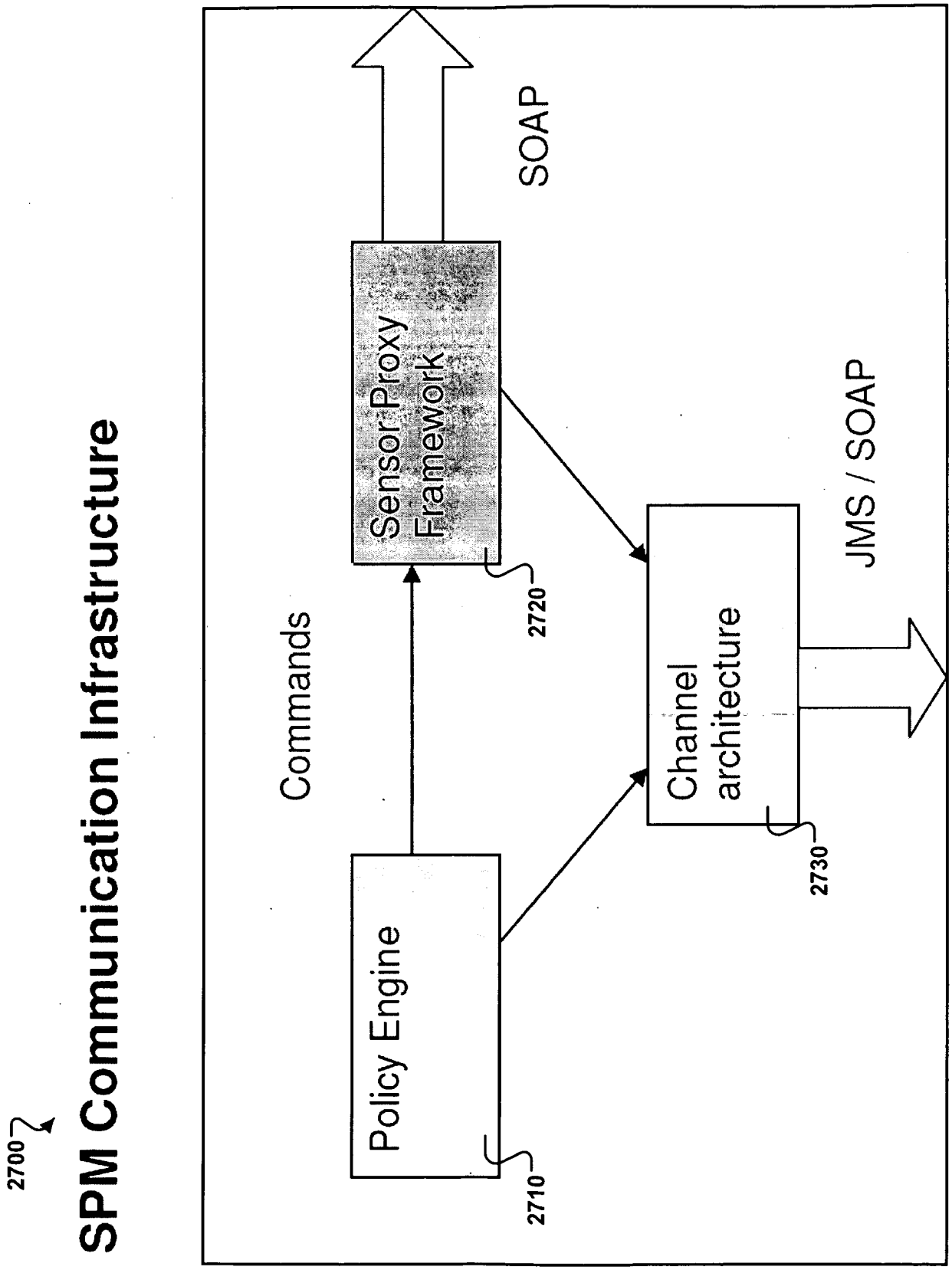

2700
SPM Communication Infrastructure
Policy Engine
2710
Commands
Sensor Proxy Framework
2720
SOAP
Channel architecture
2730
JMS / SOAP

FIG. 32

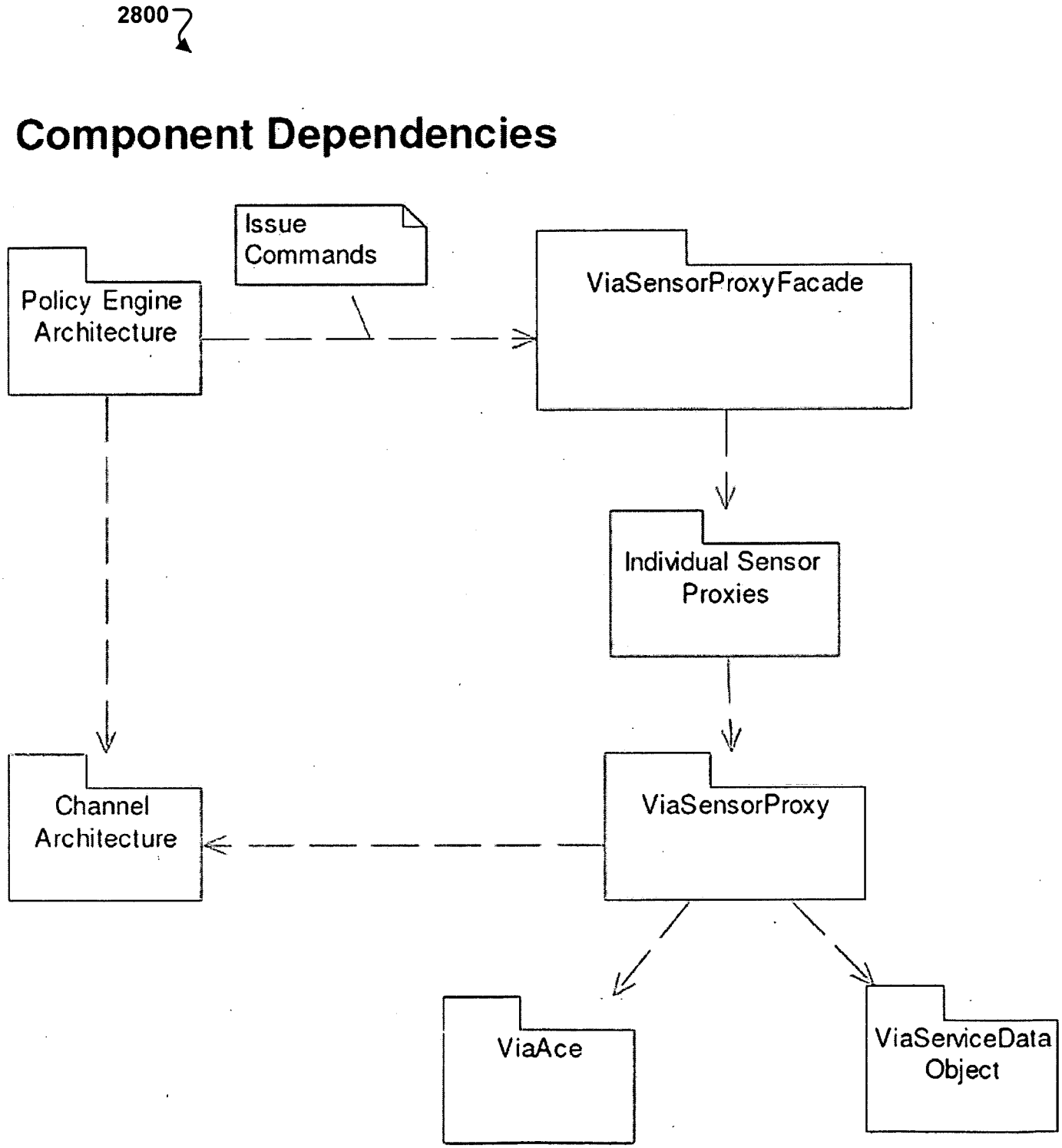

2800
Component Dependencies
Issue Commands
Policy Engine Architecture
ViaSensorProxyFacade
Individual Sensor Proxies
Channel Architecture
ViaSensorProxy
ViaAce
ViaServiceData Object

FIG. 33

2900

Layer name	TCP/IP Mappings. Framework usages	Services
Physical (1)	Physical cabling	Physical medium.
Data-Link(2)	Ethernet	Point-to-point communications on same physical network.
Network(3)	IP	Provide network addressing across subnets – routing.
Transport(4)	TCP (connection-oriented), UDP (connectionless). HTTP is treated as a quasi-transport layer by the sensor proxy framework.	Multiplexing (sockets.) End-to-end reliability (for connection-oriented only)
Application(5)	Standard TCP application protocols such as SNMP, Telnet. SOAP. De-facto standards such as TCP-ModBus. XML-RPC. Various proprietary protocols.	Carries the data encapsulated within own protocol.

FIG. 34

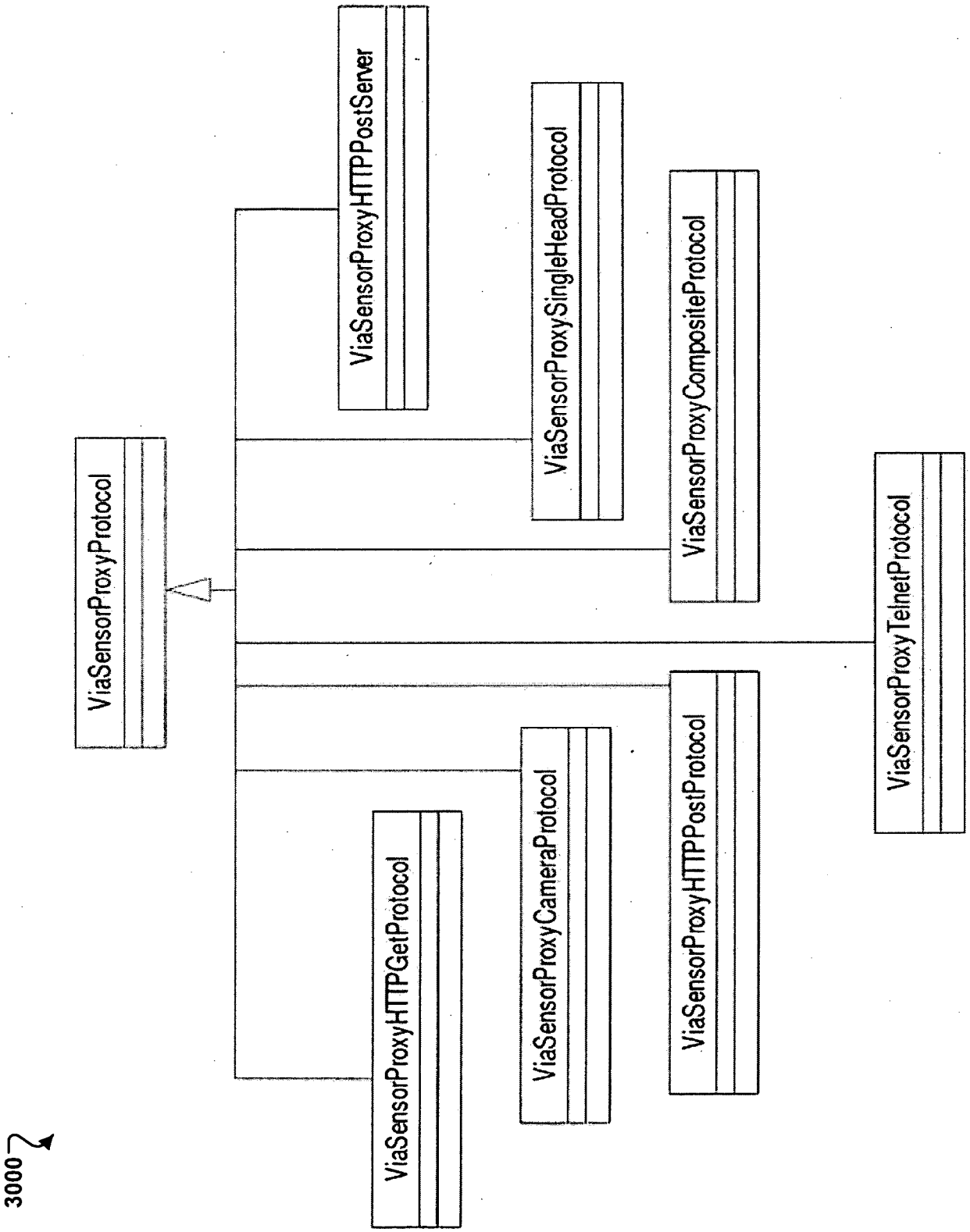

3000
ViaSensorProxyProtocol
ViaSensorProxyHTTPPostServer
ViaSensorProxySingleHeadProtocol
ViaSensorProxyCompositeProtocol
ViaSensorProxyTelnetProtocol
ViaSensorProxyHTTPGetProtocol
ViaSensorProxyCameraProtocol
ViaSensorProxyHTTPPostProtocol

FIG. 35

3100

Transport Protocol	Base Class
TCP Client	ViaSensorProxyProtocol
TCP Server	ViaSensorProxyProtocol
UDP	ViaSensorProxyProtocol
Simulation only (no real protocol)	ViaSensorProxyProtocol
HTTP Post (client)	ViaSensorProxyHTTPPostProtocol
HTTP Post (server)	ViaSensorProxyHTTPPostServer
HTTP Get (client)	ViaSensorProxyHTTPGetProtocol
HTTP Get – for camera images	ViaSensorProxyCameraProtocol
Sensor Group	ViaSensorProxyCompoundProtocol
Sensor Array	ViaSensorProxySingleHeadProtocol
SNMP Manager (Trap receiver.)	ViaSensorProxySingleHeadProtocol

FIG. 36

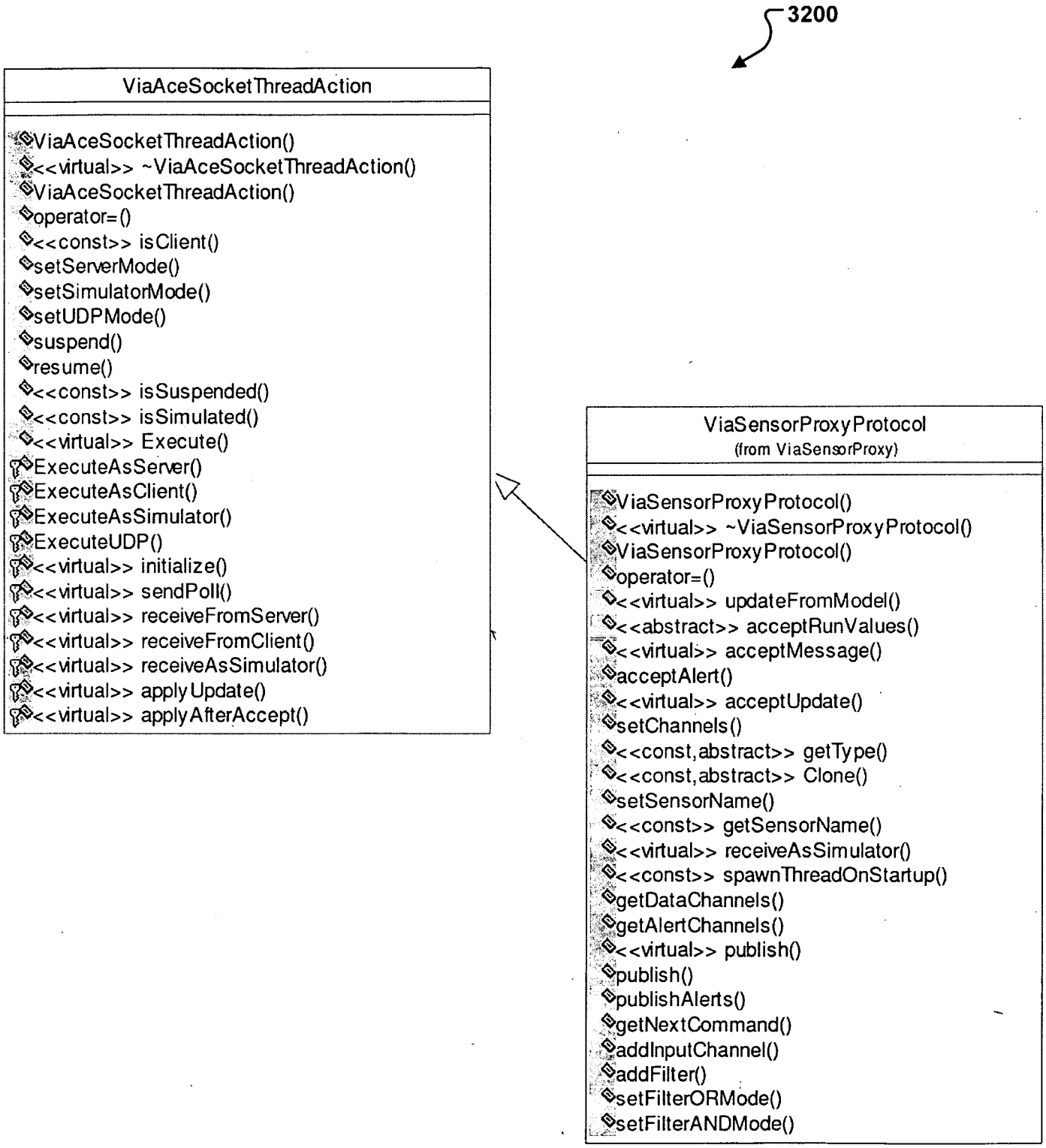

3200
ViaAceSocketThreadAction
ViaAceSocketThreadAction()
<<virtual>> ~ViaAceSocketThreadAction()
ViaAceSocketThreadAction()
operator=()
<<const>> isClient()
setServerMode()
setSimulatorMode()
setUDPMode()
suspend()
resume()
<<const>> isSuspended()
<<const>> isSimulated()
<<virtual>> Execute()
ExecuteAsServer()
ExecuteAsClient()
ExecuteAsSimulator()
ExecuteUDP()
<<virtual>> initialize()
<<virtual>> sendPoll()
<<virtual>> receiveFromServer()
<<virtual>> receiveFromClient()
<<virtual>> receiveAsSimulator()
<<virtual>> applyUpdate()
<<virtual>> applyAfterAccept()
ViaSensorProxyProtocol
(from ViaSensorProxy)
ViaSensorProxyProtocol()
<<virtual>> ~ViaSensorProxyProtocol()
ViaSensorProxyProtocol()
operator=()
<<virtual>> updateFromModel()
<<abstract>> acceptRunValues()
<<virtual>> acceptMessage()
acceptAlert()
<<virtual>> acceptUpdate()
setChannels()
<<const,abstract>> getType()
<<const,abstract>> Clone()
setSensorName()
<<const>> getSensorName()
<<virtual>> receiveAsSimulator()
<<const>> spawnThreadOnStartup()
getDataChannels()
getAlertChannels()
<<virtual>> publish()
publish()
publishAlerts()
getNextCommand()
addInputChannel()
addFilter()
setFilterORMode()
setFilterANDMode()

FIG. 37

3300

Method name	Usage:
ViaAceSocketThreadAction Methods	
virtual Execute()	Provides main loops for TCP client, TCP server, UDP and simulation mode. It is overridden where the protocol is not one of these. The subclasses of ViaSensorProxyProtocol override this (for example HTTP has its own main loop.) The sensor proxy writer may override this if there is none of the existing subclasses provides a suitable implementation.
setServerMode()	Called in the sensor proxy constructor if the sensor proxy is a TCP server.
setUDPMode()	Called in the sensor proxy constructor if the underlying protocol is UDP.
setSimulatorMode()	This is invoked from the configuration file automatically.
suspend()	Suspend operation of the sensor proxy. Can be called by configuration option or as the result of receiving a command at the sensor proxy.
resume()	Resume operation of the sensor proxy. Can be called as the result of receiving a command at the sensor proxy.
virtual initialize()	Default implementation is to do nothing. Overridden if any additional initialization is required prior to entering the main loop.
virtual sendPoll()	Default implementation is to do nothing. Overridden if the sensor is passive and has to be pooled in order to send data.
virtual receiveFromServer()	Overridden for TCP Clients that have to process received data from a sensor – i.e Almost all of them. The default implementation is to do nothing.
Virtual receiveFromClient()	Similar to receiveFromServer () but used by TCP server sensor proxies.
Virtual receiveAsSimulator()	Invoked once every poll period in simulation mode. The default behavior for simulation mode is to publish on the data channel once every poll period. (The data from the configuration file is used.)
Virtual applyUpdate()	Add a command to the input queue of commands for this sensor proxy.
Virtual applyAfterAccept()	Used by TCP Server sensor proxies. It is automatically invoked after the server has accepted a connection from a remote client.

FIG. 38

3400

ViaSensorProxyProtocol Methods	
Virtual updateFromModel()	Observer pattern notify method. It is invoked automatically when the model object notifies its observers. Used by Single head sensor proxies and proxies that have input channels (usually compound sensor proxies.)
Virtual acceptRunValues()	This must be overridden for all sensor proxies. The sensor proxy is provided with its configuration Service Data Object (SDO) and this method is used to extract the valid information. It is invoked from within the framework.
acceptMessage()	Load message SDO. Invoked from the framework.
acceptAlert()	Load alert message SDO. Invoked from the framework.
Virtual acceptUpdate()	This is used to check if our sensor proxy has received a command and action it if so. It must be overridden.
setChannels()	Set the channels (JMS, SOAP and local) for this sensor proxy. Invoked from the framework.
Virtual getType()=0;	Return a type for the sensor proxy. Must be overridden. A new type enumeration must be added to ViaProxyEnums in the proxy framework.
Virtual Clone()=0;	Virtual constructor (prototype pattern.)
spawnThreadOnStartup()	Check if we run this sensor proxy in a separate thread. This is hard-coded and defaults to true. Single-headed sensor proxies do not run in their own threads. The SOAP Server sensor proxy provides its own thread management. All other proxies run in their own threads.
publish()	Publish on all data channels and alert channels if the data message or alert message have been updated.
publishAlerts()	Publish on all alert channels if the alert message has been updated.
getNextCommand()	Thread-safe method for retrieving the next command
	in the command input queue. Returns NULL if there is no command.
Filter methods	The capability exists to set up a simple AND/OR filter in the configuration file. These methods are invoked directly by the framework.

FIG. 39

3500

Service Data Object Class Model

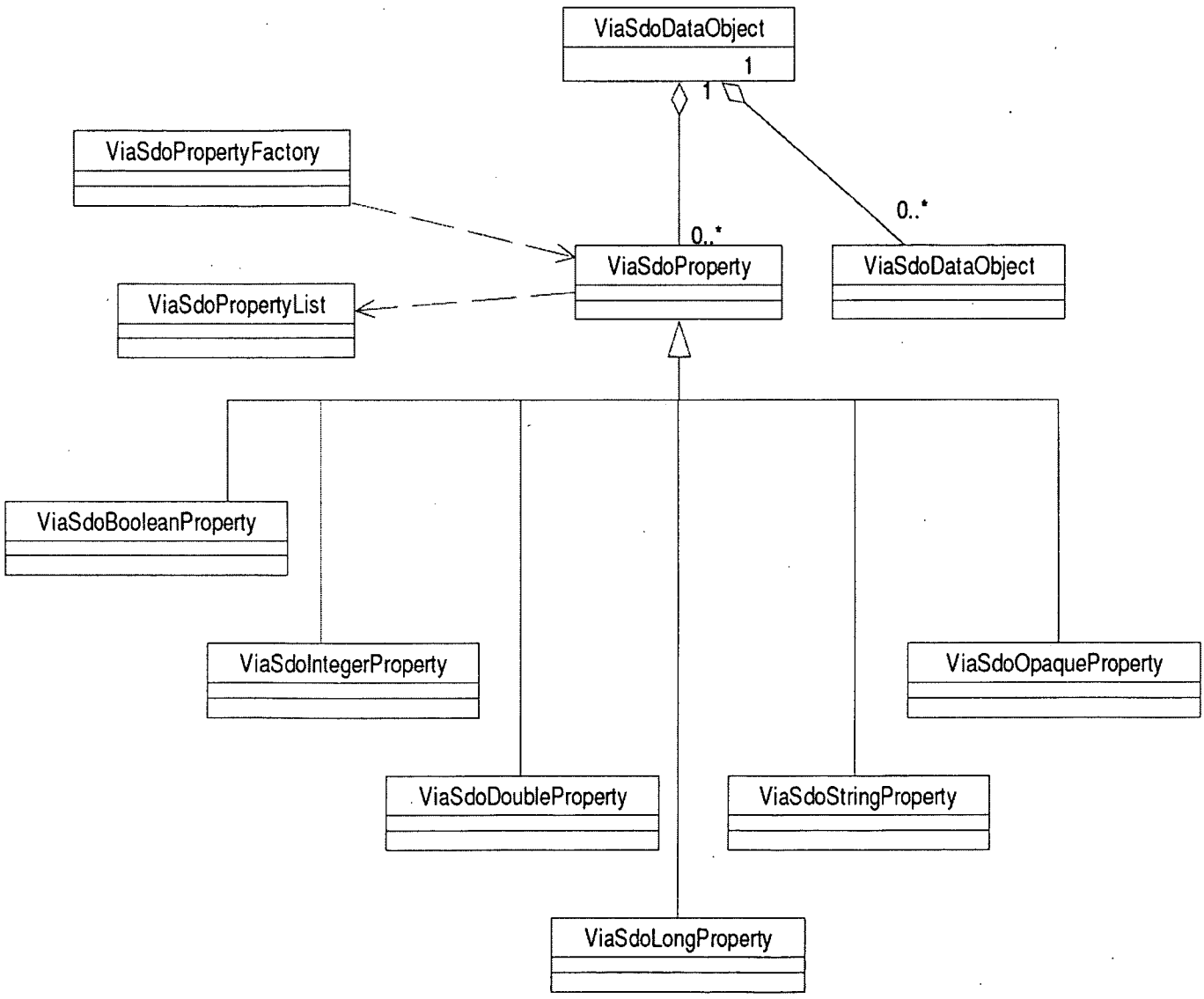


FIG. 40


3600
ViaSdoDataObject
<<const,virtual>> Clone()
<<const>> getNumProperties()
<<const>> getNumDataObjects()
<<const>> getDouble()
<<const>> getInteger()
<<const>> getString()
<<const>> getBoolean()
<<const>> getLong()
<<const>> getDouble()
<<const>> getInteger()
<<const>> getString()
<<const>> getBoolean()
<<const>> getLong()
<<const>> propertySize()
<<const>> findPropertyIndex()
<<const>> findDataObjectIndex()
<<const>> containsProperty()
<<const>> containsDataObject()
<<const>> getVariableType()
<<const>> isScalarType()
<<const>> getVariableIds()
<<const>> propertyIsSet()
<<const>> format()
<<const>> format()
<<const,virtual>> format()
getProperty()
<<const>> getProperty()
getProperty()
<<const>> getProperty()
getProperty()
<<const>> getProperty()
getDataObject()
<<const>> getDataObject()
getDataObject()
<<const>> getDataObject()
<<const>> getDataObjectNames()
<<const>> getSubObjectsSize()
getDataObject()
<<const>> getDataObject()
<<const>> expose()
initVariables()
setDouble()
setInteger()
setString()
setBoolean()
setLong()
parse()
setDouble()
setInteger()
setString()
setBoolean()
setLong()
parse()
addProperty()
addDataObject()
rmDataObject()
clearDataObjects()
clearProperties()
setAllProperties()
unsetAllProperties()
resetId()
containsSetProperties()
clear()
<<virtual>> accept()
3610
ViaSdoProperty
ViaSdoProperty()
ViaSdoProperty()
ViaSdoProperty()
operator=()
<<virtual>> ~ViaSdoProperty()
<<const,virtual>> IsA()
<<const>> getPropertyType()
<<const>> getArraySize()
<<const,virtual>> Clone()
<<const,virtual>> CloneElement()
<<const>> getId()
<<const,virtual>> format()
<<const,virtual>> format()
<<const,virtual>> size()
<<const>> isMany()
<<const>> getIsMandatory()
<<const>> isSet()
getAttributes()
<<const>> getPropertyList()
setId()
<<virtual>> resize()
<<virtual>> parse()
<<virtual>> parse()
<<virtual>> unset()
set()
addAttribute()
rmAttribute()
clearAttributes()
setMany()
unsetMany()
<<const>> operator==()
<<const>> operator!=()
<<const>> operator<()
<<const>> operator>()
<<const>> operator<=()
<<const>> operator>=()
<<virtual>> setBounds()
<<virtual>> unsetBounds()
<<static>> setPrecision()
<<static>> getPrecision()

FIG. 41

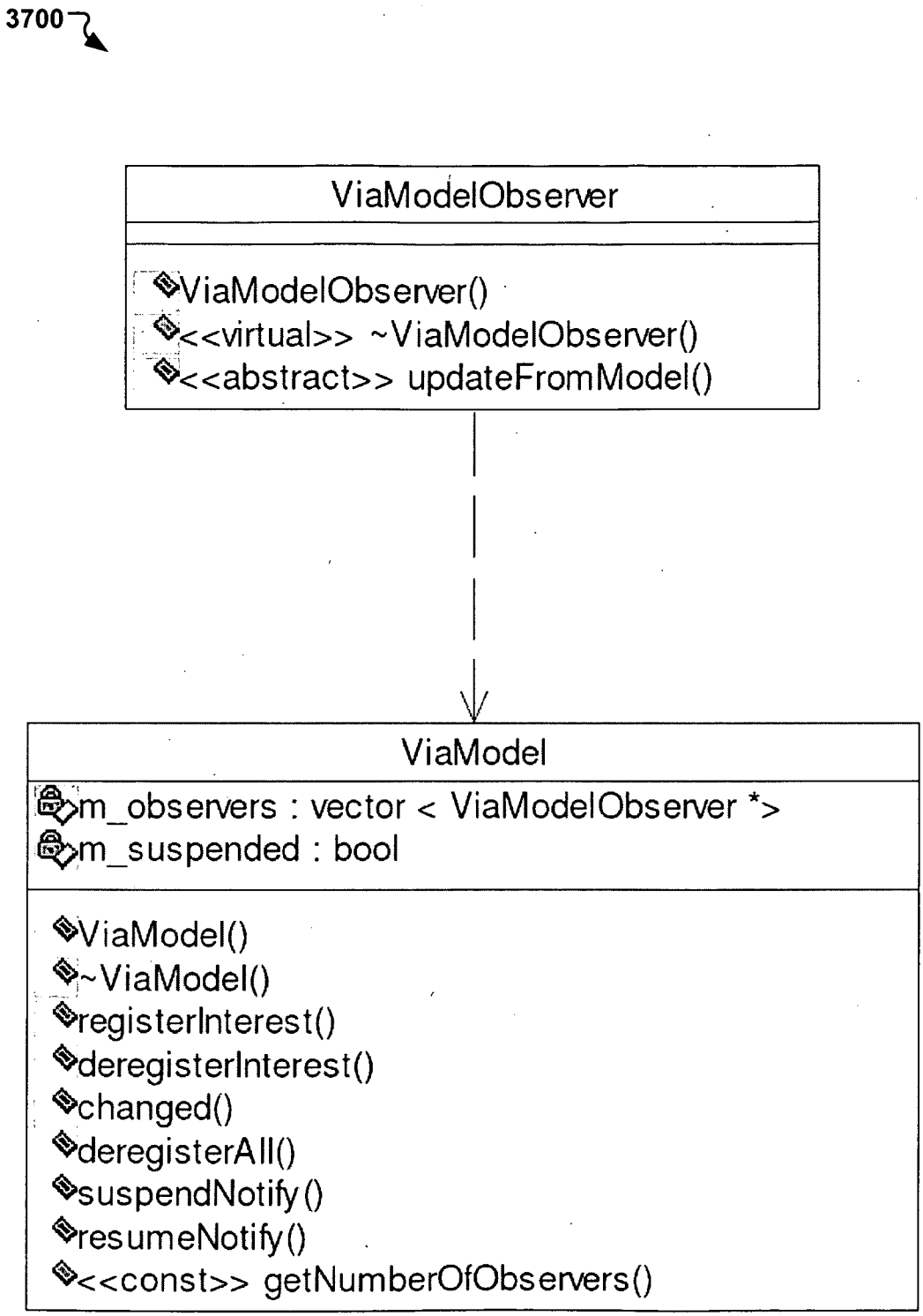

3700
ViaModelObserver
ViaModelObserver()
<<virtual>> ~ViaModelObserver()
<<abstract>> updateFromModel()
ViaModel
m_observers : vector < ViaModelObserver *>
m_suspended : bool
ViaModel()
~ViaModel()
registerInterest()
deregisterInterest()
changed()
deregisterAll()
suspendNotify()
resumeNotify()
<<const>> getNumberOfObservers()

FIG. 42

3800

```
<xs:complexType name="subSectionEntryType">
        <xs:sequence maxOccurs="unbounded">
                <xs:choice>
                        <!--- Properties. -->
                        <xs:element name="localIPAddress" type="xs:string"/>
                        <xs:element name="remoteIPAddress" type="xs:string"/>
                        <xs:element name="localIPPort" type="xs:int"/>
                        <xs:element name="remoteIPPort" type="xs:int"/>
                        <xs:element name="SensorName" type="xs:string"/>
                        ...
                </xs:choice>
        </xs:sequence >
<xs:complexType>
```

FIG. 43

3900

```
<configSection>
        <sectionName>SensorProxyTypes</sectionName>
        <sectionEntry>
                <subSection>
                        <subSectionName>SensorProxyTypes</subSectionName>
                        <subSectionEntry>
                                <SP_Type>ThermoElectron</SP_Type>
                                <SP_Type>ITrans</SP_Type>
                                <SP_Type>ITransChannel</SP_Type>
                                <SP_Type>AreCont</SP_Type>
                        </subSectionEntry>
                </subSection>
        </sectionEntry>
</configSection>
```

FIG. 44

4000

```
<configSection>
        <sectionName>DataTypes</sectionName>
        <sectionEntry>
                <subSection>
                        <subSectionName>GasData</subSectionName>
                        <subSectionEntry>
                        <Timestamp>000000000000</Timestamp>
                        <NameofdetectedGas>Unknown</NameofdetectedGas>
                        ...
                        <Concentration>0.00</Concentration>
                </subSectionEntry>
                </subsection>
        </sectionEntry>
</configEntry>
```

FIG. 45

4100

```
<configSection>
        <sectionName>ITrans</sectionName>
        <sectionEntry>
                <subSection>
                        <subSectionName>iTransSP</subSectionName>
                        <subSectionEntry>
                        <SensorName>ITransSP</SensorName>
                        <remoteIPAddress>64.210.18.21</remoteIPAddress>
                        <remoteIPPort>2101</remoteIPPort>
                        <PollingRate>1</PollingRate>
                        <Enabled>1</Enabled>
                        <DataMessageType>GasData</DataMessageType>
                        <AlertMessageType>NullAlert</AlertMessageType>
                        </subSectionEntry>
                </subSection>
        </sectionEntry>
</configSection>
```

FIG. 46

4200

```
<configSection>
    <sectionName>ITransChannel</sectionName>
    <sectionEntry>
        <subSection>
            <subSectionName>ITransChan0</subSectionName>
            <subSectionEntry>
                <SensorName>ITransChan0</SensorName>
                <FilterAttribute>NameofdetectedGas</FilterAttribute>
                <FilterValue>CO_Carbon_Monoxide</FilterValue>
                <FilterMode>OR</FilterMode>
                <FilterType>2</FilterType>
                <Enabled>1</Enabled>
                <SP_Model>ITransSP</SP_Model>
                <DataMessageType>NullData</DataMessageType>
                <AlertMessageType>NullAlert</AlertMessageType>
                <jmsTopic>ITrans.Gas.CarbonMonoxide</jmsTopic>
            </subSectionEntry>
        </subSection>
        <subSection>
            <subSectionName>ITransChan1</subSectionName>
            <subSectionEntry>
                <SensorName>ITransChan1</SensorName>
                <Enabled>1</Enabled>
                <SP_Model>ITransSP</SP_Model>
                <FilterAttribute>NameofdetectedGas</FilterAttribute>
                <FilterValue>CH4_Methane</FilterValue>
                <FilterMode>OR</FilterMode>
                <FilterType>2</FilterType>
                <DataMessageType>NullData</DataMessageType>
                <AlertMessageType>NullAlert</AlertMessageType>
                <jmsTopic>ITrans.Gas.Flammables</jmsTopic>
            </subSectionEntry>
        </subSection>
    </sectionEntry>
</configSection>
```

FIG. 47

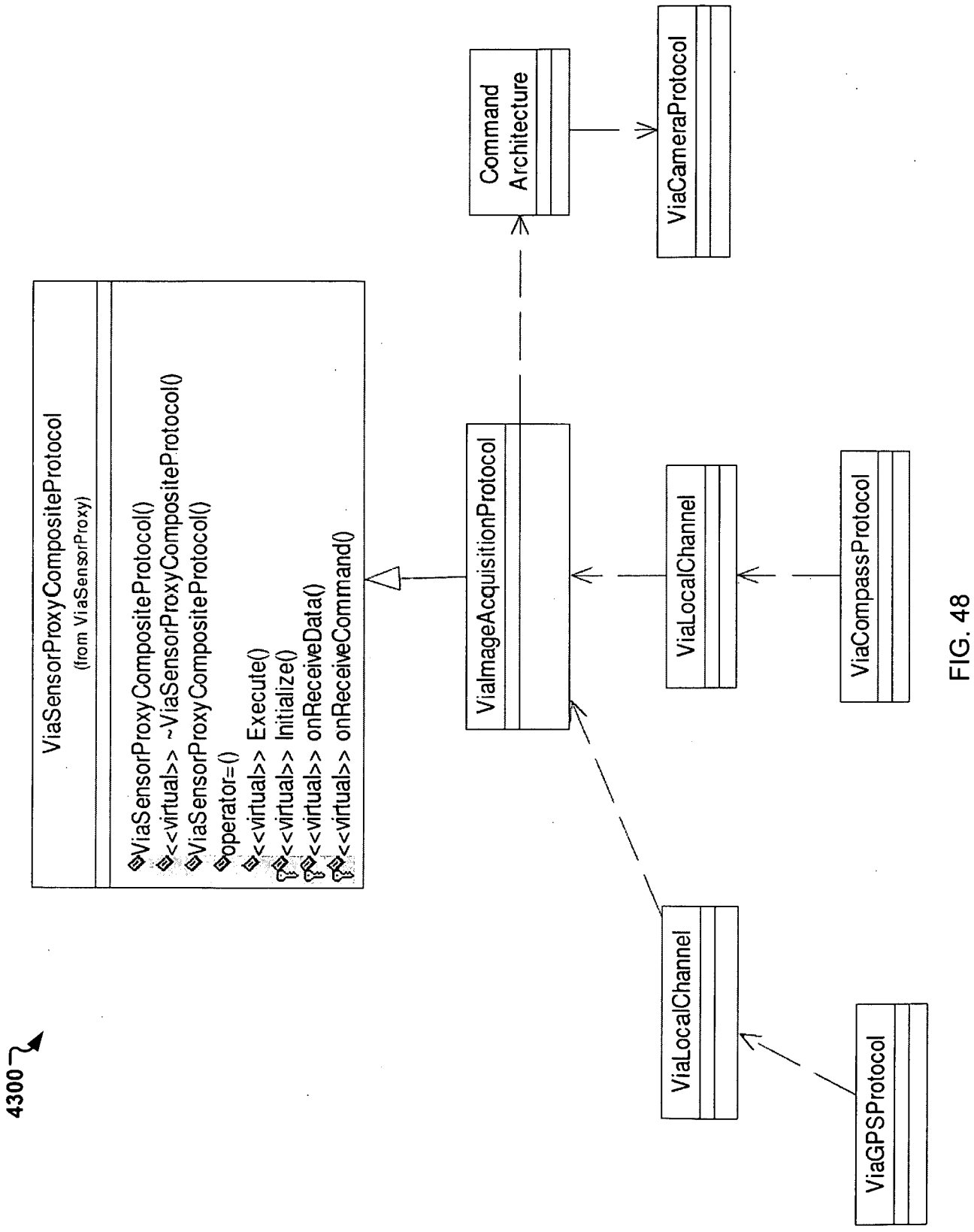
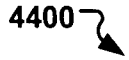

4300
ViaSensorProxyCompositeProtocol
(from ViaSensorProxy)
ViaSensorProxyCompositeProtocol()
<<virtual>> ~ViaSensorProxyCompositeProtocol()
ViaSensorProxyCompositeProtocol()
operator=()
<<virtual>> Execute()
<<virtual>> Initialize()
<<virtual>> onReceiveData()
<<virtual>> onReceiveCommand()
Command Architecture
ViaCameraProtocol
ViaImageAcquisitionProtocol
ViaLocalChannel
ViaCompassProtocol
ViaLocalChannel
ViaGPSProtocol

FIG. 48




```
<configSection>
        <sectionName>ImageAcquisition </sectionName>
        <sectionEntry>
                <subSection>
                        <subsection <Name>BorderSecuritySP</subSectionName>
                        <subSectionEntry>
                                <SensorName>BorderSecuritySP</SensorName>
                                <PollingRate>1</PollingRate>
                                <Enabled>1</Enabled>
                                <localInputChannel>GPS1Chan</localInputChannel>
                                <localInputChannel>Compass1n</localInputChannel>
                                <Camera1tName>QuickSet1SP</Camera1Name>
                                ...
                                </subSectionEntry>
                        </subSection>
                </sectionEntry>
        </configSection>
```

FIG. 49

4500

```
// Pass the first start token to the packet assembler.
fastarray<unsigned char> startTok;
startTok.resize(1);
startTok[0] = ACK;
m_assembler.setStartToken(startTok);
// Pass the second start token to the packet assembler.
startTok[0] = NAK;
m_assembler.setStartToken(startTok);

// Pass the end token to the packet assembler.
fastarray<unsigned char> endTok;
endTok.resize(1);
endTok[0] = ETX;
m_assembler.setEndToken(endTok);

// Register interest. Set up an observer relationship..
m_assembler.registerInterest(this);
```

FIG. 50

```
void ViaMyProtocol::receiveFromServer()
{
        int size;
        unsigned char *data = Receive(size);
        if ( (size > 0) && (data) )
        {
                m_assembler.handleInput(data, size);
                ...
```

FIG. 51

```
void ViaMyProtocol::updateFromModel(void *clientData)
{
        // For now we take this as a status response.

        int index = m_assembler.getStartTokenIndex();
        fastarray<unsigned char> message = m_assembler.getMessage();
        ...
```

FIG. 52

```
<sectionEntry>
        <subSection>
                <subSectionName>MySP</subSectionName>
                <subSectionEntry>
                        ...
                        <FilterAttribute>AlarmType</FilterAttribute>
                        <FilterValue>4</FilterValue>
                        <FilterAttribute>AlarmSubType</FilterAttribute>
                        <FilterValue>2</FilterValue>
                        <FilterMode>AND</FilterMode>
                        <FilterType>2</FilterType>
                        <jmsTopic>ITrans.Gas.CarbonMonoxide</jmsTopic>
                </subSectionEntry>
        </subSection>
</sectionEntry>
```

FIG. 53

```
<configSection>
    <sectionName>SensorGroups</sectionName>
    <sectionEntry>
        <subSection>
            <subSectionName>Tivella1</subSectionName>
            <subSectionEntry>
                <SensorGroupName>Tivella1</SensorGroupName>
                <SensorGroupType>Tivella</SensorGroupType>
                <SensorName>TivellaSP1</SensorName>
                  </subSectionEntry>
            </subSection>
            <subSection>
            <subSectionName>Tivella2</subSectionName>
            <subSectionEntry>
                <SensorGroupName>Tivella2</SensorGroupName>
                <SensorGroupType>Tivella</SensorGroupType>
                <SensorName>TivellaSP2</SensorName>
                <SensorName>TivellaSP3</SensorName>
            </subSectionEntry>
            </subSection>
            <subSection>
            <subSection>
            <subSectionName>INova1</subSectionName>
            <subSectionEntry>
                <SensorGroupName>INova1</SensorGroupName>
                <SensorGroupType>Inova</SensorGroupType>
                <SensorName>INovaSP1</SensorName>
            </subSectionEntry>
            </subSection>
    </sectionEntry>
</configSection>
```

FIG. 54

```
ViaSensorProxyGroupManager *spg =
ViaSensorProxyGroupManager::Instance();
const map< string, vector<string> > *tivellaGroups
                = spg->getGroupsForType("Tivella");
const map< string, vector<string> > *inovaGroups
                = spg->getGroupsForType("Inova");
```

FIG. 55

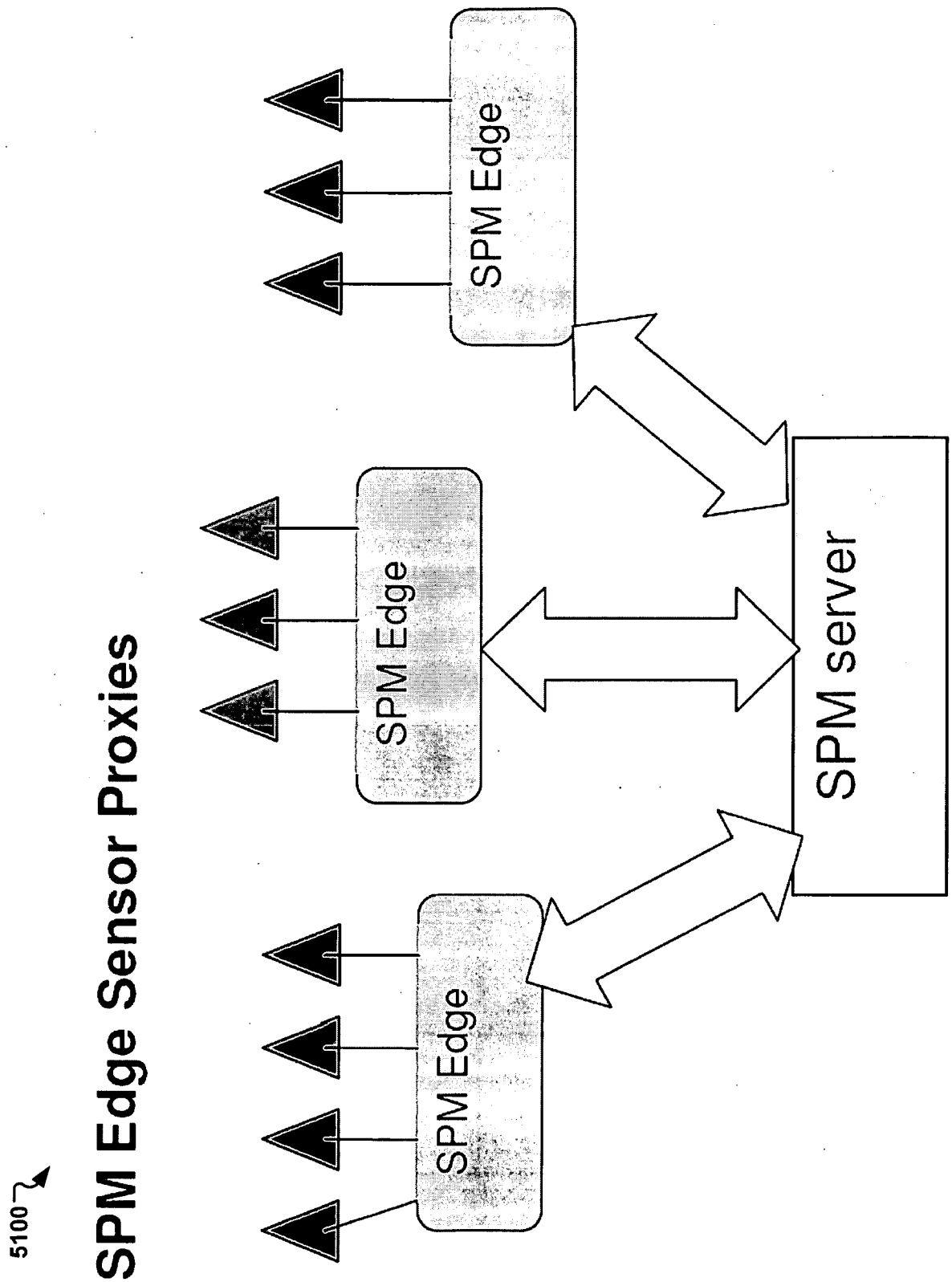

5100
SPM Edge Sensor Proxies
SPM Edge
SPM Edge
SPM Edge
SPM server

FIG. 56

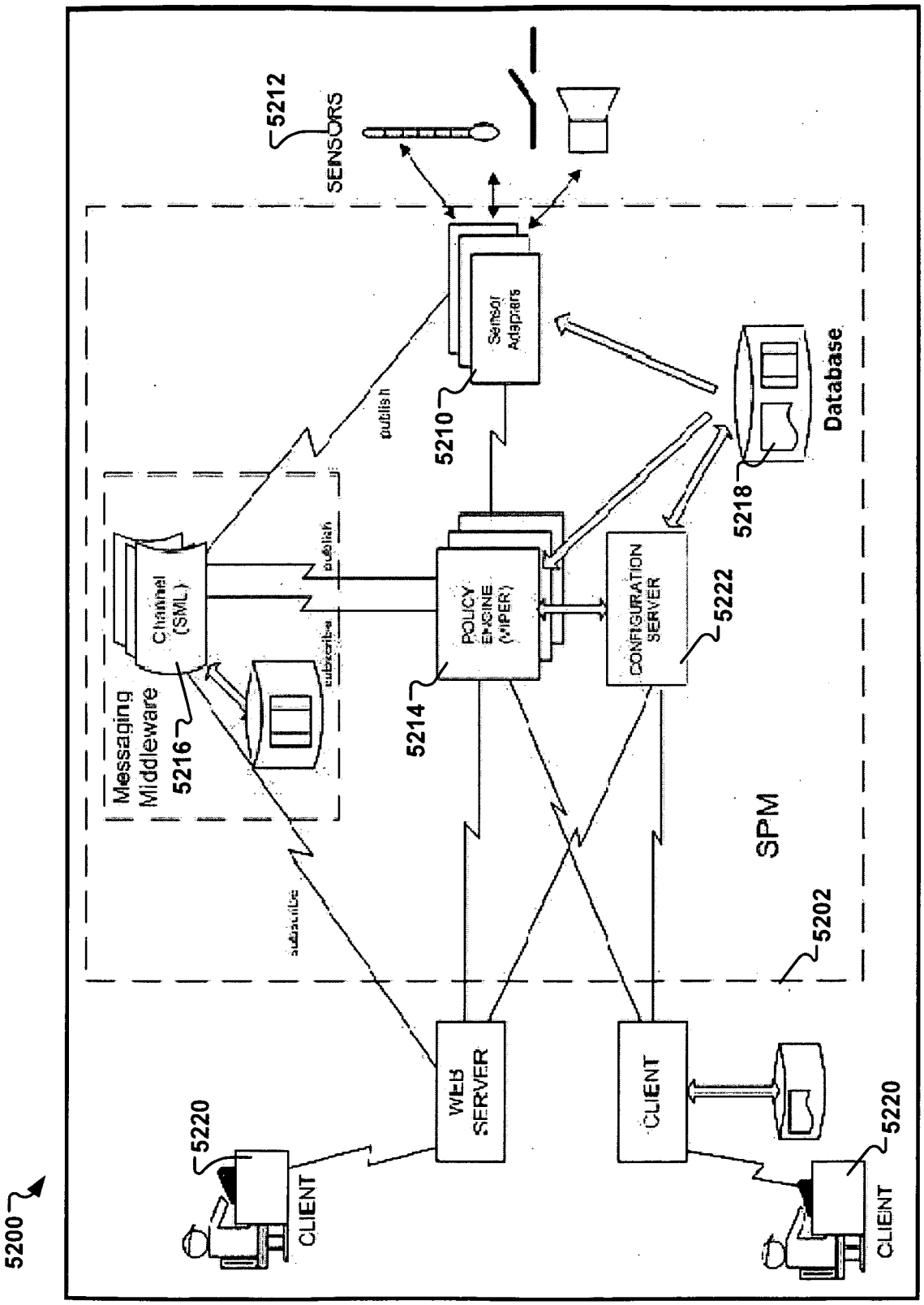

5200
5212
SENSORS
Sensor Adapters
5210
publish
Database
5218
Channel (SML)
Messaging Middleware
5216
publish
subscribe
POLICY ENGINE (VIPER)
5214
CONFIGURATION SERVER
5222
SPM
5202
subscribe
Web SERVER
CLIENT
5220
CLIENT
5220

FIG. 57

5300

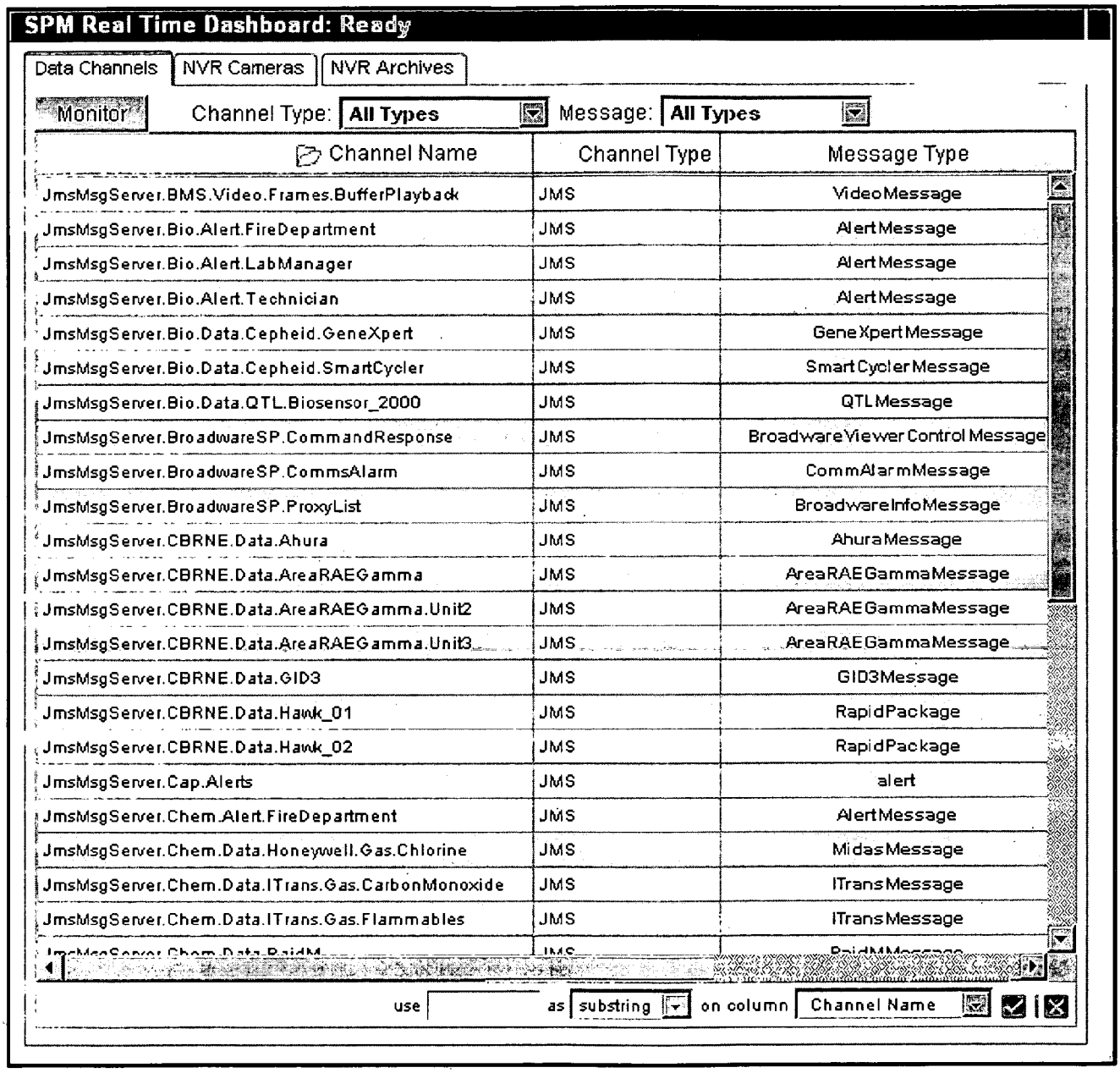

SPM Real Time Dashboard: Ready
Data Channels
NVR Cameras
NVR Archives
Monitor
Channel Type: All Types
Message: All Types
Channel Name
Channel Type
Message Type
JmsMsgServer.BMS.Video.Frames.BufferPlayback JMS VideoMessage
JmsMsgServer.Bio.Alert.FireDepartment JMS AlertMessage
JmsMsgServer.Bio.Alert.LabManager JMS AlertMessage
JmsMsgServer.Bio.Alert.Technician JMS AlertMessage
JmsMsgServer.Bio.Data.Cepheid.GeneXpert JMS GeneXpertMessage
JmsMsgServer.Bio.Data.Cepheid.SmartCycler JMS SmartCyclerMessage
JmsMsgServer.Bio.Data.QTL.Biosensor_2000 JMS QTLMessage
JmsMsgServer.BroadwareSP.CommandResponse JMS BroadwareViewerControlMessage
JmsMsgServer.BroadwareSP.CommsAlarm JMS CommAlarmMessage
JmsMsgServer.BroadwareSP.ProxyList JMS BroadwareInfoMessage
JmsMsgServer.CBRNE.Data.Ahura JMS AhuraMessage
JmsMsgServer.CBRNE.Data.AreaRAEGamma JMS AreaRAEGammaMessage
JmsMsgServer.CBRNE.Data.AreaRAEGamma.Unit2 JMS AreaRAEGammaMessage
JmsMsgServer.CBRNE.Data.AreaRAEGamma.Unit3 JMS AreaRAEGammaMessage
JmsMsgServer.CBRNE.Data.GID3 JMS GID3Message
JmsMsgServer.CBRNE.Data.Hawk_01 JMS RapidPackage
JmsMsgServer.CBRNE.Data.Hawk_02 JMS RapidPackage
JmsMsgServer.Cap.Alerts JMS alert
JmsMsgServer.Chem.Alert.FireDepartment JMS AlertMessage
JmsMsgServer.Chem.Data.Honeywell.Gas.Chlorine JMS MidasMessage
JmsMsgServer.Chem.Data.ITrans.Gas.CarbonMonoxide JMS ITransMessage
JmsMsgServer.Chem.Data.ITrans.Gas.Flammables JMS ITransMessage
use
as substring
on column Channel Name

FIG. 58

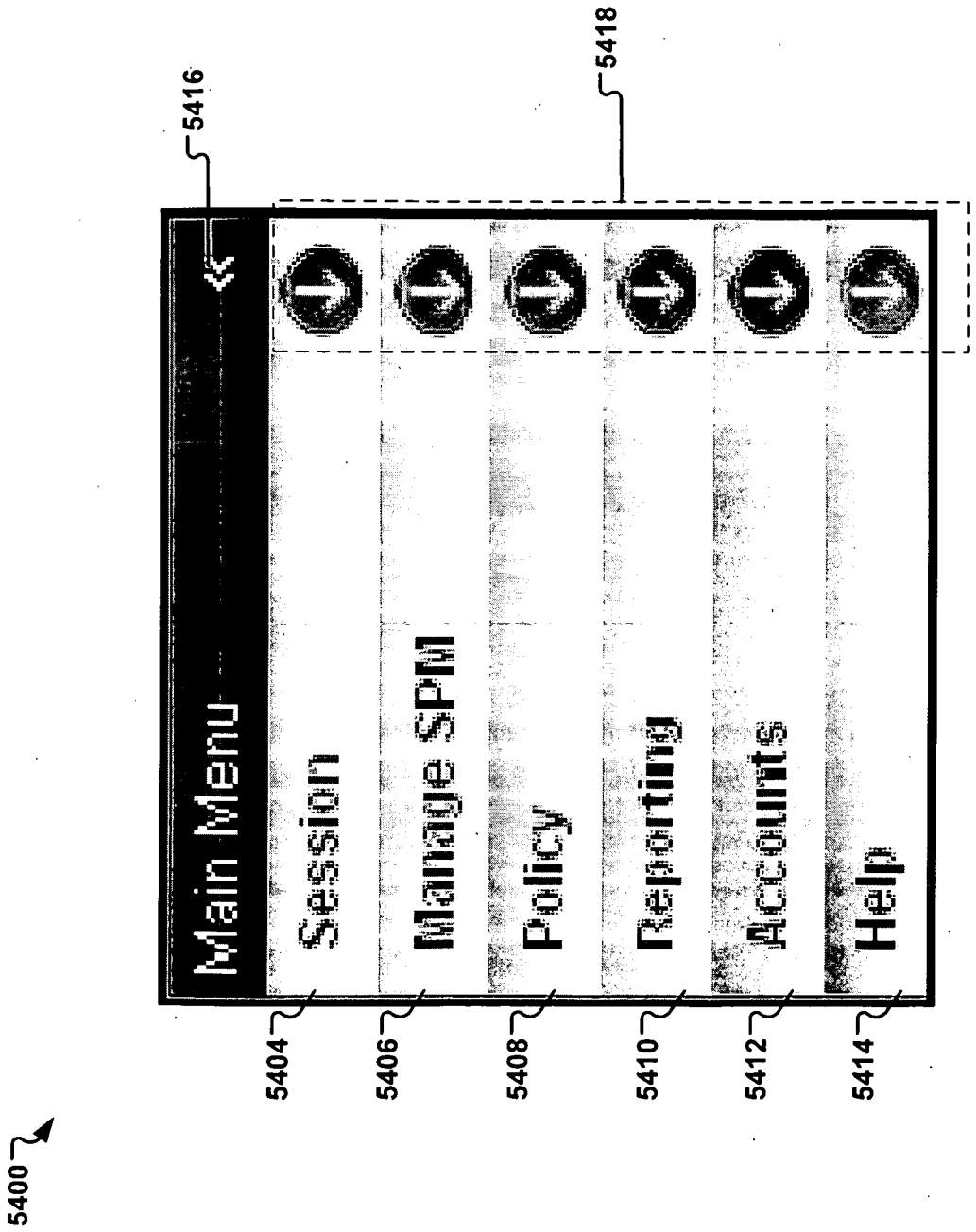

5400
5416
5418
Main Menu
Session
Manage SPM
Policy
Reporting
Accounts
Help
5404
5406
5408
5410
5412
5414

FIG. 59

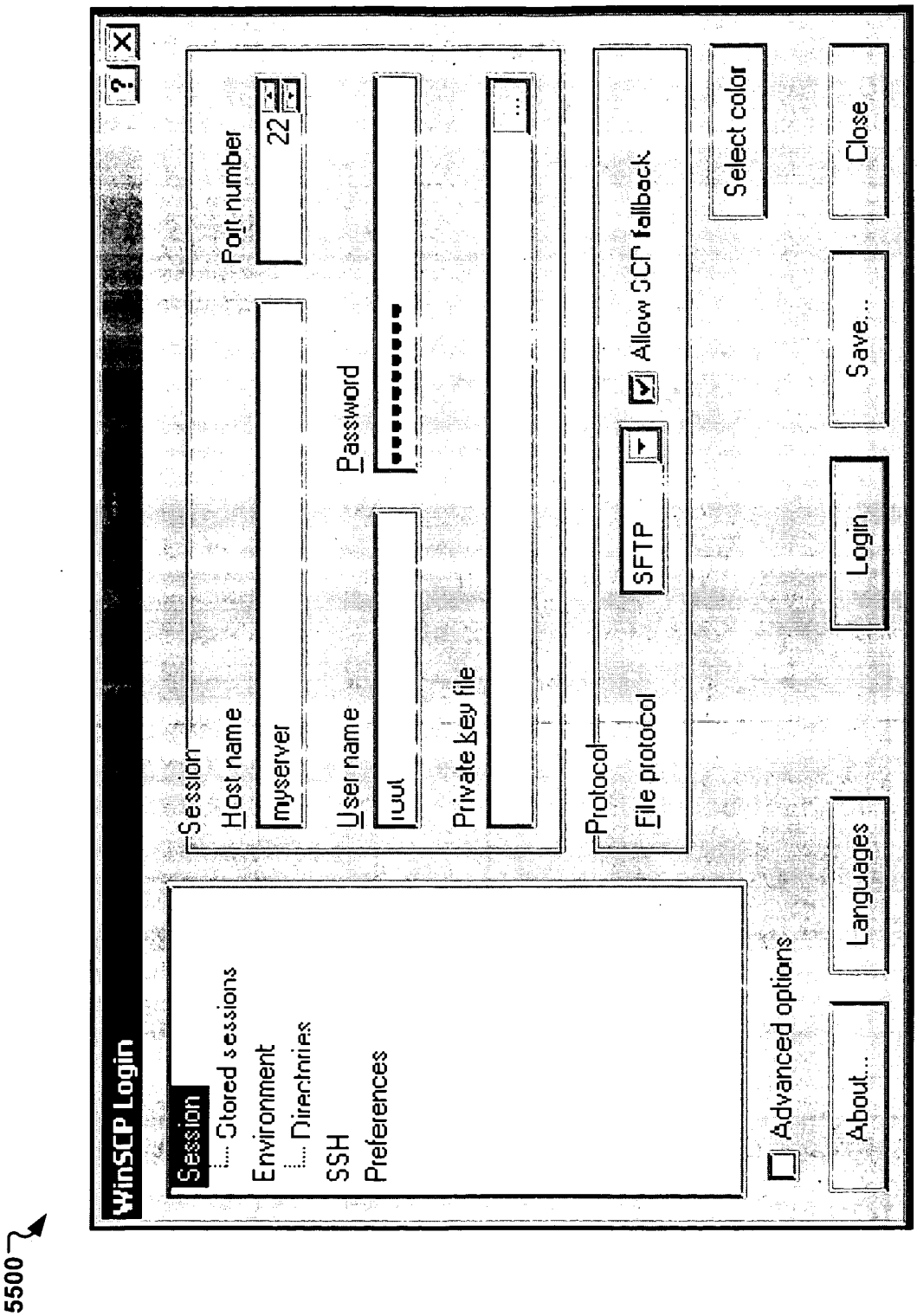
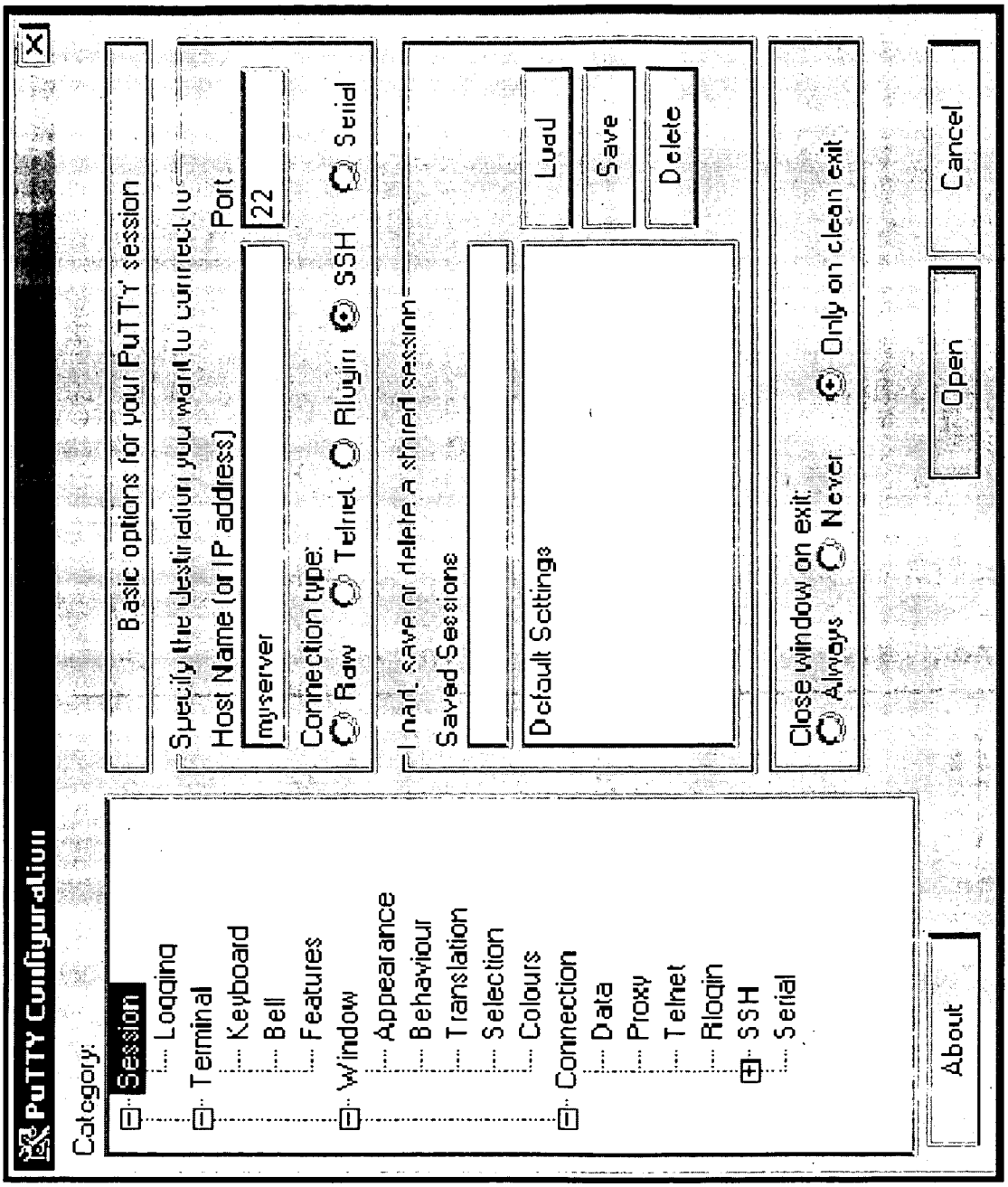

5500
WinSCP Login
Session
Stored sessions
Environment
Directories
SSH
Preferences
Session
Host name
myserver
Port number
22
User name
Password
Private key file
Protocol
File protocol
SFTP
Allow SCP fallback
Select color
Advanced options
About...
Languages
Login
Save...
Close

FIG. 60


PuTTY Configuration
Category:
Session
Logging
Terminal
Keyboard
Bell
Features
Window
Appearance
Behaviour
Translation
Selection
Colours
Connection
Data
Proxy
Telnet
Rlogin
SSH
Serial
Basic options for your PuTTY session
Specify the destination you want to connect to
Host Name (or IP address)
Port
myserver
22
Connection type:
Raw
Telnet
Rlogin
SSH
Serial
Load, save or delete a stored session
Saved Sessions
Default Settings
Load
Save
Delete
Close window on exit:
Always
Never
Only on clean exit
About
Open
Cancel

5600

FIG. 61

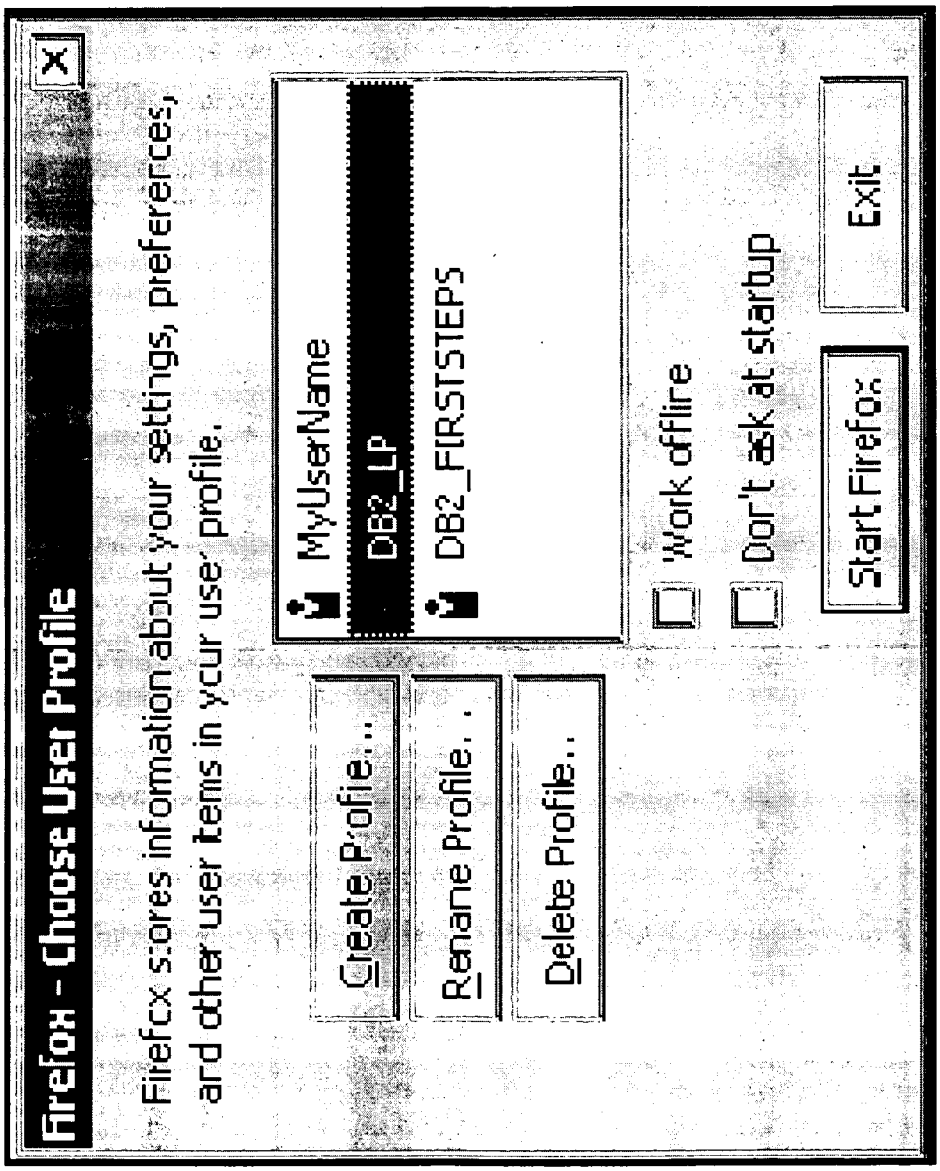

5700
Firefox - Choose User Profile
Firefox stores information about your settings, preferences,
and other user items in your user profile.
MyUserName
DB2_LP
DB2_FIRSTSTEPS
Create Profile...
Rename Profile...
Delete Profile...
Work offline
Don't ask at startup
Start Firefox
Exit

FIG. 62

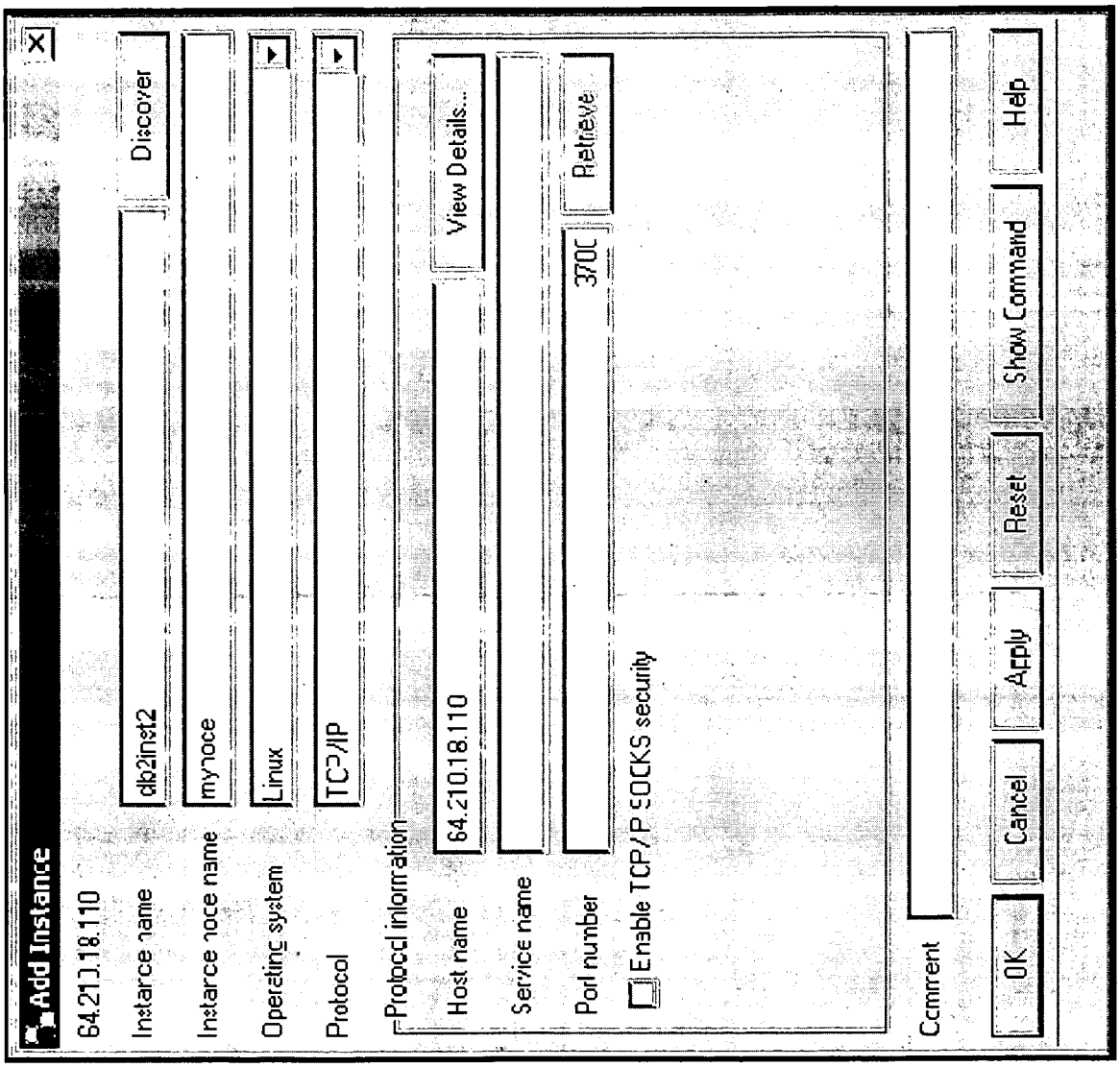

5800
Add Instance
64.210.18.110
Instance name
db2inst2
Discover
Instance node name
mynode
Operating system
Linux
Protocol
TCP/IP
Protocol information
Host name
64.210.18.110
View Details...
Service name
Port number
3700
Retrieve
Enable TCP/IP SOCKS security
Comment
OK
Cancel
Apply
Reset
Show Command
Help

FIG. 63

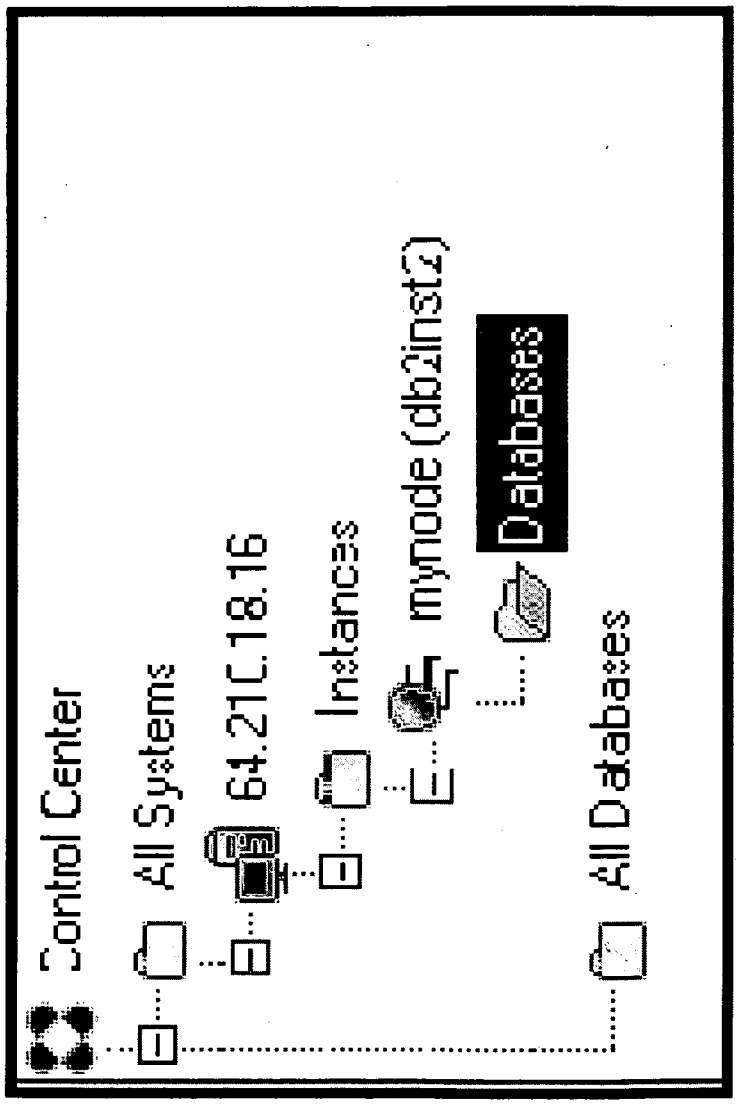

Control Center
All Systems
64.21C.18.16
Instances
mynode (db2inst2)
Databases
All Databases
5900

FIG. 64

6000

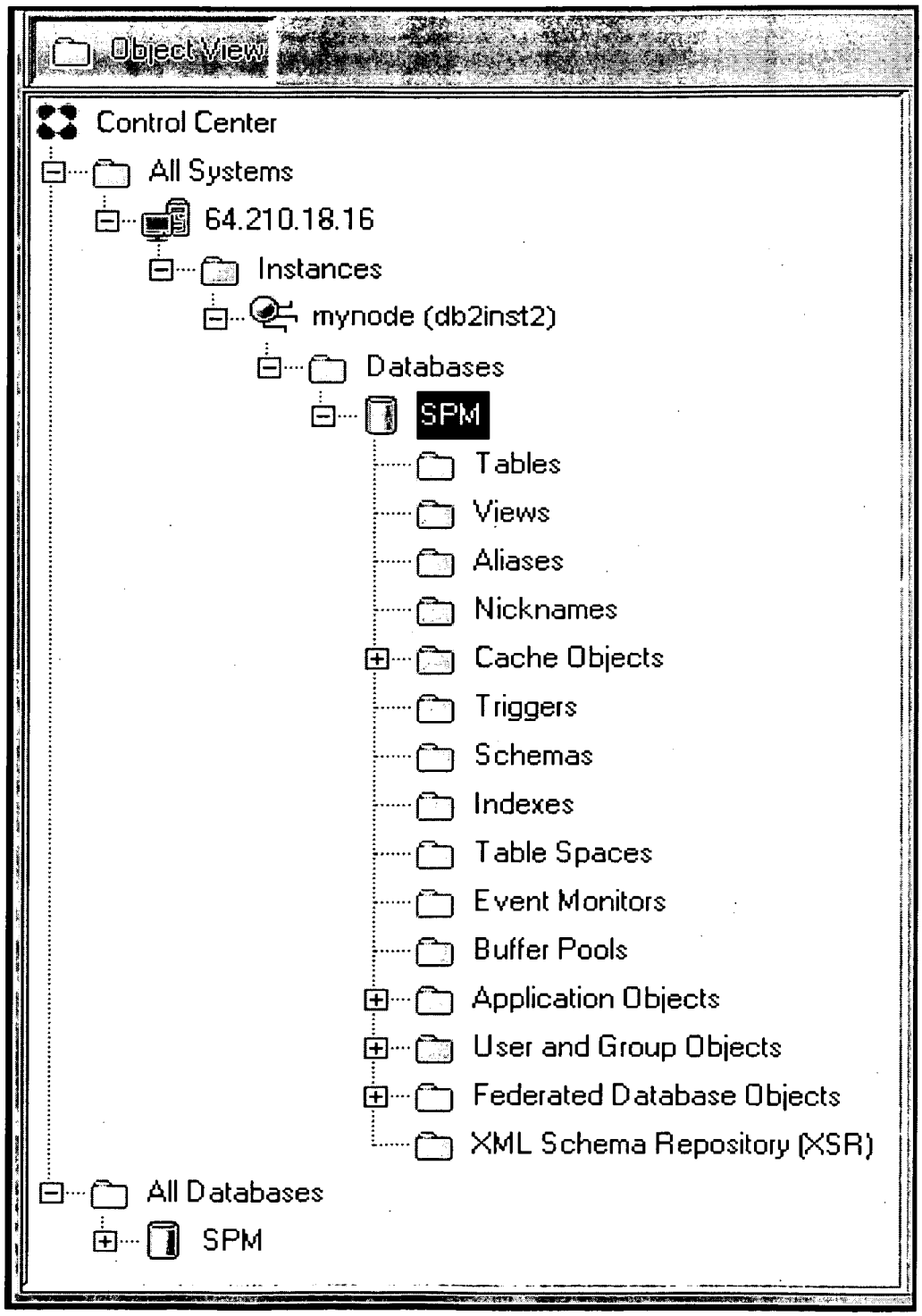

Object View
Control Center
All Systems
64.210.18.16
Instances
mynode (db2inst2)
Databases
SPM
Tables
Views
Aliases
Nicknames
Cache Objects
Triggers
Schemas
Indexes
Table Spaces
Event Monitors
Buffer Pools
Application Objects
User and Group Objects
Federated Database Objects
XML Schema Repository (XSR)
All Databases
SPM

FIG. 65

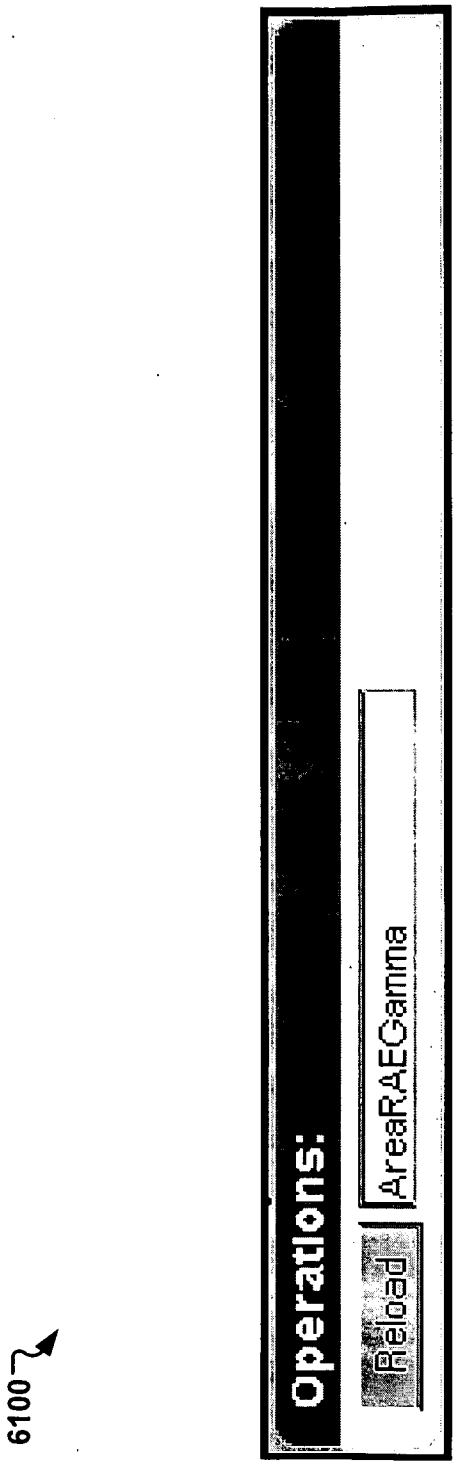


FIG. 66

6200

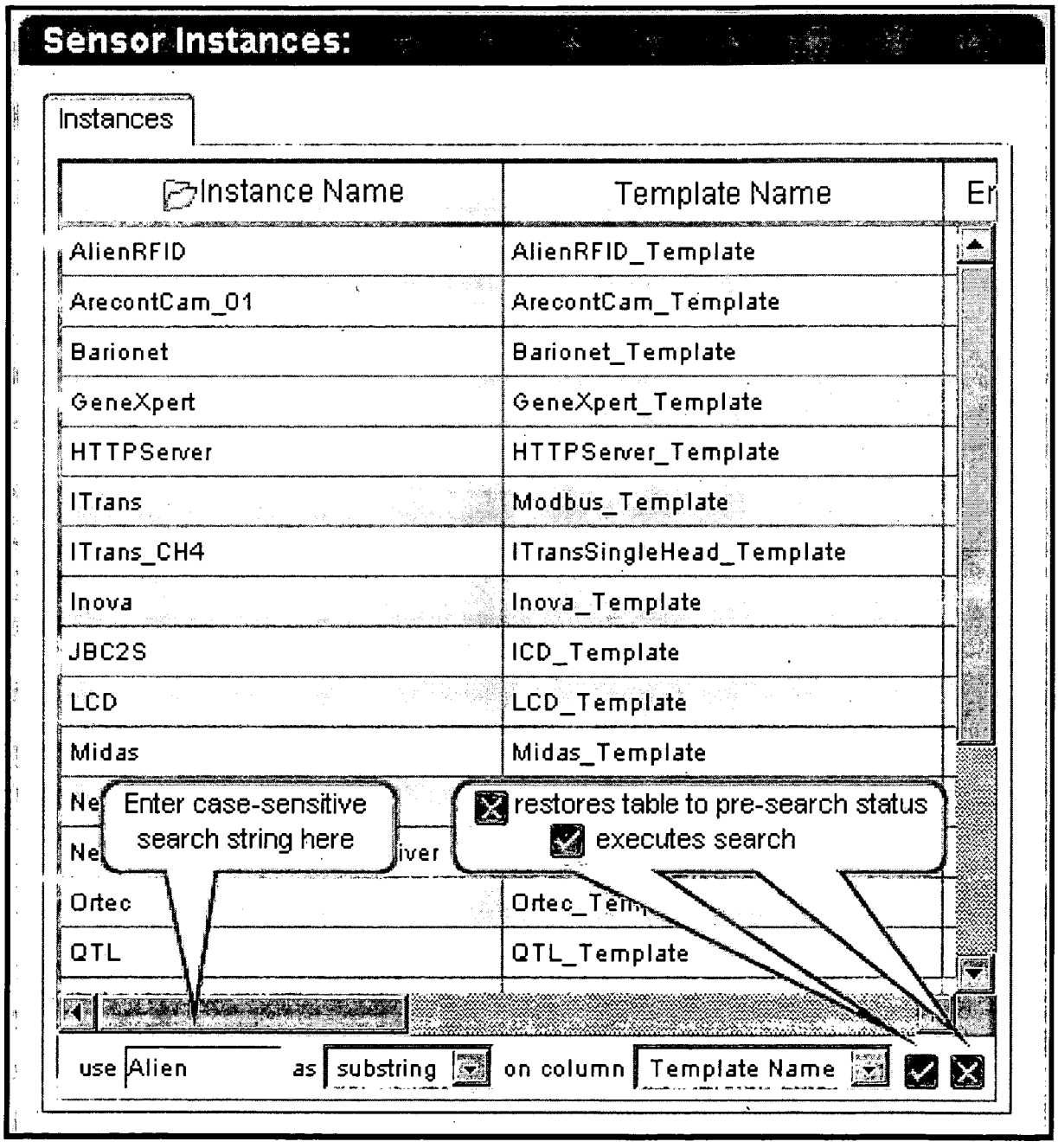

Sensor Instances:
Instances
Instance Name
Template Name
AlienRFID
AlienRFID_Template
ArecontCam_01
ArecontCam_Template
Barionet
Barionet_Template
GeneXpert
GeneXpert_Template
HTTPServer
HTTPServer_Template
ITrans
Modbus_Template
ITrans_CH4
ITransSingleHead_Template
Inova
Inova_Template
JBC2S
ICD_Template
LCD
LCD_Template
Midas
Midas_Template
Ortec
QTL
QTL_Template
Enter case-sensitive search string here
restores table to pre-search status
executes search
use Alien
as substring
on column Template Name

FIG. 67

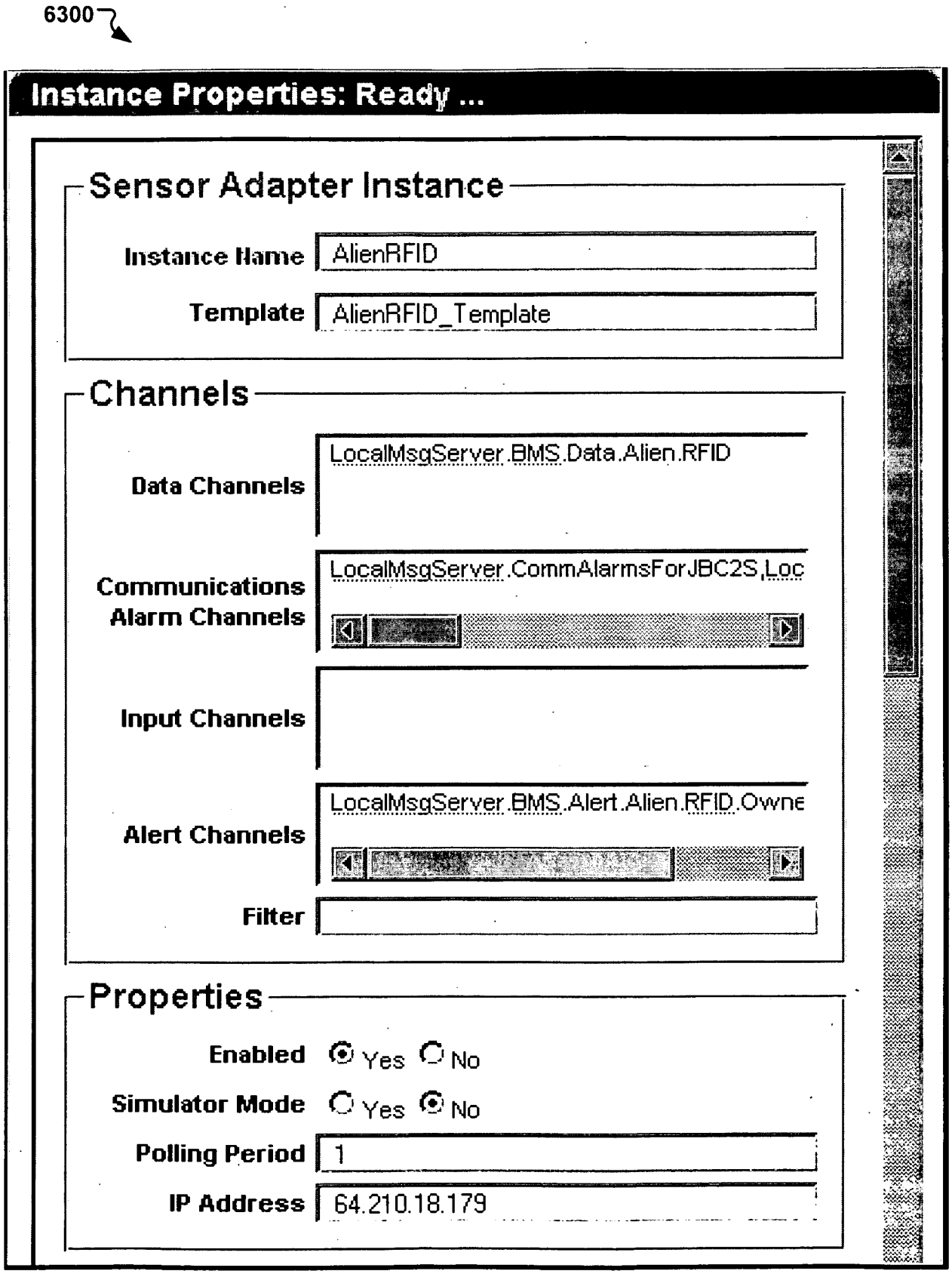

6300
Instance Properties: Ready ...
Sensor Adapter Instance
Instance Name
AlienRFID
Template
AlienRFID_Template
Channels
Data Channels
LocalMsgServer.BMS.Data.Alien.RFID
Communications Alarm Channels
LocalMsgServer.CommAlarmsForJBC2S,Loc
Input Channels
Alert Channels
LocalMsgServer.BMS.Alert.Alien.RFID.Owne
Filter
Properties
Enabled
Yes
No
Simulator Mode
Yes
No
Polling Period
1
IP Address
64.210.18.179

FIG. 68

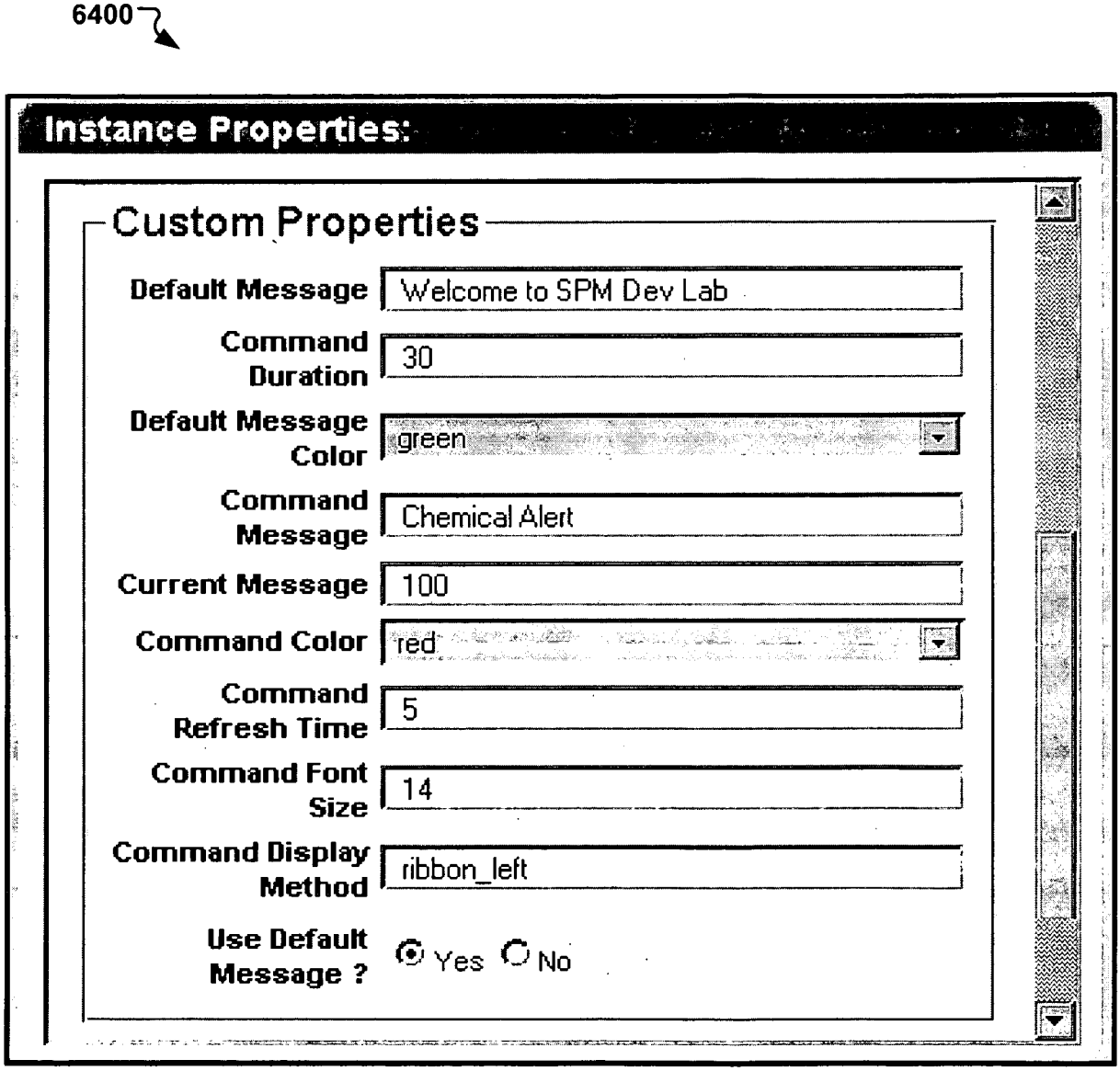

6400
Instance Properties:
Custom Properties
Default Message
Welcome to SPM Dev Lab
Command Duration
30
Default Message Color
green
Command Message
Chemical Alert
Current Message
100
Command Color
red
Command Refresh Time
5
Command Font Size
14
Command Display Method
ribbon_left
Use Default Message ?
Yes
No

FIG. 69

6500

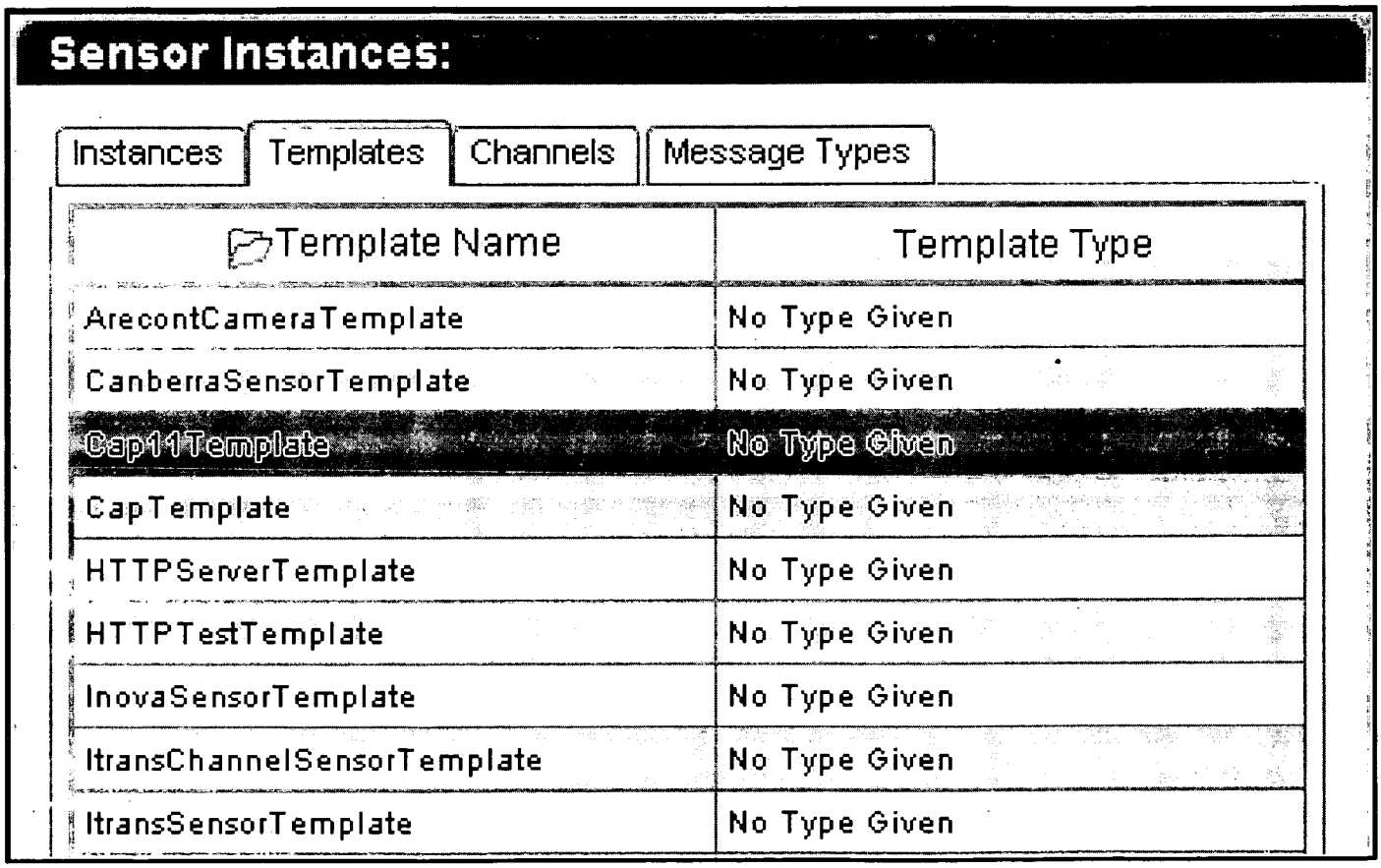

Sensor Instances:
Instances
Templates
Channels
Message Types
Template Name
Template Type
ArecontCameraTemplate
No Type Given
CanberraSensorTemplate
No Type Given
Cap11Template
No Type Given
CapTemplate
No Type Given
HTTPServerTemplate
No Type Given
HTTPTestTemplate
No Type Given
InovaSensorTemplate
No Type Given
ItransChannelSensorTemplate
No Type Given
ItransSensorTemplate
No Type Given

FIG. 70

6600

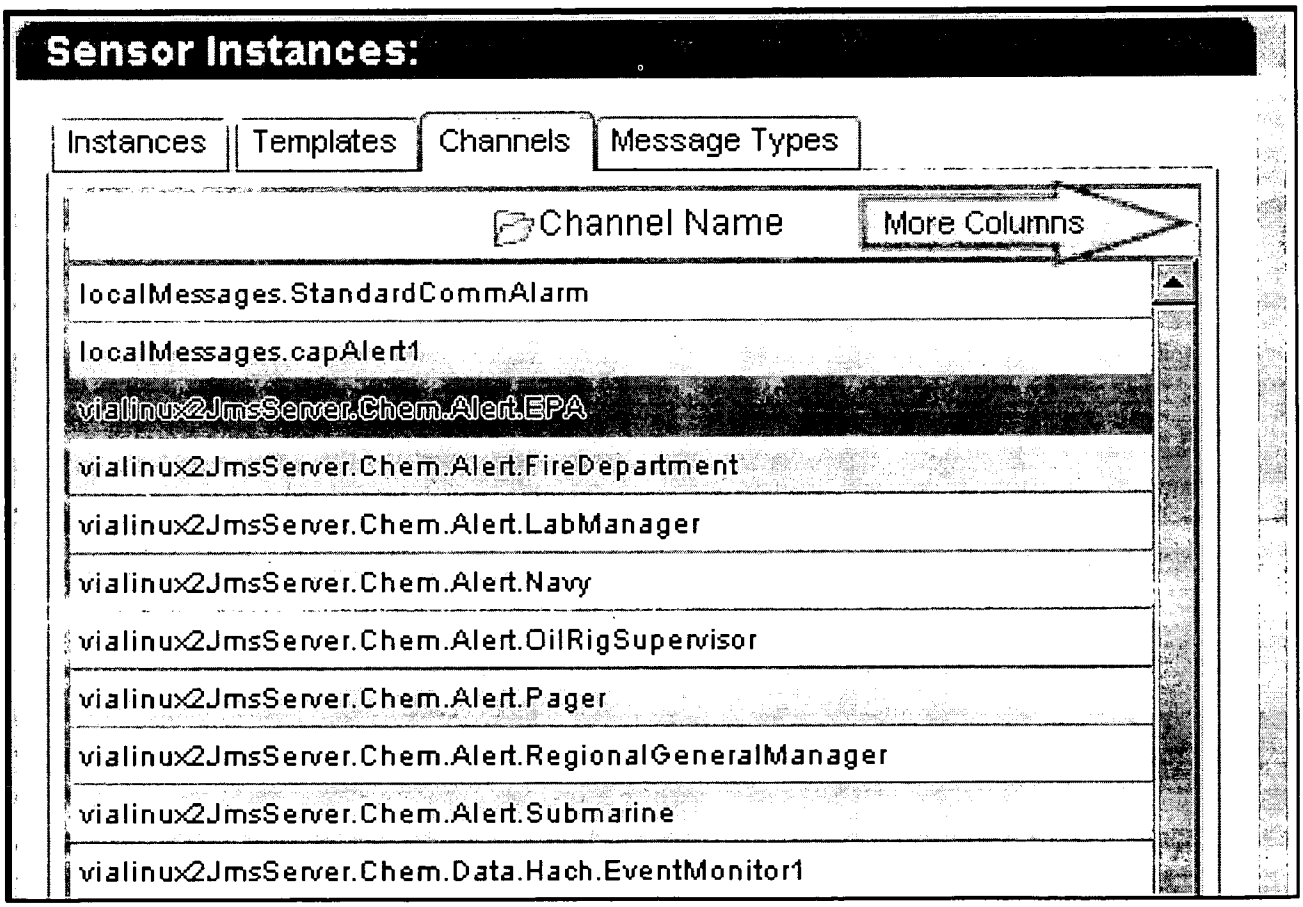

Sensor Instances:
Instances
Templates
Channels
Message Types
Channel Name
More Columns
localMessages.StandardCommAlarm
localMessages.capAlert1
vialinux2JmsServer.Chem.Alert.EPA
vialinux2JmsServer.Chem.Alert.FireDepartment
vialinux2JmsServer.Chem.Alert.LabManager
vialinux2JmsServer.Chem.Alert.Navy
vialinux2JmsServer.Chem.Alert.OilRigSupervisor
vialinux2JmsServer.Chem.Alert.Pager
vialinux2JmsServer.Chem.Alert.RegionalGeneralManager
vialinux2JmsServer.Chem.Alert.Submarine
vialinux2JmsServer.Chem.Data.Hach.EventMonitor1

FIG. 71

6700

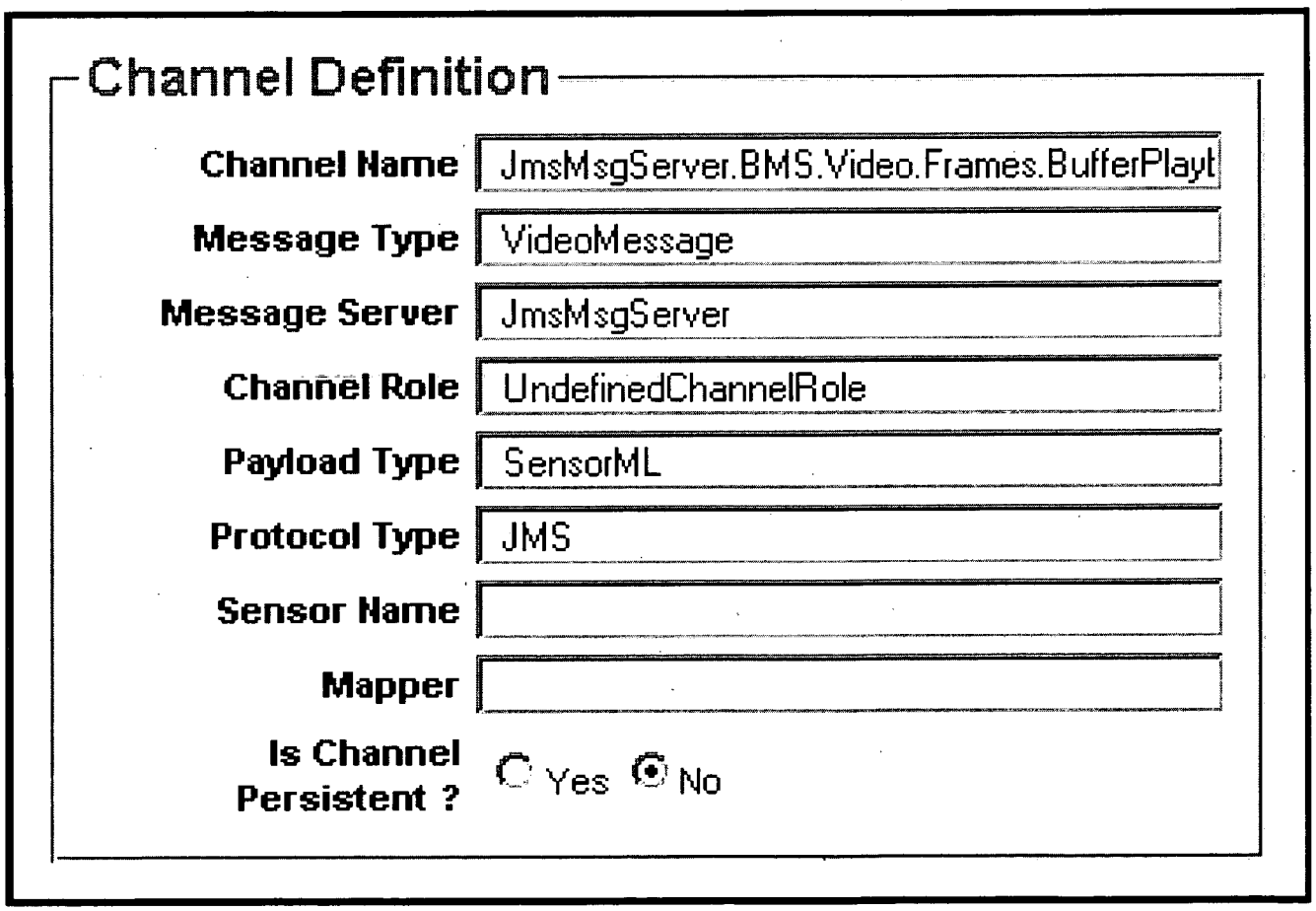

Channel Definition
Channel Name
JmsMsgServer.BMS.Video.Frames.BufferPlayt
Message Type
VideoMessage
Message Server
JmsMsgServer
Channel Role
UndefinedChannelRole
Payload Type
SensorML
Protocol Type
JMS
Sensor Name
Mapper
Is Channel Persistent ?
Yes
No

FIG. 72

6800

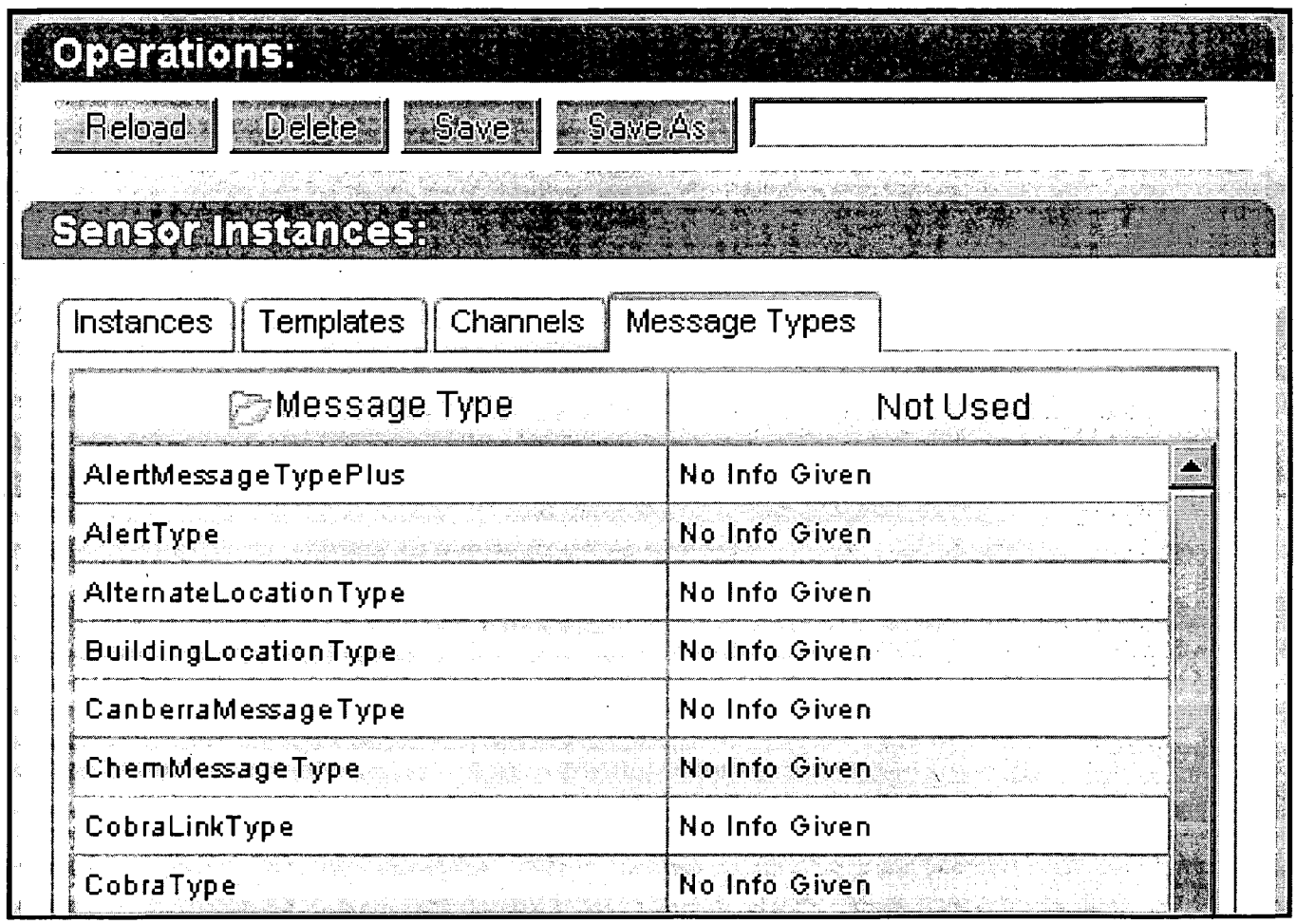

Operations:
Reload
Delete
Save
Save As
Sensor Instances:
Instances
Templates
Channels
Message Types
Message Type
Not Used
AlertMessageTypePlus
No Info Given
AlertType
No Info Given
AlternateLocationType
No Info Given
BuildingLocationType
No Info Given
CanberraMessageType
No Info Given
ChemMessageType
No Info Given
CobraLinkType
No Info Given
CobraType
No Info Given

FIG. 73

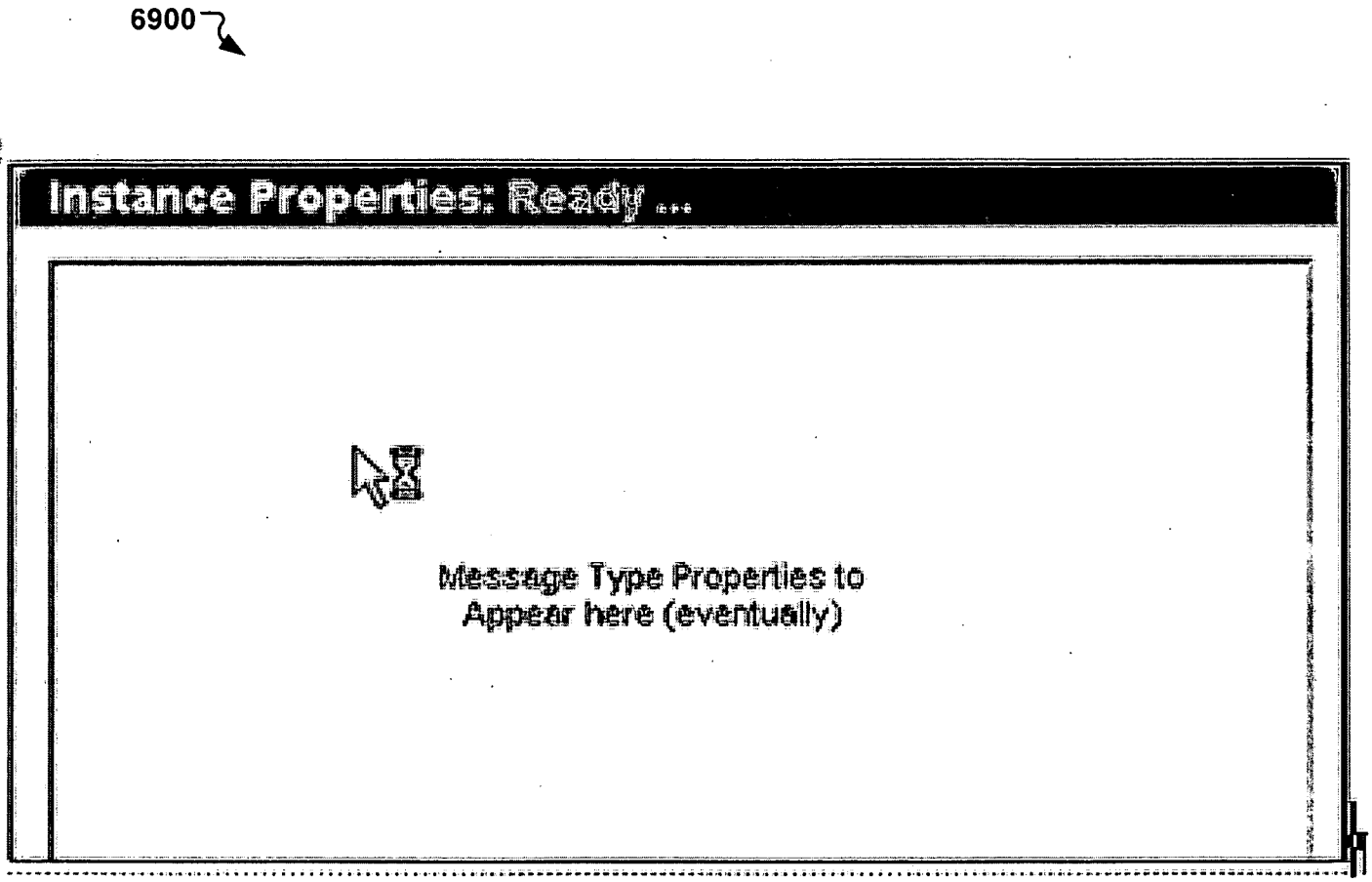

6900
Instance Properties: Ready ...
Message Type Properties to
Appear here (eventually)

FIG. 74

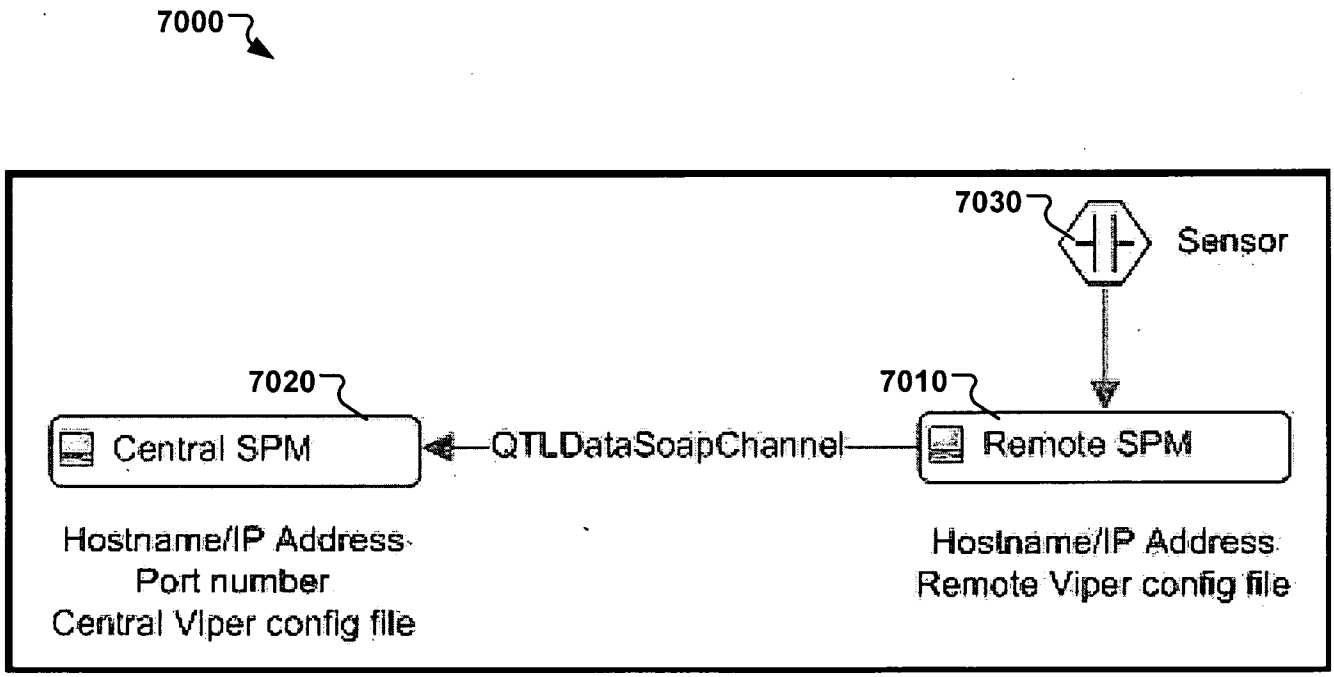

7000
7030
Sensor
7020
7010
Central SPM
QTLDataSoapChannel
Remote SPM
Hostname/IP Address
Port number
Central Viper config file
Hostname/IP Address
Remote Viper config file

FIG. 75

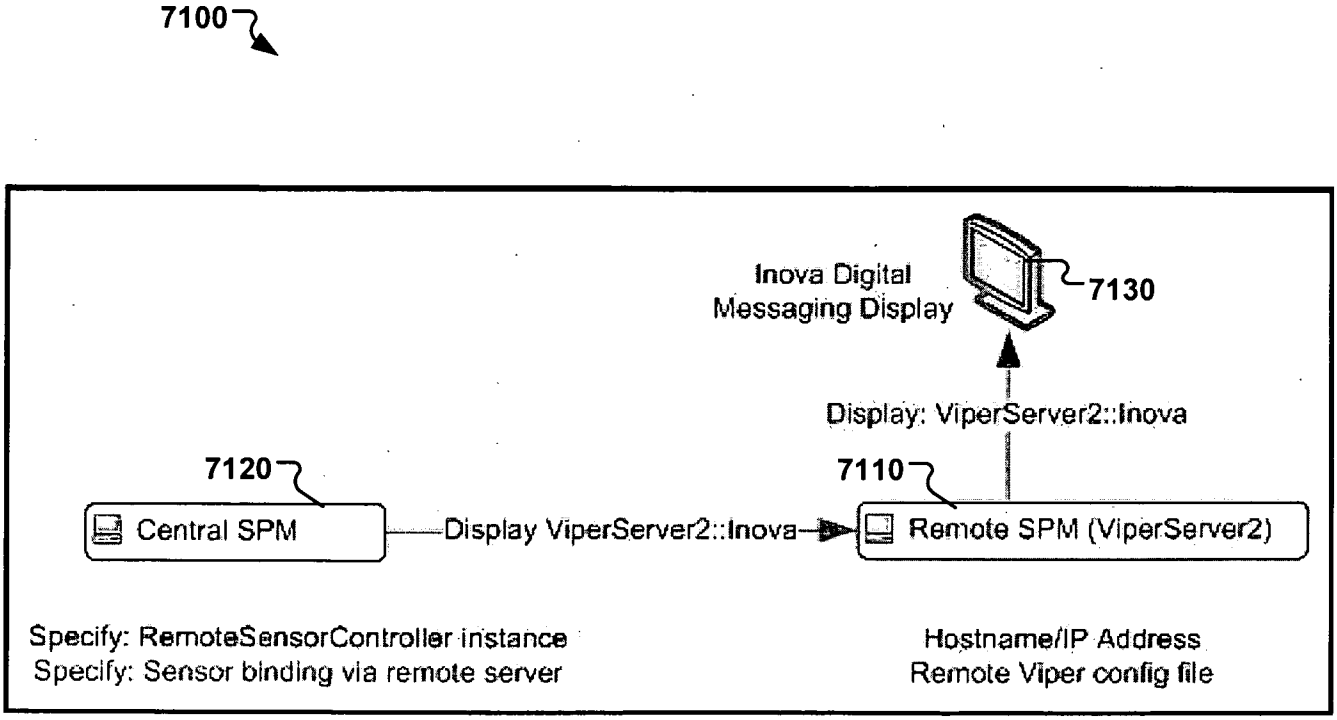

7100
Inova Digital
Messaging Display
7130
Display: ViperServer2::Inova
7120
7110
Central SPM
Display ViperServer2::Inova
Remote SPM (ViperServer2)
Specify: RemoteSensorController instance
Specify: Sensor binding via remote server
Hostname/IP Address
Remote Viper config file

FIG. 76

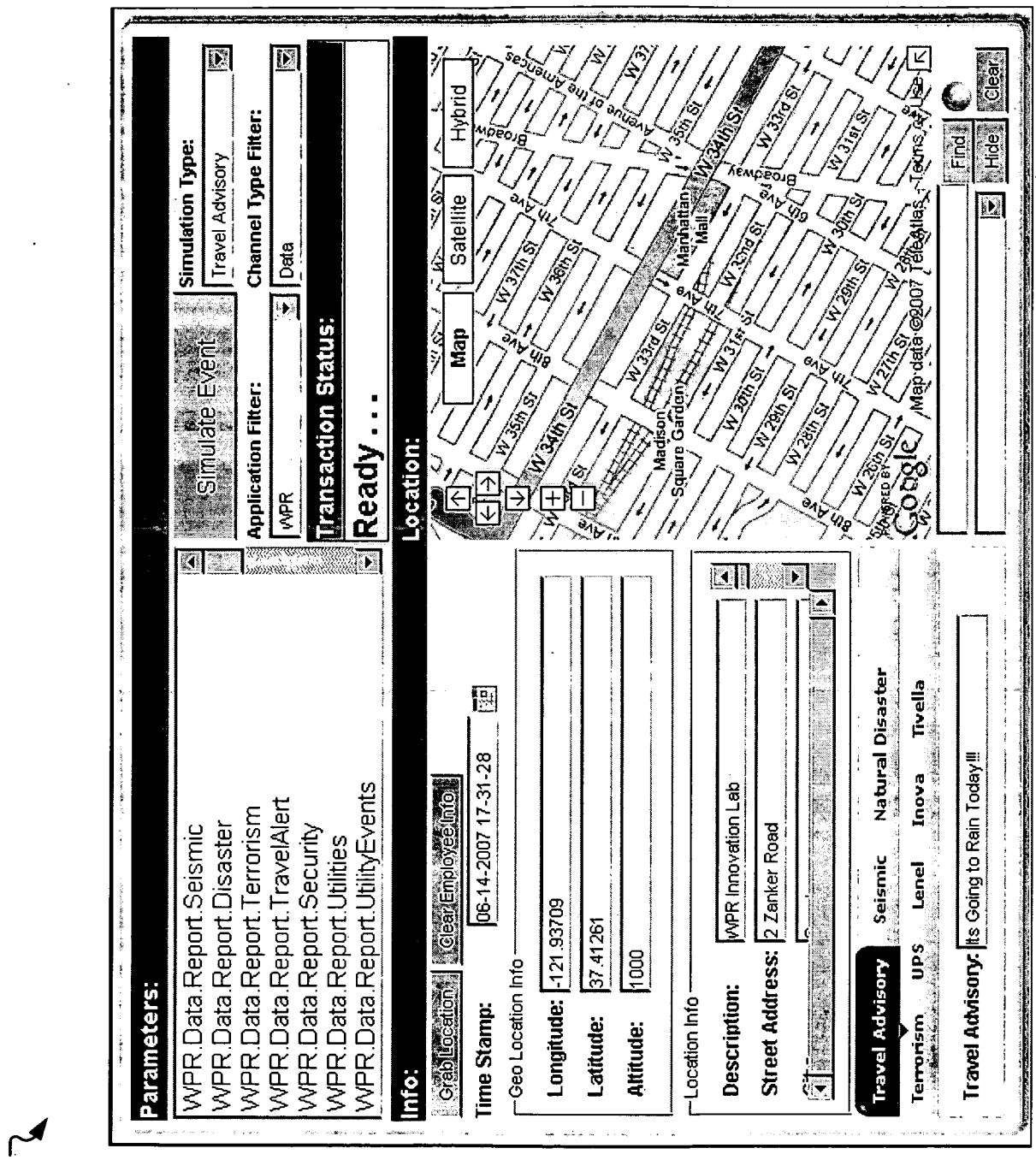
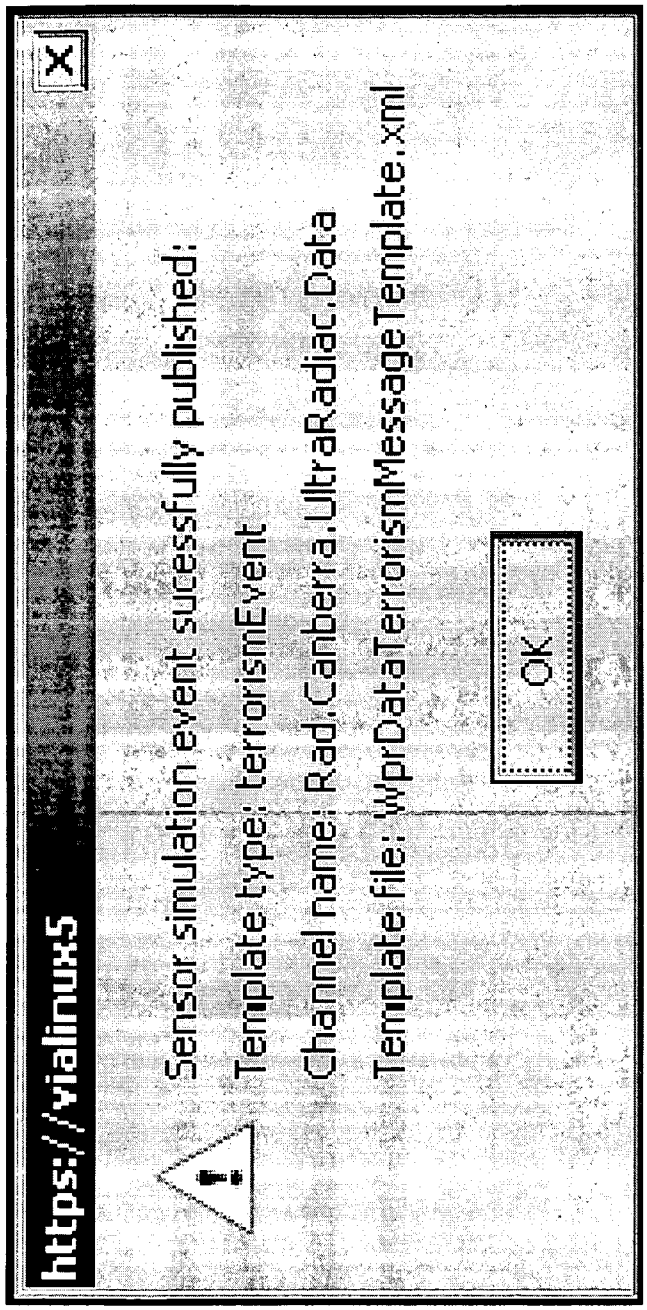

7200
Parameters:
WPR.Data.Report.Seismic
WPR.Data.Report.Disaster
WPR.Data.Report.Terrorism
WPR.Data.Report.TravelAlert
WPR.Data.Report.Security
WPR.Data.Report.Utilities
WPR.Data.Report.UtilityEvents
Simulate Event
Simulation Type:
Travel Advisory
Application Filter:
WPR
Channel Type Filter:
Data
Transaction Status:
Ready...
Info:
Location:
Grab Location
Clear Employee Info
Time Stamp:
06-14-2007 17-31-28
Geo Location Info
Longitude:
-121.93709
Latitude:
37.41261
Altitude:
1000
Location Info
Description:
WPR Innovation Lab
Street Address:
2 Zanker Road
Travel Advisory
Seismic
Natural Disaster
Terrorism
UPS
Lenel
Inova
Tivella
Travel Advisory:
Its Going to Rain Today!!!
Map
Satellite
Hybrid
Find
Hide
Clear

FIG. 77


https://vialinux5
Sensor simulation event sucessfully published:
Template type: terrorismEvent
Channel name: Rad.Canberra.UltraRadiac.Data
Template file: WprDataTerrorismMessageTemplate.xml
OK

7300

FIG. 78

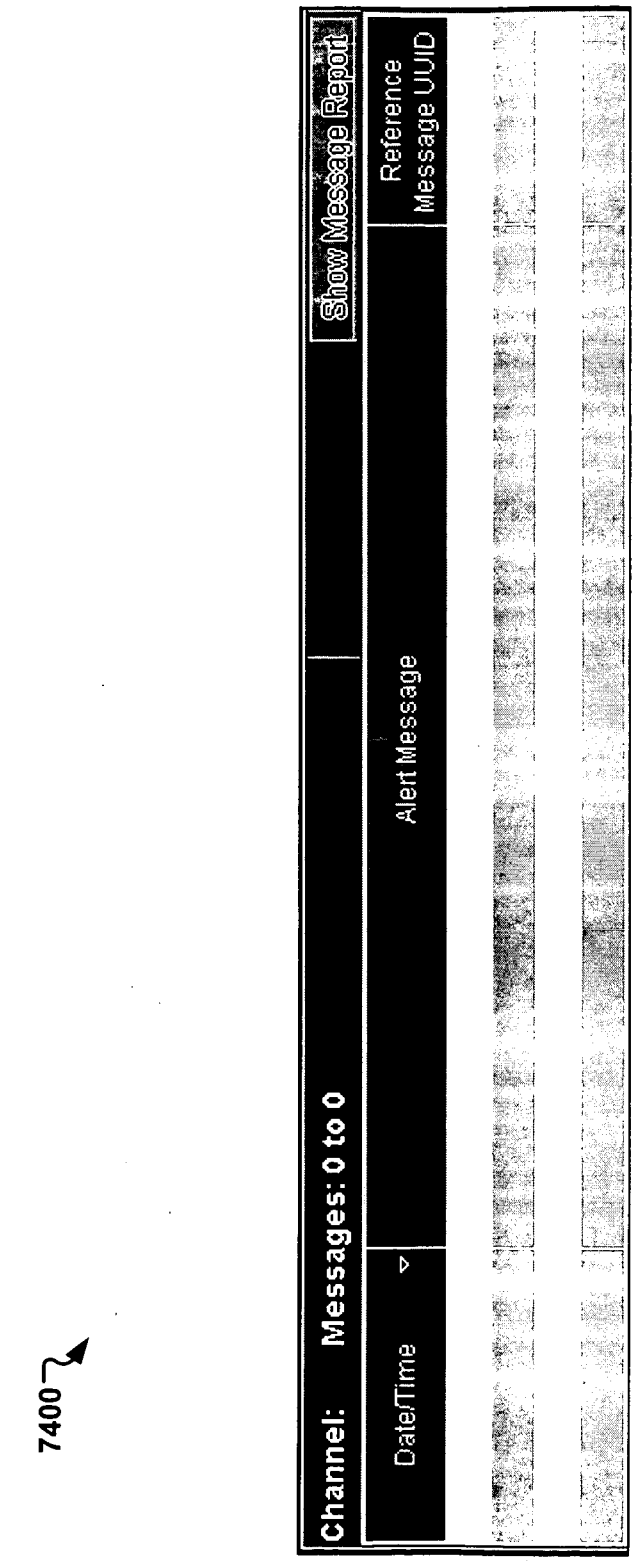
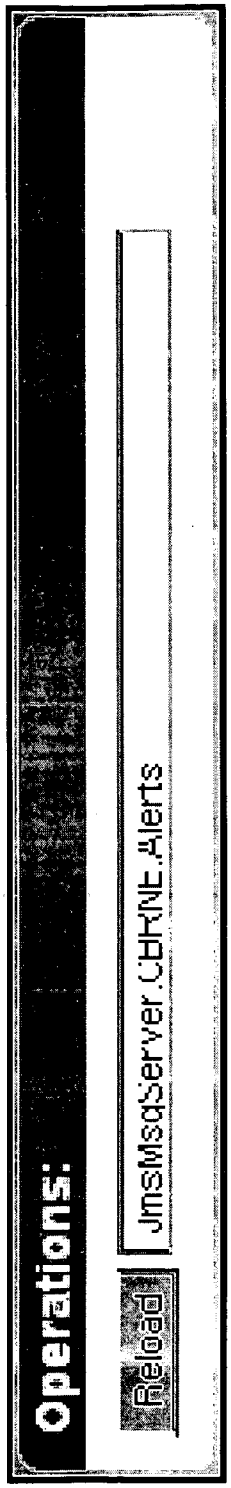

7400
Channel: Messages: 0 to 0
Show Message Report
Date/Time
Alert Message
Reference Message UUID

FIG. 79



7500

FIG. 80

7600

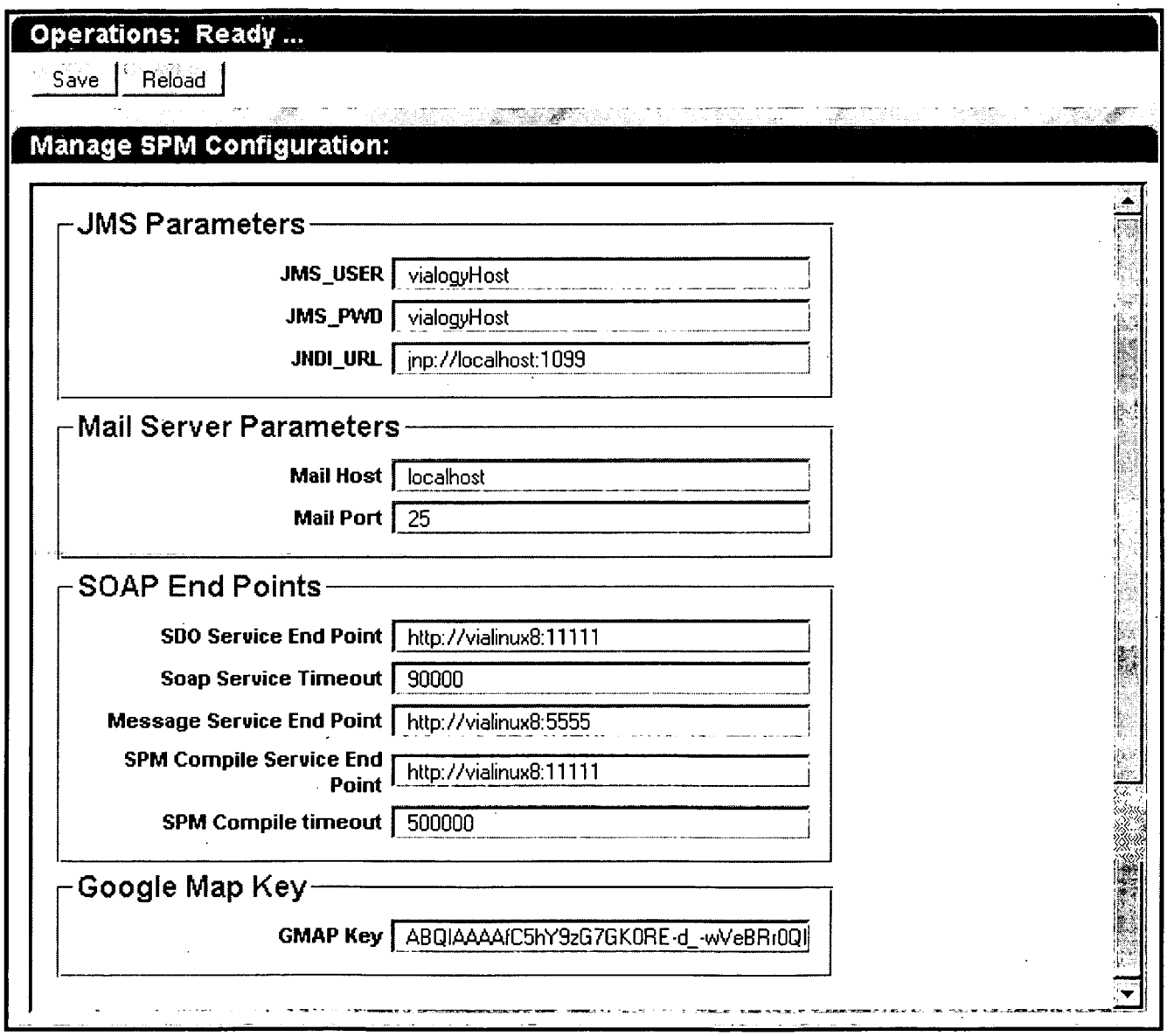

Operations: Ready ...
Save
Reload
Manage SPM Configuration:
JMS Parameters
JMS_USER
vialogyHost
JMS_PWD
vialogyHost
JNDI_URL
jnp://localhost:1099
Mail Server Parameters
Mail Host
localhost
Mail Port
25
SOAP End Points
SDO Service End Point
http://vialinux8:11111
Soap Service Timeout
90000
Message Service End Point
http://vialinux8:5555
SPM Compile Service End Point
http://vialinux8:11111
SPM Compile timeout
500000
Google Map Key
GMAP Key
ABQIAAAAfC5hY9zG7GK0RE-d_-wVeBRr0QI

FIG. 81

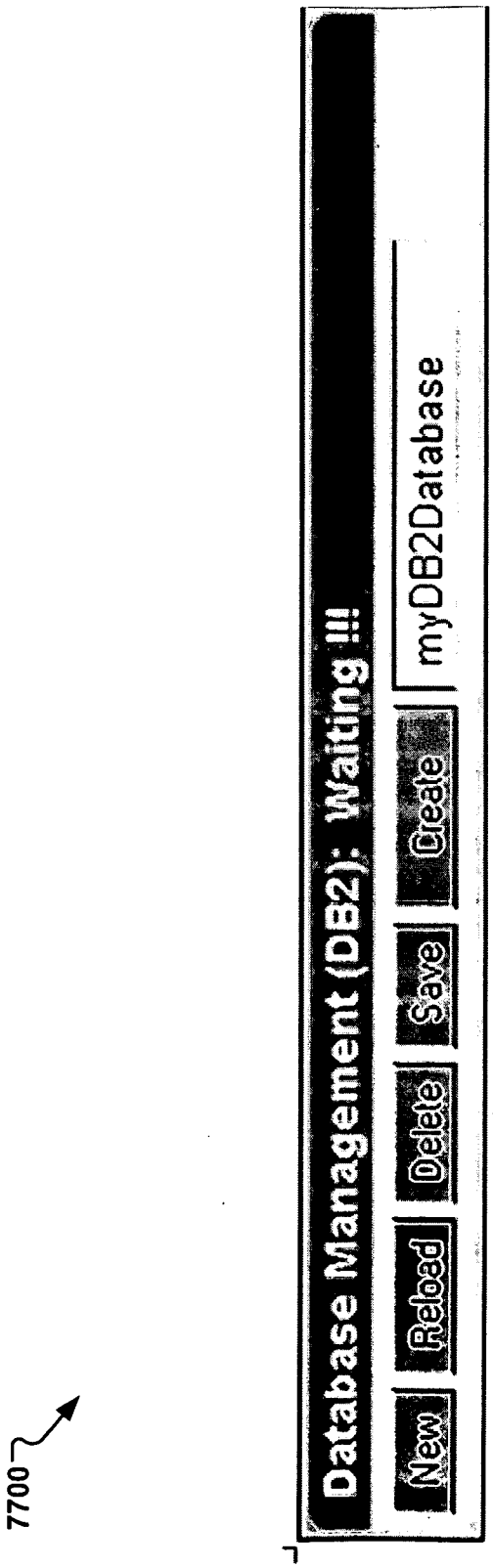

7700
Database Management (DB2): Waiting !!!
New
Reload
Delete
Save
Create
myDB2Database

FIG. 82

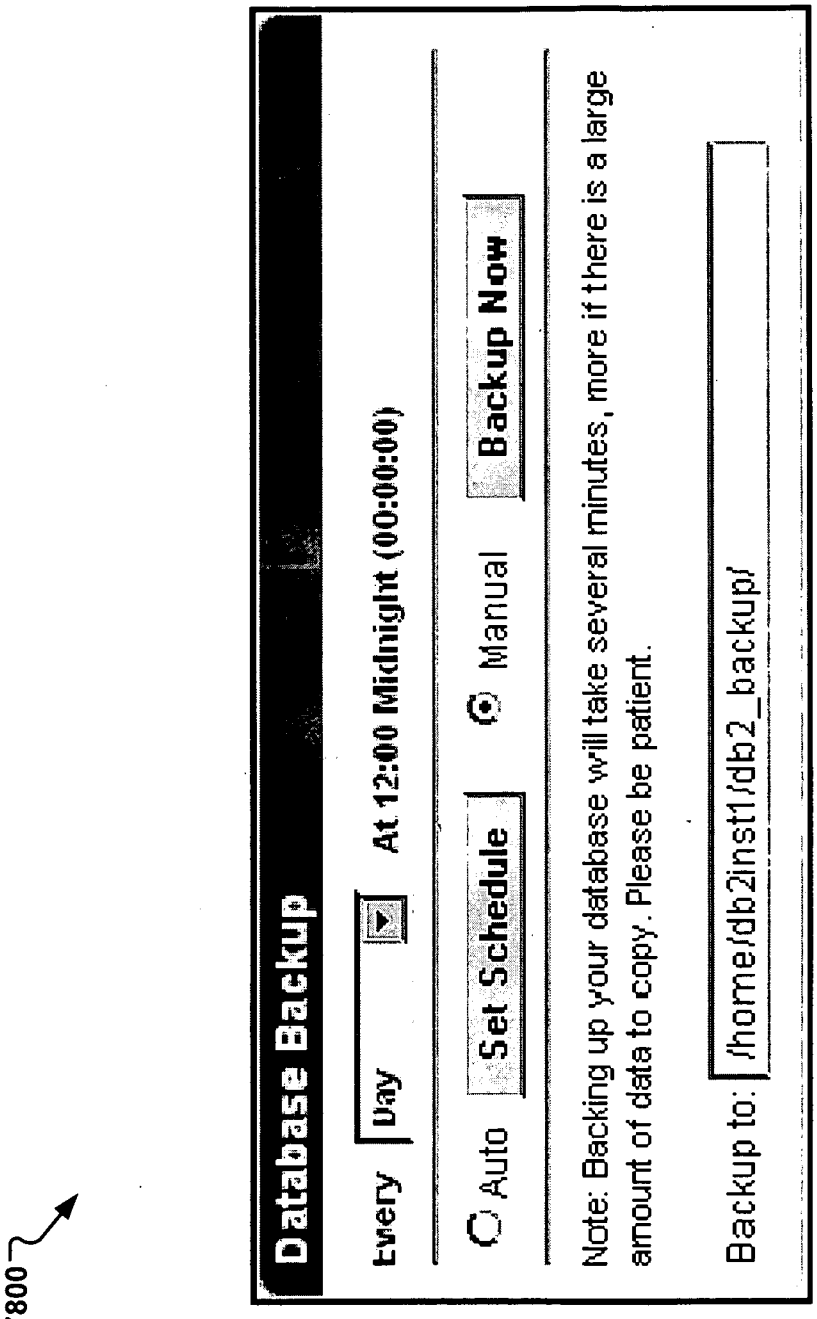

7800
Database Backup
Every Day
At 12:00 Midnight (00:00:00)
Auto
Set Schedule
Manual
Backup Now
Note: Backing up your database will take several minutes, more if there is a large amount of data to copy. Please be patient.
Backup to: /home/db2inst1/db2_backup/

FIG. 83

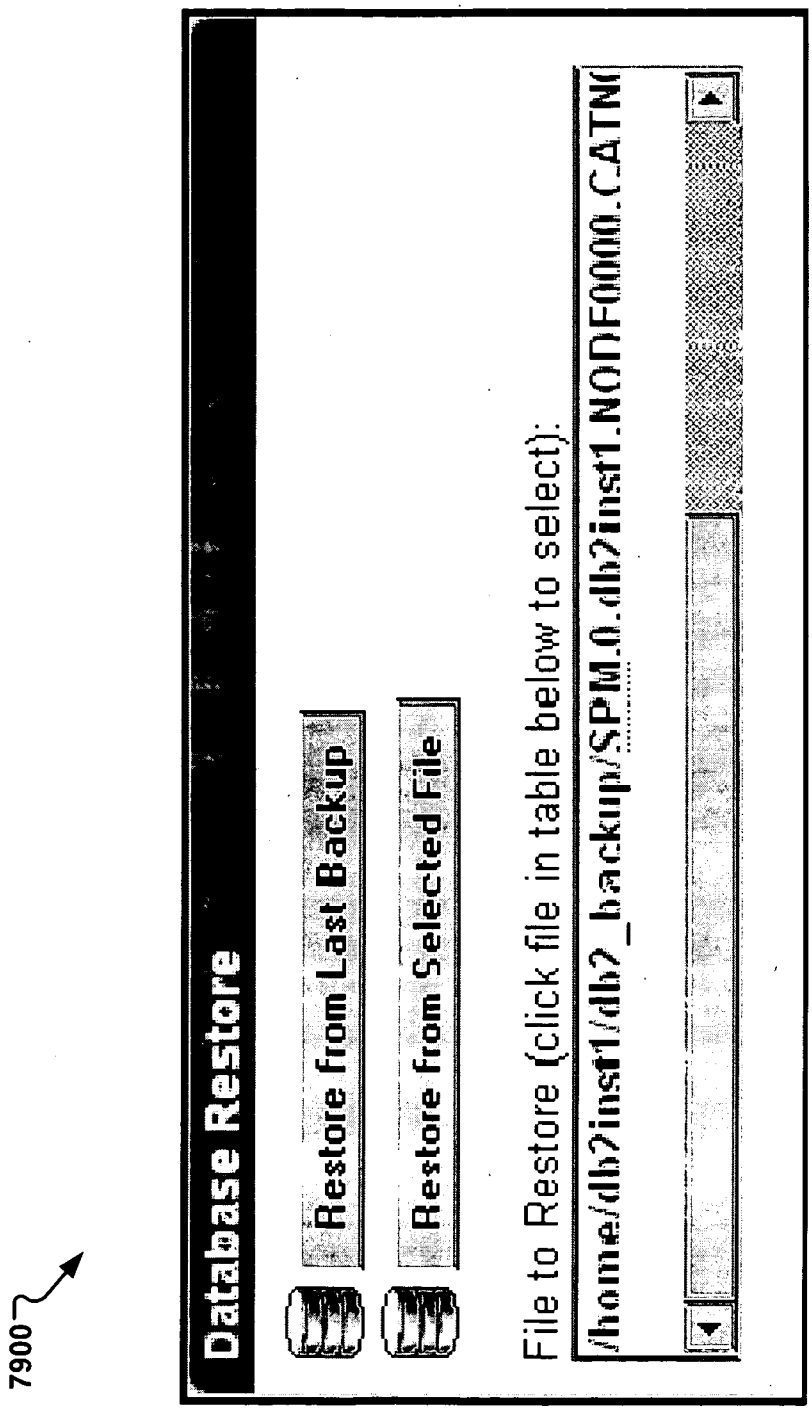

7900
Database Restore
Restore from Last Backup
Restore from Selected File
File to Restore (click file in table below to select):
/home/db2inst1/db2_backup/SPM.0.db2inst1.NODE0000.CATN

FIG. 84

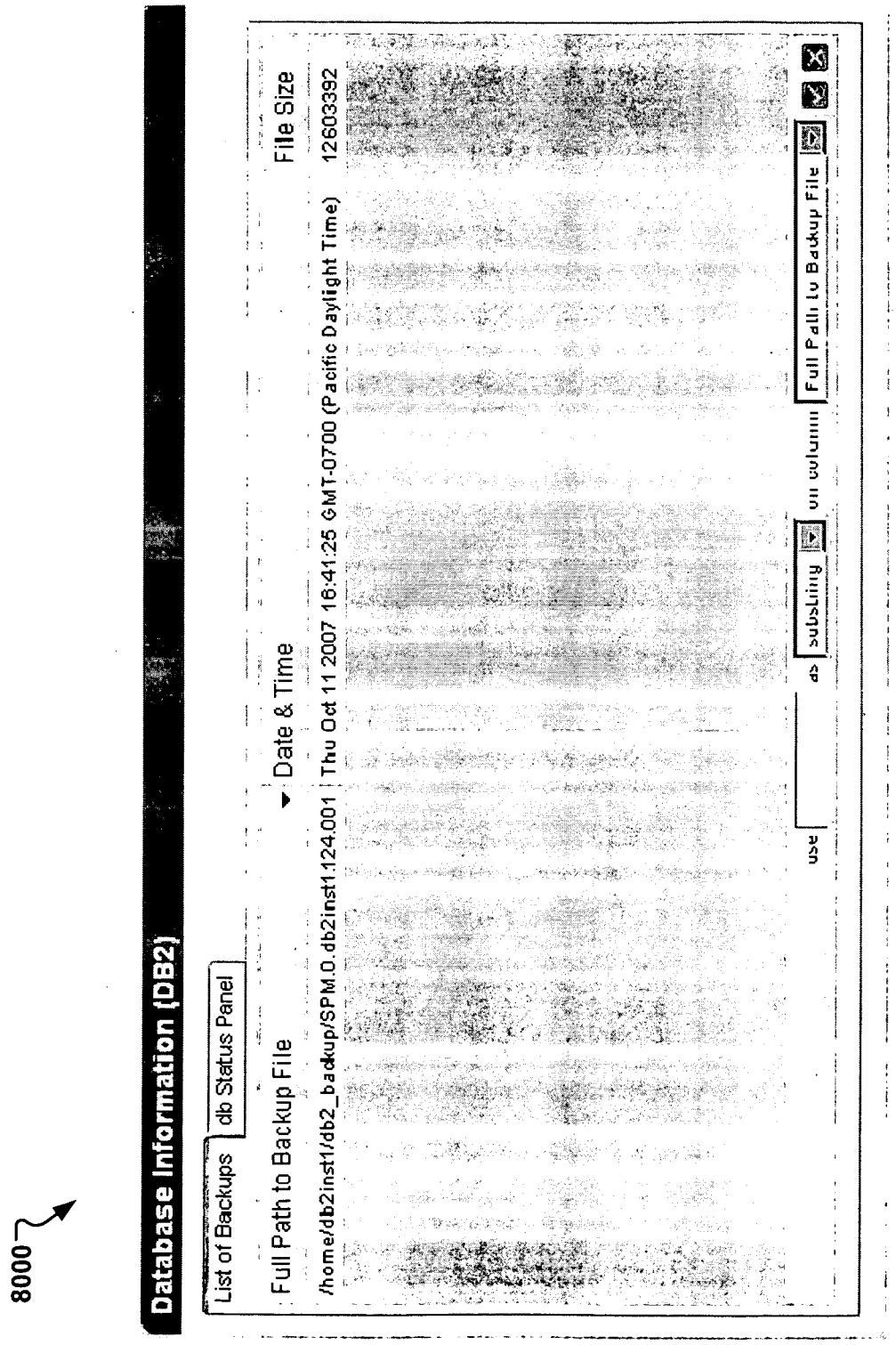

8000
Database Information (DB2)
List of Backups
db Status Panel
Full Path to Backup File
Date & Time
File Size
/home/db2inst1/db2_backup/SPM.0.db2inst1.124.001
Thu Oct 11 2007 16:41:25 GMT-0700 (Pacific Daylight Time)
12603392
use
substring
Full Path to Backup File

FIG. 85

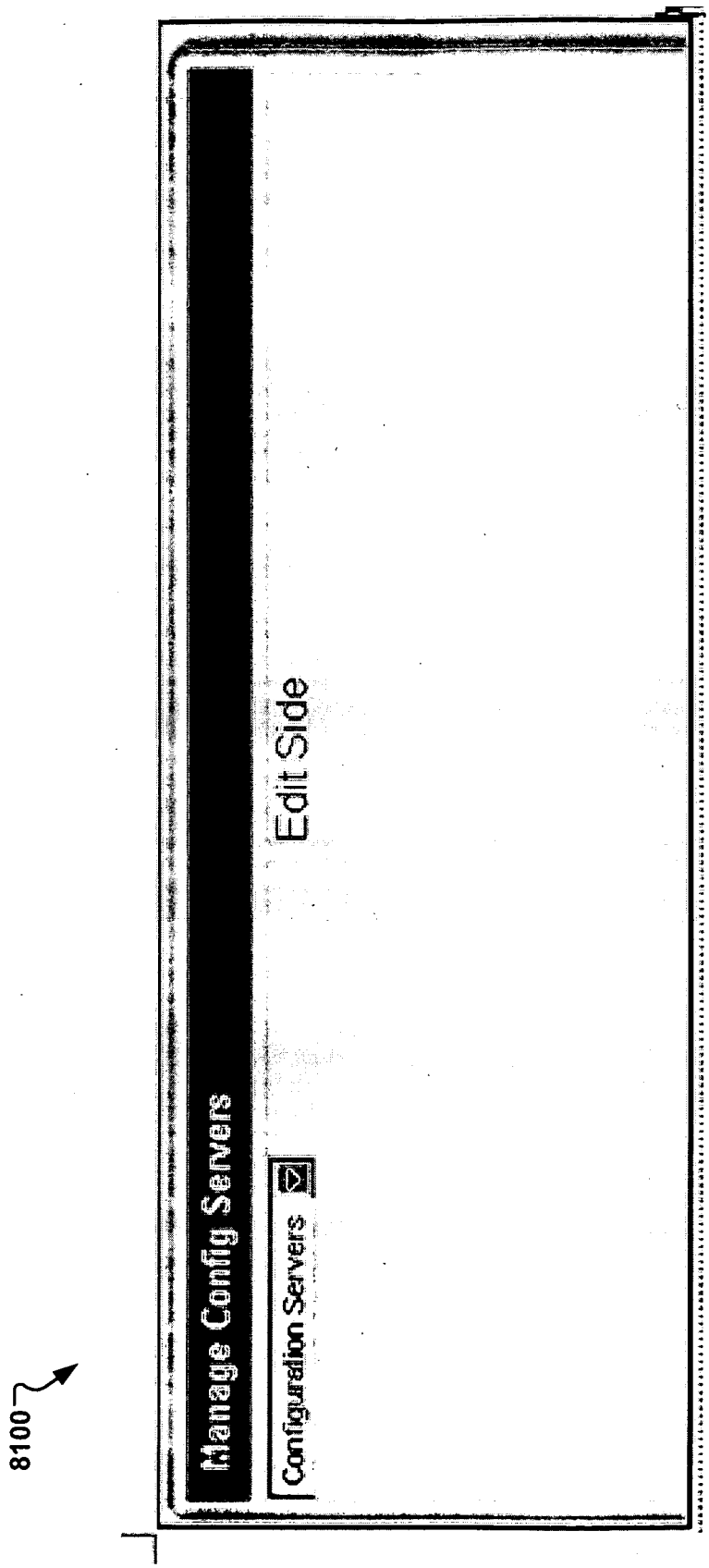

8100
Manage Config Servers
Configuration Servers
Edit Side

FIG. 86

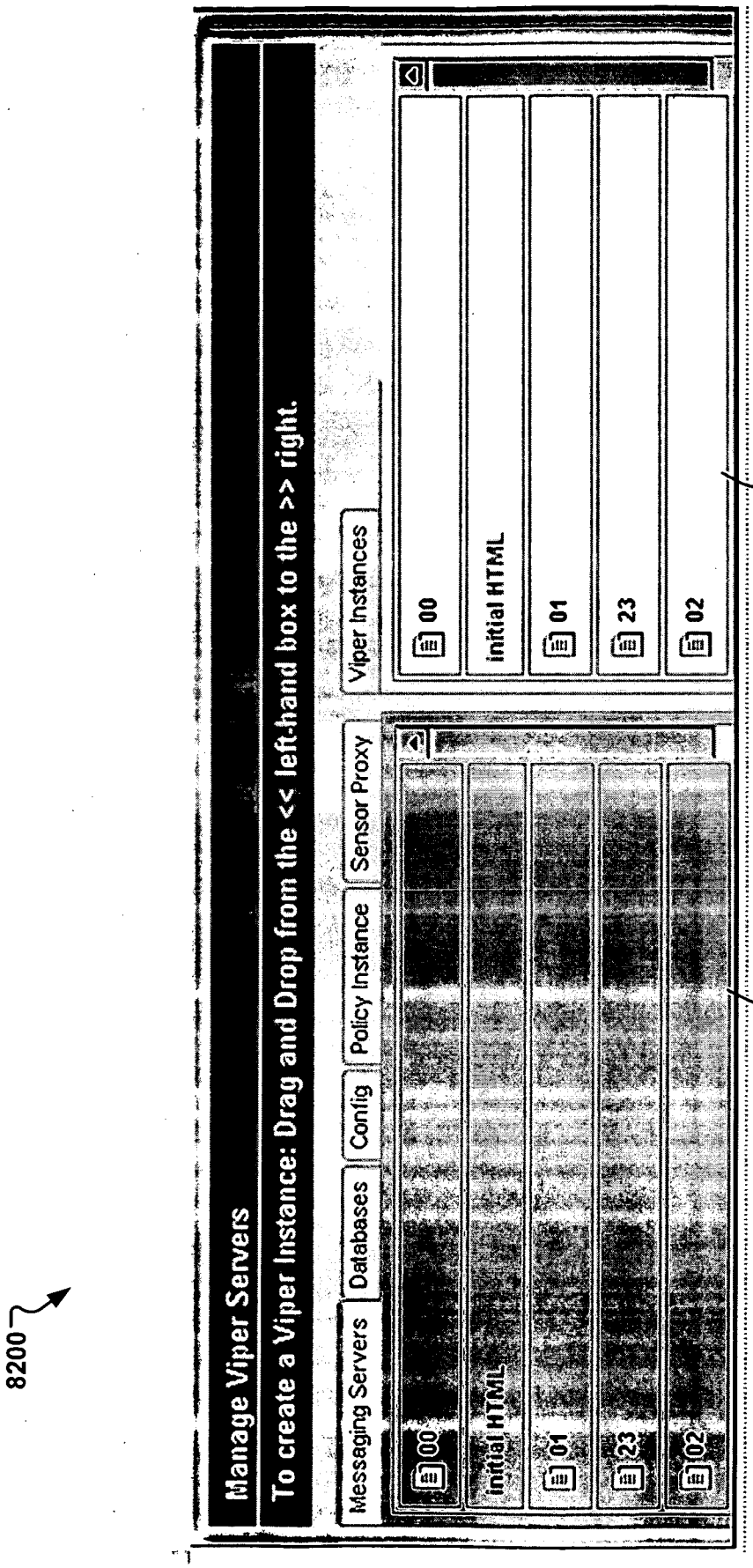

8200
Manage Viper Servers
To create a Viper Instance: Drag and Drop from the << left-hand box to the >> right.
Messaging Servers
Databases
Config
Policy Instance
Sensor Proxy
Viper Instances
00
initial HTML
01
23
02
8210
8220

FIG. 87

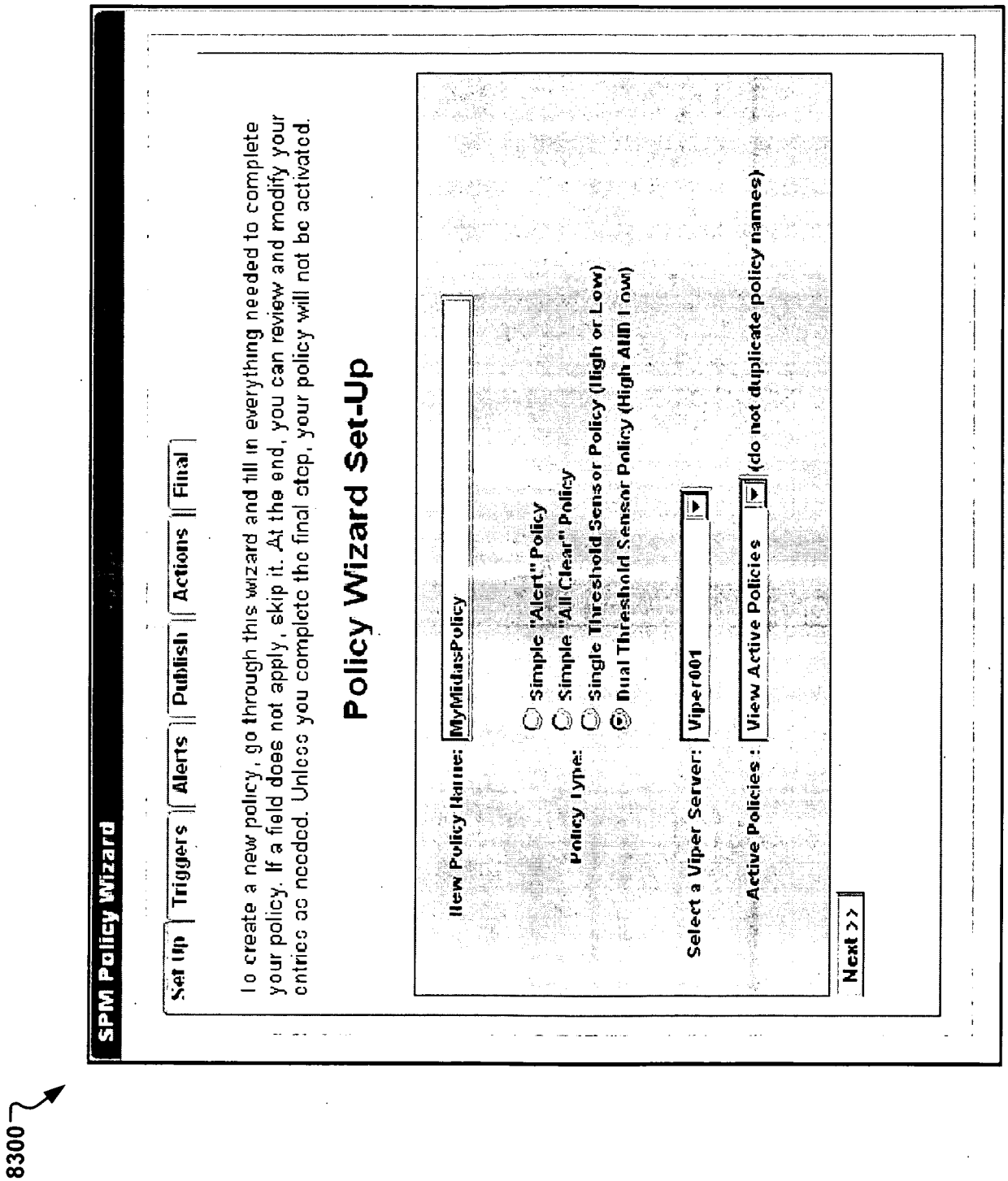

8300
SPM Policy Wizard
Set Up
Triggers
Alerts
Publish
Actions
Final
To create a new policy, go through this wizard and fill in everything needed to complete your policy. If a field does not apply, skip it. At the end, you can review and modify your entries as needed. Unless you complete the final step, your policy will not be activated.
Policy Wizard Set-Up
New Policy Name:
MyMidasPolicy
Policy Type:
Simple "Alert" Policy
Simple "All Clear" Policy
Single Threshold Sensor Policy (High or Low)
Dual Threshold Sensor Policy (High AND Low)
Select a Viper Server:
Viper001
Active Policies :
View Active Policies
(do not duplicate policy names)
Next >>

FIG. 88

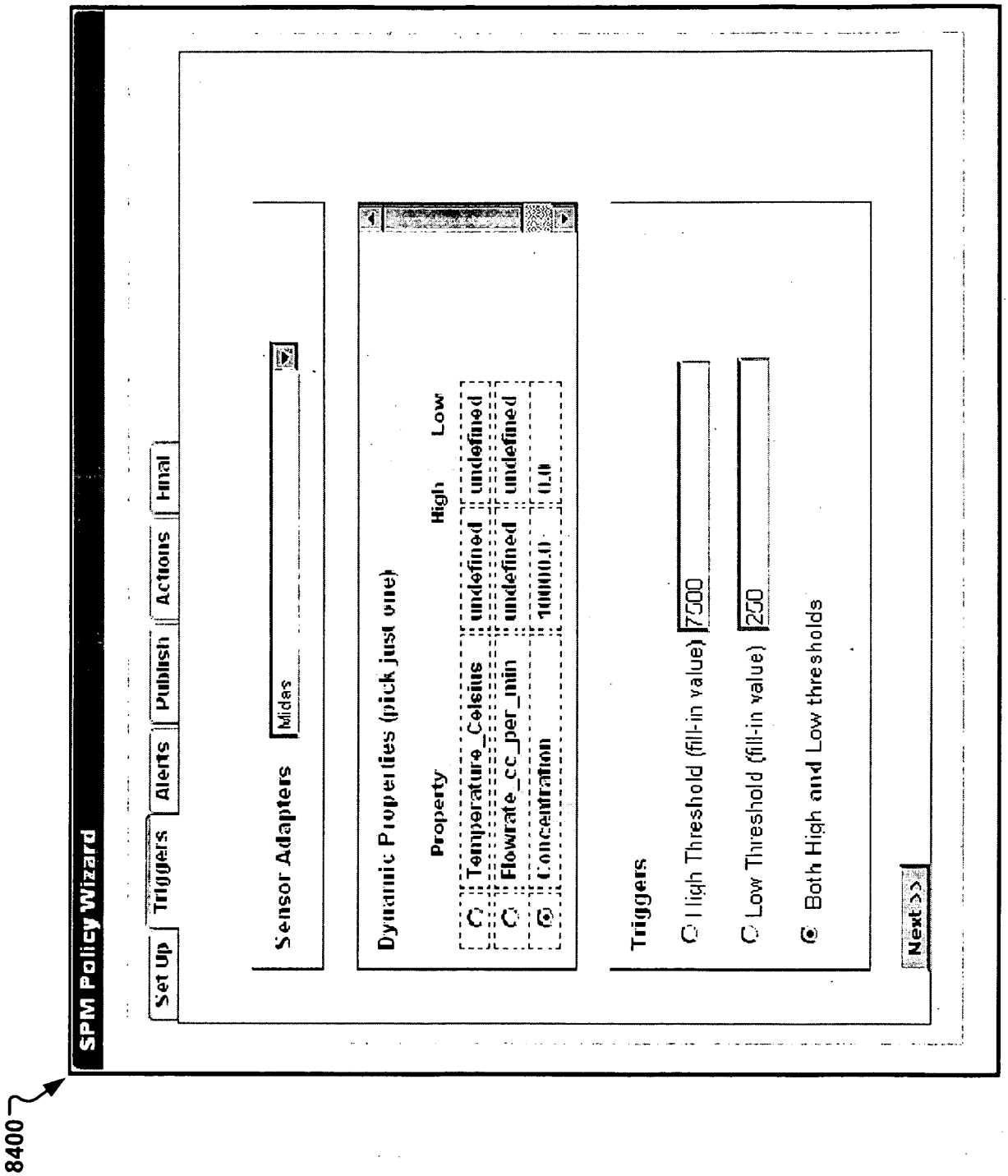

8400
SPM Policy Wizard
Set Up
Triggers
Alerts
Publish
Actions
Final
Sensor Adapters
Midas
Dynamic Properties (pick just one)
Property
High
Low
Temperature_Celsius
undefined
undefined
Flowrate_cc_per_min
undefined
undefined
Concentration
Triggers
High Threshold (fill-in value)
7500
Low Threshold (fill-in value)
250
Both High and Low thresholds
Next >>

FIG. 89

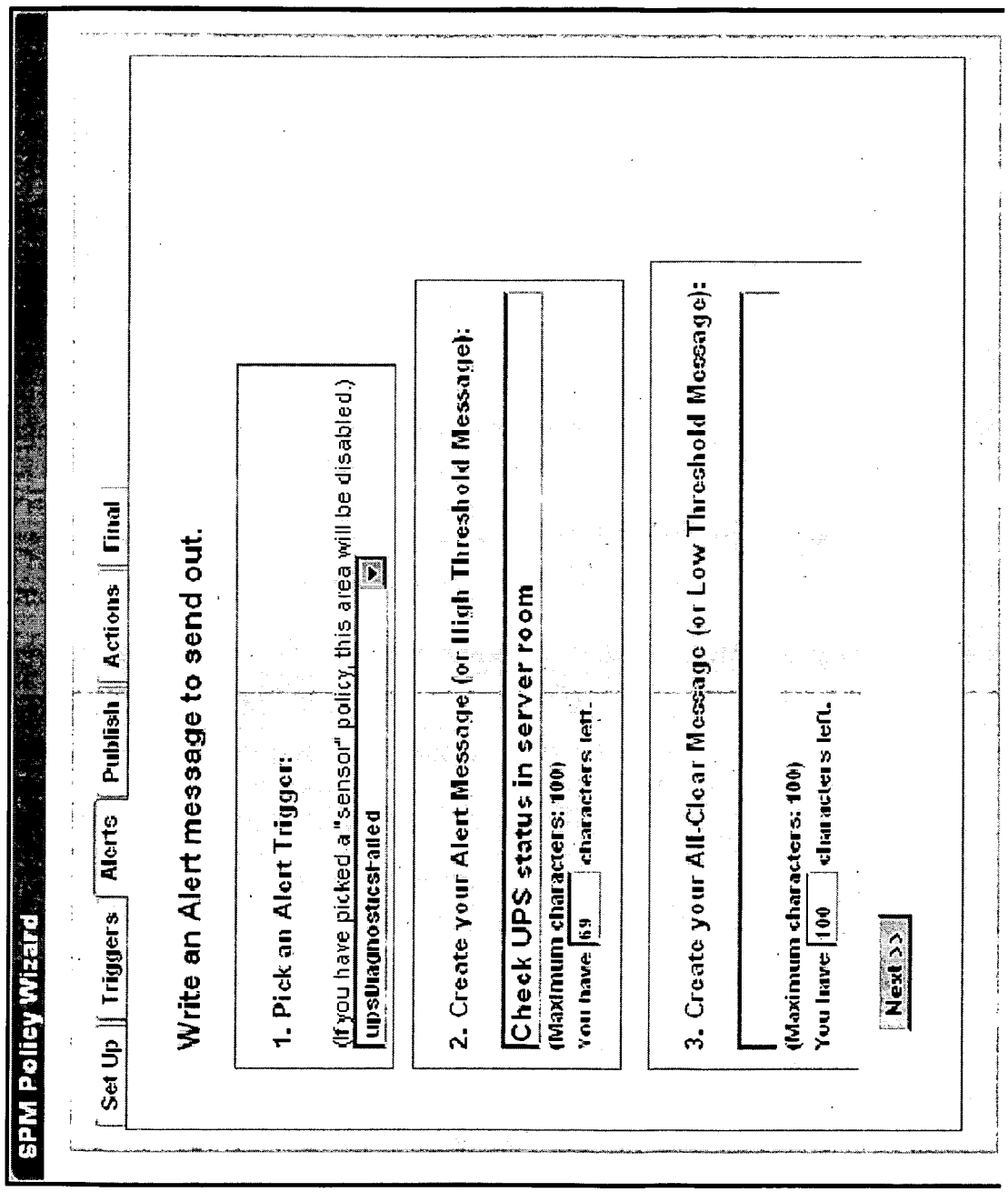

8500
SPM Policy Wizard
Set Up
Triggers
Alerts
Publish
Actions
Final
Write an Alert message to send out.
1. Pick an Alert Trigger:
(If you have picked a "sensor" policy, this area will be disabled.)
upsDiagnosticsFailed
2. Create your Alert Message (or High Threshold Message):
Check UPS status in server room
(Maximum characters: 100)
You have 69 characters left.
3. Create your All-Clear Message (or Low Threshold Message):
(Maximum characters: 100)
You have 100 characters left.
Next >>

FIG. 90

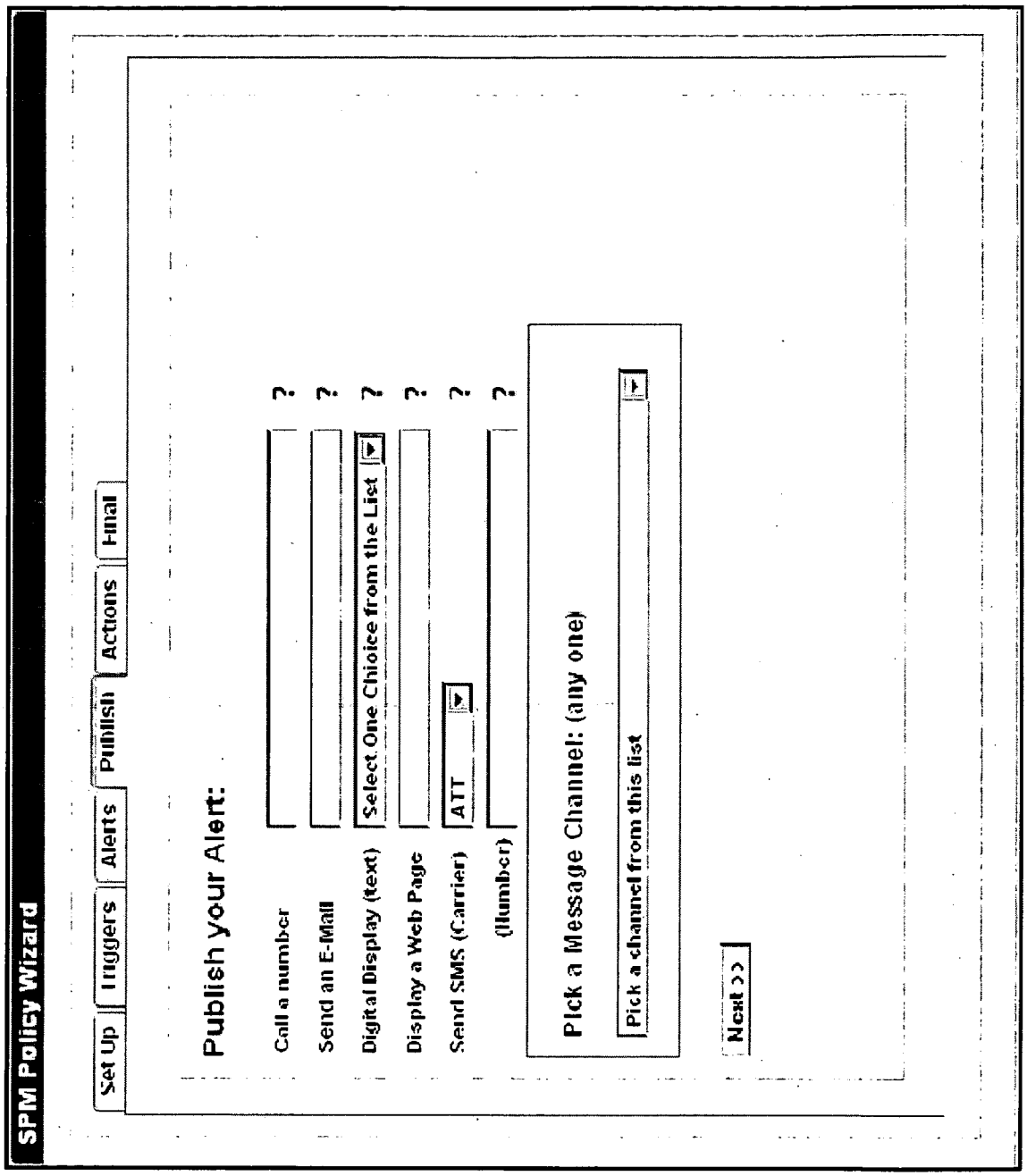

8600
SPM Policy Wizard
Set Up
Triggers
Alerts
Publish
Actions
Final
Publish your Alert:
Call a number
Send an E-Mail
Digital Display (text)
Select One Chioice from the List
Display a Web Page
Send SMS (Carrier)
ATT
(Number)
Pick a Message Channel: (any one)
Pick a channel from this list
Next >>

FIG. 91

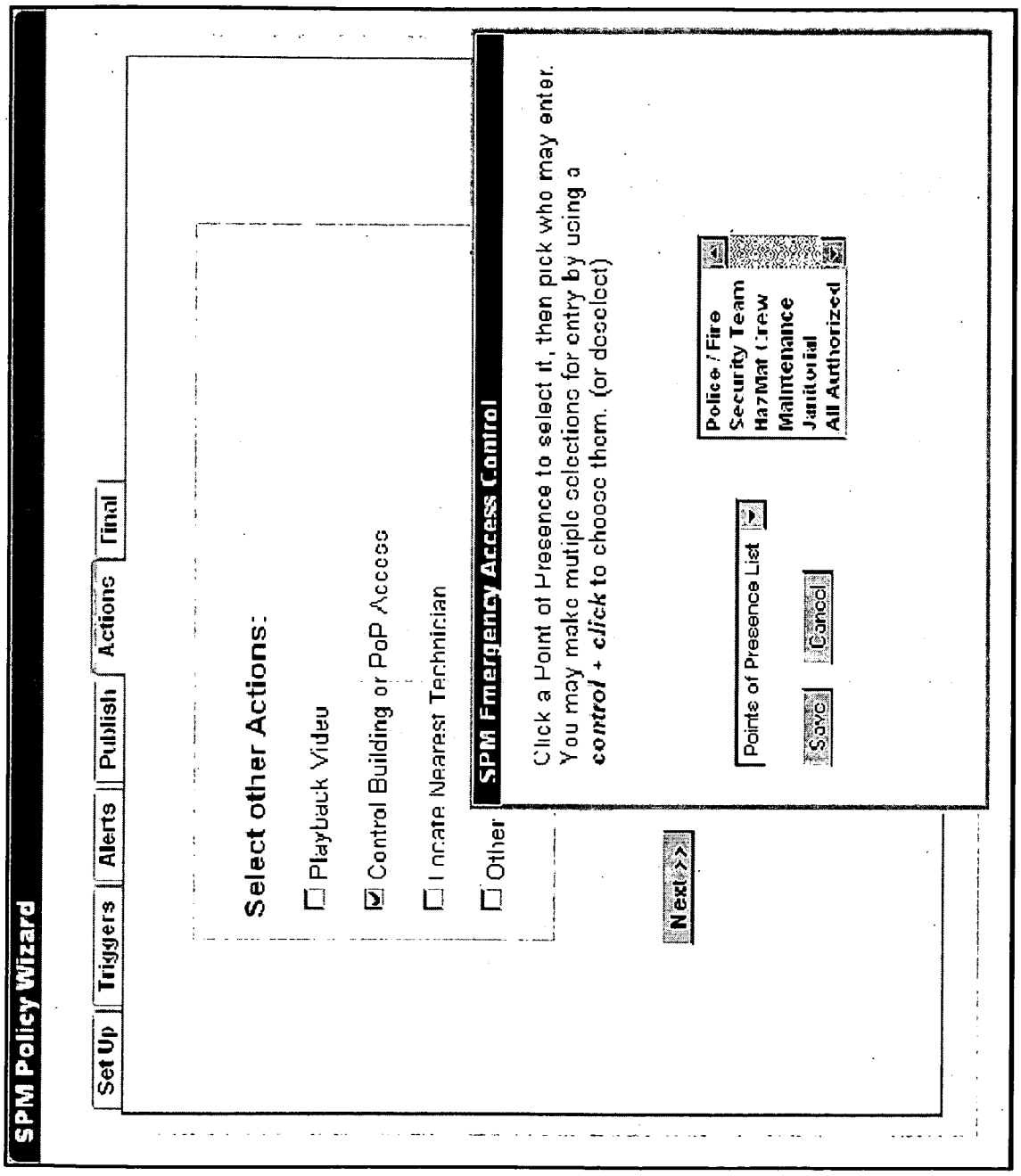

8700
SPM Policy Wizard
Set Up
Triggers
Alerts
Publish
Actions
Final
Select other Actions:
Playback Video
Control Building or PoP Access
Locate Nearest Technician
Other
Next >>
SPM Emergency Access Control
Click a Point of Presence to select it, then pick who may enter.
You may make multiple selections for entry by using a
control + click to choose them. (or deselect)
Points of Presence List
Save
Cancel
Police / Fire
Security Team
HazMat Crew
Maintenance
Janitorial
All Authorized

FIG. 92

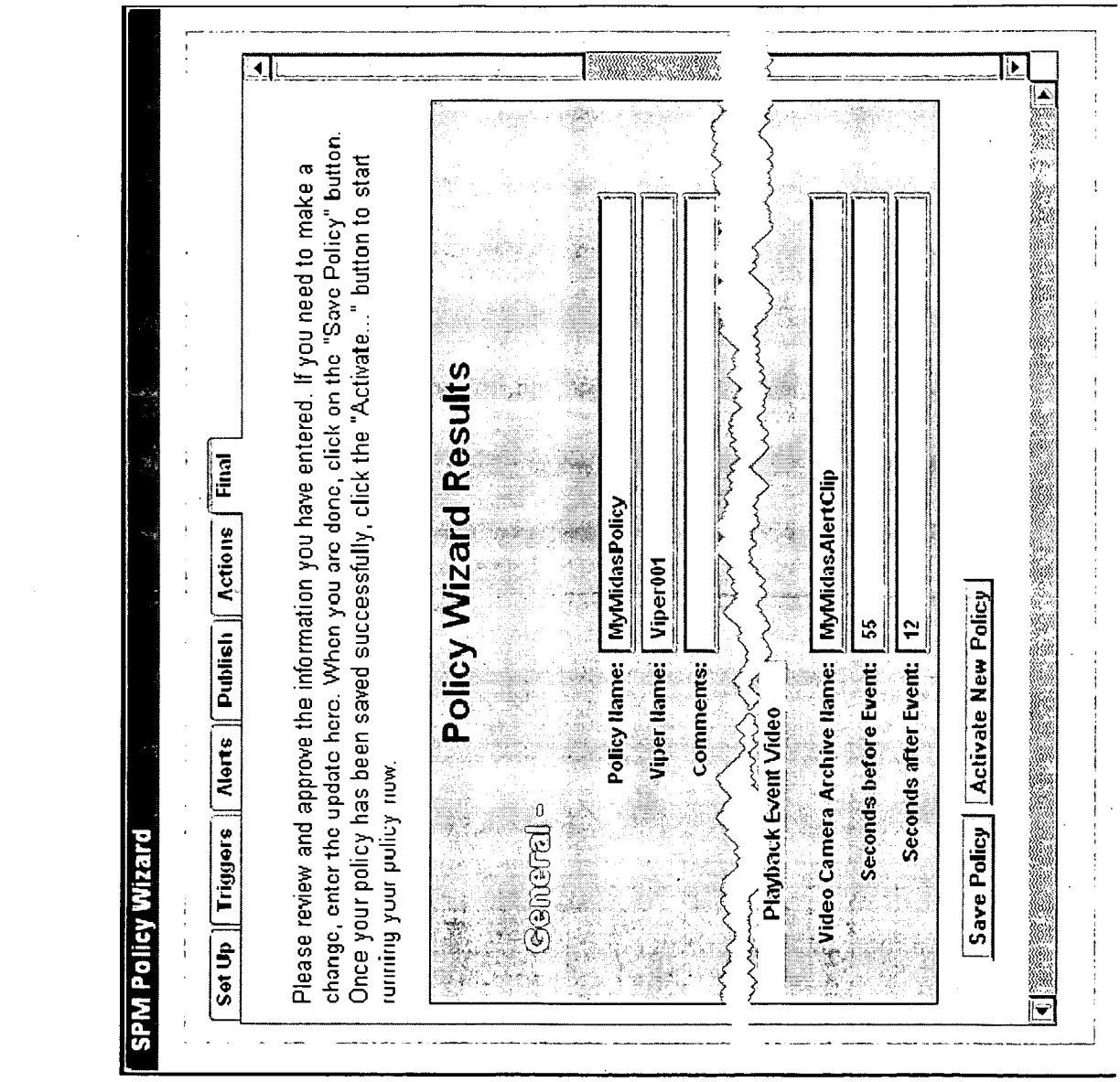

8800
SPM Policy Wizard
Set Up
Triggers
Alerts
Publish
Actions
Final
Please review and approve the information you have entered. If you need to make a change, enter the update here. When you are done, click on the "Save Policy" button. Once your policy has been saved successfully, click the "Activate..." button to start running your policy now.
Policy Wizard Results
General
Policy Name:
MyMidasPolicy
Viper Name:
Viper001
Comments:
Playback Event Video
Video Camera Archive Name:
MyMidasAlertClip
Seconds before Event:
55
Seconds after Event:
12
Save Policy
Activate New Policy

FIG. 93

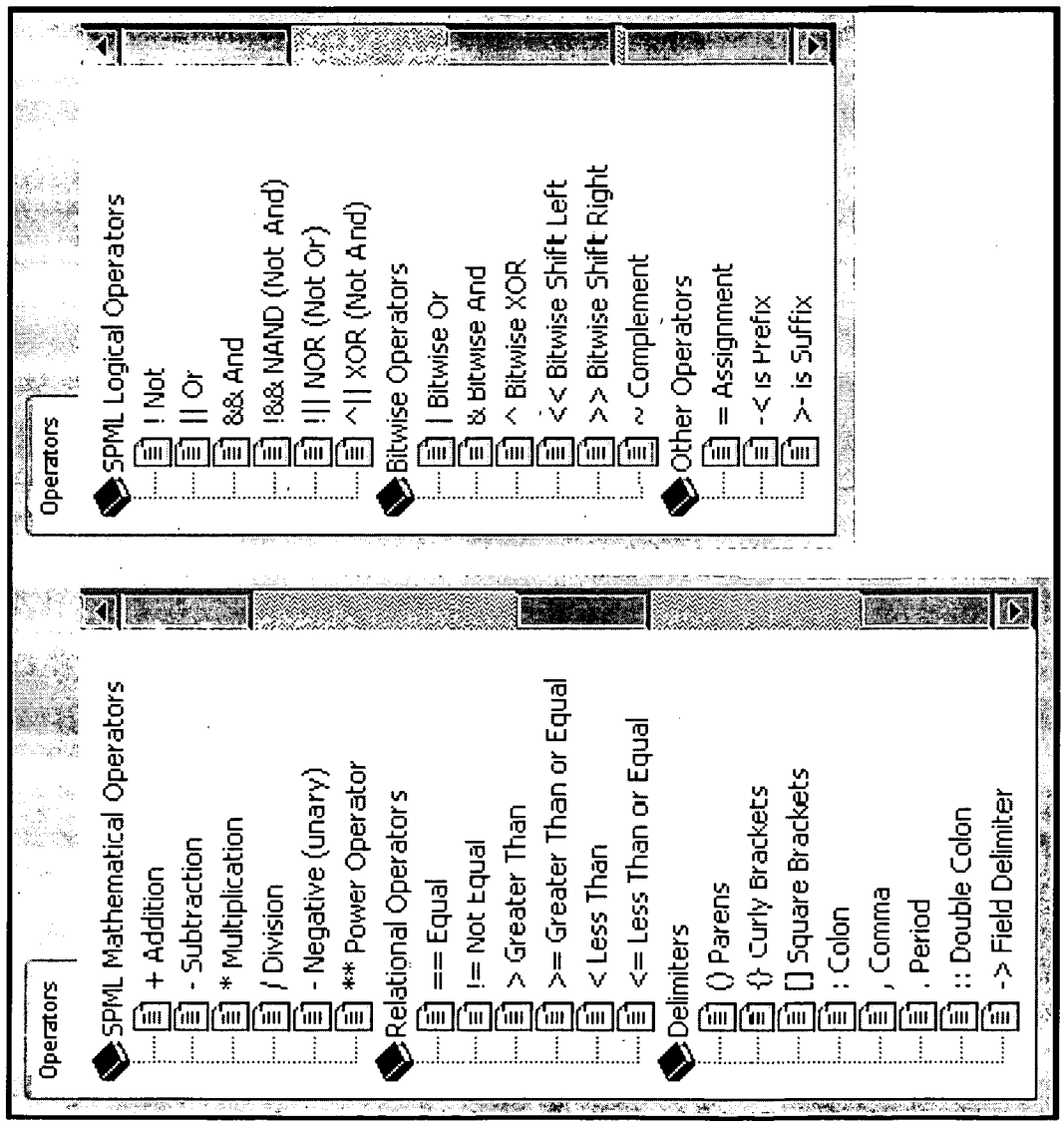

Operators
SPML Mathematical Operators
+ Addition
- Subtraction
* Multiplication
/ Division
- Negative (unary)
** Power Operator
Relational Operators
== Equal
!= Not Equal
> Greater Than
>= Greater Than or Equal
< Less Than
<= Less Than or Equal
Delimiters
() Parens
{} Curly Brackets
[] Square Brackets
: Colon
, Comma
. Period
:: Double Colon
-> Field Delimiter
Operators
SPML Logical Operators
! Not
|| Or
&& And
!&& NAND (Not And)
!|| NOR (Not Or)
^|| XOR (Not And)
Bitwise Operators
| Bitwise Or
& Bitwise And
^ Bitwise XOR
<< Bitwise Shift Left
>> Bitwise Shift Right
~ Complement
Other Operators
= Assignment
-< is Prefix
>- is Suffix

FIG. 94

Operators	Type	Name or Synonym	Description
+	Mathematical	Addition	Sums a and b
+	Mathematical	Concatenation	Concatenation if either operand is a string
-	Mathematical	Subtraction	Subtracts b from a
-	Mathematical	Negation (unary)	Changes sign of operand
*	Mathematical	Multiplication	Multiplies a by b
/	Mathematical	Division	Divides a by b
**	Mathematical	POWER	Raises a to power of b
==	Relational	Equal	Returns true if a and b are equal; false otherwise
!=	Relational	Not Equal	Returns true if a and b are not equal; false otherwise
>	Relational	Greater Than	Returns true if a is greater than b; returns false otherwise
>=	Relational	Greater Than Or Equal	Returns true if a is greater than b or if both a and b are equal; returns false otherwise
<	Relational	Less Than	Returns true if a is less than b; returns false otherwise
<=	Relational	Less Than Or Equal	Returns true if a is less than b or if both a and b are equal; returns false otherwise
( )	Delimiter	Parentheses	Delimits Function Parameters
{ }	Delimiter	Curly Brackets	Delimits Structures
[ ]	Delimiter	Square Brackets	Delimits Arrays
:	Delimiter	Colon	Equivalence in type maps and bindings
,	Delimiter	Comma	Separates series of values, operands or parameters
.	Delimiter	Period	Delimits names in a distinguished name
::	Delimiter	Double Colon	Scope delimiter, e.g., MsgServerName::ChannelName
->	Delimiter	Field Delimiter	Field Delimiter, e.g., ChannelName->FieldName

FIG. 95

Operators	Type	Name or Synonym	Description
->	Delimiter	Field Delimiter	Field Delimiter, e.g., ChannelName->FieldName
!	Logical	NOT (Unary)	Negation
||	Logical	OR	Returns true if either a or b is true
&&	Logical	AND	Returns true if and only if both a and b are true
!&&	Logical	NAND	Returns false if and only if both a and b are true
!||	Logical	NOR	Returns true if and only if both a and b are false
x||	Logical	Exclusive Or; XOR	Returns true if and only if exactly one operand is true
|	Bitwise	Bitwise OR	10011 | 10100 -> 10111
&	Bitwise	Bitwise AND	10011 & 10100 -> 10000
^	Bitwise	Bitwise XOR	10011 ^ 10100 -> 00111
<<	Bitwise	Bitwise Shift Left	Shifts operand a the specified number of bits to the left. a is a 4-byte integer. b specifies the number of bits to shift. Bits shifted out to the left are discarded. New bits coming in from the right are zeros. Left shifting a by b places is the same as multiplying it by 2^b. Sign is preserved.
>>	Bitwise	Bitwise Shift Right	Shifts operand a the specified number of bits to the right. Bits shifted out to the right are discarded. New bits coming in from the left are zeros for positive numbers or ones for negative numbers. Right shifting by b places is the same as dividing by 2^b. Sign is preserved.
~	Bitwise	Ones' Complement or Bitwise NOT (unary)	Performs logical negation on each bit. All zeros become ones, and vice versa.
=	Misc	Assignment, Equal	Assigns value of b to a
-<	Misc	isPrefix	Returns true if b is a prefix of a
>-	Misc	isSuffix	Returns true if b is a suffix of a
1) In binary operations, "a" and "b" refer to the left (first) and right (second) operands, respectively.			

FIG. 96

Keyword¤	Keyword¤	Keyword¤
action_template¤	field_bindings¤	register_reference¤
alert_channels¤	filter¤	remote_viper_servers¤
alert_message¤	host¤	SDO¤
boolean¤	ICD100¤	sensors¤
CAP1_1¤	input_channels¤	sensor_adapters¤
channel¤	integer¤	sensor_adapter_instance¤
channels¤	jms_message_server¤	sensor_adapter_template¤
channel_bindings¤	local_message_server¤	sensor_bindings¤
channel_reference	long	set
comm_alarm_channels	message_protocol	SML
comm_alarm_message	message_servers	snmp_message_server
condition_template	message_structure_map	soap_message_server
configuration_server	message_type	soap_server_port
constants	message_types	string
constant_reference	message_value_map	tcp_message_server
control	output_channels	tcp_server_port
data_channels	persistent	template
data_message	policies	then
depth	policy_instance	timespan
double	policy_template	trigger
else	process_manager	true
enum	process_managers	type
equipment_alarm_channels	properties	type_structure_map
equipment_alarm_message	protocol	type_value_map
event_server	publish	viper_server
event_servers	reference_channels	viper_servers
false	registers	
fi	register_bindings	

FIG. 97

Function	Usage	Comments
LOWER_CASE(str)	Converts all letters in str to lower case letters.	
UPPER_CASE(str)	Converts all letters in str to upper case letters.	
CONTAINS_ANY_STRING(str1, str2, str3, ...)	Returns true if str1 contains any of the subsequent strings.	
CONTAINS_ALL_STRINGS(str1, str2, str3, ...)	Returns true if str1 contains all of the subsequent strings.	
SQRT(num)	Returns sqrt of num.	NEED TO IMPLEMENT!!!
TIMESTAMP_GAP_SEC(timestamp1, timestamp2)	Returns timestamp1 - timestamp2 (result is in seconds).	Both parameters are strings representing a timestamp (number of milliseconds since 1/1/1970 00:00:00 GMT).
FORMAT_TIME(timestamp)	Converts timestamp to a string in the form "%H:%M:%S"	
COMPOSITE_VIDEO(videoFrame1, videoFrame2, ...)	Combines all provided video frames (base64 encoded strings) into a composite image (also a base64 encoded string).	
GEO_LOCATION_DISTANCE(geoLoc1, geoLoc2)	Calculates the distance in miles between geoLocations geoLoc1 and geoLoc2.	Employs a function called calculateGeographicDistance() which is in \ViaServiceDataObject\src\ViaSdoUtil.cpp
FIND_CLOSEST_CAMERA(targetGeoLoc, camera1GeoLoc, camera2GeoLoc)	Using the provided geoLocations, calculates the camera which is closest to the target.	This should be made generic, e.g., FIND_CLOSEST_TO_LOCATION(targetGeoLoc, geoLoc1, geoLoc2, geoLoc3, ...).
UUID( )	Returns the UUID of the ????	

FIG. 98

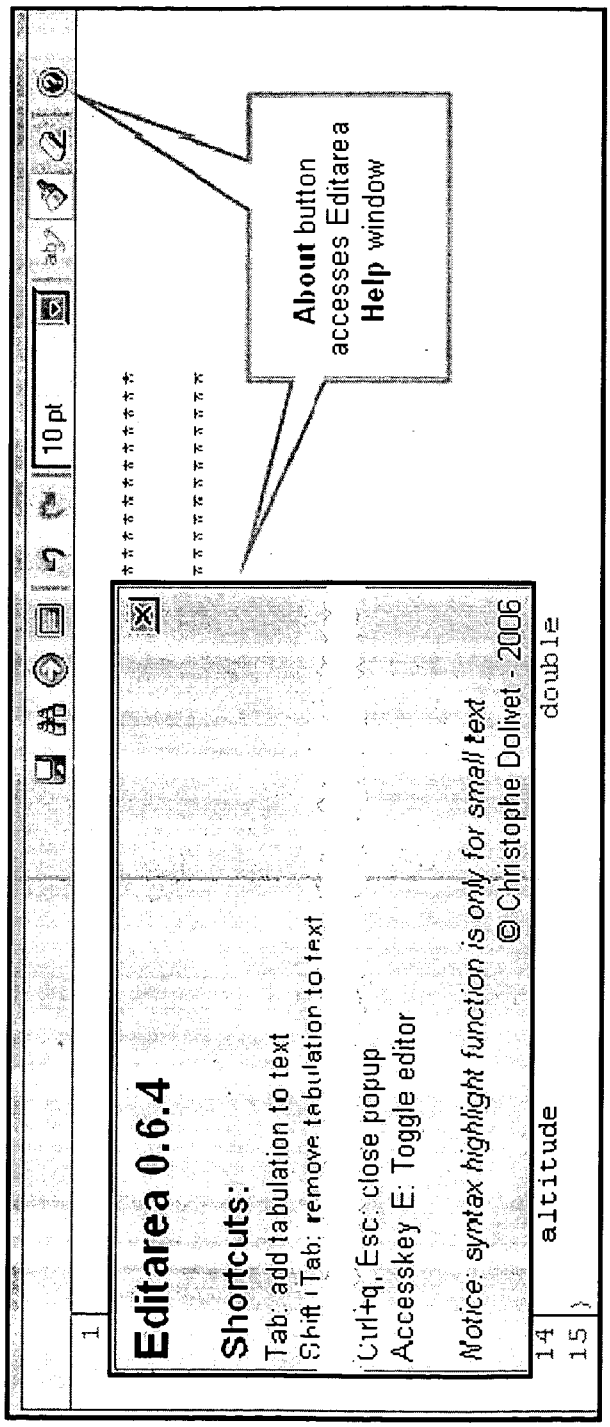

Editarea 0.6.4
Shortcuts:
Tab: add tabulation to text
Shift / Tab: remove tabulation to text
Ctrl+q, Esc: close popup
Accesskey E: Toggle editor
Notice: syntax highlight function is only for small text
© Christophe Dolivet - 2006
About button accesses Editarea Help window
10 pt
altitude
double

FIG. 99

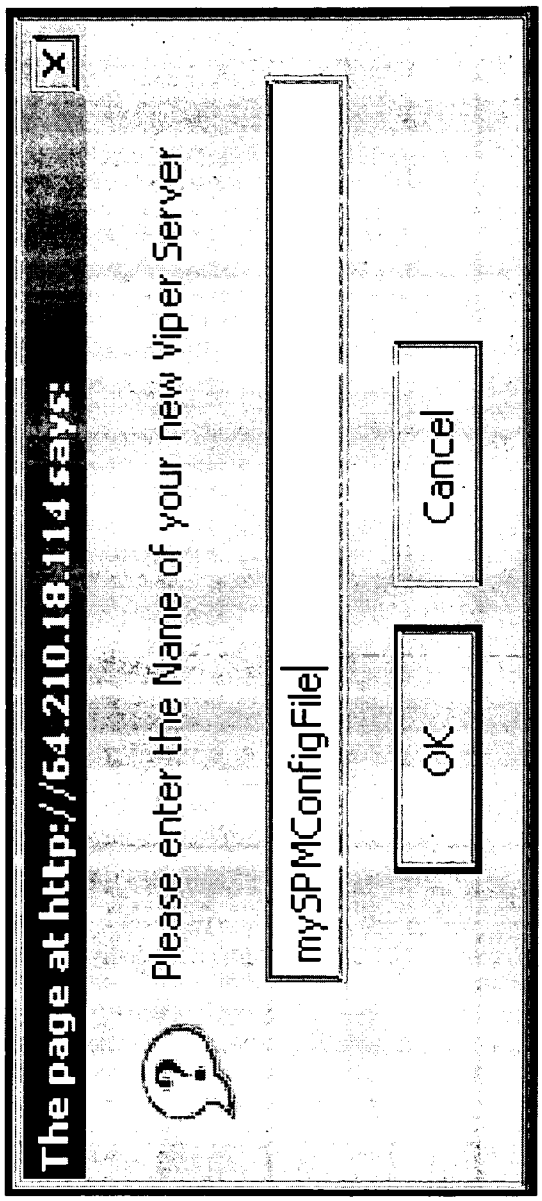

The page at http://64.210.18.114 says:
Please enter the Name of your new Viper Server
mySPMConfigFile
OK
Cancel

FIG. 100

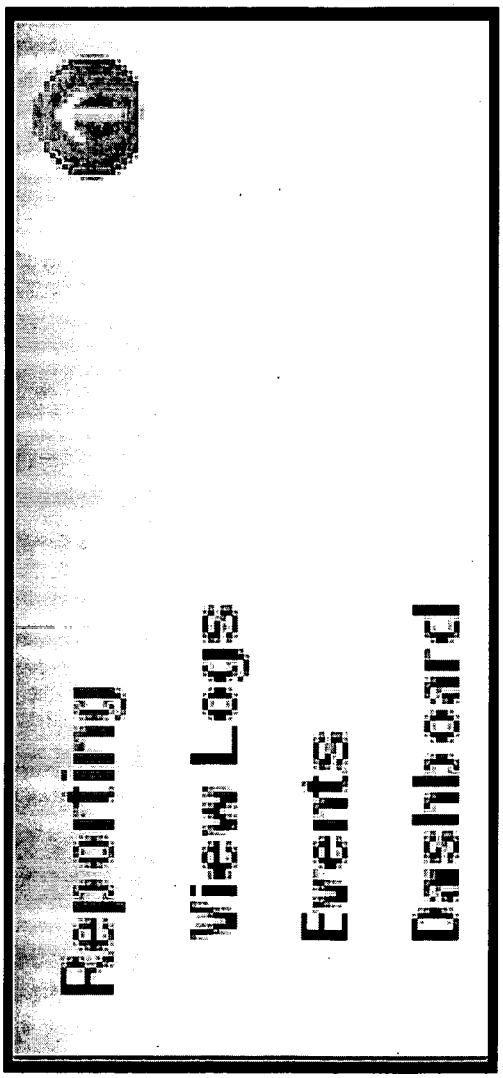


FIG. 101

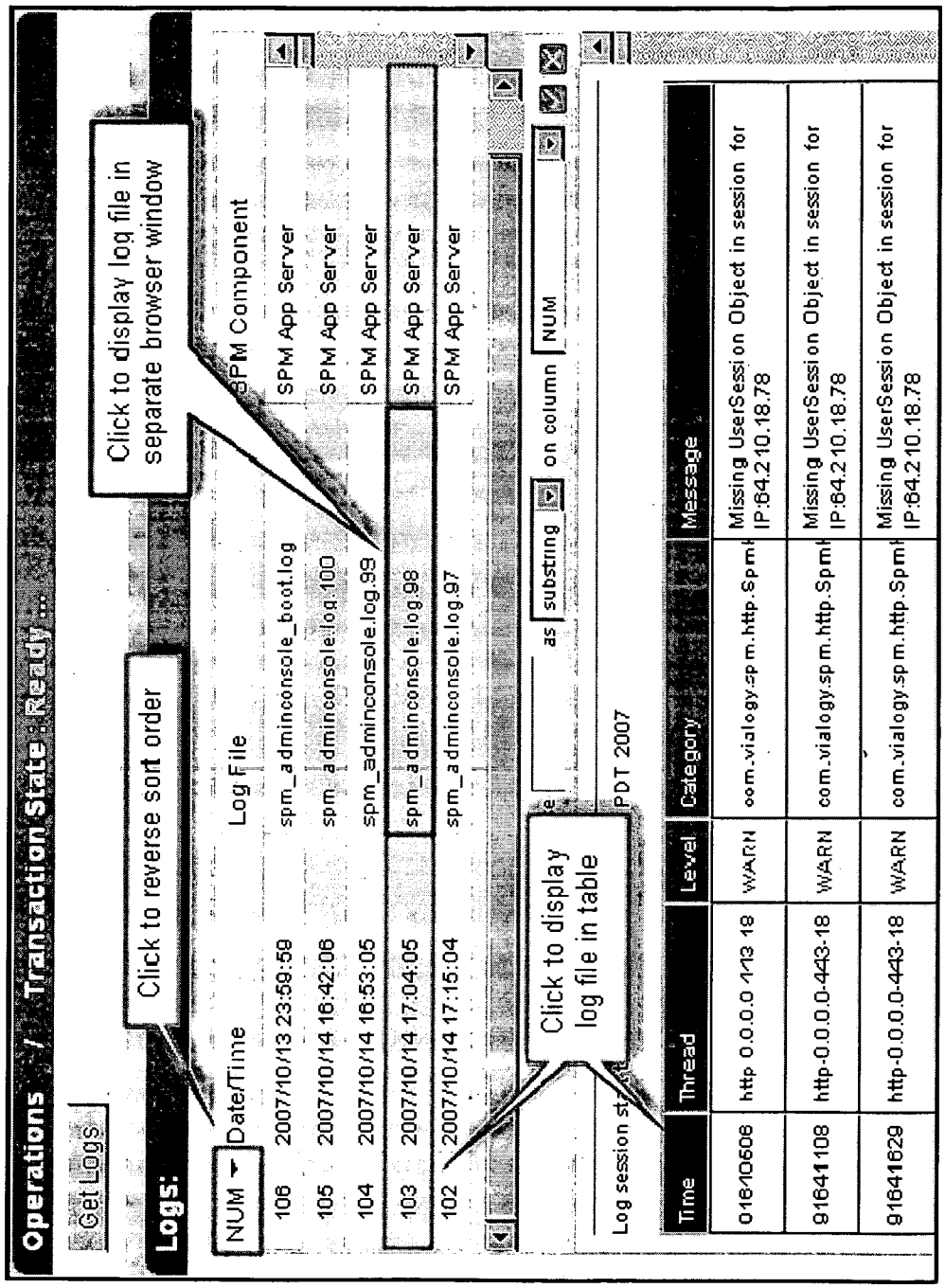

Operations / Transaction State: Ready ...
Get Logs
Logs:
Click to reverse sort order
Click to display log file in separate browser window
NUM
Date/Time
Log File
SPM Component
106 2007/10/13 23:59:59 spm_adminconsole_boot.log SPM App Server
105 2007/10/14 16:42:06 spm_adminconsole.log.100 SPM App Server
104 2007/10/14 16:53:05 spm_adminconsole.log.99 SPM App Server
103 2007/10/14 17:04:05 spm_adminconsole.log.98 SPM App Server
102 2007/10/14 17:15:04 spm_adminconsole.log.97 SPM App Server
Click to display log file in table
as substring on column NUM
PDT 2007
Time
Thread
Level
Category
Message
01610606 http 0.0.0.0 443 18 WARN com.vialogy.spm.http.Spml Missing UserSession Object in session for IP:64.210.18.78
91641108 http-0.0.0.0-443-18 WARN com.vialogy.spm.http.Spml Missing UserSession Object in session for IP:64.210.18.78
91641629 http-0.0.0.0-443-18 WARN com.vialogy.spm.http.Spml Missing UserSession Object in session for IP:64.210.18.78

FIG. 102

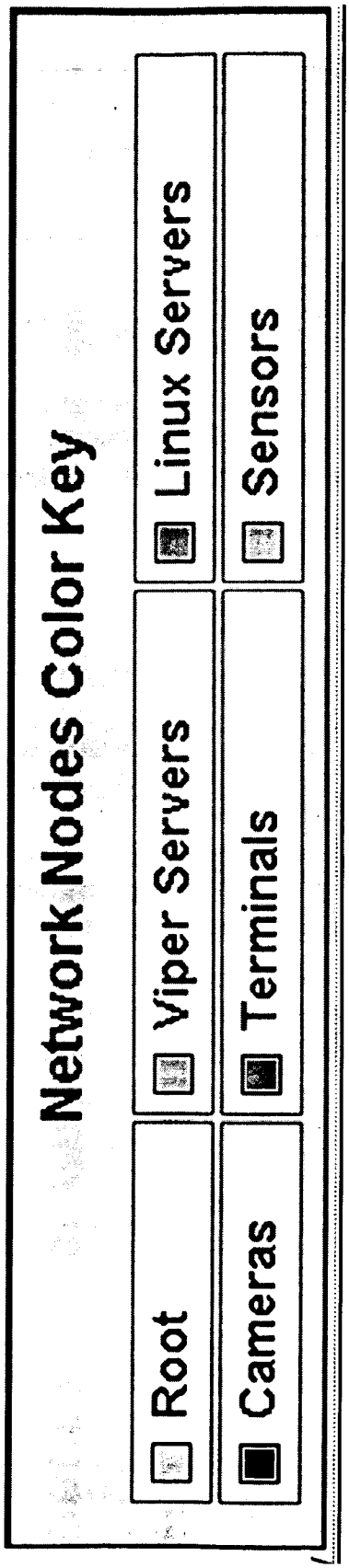

Network Nodes Color Key
Root
Viper Servers
Linux Servers
Cameras
Terminals
Sensors

FIG. 103

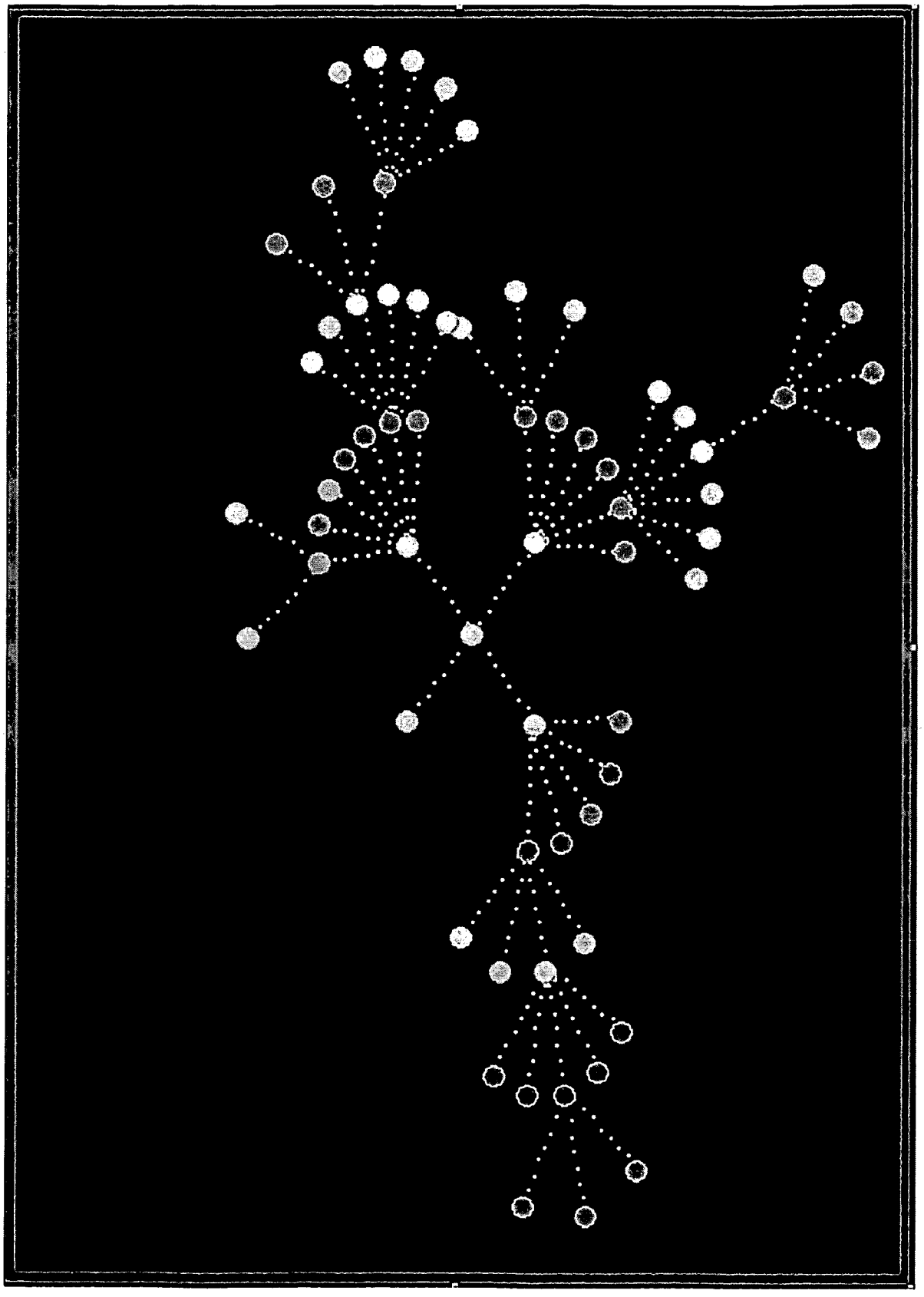


FIG. 104

Events - Page: 1 , Messages:(1 to 20) , From: JmsMsgServer.ChemAlertFireDepartment

Date/Time	Alert Level	Alert Message
03/10/2008 17:26:22:340 (GMT -7)	GREEN	SITUATION BACK TO NORMAL: AMMONIA LEVEL WITHIN LIMITS.
03/10/2008 17:26:27:397 (GMT -7)	RED	HIGH AMMONIA LEVEL ALERT.
03/10/2008 17:27:22:413 (GMT -7)	GREEN	SITUATION BACK TO NORMAL: AMMONIA LEVEL WITHIN LIMITS.
03/10/2008 17:27:27:420 (GMT -7)	RED	HIGH AMMONIA LEVEL ALERT.
03/10/2008 17:28:21:535 (GMT -7)	GREEN	SITUATION BACK TO NORMAL: AMMONIA LEVEL WITHIN LIMITS.

FIG. 105

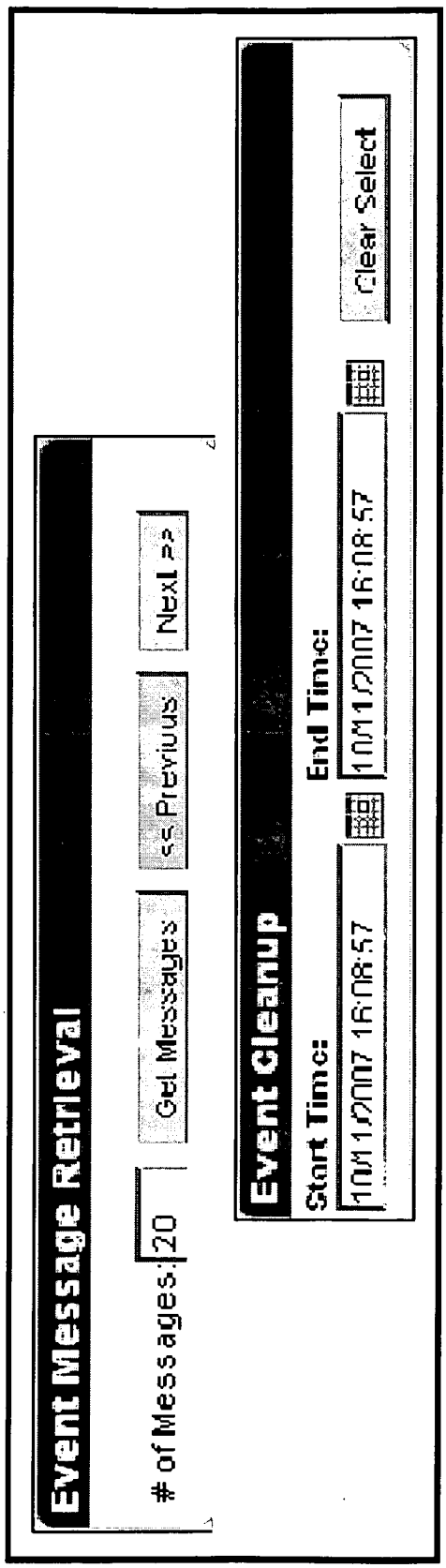

Event Message Retrieval
# of Messages: 20
Get Messages
<< Previous
Next >>
Event Cleanup
Start Time:
10/11/2007 16:08:57
End Time:
10/11/2007 16:08:57
Clear Select

FIG. 106

Events - Page: 1 , Messages:(1 to 20) . From: JmsMspServer.SPM.Alert.Sensor.Comm.Alarms

Date/Time	Alert Level	Alert Message
03/10/2008 16:36:45:399 (GMT -7)	GREEN	Connection restored (Viper001::Canberra, Type: Canberra)
03/10/2008 16:36:51:677 (GMT -7)	RED	Connection down (MaxConnectFailures exceeded) (Viper001::Canberra,
03/10/2008 16:36:54:686 (GMT -7)	GREEN	Connection restored (Viper001::Canberra, Type: Canberra)
03/10/2008 16:37:07:014 (GMT -7)	RED	Connection down (MaxConnectFailures exceeded) (Viper001::Canberra,
03/10/2008 16:37:10:025 (GMT -7)	GREEN	Connection restored (Viper001::Canberra, Type: Canberra)
03/10/2008 16:38:17:341 (GMT -7)	GREEN	Connection restored (Viper001::QTL, Type: QTLBioSensor)
03/10/2008 16:43:28:991 (GMT -7)	RED	Connection down (MaxConnectFailures exceeded) (Viper001::Canberra,
03/10/2008 16:43:34:003 (GMT -7)	GREEN	Connection restored (Viper001::Canberra, Type: Canberra)

FIG. 107

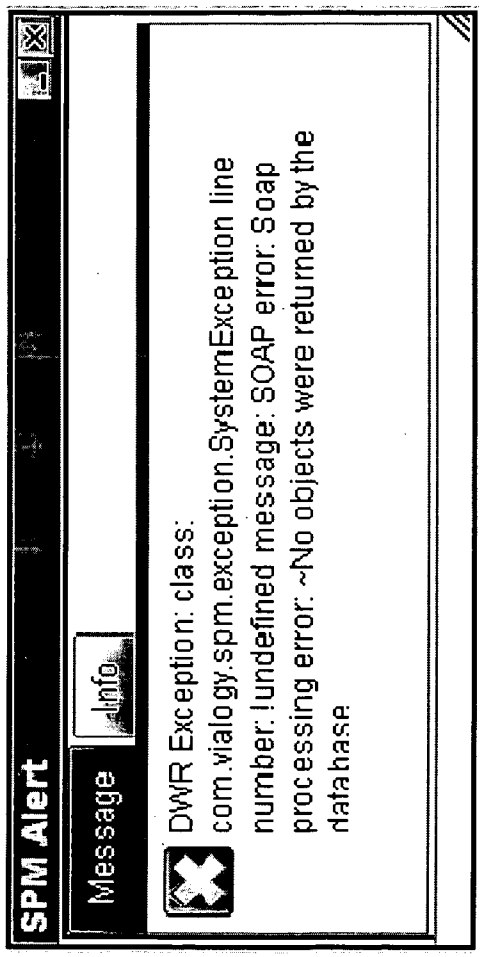

SPM Alert
Message
Info
DWR Exception: class:
com.vialogy.spm.exception.SystemException line
number: !undefined message: SOAP error: Soap
processing error: ~No objects were returned by the
database

FIG. 108

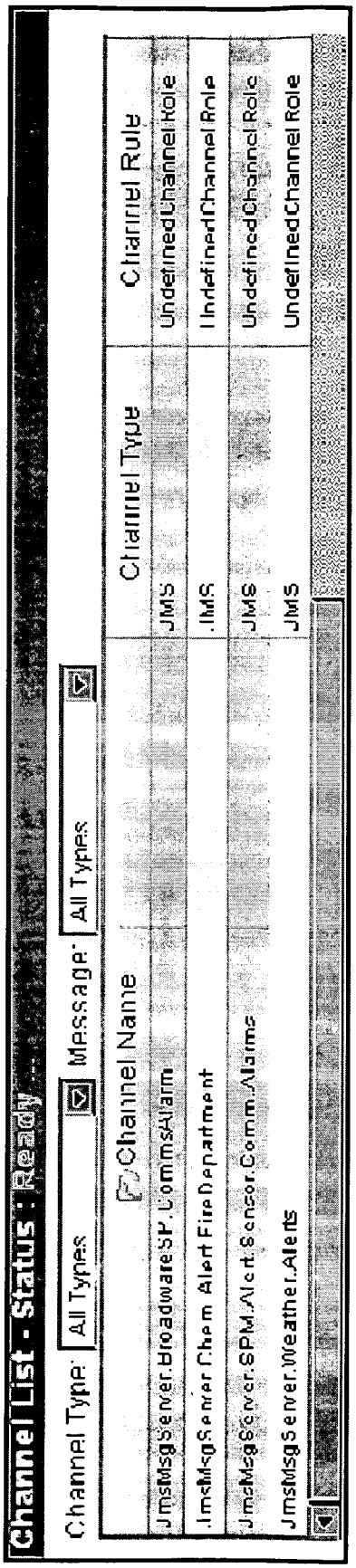

Channel List - Status : Ready
Channel Type: All Types
Message: All Types
Channel Name
Channel Type
Channel Rule
JmsMsgServer.BroadwareSP.CommsAlarm
JMS
UndefinedChannelRole
JmsMsgServer.ChemAlertFireDepartment
JMS
UndefinedChannelRole
JmsMsgServer.SPM.AlertSensor.Comm.Alarms
JMS
UndefinedChannelRole
JmsMsgServer.WeatherAlerts
JMS
UndefinedChannelRole

FIG. 109

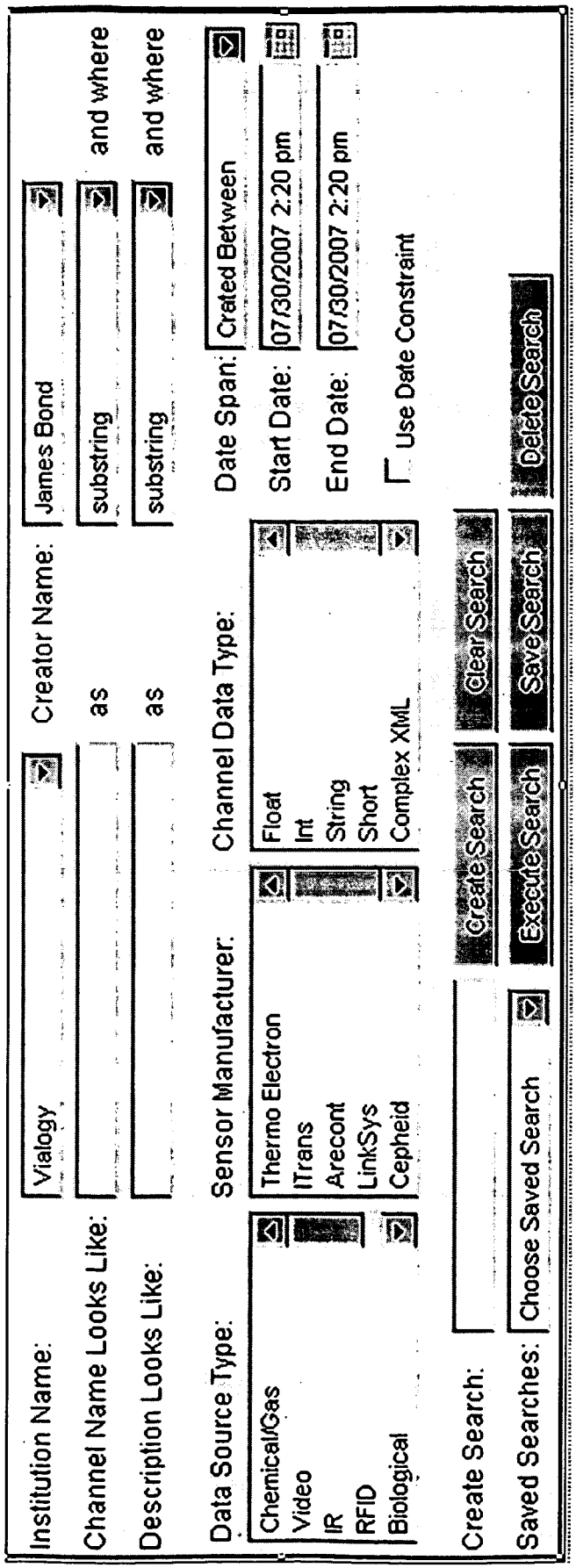

Institution Name:
Vialogy
Creator Name:
James Bond
Channel Name Looks Like:
as
substring
and where
Description Looks Like:
as
substring
and where
Data Source Type:
Sensor Manufacturer:
Channel Data Type:
Date Span:
Crated Between
Chemical/Gas
Video
IR
RFID
Biological
Thermo Electron
ITrans
Arecont
LinkSys
Cepheid
Float
Int
String
Short
Complex XML
Start Date:
07/30/2007 2:20 pm
End Date:
07/30/2007 2:20 pm
Use Date Constraint
Create Search:
Create Search
Clear Search
Saved Searches:
Choose Saved Search
Execute Search
Save Search
Delete Search

FIG. 110

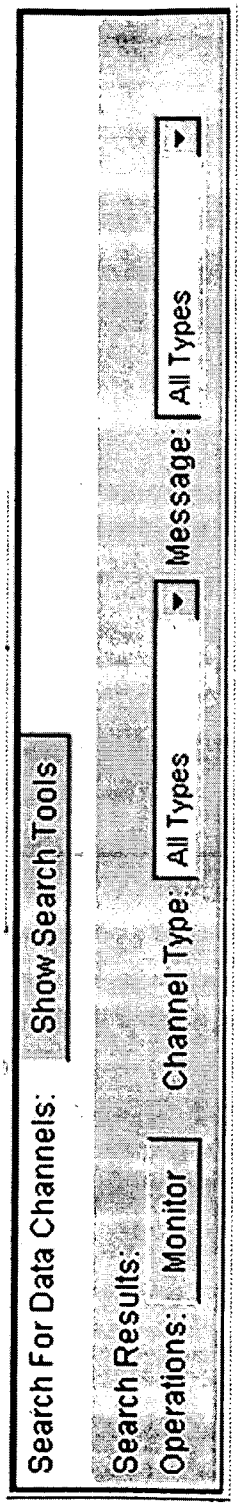

Search For Data Channels:
Show Search Tools
Search Results:
Operations:
Monitor
Channel Type:
All Types
Message:
All Types

FIG. 111

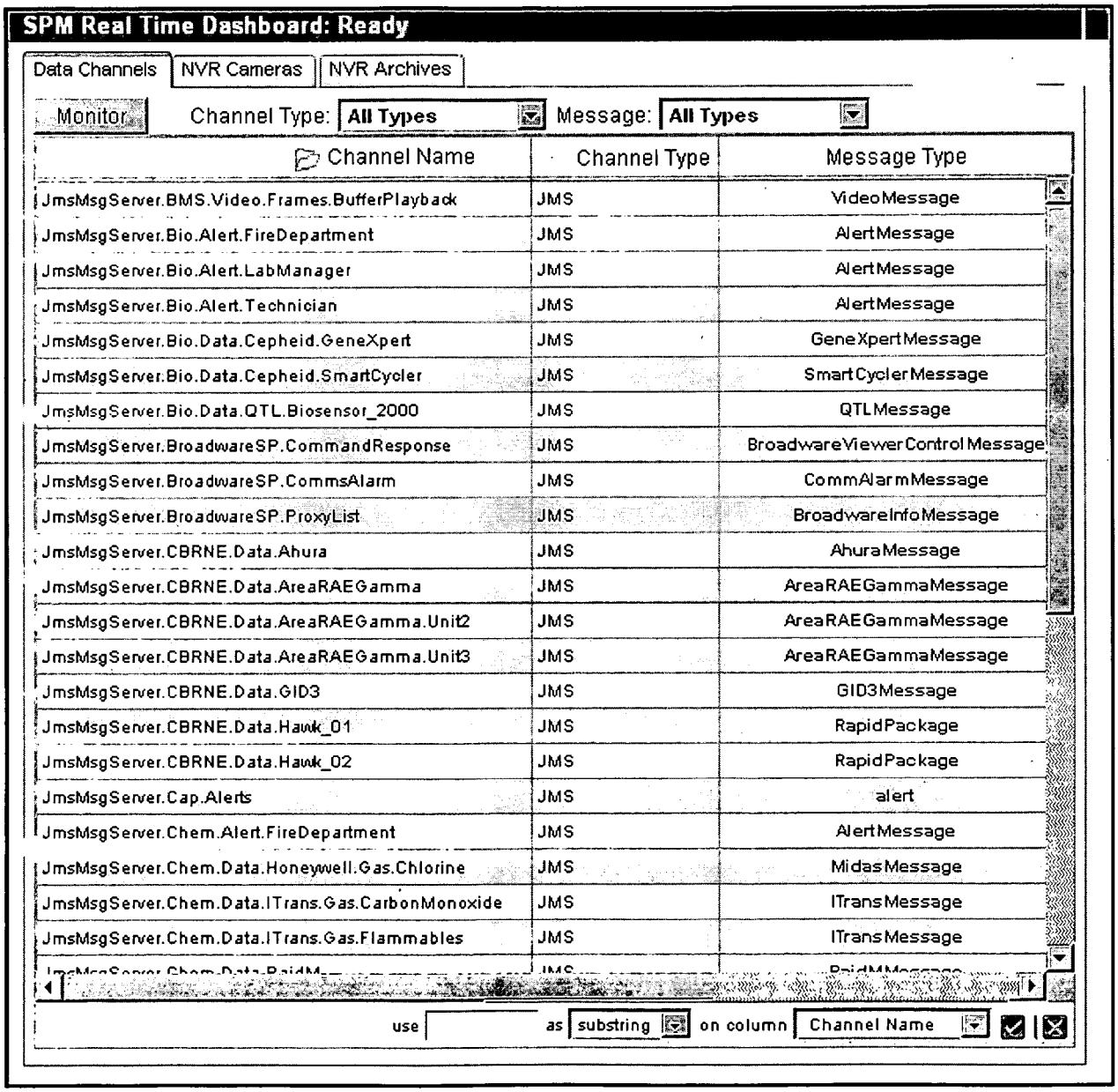

SPM Real Time Dashboard: Ready
Data Channels
NVR Cameras
NVR Archives
Monitor
Channel Type: All Types
Message: All Types
Channel Name
Channel Type
Message Type
JmsMsgServer.BMS.Video.Frames.BufferPlayback JMS VideoMessage
JmsMsgServer.Bio.Alert.FireDepartment JMS AlertMessage
JmsMsgServer.Bio.Alert.LabManager JMS AlertMessage
JmsMsgServer.Bio.Alert.Technician JMS AlertMessage
JmsMsgServer.Bio.Data.Cepheid.GeneXpert JMS GeneXpertMessage
JmsMsgServer.Bio.Data.Cepheid.SmartCycler JMS SmartCyclerMessage
JmsMsgServer.Bio.Data.QTL.Biosensor_2000 JMS QTLMessage
JmsMsgServer.BroadwareSP.CommandResponse JMS BroadwareViewerControlMessage
JmsMsgServer.BroadwareSP.CommsAlarm JMS CommAlarmMessage
JmsMsgServer.BroadwareSP.ProxyList JMS BroadwareInfoMessage
JmsMsgServer.CBRNE.Data.Ahura JMS AhuraMessage
JmsMsgServer.CBRNE.Data.AreaRAEGamma JMS AreaRAEGammaMessage
JmsMsgServer.CBRNE.Data.AreaRAEGamma.Unit2 JMS AreaRAEGammaMessage
JmsMsgServer.CBRNE.Data.AreaRAEGamma.Unit3 JMS AreaRAEGammaMessage
JmsMsgServer.CBRNE.Data.GID3 JMS GID3Message
JmsMsgServer.CBRNE.Data.Hawk_01 JMS RapidPackage
JmsMsgServer.CBRNE.Data.Hawk_02 JMS RapidPackage
JmsMsgServer.Cap.Alerts JMS alert
JmsMsgServer.Chem.Alert.FireDepartment JMS AlertMessage
JmsMsgServer.Chem.Data.Honeywell.Gas.Chlorine JMS MidasMessage
JmsMsgServer.Chem.Data.ITrans.Gas.CarbonMonoxide JMS ITransMessage
JmsMsgServer.Chem.Data.ITrans.Gas.Flammables JMS ITransMessage
use
as substring
on column Channel Name

FIG. 112

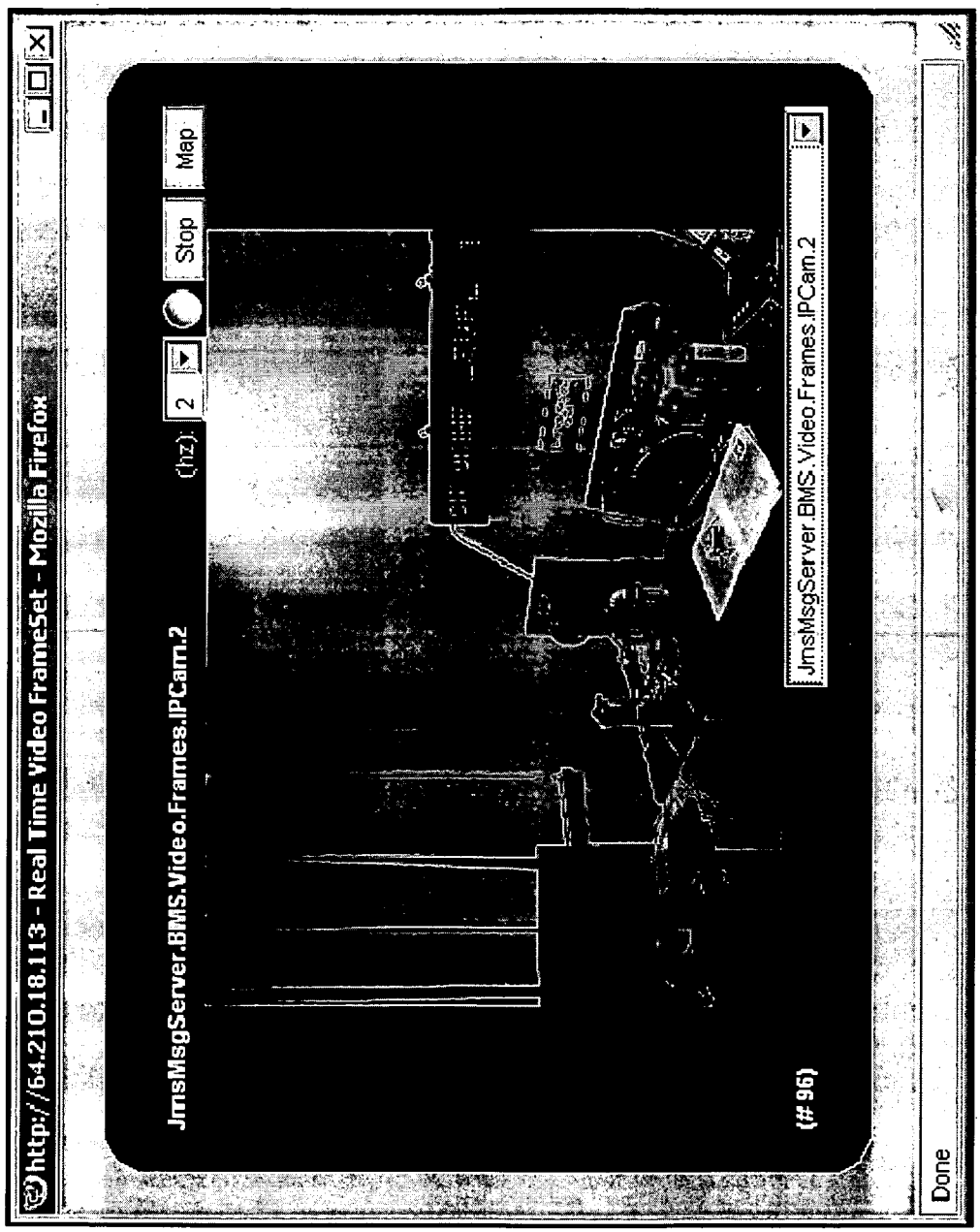

http://64.210.18.113 - Real Time Video FrameSet - Mozilla Firefox
JmsMsgServer.BMS.Video.Frames.IPCam.2
(hz):
2
Stop
Map
(# 96)
JmsMsgServer.BMS.Video.Frames.IPCam.2
Done

FIG. 113

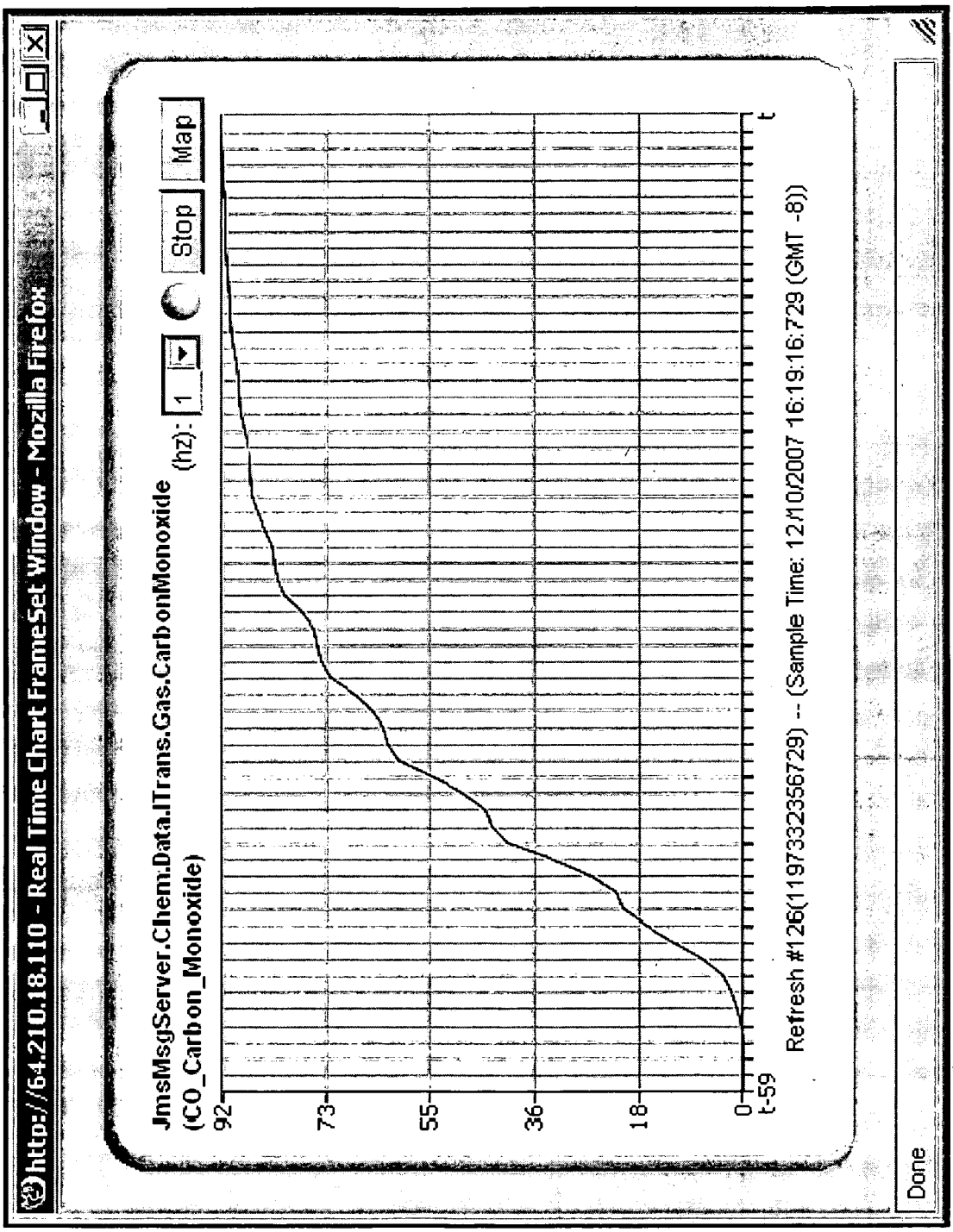

http://64.210.18.110 - Real Time Chart FrameSet Window - Mozilla Firefox
JmsMsgServer.Chem.Data.ITrans.Gas.CarbonMonoxide
(CO_Carbon_Monoxide)
(hz):
1
Stop
Map
92
73
55
36
18
0
t-59
t
Refresh #126(119733235729) -- (Sample Time: 12/10/2007 16:19:16:729 (GMT -8))
Done

FIG. 114

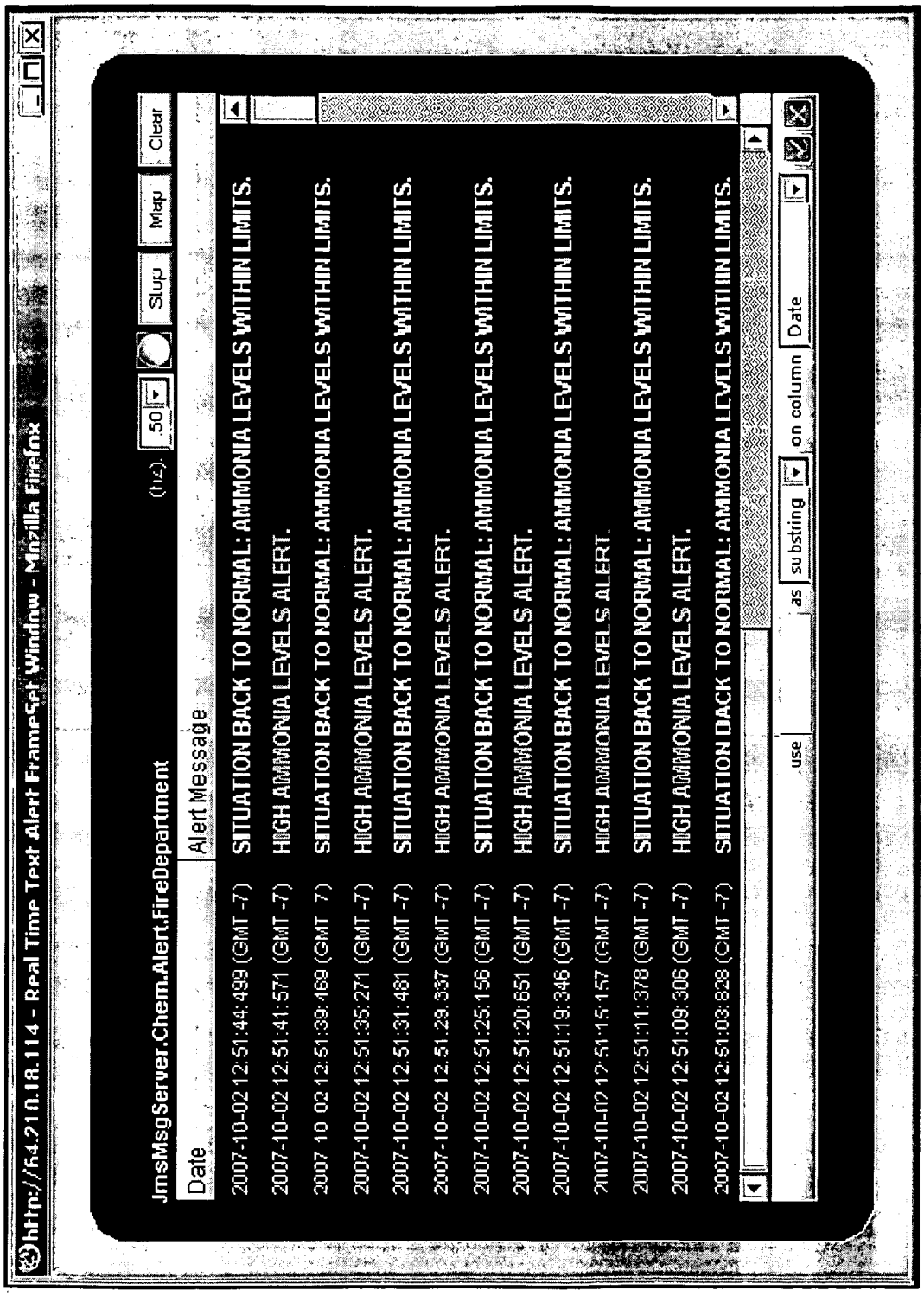

http://64.210.18.114 - Real Time Text Alert FrameSet Window - Mozilla Firefox
JmsMsgServer.Chem.Alert.FireDepartment
(hz)
50
Stop
Map
Clear
Date
Alert Message
2007-10-02 12:51:44:499 (GMT -7) SITUATION BACK TO NORMAL: AMMONIA LEVELS WITHIN LIMITS.
2007-10-02 12:51:41:571 (GMT -7) HIGH AMMONIA LEVELS ALERT.
2007 10 02 12:51:39:469 (GMT -7) SITUATION BACK TO NORMAL: AMMONIA LEVELS WITHIN LIMITS.
2007-10-02 12:51:35:271 (GMT -7) HIGH AMMONIA LEVELS ALERT.
2007-10-02 12:51:31:481 (GMT -7) SITUATION BACK TO NORMAL: AMMONIA LEVELS WITHIN LIMITS.
2007-10-02 12:51:29:337 (GMT -7) HIGH AMMONIA LEVELS ALERT.
2007-10-02 12:51:25:156 (GMT -7) SITUATION BACK TO NORMAL: AMMONIA LEVELS WITHIN LIMITS.
2007-10-02 12:51:20:651 (GMT -7) HIGH AMMONIA LEVELS ALERT.
2007-10-02 12:51:19:346 (GMT -7) SITUATION BACK TO NORMAL: AMMONIA LEVELS WITHIN LIMITS.
2007-10-02 12:51:15:157 (GMT -7) HIGH AMMONIA LEVELS ALERT.
2007-10-02 12:51:11:378 (GMT -7) SITUATION BACK TO NORMAL: AMMONIA LEVELS WITHIN LIMITS.
2007-10-02 12:51:09:306 (GMT -7) HIGH AMMONIA LEVELS ALERT.
2007-10-02 12:51:03:828 (GMT -7) SITUATION BACK TO NORMAL: AMMONIA LEVELS WITHIN LIMITS.
use
as substring
on column Date

FIG. 115

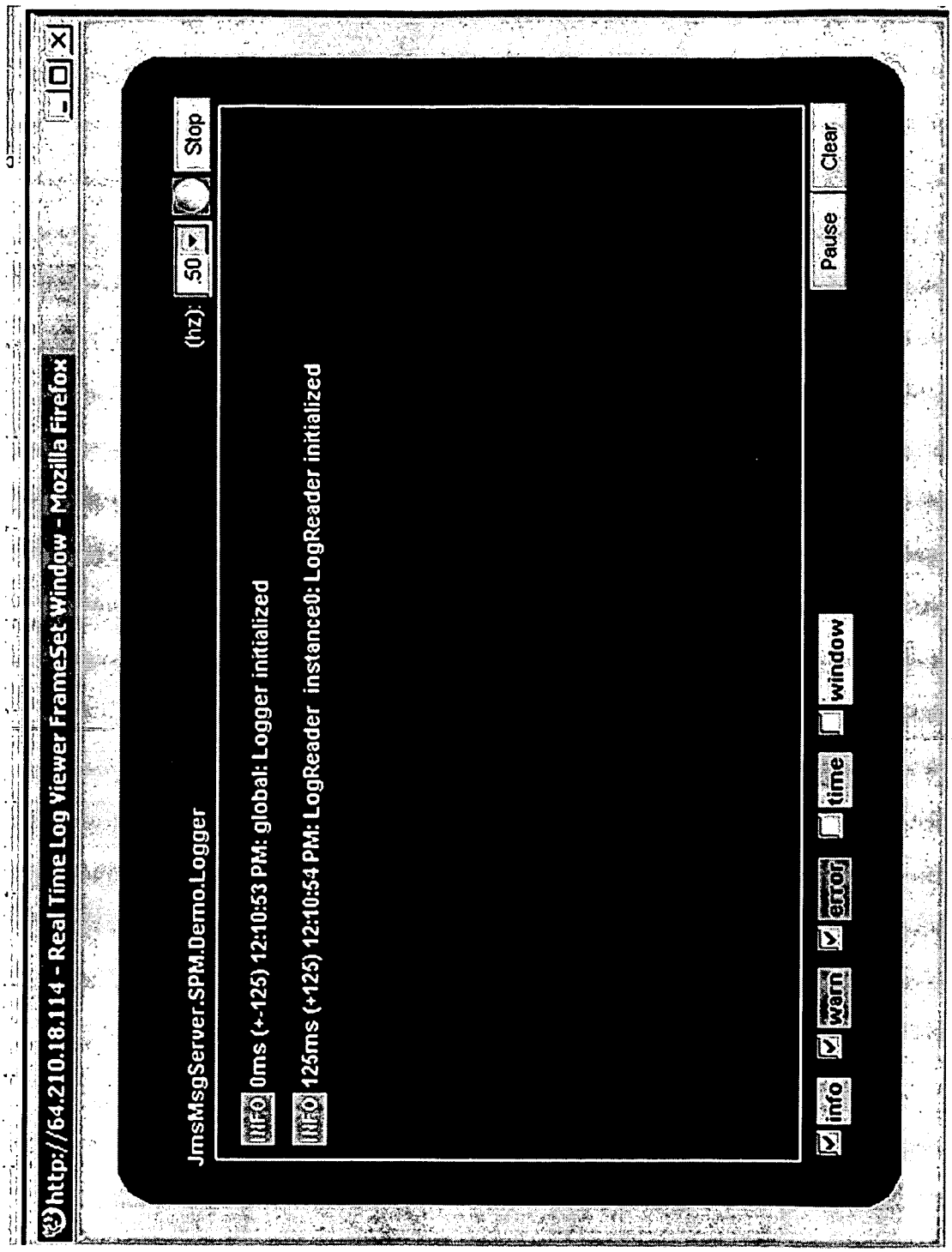

http://64.210.18.114 - Real Time Log Viewer FrameSet Window - Mozilla Firefox
JmsMsgServer.SPM.Demo.Logger
(hz):
50
Stop
INFO 0ms (+-125) 12:10:53 PM: global: Logger initialized
INFO 125ms (+125) 12:10:54 PM: LogReader instance0: LogReader initialized
info
warn
error
time
window
Pause
Clear

FIG. 116

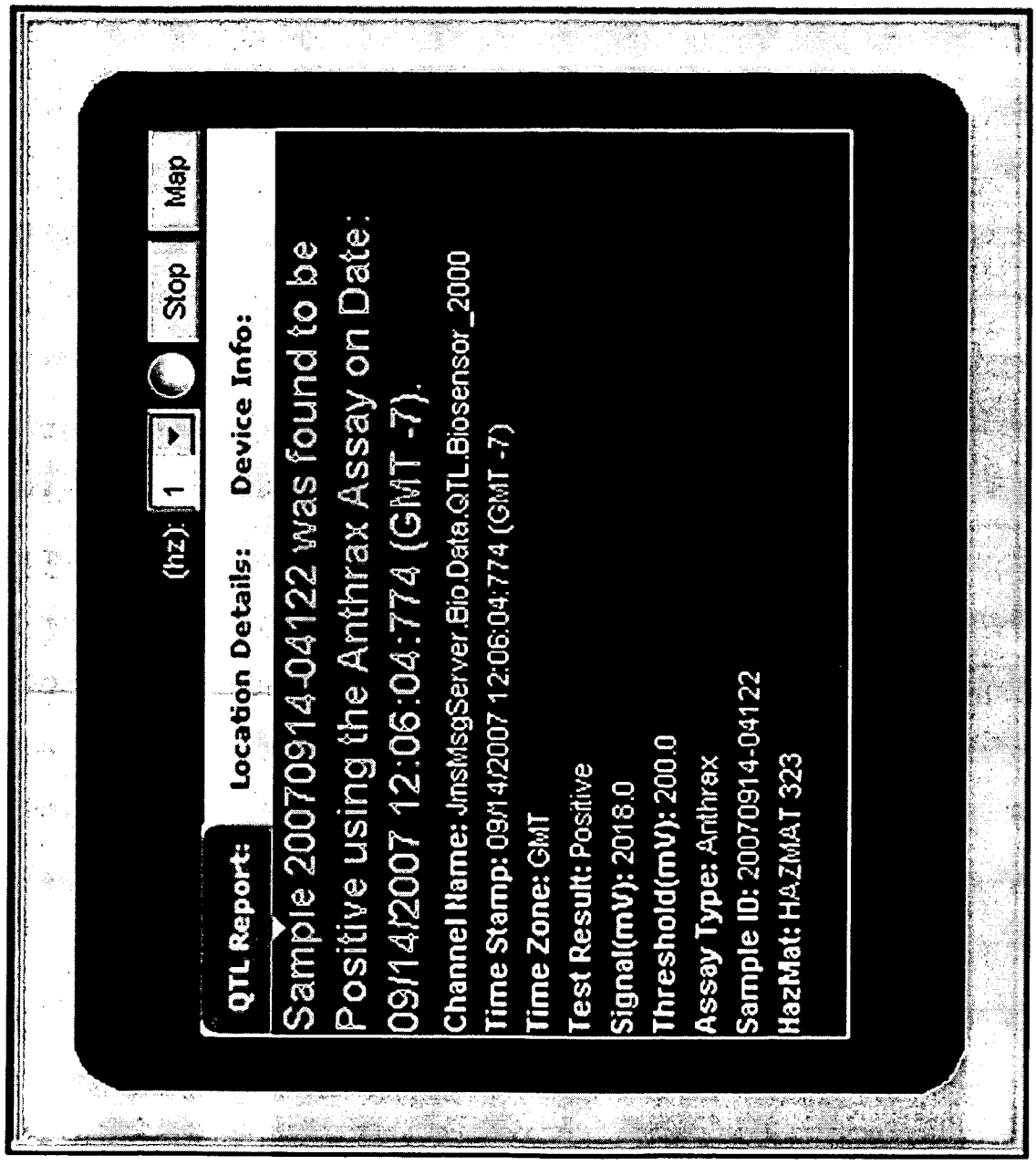

(hz):
1
Stop
Map
QTL Report:
Location Details:
Device Info:
Sample 20070914-04122 was found to be Positive using the Anthrax Assay on Date: 09/14/2007 12:06:04:774 (GMT -7).
Channel Name: JmsMsgServer.Bio.Data.QTL.Biosensor_2000
Time Stamp: 09/14/2007 12:06:04:774 (GMT -7)
Time Zone: GMT
Test Result: Positive
Signal(mV): 2018.0
Threshold(mV): 200.0
Assay Type: Anthrax
Sample ID: 20070914-04122
HazMat: HAZMAT 323

FIG. 117

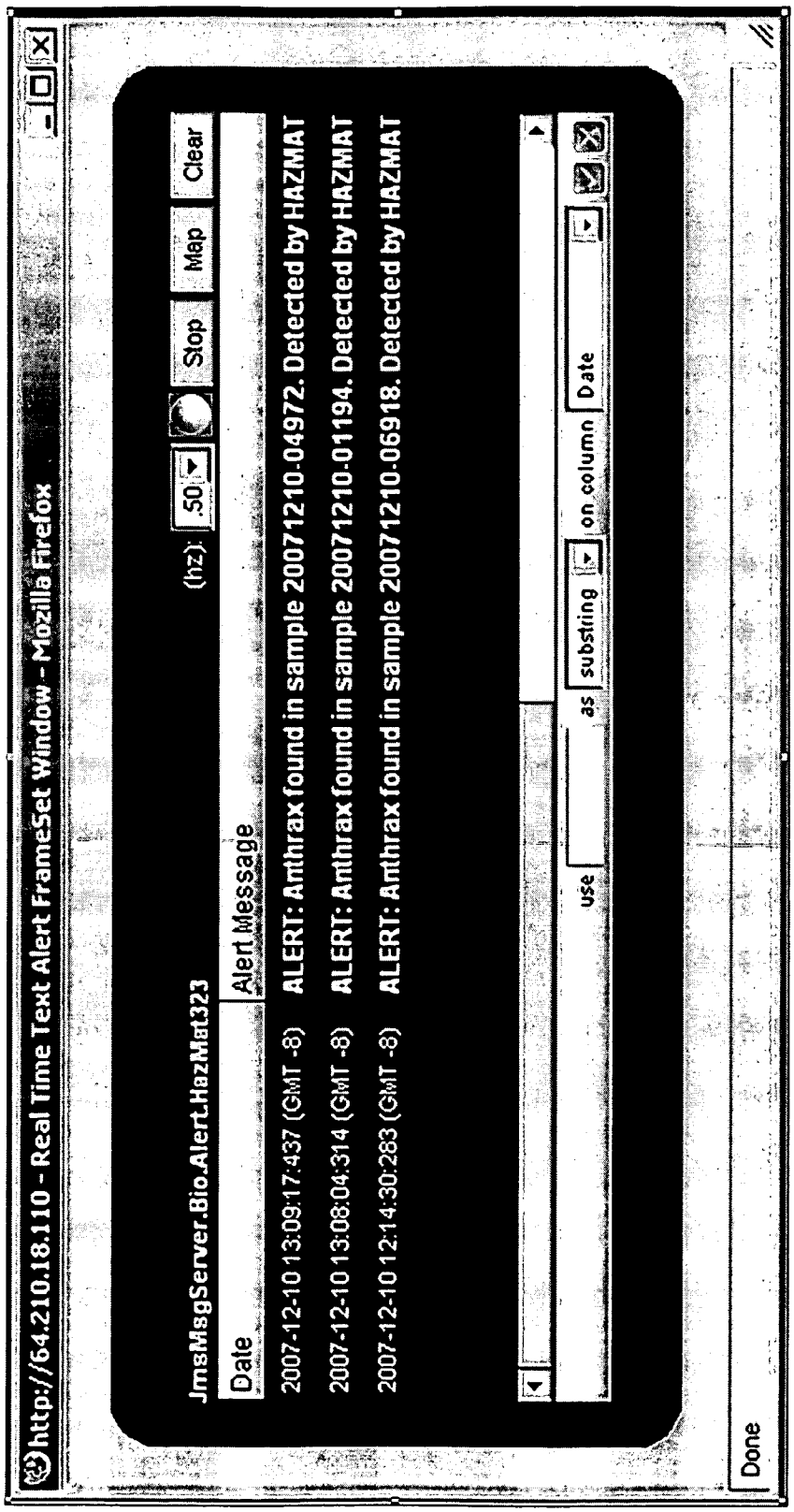

http://64.210.18.110 - Real Time Text Alert FrameSet Window - Mozilla Firefox
JmsMsgServer.Bio.Alert.HazMat323
(hz)
.50
Stop
Map
Clear
Date
Alert Message
2007-12-10 13:09:17:437 (GMT -8)
ALERT: Anthrax found in sample 20071210.04972. Detected by HAZMAT
2007-12-10 13:08:04:314 (GMT -8)
ALERT: Anthrax found in sample 20071210.01194. Detected by HAZMAT
2007-12-10 12:14:30:283 (GMT -8)
ALERT: Anthrax found in sample 20071210.06918. Detected by HAZMAT
use
as
substring
on column
Date
Done

FIG. 118

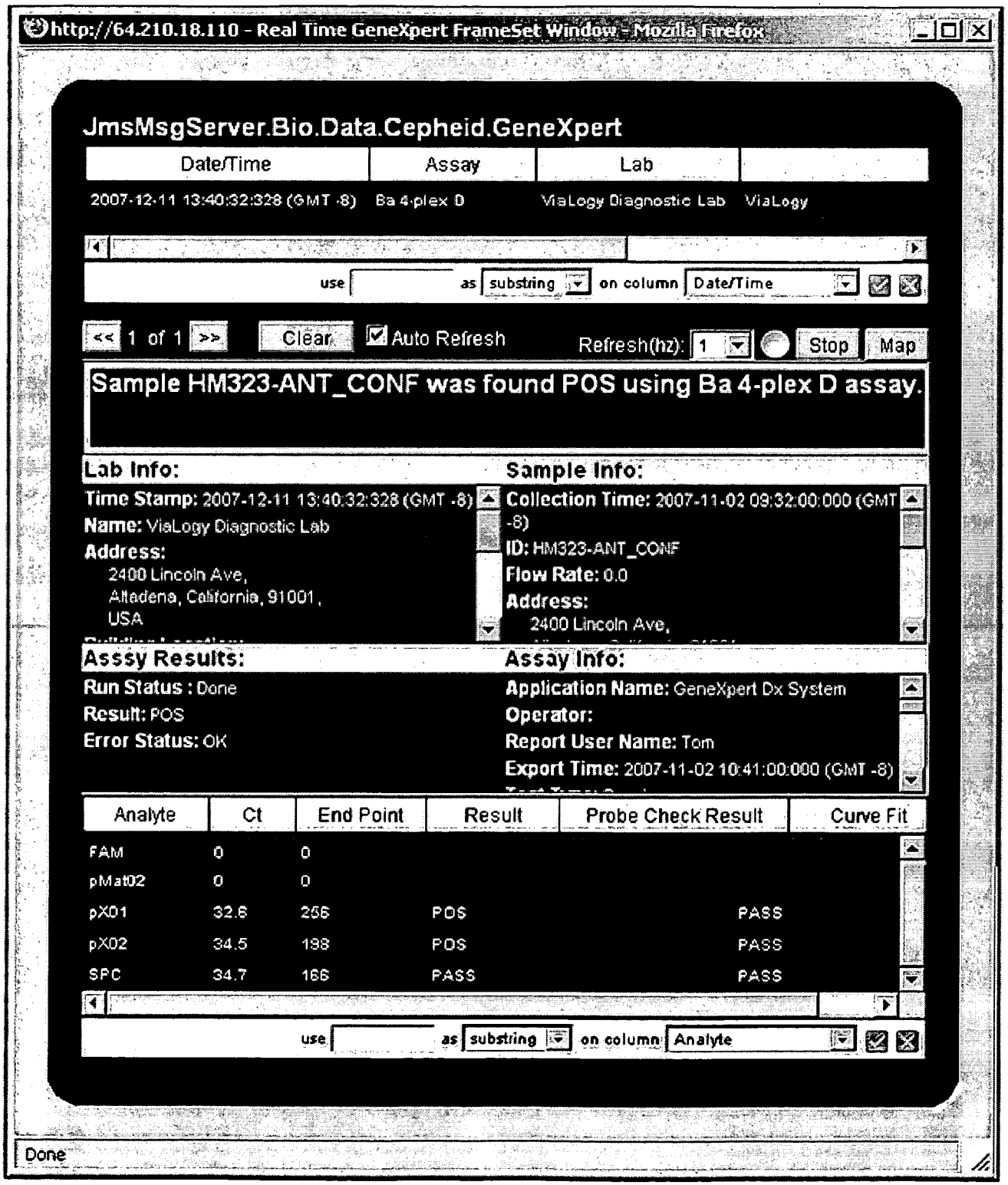

http://64.210.18.110 - Real Time GeneXpert FrameSet Window - Mozilla Firefox
JmsMsgServer.Bio.Data.Cepheid.GeneXpert
Date/Time
Assay
Lab
2007-12-11 13:40:32:328 (GMT -8)
Ba 4-plex D
ViaLogy Diagnostic Lab
ViaLogy
use
as substring
on column Date/Time
<< 1 of 1 >>
Clear
Auto Refresh
Refresh(hz): 1
Stop
Map
Sample HM323-ANT_CONF was found POS using Ba 4-plex D assay.
Lab Info:
Time Stamp: 2007-12-11 13:40:32:328 (GMT -8)
Name: ViaLogy Diagnostic Lab
Address:
2400 Lincoln Ave,
Altadena, California, 91001,
USA
Sample Info:
Collection Time: 2007-11-02 08:32:00:000 (GMT -8)
ID: HM323-ANT_CONF
Flow Rate: 0.0
Address:
2400 Lincoln Ave,
Assay Results:
Run Status : Done
Result: POS
Error Status: OK
Assay Info:
Application Name: GeneXpert Dx System
Operator:
Report User Name: Tom
Export Time: 2007-11-02 10:41:00:000 (GMT -8)
Analyte
Ct
End Point
Result
Probe Check Result
Curve Fit
FAM 0 0
pMat02 0 0
pX01 32.8 256 POS PASS
pX02 34.5 198 POS PASS
SPC 34.7 166 PASS PASS
use
as substring
on column Analyte
Done

FIG. 119

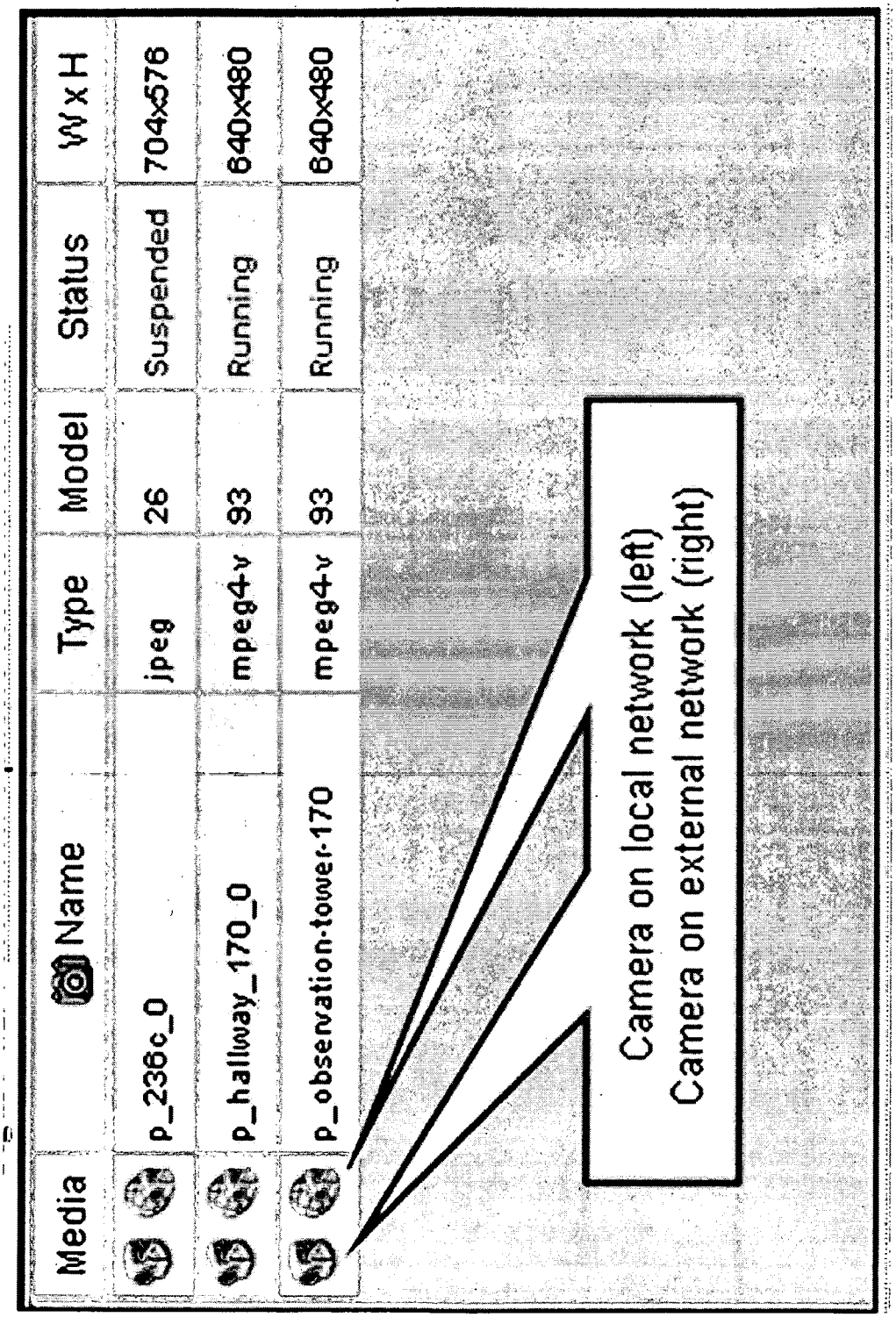

Media
Name
Type
Model
Status
W x H
p_236c_0
jpeg
26
Suspended
704x576
p_hallway_170_0
mpeg4-v
93
Running
640x480
p_observation-tower-170
mpeg4-v
93
Running
640x480
Camera on local network (left)
Camera on external network (right)

FIG. 120

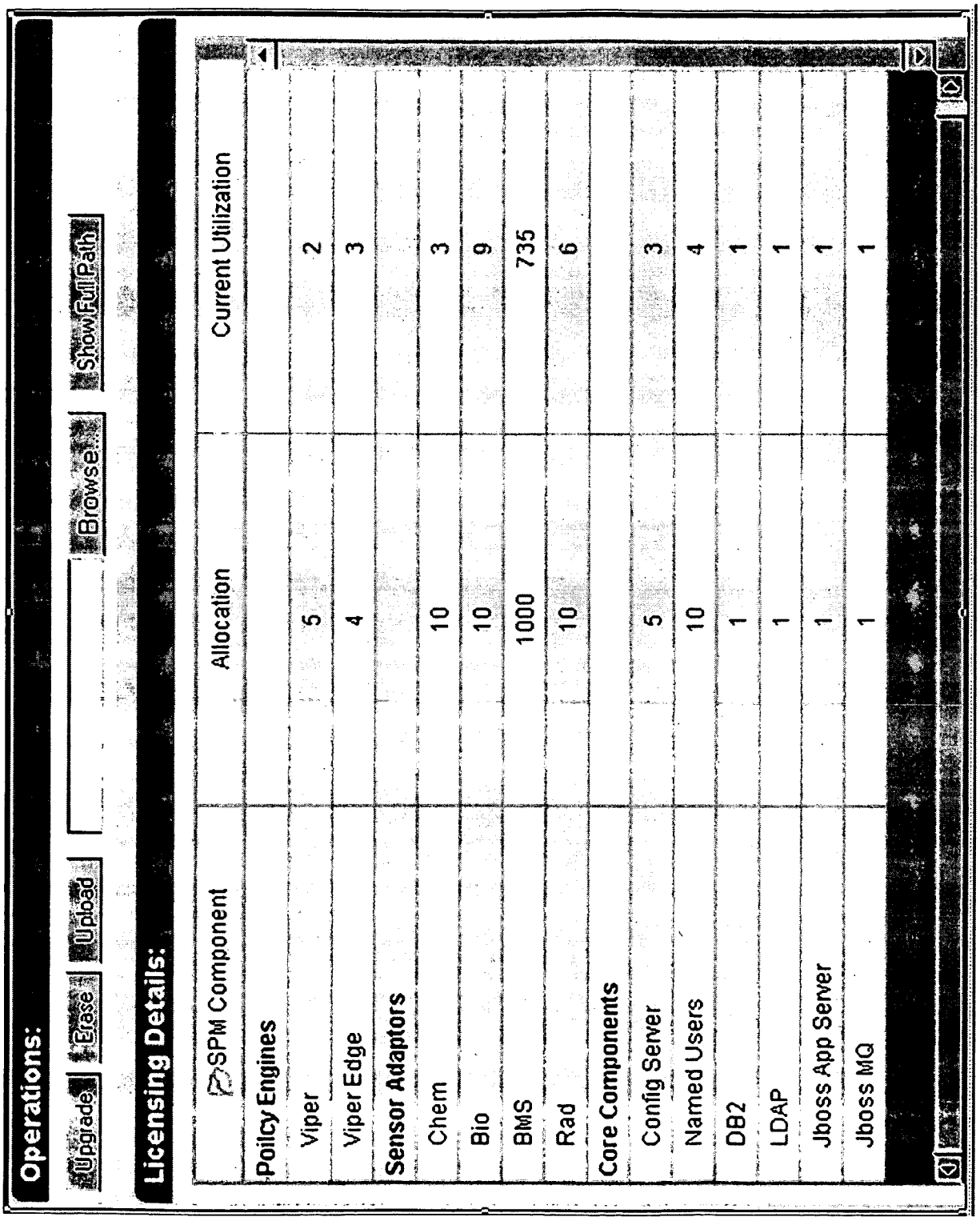

Operations:
Upgrade
Erase
Upload
Browse...
Show Full Path
Licensing Details:
SPM Component
Allocation
Current Utilization
Policy Engines
Viper
5
2
Viper Edge
4
3
Sensor Adaptors
Chem
10
3
Bio
10
9
BMS
1000
735
Rad
10
6
Core Components
Config Server
5
3
Named Users
10
4
DB2
1
1
LDAP
1
1
Jboss App Server
1
1
Jboss MQ
1
1

FIG. 121

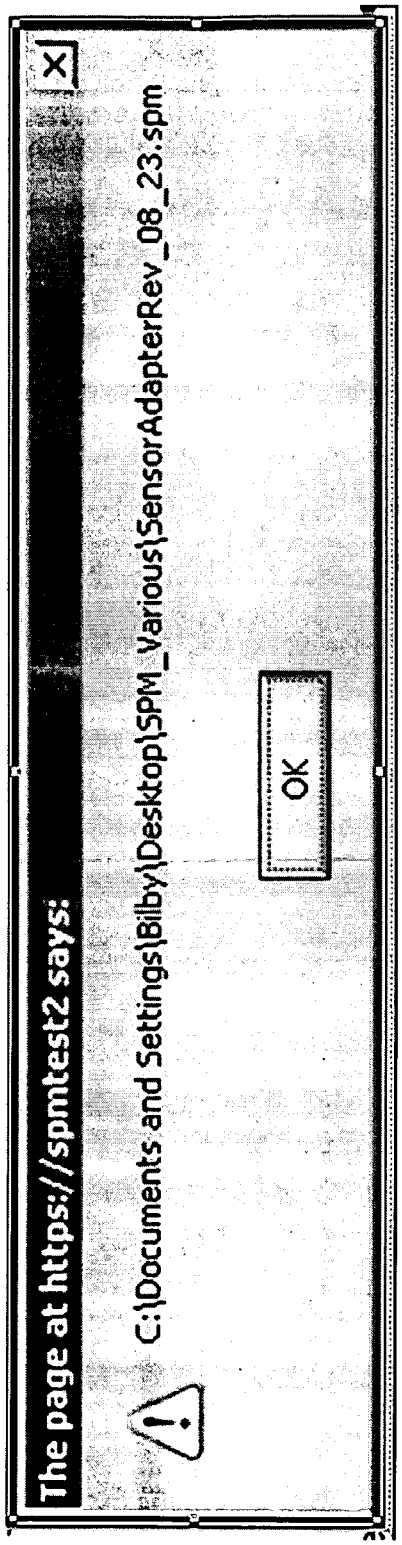

The page at https://spmtest2 says:
C:\Documents and Settings\Bilby\Desktop\SPM_Various\SensorAdapterRev_08_23.spm
OK

FIG. 122

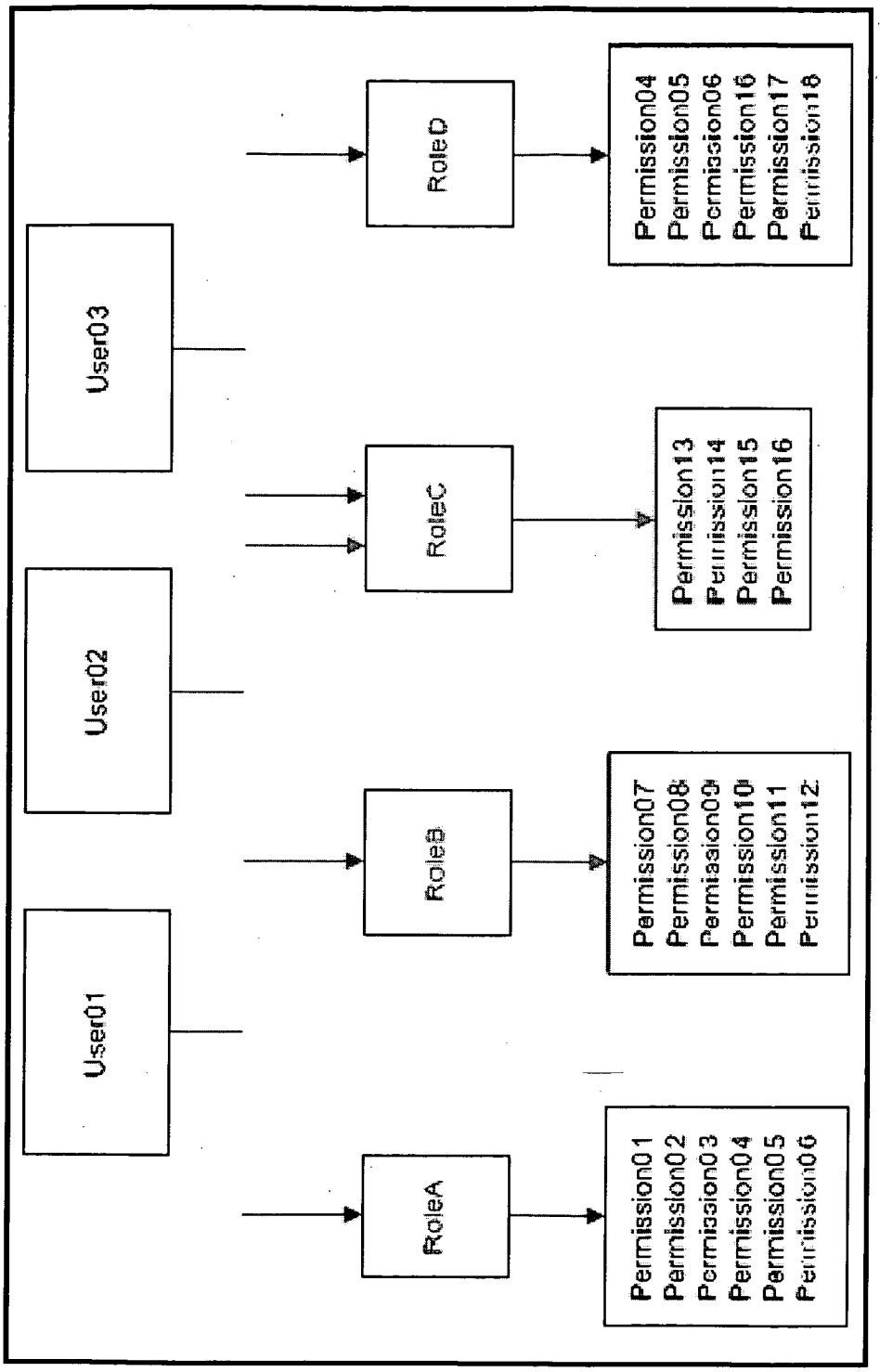

User01
User02
User03
RoleA
RoleB
RoleC
RoleD
Permission01
Permission02
Permission03
Permission04
Permission05
Permission06
Permission07
Permission08
Permission09
Permission10
Permission11
Permission12
Permission13
Permission14
Permission15
Permission16
Permission04
Permission05
Permission06
Permission16
Permission17
Permission18

FIG. 123

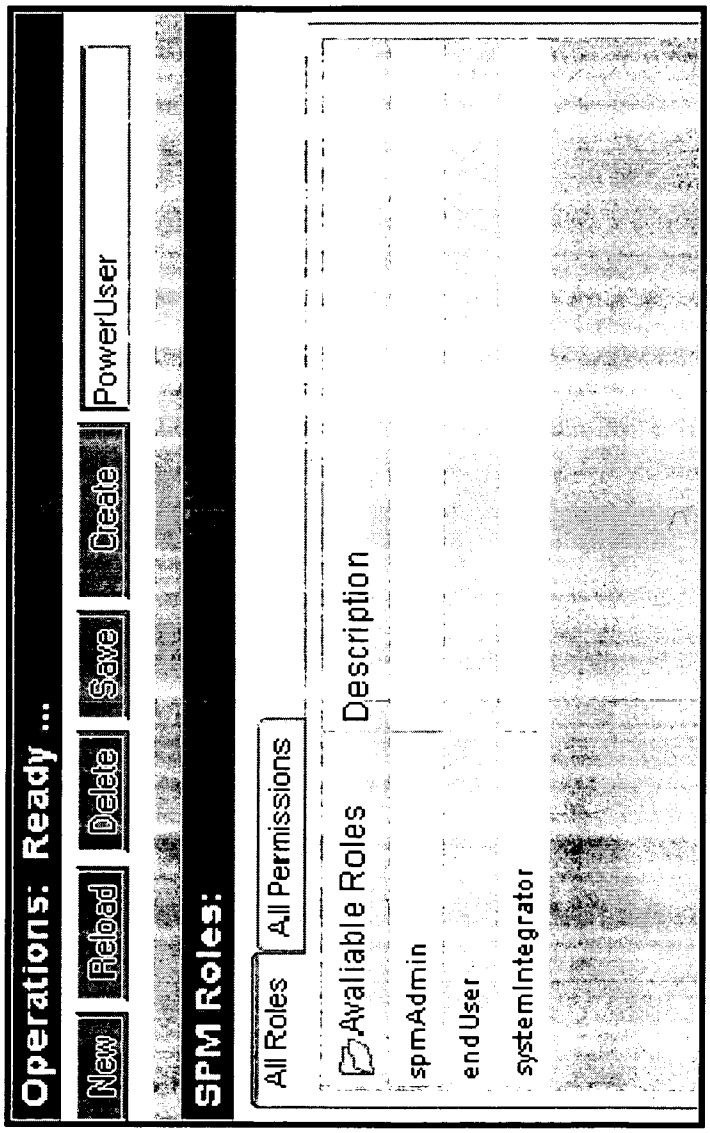

Operations: Ready ...
New
Reload
Delete
Save
Create
PowerUser
SPM Roles:
All Roles
All Permissions
Avaliable Roles
Description
spmAdmin
endUser
systemIntegrator

FIG. 124

		Predefined Role		
Config Object	Permissions	SysInt	Expert	User
actionTemplate	delete > write > read > list	D	D	R
all*		X		
channel	delete > write > read > list	D	D	R
channelValue	delete > write > read > list	D	D	R
channelValue deleted sometime before 6/30/08; not know by whom or for what reason				
conditionTemplate	delete > write > read > list	D	D	R
configServer	delete > write > read > list	D	D	R
mapper	delete > write > read > list	D	R	R
mapper deleted sometime before 6/26/08				
messageType	delete > write > read > list	D	D	R
mibFile	list	L	L	L
mibFile added sometime before 6/26/08				
msgDbInterface	delete > write > read > list	D	R	R
msgServerConfig	delete > write > read > list	D	D	R
permission	list	L	L	L
removed the nonexistent permissions list and replaced it with permission list on 5/5/08; this allows EndUser to see All Permissions list				
policyInstance	delete > write > read > list	D	D	R
policyTemplate	delete > write > read > list	D	D	R
processManager	delete > write > read > list	D	D	R
register	delete > write > read > list	D	R	R
role	delete > write > read > list	D	R	-
as of 5/5/08 the EndUser does not have role list permission by default; the roles list that is currently loaded is an error; added role list on 5/05/08 with result that EndUser can see contents of All Roles list in Account Management Users				
rolepermission	delete > write > list	D	W	L
The All Permissions list does not seem to include a rolepermission read but there are rolepermission list, .write and .delete; added rolepermission list to EndUser on 5/5/08; this allows EndUser to see the permissions for a selected role, but not the contents of the All Permissions list; but see the note above to permission list				

FIG. 125

		Predefined Role		
Config Object	Permissions	SysInt	Expert	User
sdoStructureMap	delete > write > read > list	D	?	?
sdoStructureMap added sometime before 6/26/08				
sdoValueMap	delete > write > read > list	D	?	?
sdoValueMap added sometime before 6/26/08				
sensorProxy (sensor adapter)	delete > write > read > list	D	R	R
serviceDataObject	delete > write > read > list	D	R	R
sessiontimeout	write	W	W	W
as of 5/5/08 SystemIntegrator has sessiontimeout read only; but All Permissions list contains only a sessiontimeout write permission;				
spmlHandler	write > read	W	R	R
trap	list	I	I	I
trap added sometime before 6/26/08				
user	delete > write > read > list	D	R	L
as of 5/5/08 the EndUser does not have user list permission; this means that the user read and uer write permissions are useless; added user list to EndUser on 5/5/08;				
userpassword	write* > read	R	R	R
The lack of a userpassword write permission means, among other things, that a user cannot change his own passwords; passwords can be created only when an admin creates a new user; passwords are destroyed by deleting the user.				
userrole	delete > write > list	D	L	L
as of 5/5/08, All Permissions did not include a userrole read permission; removed the nonexistent userroles list from EndUser as of 5/5/08; added userrole list to ExpertUser and EndUser;				
varbind	read	R	R	R
viperServer	delete > write > start > stop > read >list	D	R	R
The 'all' permission disables permission checking within the SPM system. Thus, no prohibitions are in effect.				

FIG. 126

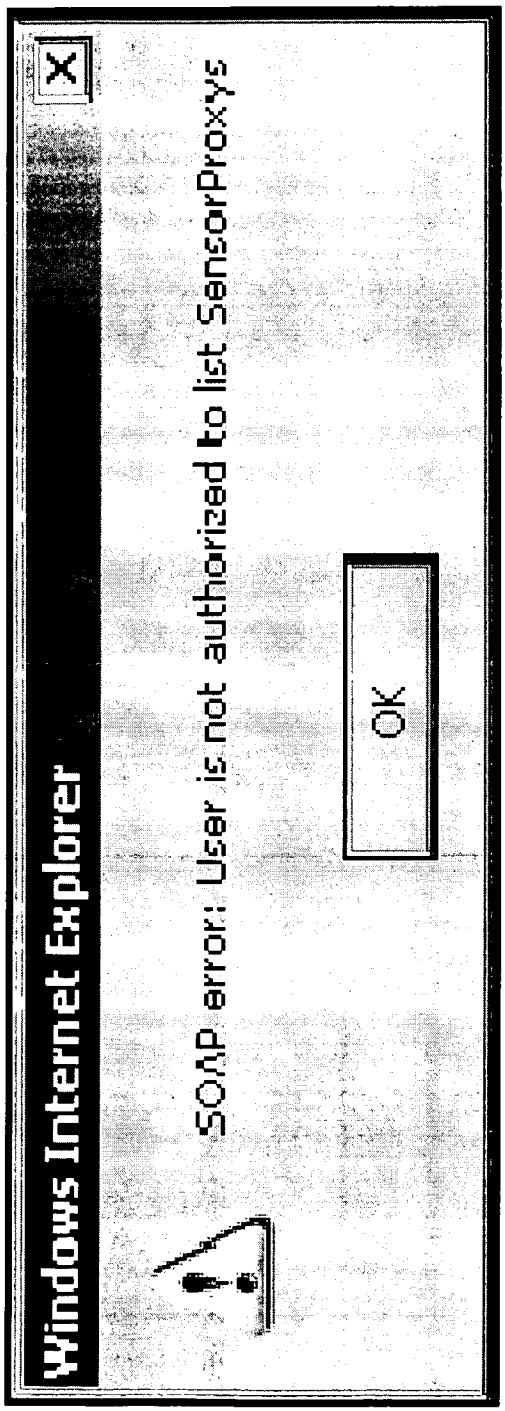

Windows Internet Explorer
SOAP error: User is not authorized to list SensorProxys
OK

FIG. 127

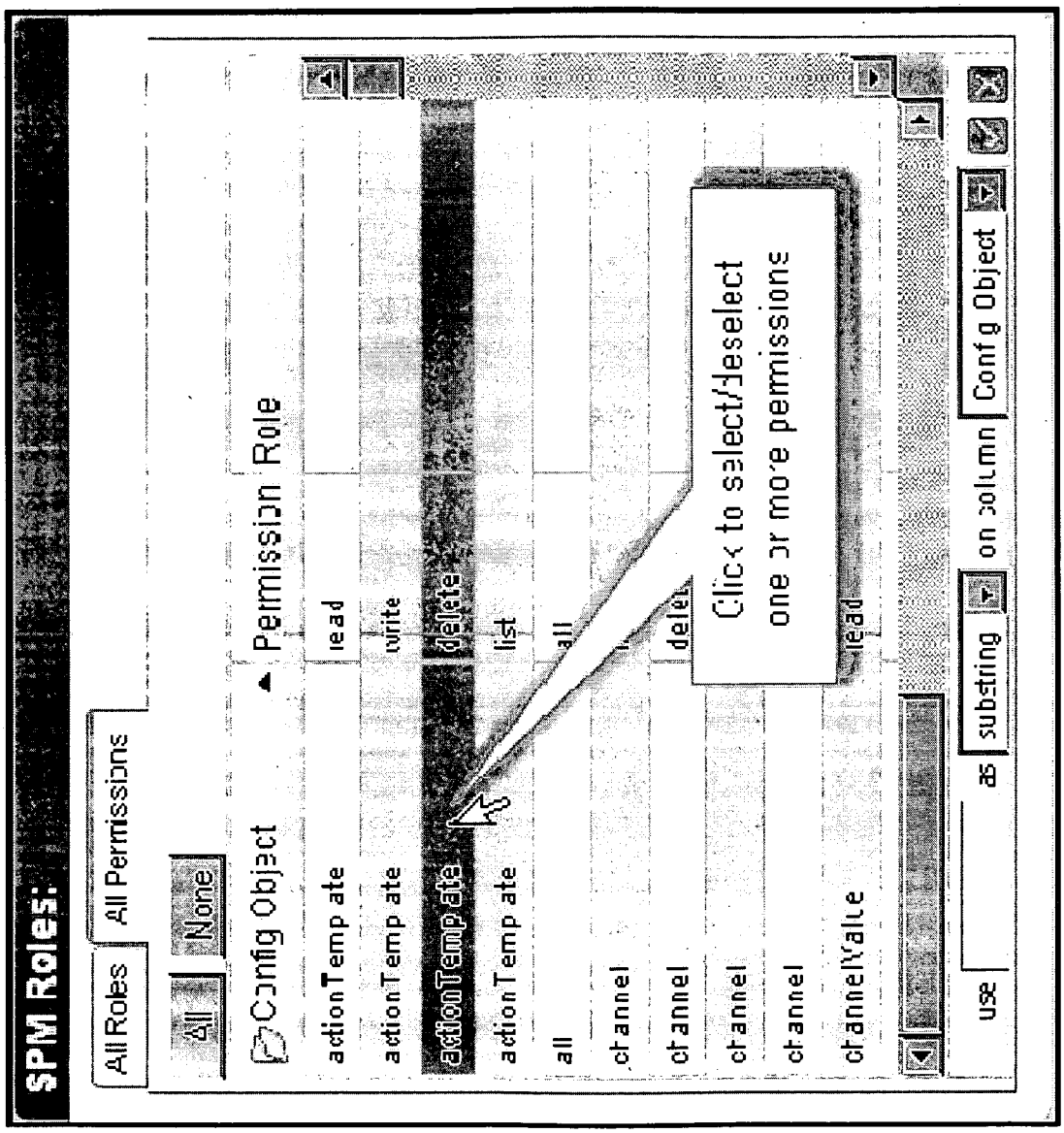

SPM Roles:
All Roles
All Permissions
All
None
Config Object
Permission Role
actionTemplate
actionTemplate
actionTemplate
actionTemplate
all
channel
channel
channel
channel
channelValue
read
write
delete
list
all
read
Click to select/deselect
one or more permissions
use
as substring
on column
Config Object

FIG. 128

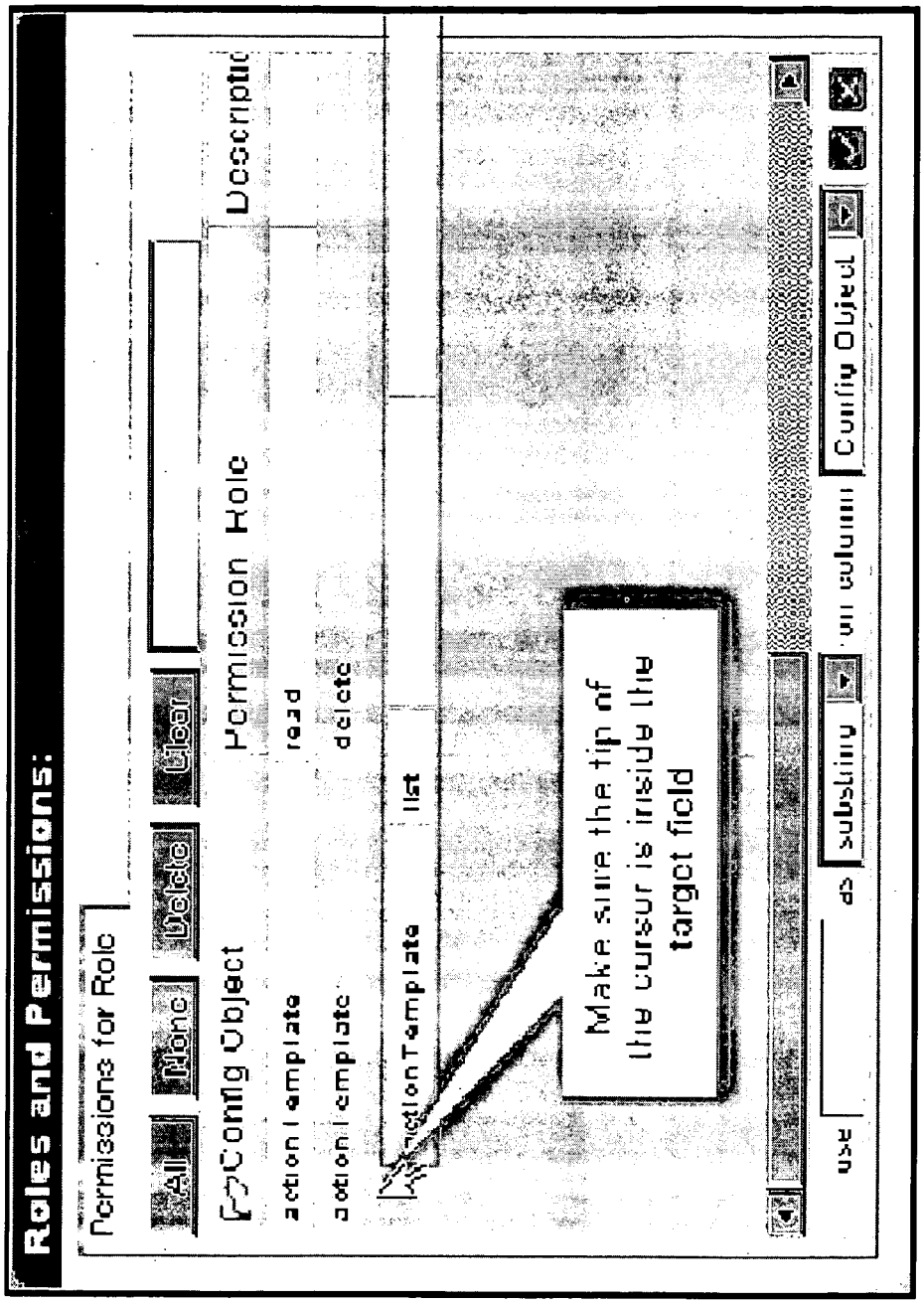

Roles and Permissions:
Permissions for Role
All
None
Delete
Clear
Permission Role
Description
action1emplate
read
action1emplate
delete
ActionTemplate
list
Make sure the tip of the cursor is inside the target field
use
as
substring
on column
Config Object

FIG. 129

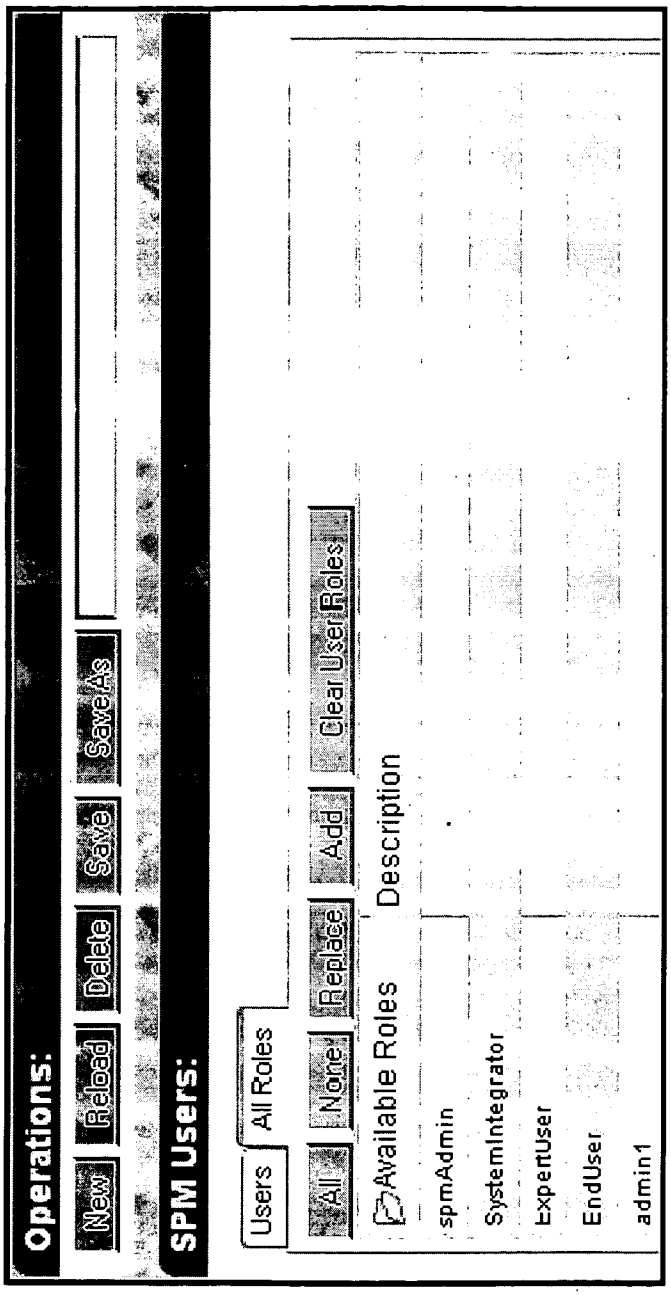

Operations:
New
Reload
Delete
Save
Save As
SPM Users:
Users
All Roles
All
None
Replace
Add
Clear User Roles
Available Roles
Description
spmAdmin
SystemIntegrator
ExpertUser
EndUser
admin1

FIG. 130

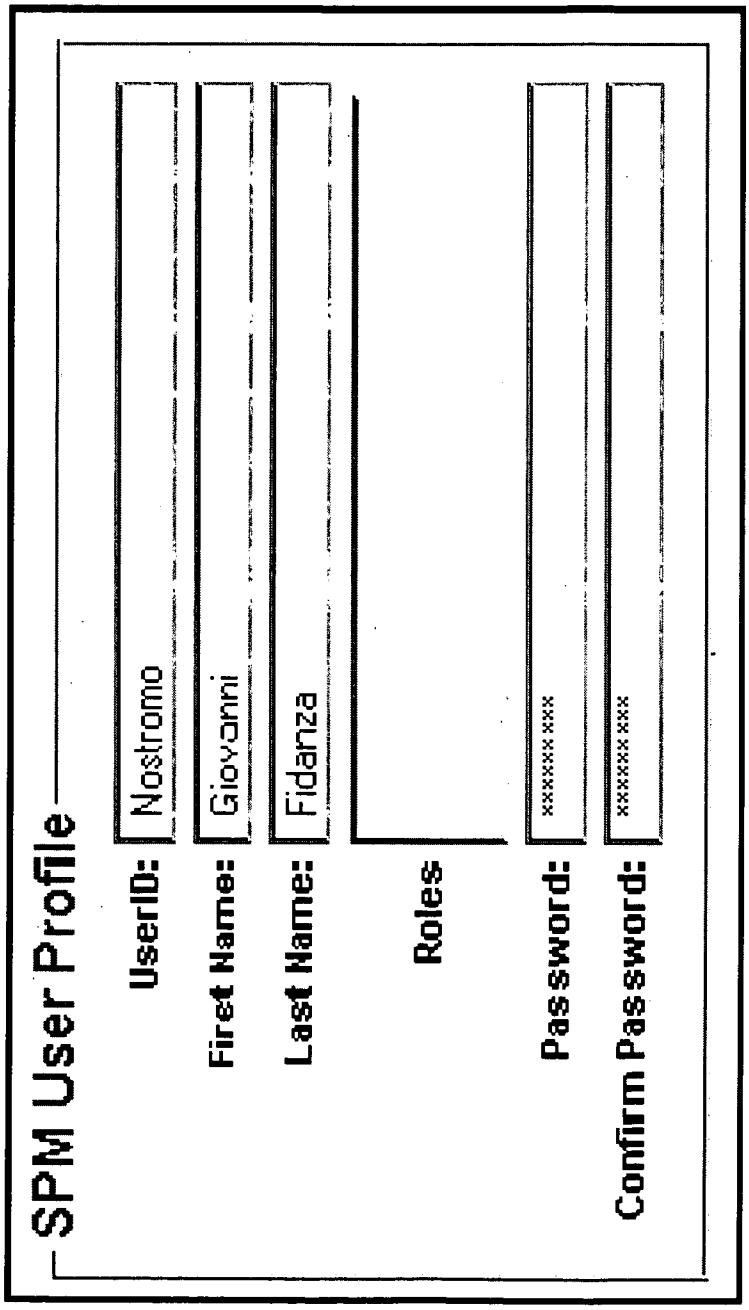

SPM User Profile
UserID:
Nostromo
First Name:
Giovanni
Last Name:
Fidanza
Roles
Password:
Confirm Password:

FIG. 131

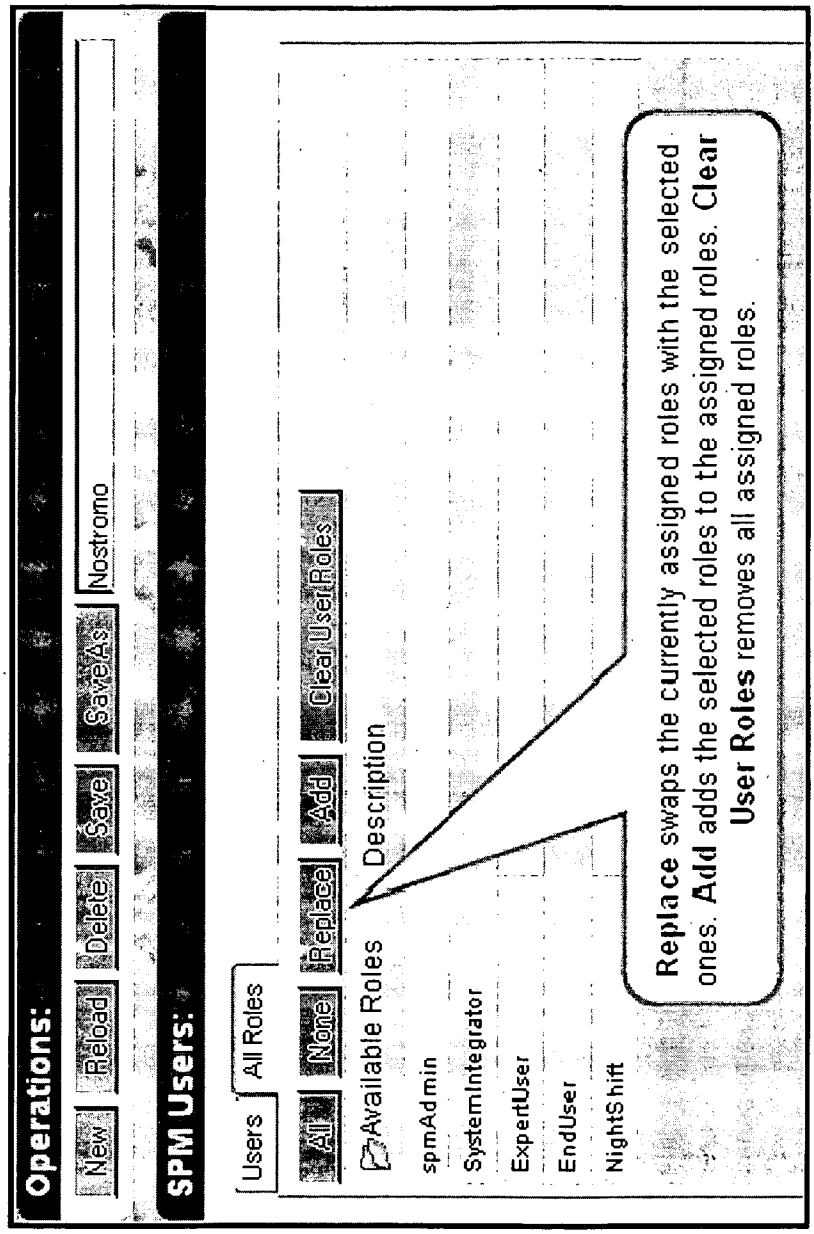

Operations:
New
Reload
Delete
Save
Save As
Nostromo
SPM Users:
Users
All Roles
All
None
Replace
Add
Clear User Roles
Available Roles
Description
spmAdmin
SystemIntegrator
ExpertUser
EndUser
NightShift
Replace swaps the currently assigned roles with the selected ones. Add adds the selected roles to the assigned roles. Clear User Roles removes all assigned roles.

FIG. 132

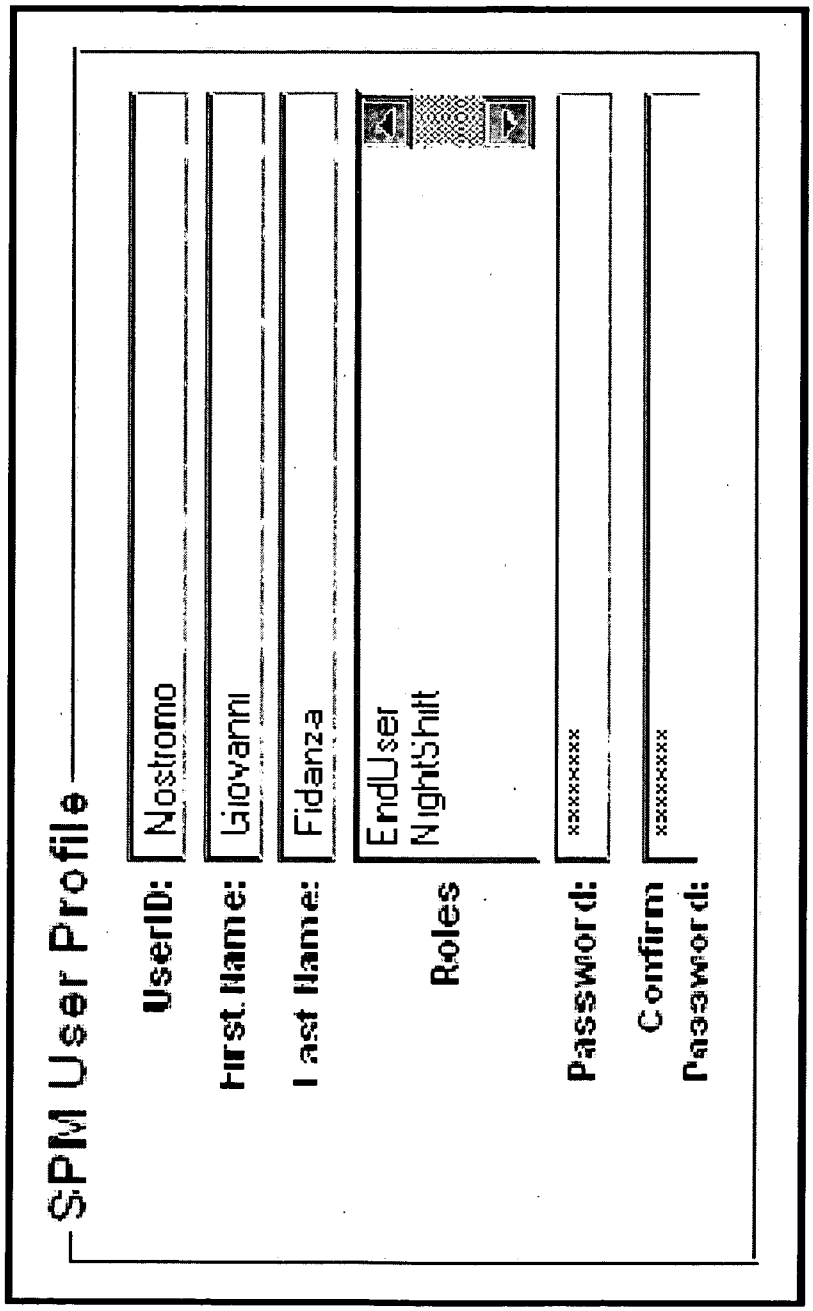

SPM User Profile
UserID:
Nostromo
First Name:
Giovanni
Last Name:
Fidanza
Roles
EndUser
NightShift
Password:
Confirm Password:

FIG. 133

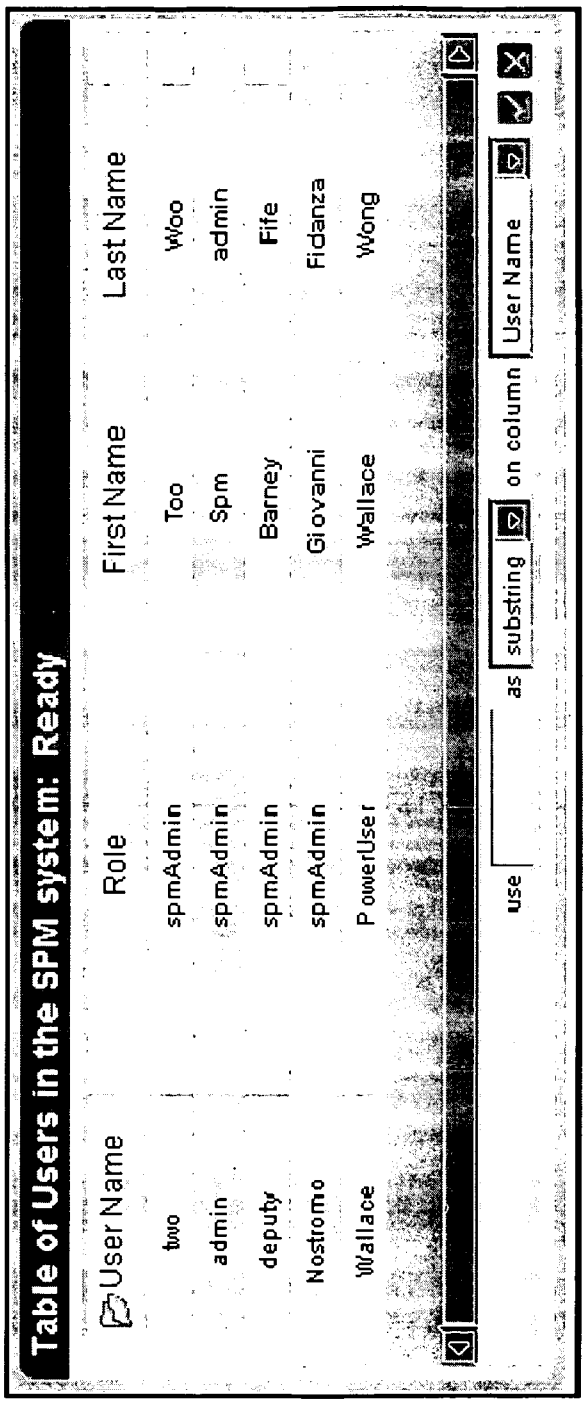


Table of Users in the SPM system: Ready
User Name
Role
First Name
Last Name
two
spmAdmin
Too
Woo
admin
spmAdmin
Spm
admin
deputy
spmAdmin
Barney
Fife
Nostromo
spmAdmin
Giovanni
Fidanza
Wallace
PowerUser
Wallace
Wong
use
as
substring
on column
User Name

FIG. 134

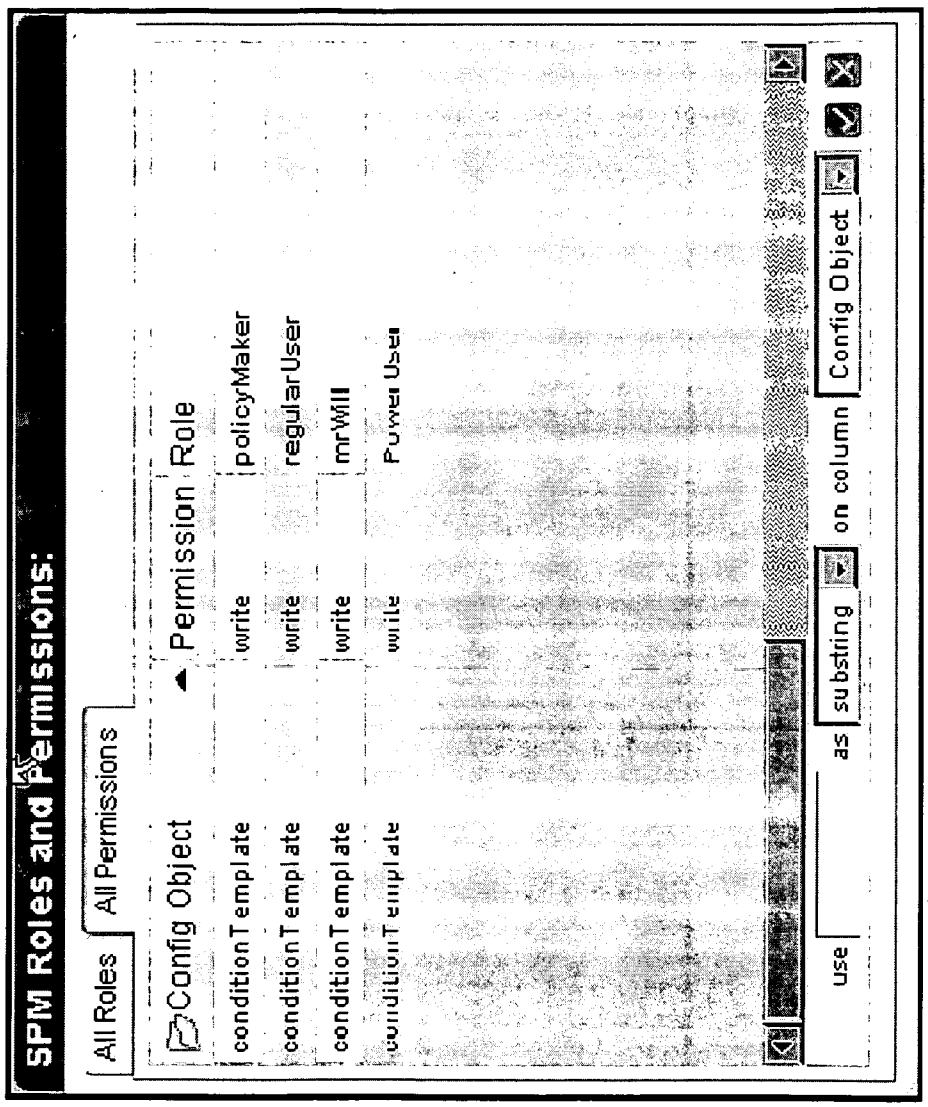

SPM Roles and Permissions:
All Roles
All Permissions
Config Object
Permission
Role
conditionTemplate
write
policyMaker
conditionTemplate
write
regularUser
conditionTemplate
write
mrWill
conditionTemplate
write
Power User
use
as
substring
on column
Config Object

FIG. 135

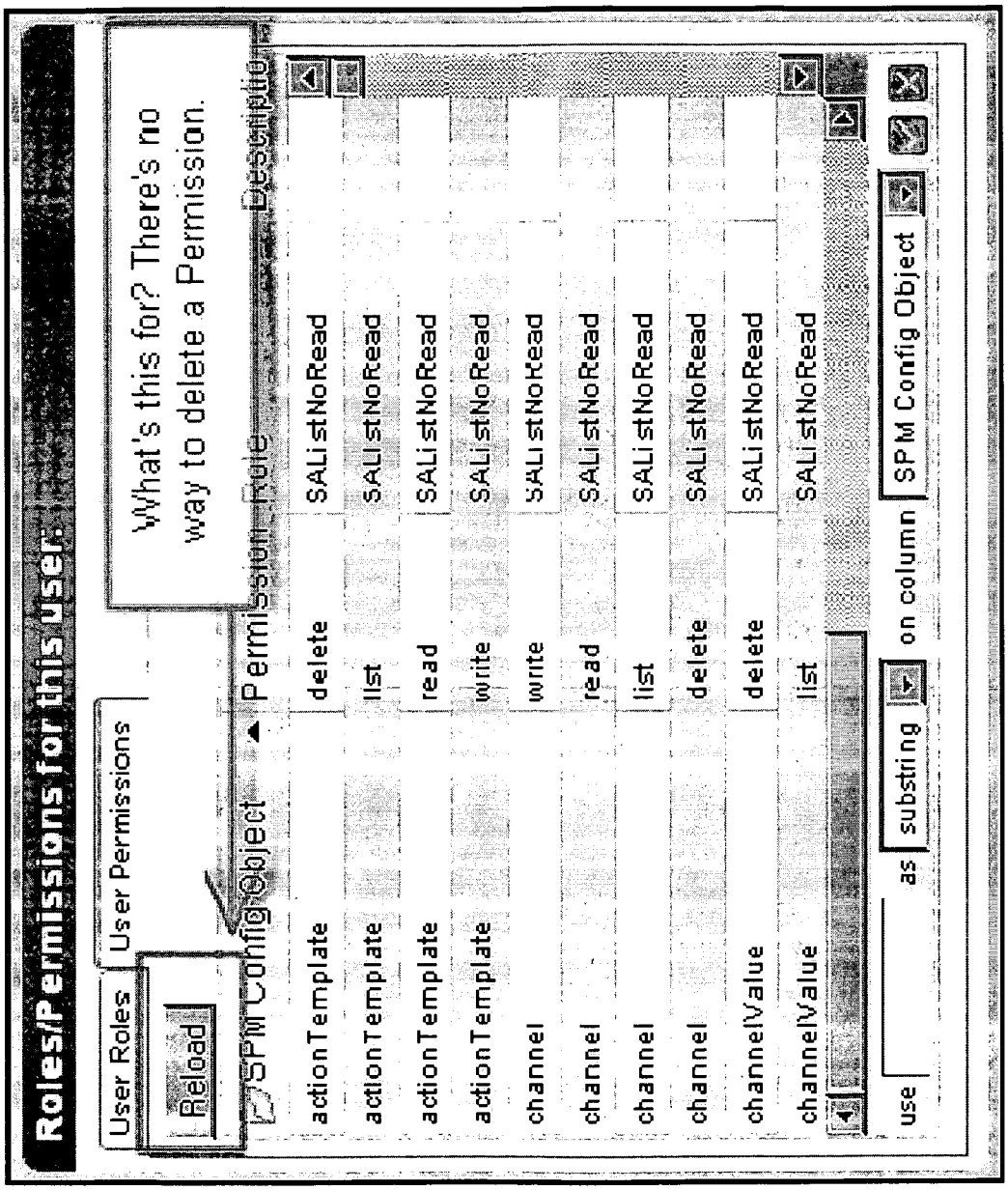

Roles/Permissions for this user
User Roles
User Permissions
Reload
What's this for? There's no way to delete a Permission.
SPMConfigObject
Permission
Role
Description
actionTemplate delete SAListNoRead
actionTemplate list SAListNoRead
actionTemplate read SAListNoRead
actionTemplate write SAListNoRead
channel write SAListNoRead
channel read SAListNoRead
channel list SAListNoRead
channel delete SAListNoRead
channelValue delete SAListNoRead
channelValue list SAListNoRead
use
as substring
on column
SPM Config Object

FIG. 136

Property name	Description	Value
Timestamp	Generic field; maintained within SPM as the number of milliseconds since 1/1/1970 00:00:00 GMT; the value is stored as a string to avoid dropping digits; with this value and the Timezone, the consumer of the data (such as the SPM GUI) converts it into the desired reader-friendly representation	String
Timezone	Generic field	String
UUID	Generic field	Sensor adapter UUID
TagID	Unique tag ID	Hexadecimal digits
TagDescription	Tag descriptions are passed in messages and alerts. They should describe what the tag is attached to in a very few words. The format for this field is: "tagid1:description1;tagid2:description2;tagid3:description3" Do not use ordinary apostrophes in tag descriptions; instead use the grave accent (lower case on Tilde key above Tab key [Alt, 096). Tag descriptions are used in Coupled Tags. Tag coupling invokes a message or alert when an owned tag is detected without its owner. For example, if Bob's Laptop is detected leaving the building without Bob, an alert condition is generated.	"1:Bob;2:Bobs Laptop;3:Bobs Briefcase"
ReadCount	Number of times the tag has been read in the current session	"3"
DiscoveryTime	Time tag was first seen. See discussion of Timestamp above.	Timestamp; 0 if it was given from the coupling list. (what is the coupling list??)
LastObservedTime	Time tag was last seen.	As above
WindowStartTime	Time current session started.	As above
CouplingType	Not coupled, or Owner, or Owned. An "Owner" tag can be detected by ifself. When an "Owned" tag is detected without its owner, an alert will be generated.	"Not coupled" or "Owner" or "Owned"
CoupledWith	An owner's owned object, or an owned object's owner or "N/A".	"N/A" or "OwnerName or "OwnedObjectName"
AntennaID	Antenna ID the tag was last read from	From Reader
ProtocolID	This refers to an Alien protocol, not to an SPM protocol.	From Reader
alert	Not used	

FIG. 137

Property name	Description	Value
Timestamp	See previous table	See previous table
Timezone	See previous table	See previous table
UUID	See previous table	
OwnerTagID	string	See previous table
OwnerTagDescription	string	As above
OwnedTagID	string	As above
OwnedTagDescription	string	As above
OwnedJustLeft	boolean	Always true !
OwnerIsPresent	boolean	true|false.

FIG. 138

Property name	Description	Value
Enabled	Sensor adapter is enabled or not	1
SimulatorMode	Integer; Most sensor adapters have the SimulatorMode property.	0 – sensor adapter uses real data from sensor 1 – runs in simulation mode: it does not retrieve real data from sensor but simply publishes the default data message according to the polling period
PollingPeriod	integer, seconds. This is sometimes also (imprecisely) called PollingRate (but it is actually a period and not a rate)	1
WindowTimeout	double, seconds The timespan over a number of polling periods in which to accumulate data: if a tag is seen within that timespan, it is declared present, even if it has a few dropouts.	2.0
DropTimeout	double, seconds – typically substantially longer than WindowTimeout: after a tag has not been detected for a time equal to DropTimeout, it is not longer classified as recently seen tag. Thus if the same tag reappears after this time, it will be registered as a new tag.	30.0
IPAddress	string	"nnn.nnn.nnn.nnn"
IPPort	integer; default port for Telnet	23

Property name	Description	Value
LoginUserName	string; default login given to the RFID reader, needed to communicate with the reader and send commands to it.	"alien"
LoginPassword	string; default password, as above	"password"
ConfigCommands	string; these commands tell the RFID reader how it should run. AutoModeReset, resets all auto mode values to their default states Automode allows unattended readers to look for tags and send notifications to SPM when certain conditions arise. PersistTime=-1, indicates that all tags are kept in a list on the reader until retrieved by SPM, at which time they are erased from the reader.	"AutoModeReset;AutoMode=ON; PersistTime=-1"
PollCommands	string; commands given to the RFID reader, of when to read data and transmit back to SPM. The command 'get TagList' tells the reader to scan the field immediately for tags and report results.	"get TagList"
CoupledTags	string of coupled tags; if the second member of the tag pair is observed without the first, an alert is generated. Referring to the TagDescriptions immediately below, the string to the right specifies that if Bob's laptop ("2") or his briefcase ("3") is detected, but Bob ("1") is not, an alert is generated. The numerals are the actual tag values that are read by the scanner.	"1-2;1-3"
TagDescriptions	string; these strings are passed in messages and alerts; see Table 6-1a	"1:Bob;2:Bobs Laptop;3:Bobs Briefcase"

FIG. 139

Property name	Description	Value
Enabled	Sensor proxy is enabled or not	integer
SimulatorMode		integer
IPAddress		string
PollingRate		double
Relay		string
Value		string

FIG. 140

Property name	Description	Value
Enabled	Sensor proxy is enabled or not	1
IPAddress		
Port		
PollingRate		
ReceiveTimeOut		
Timestamp	string	
ObservationNumber	integer	
FilteredDoseRate	double	
UnfilteredDoseRate	double	
MissionDose	double	
PeakRate	double	
Temperature	double	
location	LocationType	
geoLocation	GeoLocationType	

FIG. 141

Property name	Description	Value
Enabled	Sensor proxy is enabled or not	1
IPAddress	string	"11.22.33.212"
port	Integer	2902

FIG. 142

Property name	Description	Value
Sample_ID	<to come>	string
Flow_Rate	<to come>	double
Collection_time	See TimeStamp in Table 6-1a.	string
Collection_time_timezone	See TimeZone in Table 6-1a.	string
Location	Location	LocationType
geoLocation	GeoLocation	GeoLocationType
Lab_Name		string
Location	Location	LocationType
geoLocation	GeoLocation	GeoLocationType

FIG. 143

Property name	Description	Value
Application_Name		string
File_export_time		string
File_export_time_timezone		string
Report_User_Name		string
Test_Type		string
Assay		string
Assay_Version		string
Assay_Type		string
Operator		string
Notes		string
Start_time		string
Start_time_timezone		string
End_time		string
End_time_timezone		string
Reagent_Lot_ID		string
Expiration_Date		string
Cartridge_SN		string
Module_Name		string
Module_SN		string
Instrument_SN		string
SW_Version		string

FIG. 144

Property name	Description	Value
DataGroupInUse		integer
Ct		double
EndPt		double
Analyte_Result		string
Probe_Check_Result		string
Curve_Fit		string

FIG. 145

Property name	Description	Value
Status		string
Error_Status		string
Test_Result		string
Bg		GeneXpertIndividualAssayResult
FAM		GeneXpertIndividualAssayResult
H5Sub		GeneXpertIndividualAssayResult
H7Sub		GeneXpertIndividualAssayResult
IC		GeneXpertIndividualAssayResult
LIZ		GeneXpertIndividualAssayResult
pMat01		GeneXpertIndividualAssayResult
pMat02		GeneXpertIndividualAssayResult
pX01		GeneXpertIndividualAssayResult
pX02		GeneXpertIndividualAssayResult
SPC		GeneXpertIndividualAssayResult

FIG. 146

Property name	Description	Value
Timestamp	string	string
Timezone	string	string
UUID	string	string
Headline	string	string
Sample_info	Sample information	GeneXpertSampleInfo
Lab_info	Lab demographic information	GeneXpertLabInfo
Assay_info	Assay information	GeneXpertAssayInfo
Assay_results	Assay result	GeneXpertAssayResults

FIG. 147

Property name	Description	Value
Enabled	string	string
SimulatorMode	string	string
PollingRate	string	string
//ExportPath	pathname; as of 8/2/07 parser cannot handle path yet due to embedded double back slashes; path provisionsally hard-coded in C++ code	C:\\GeneXpert\Export\
Sample_info	Sample information	GeneXpertSampleInfo
Lab_info	Lab demographic information	GeneXpertLabInfo
Location	Location	LocationType
geoLocation	decimal longitude and latitude	GeoLocationType

FIG. 148

Gas	PPM		PPM		PPM
Ammonia	0-100	Arsine	0-0.2	Boron Trichloride	0-8
Boron Trifluoride	0-8	Bromine	0-0.4	Carbon Dioxide	0-2%
Carbon Monoxide	0-100	Chlorine	0-2	Chlorine Dioxide	0-0.4
Chlorine Trifluoride	0-0.8	Diborane	0-0.4	Dichlorosilane	0-8
Disilane	0-20	Fluorine	0-4	Germane	0-0.8
Hydrogen (LEL)	0-100%	Hydrogen (ppm)	0-1000	Hydrogen Bromide	0-8
Hydrogen Chloride	0-8	Hydrogen Cyanide	0-20	Hydrogen Fluoride	0-12
Hydrogen Selenide	0-0.4	Hydrogen Sulfide	0-40	Methane (LEL)	0-100%
Nitrogen Dioxide	0-12	Nitrogen Oxide	0-100	Nitrogen Trifluoride	0-40
Oxygen	0-25%	Ozone	0-0.4	Phosphine	0-1.2
Phosphorous Oxychloride	0-0.8	Silane	0-20	Silane (low level)	0-2
Sulfur Dioxide	0-8	Sulfur Tetrafluoride	0-0.8	Tetra Ethyl Ortho Silicate	0-40
Tungsten Hexafluoride	0-12				

FIG. 149

Property	Description	Value
Enabled	integer	integer
SimulatorMode	integer	integer
IPAddress	string	string
port	integer	integer
PollingRate	double	double
buildingLocation	Building Location	BuildingLocationType
geoLocation	GeoLocation	GeoLocationType

FIG. 150

Property name	Description	Value
DeviceState	string	"OK"
CommunicationState	string	"OK"
UpdateTime	string	timezone and time
DeviceName	string: hostname of SPM server or computer to which sensor is attached	"SPM-Serv1", "Device04"
DeviceCategory	string; role of computer	node server|sensor server
DeviceType	string; description name	"Motion Detector"
Longitude	string; degrees	
Latitude	string; degrees	
Altitude	string; meters	
Range	string; distance to observed object	"1000"
ElevationAngle	string; degrees from horizontal	0 to 90
Azimuth	string: degrees from true north	-180 to 180
FieldOfView	string: width of field	"45"
ID	string: UUID of detection event	
DetectionEvent	string: name of event or event tyoe	"Biological Agent"
Details	string: comment on event	
Assessment	string; assessment	"Positive", "Negative"
Annotation	string: comment on Assessment	

FIG. 151

Property name	Description	Value
Enabled	Sensor proxy is enabled or not	1
IPAddress	IP address of Inova server	"11.22.33.212"
port	integer	nnnn
PollingRate	double	1.0
userName	string	"admin"
password	string	"trustno1""
defaultMessage	string; used if currentMessage is not specified	"SPM - ViaLogy"
commandDuration	integer	60
defaultColor	string	"green"
commandMessage	string	""
currentMessage	integer; numerical message identifier	100
commandColor	string	"red"
commandRefreshTime	string	"5"
commandFontSize	string	'14"
commandDisplayMethod	string	"ribbon_left"|"ribbon_right"|"scroll_up"|"scroll_down"
useDefaultMessage	integer	1

FIG. 152

Properties	Description	Value
systemId	integer	
gBar	integer; bar indicator showing concentration of nerve agent	<range??>
hBar	integer; bar indicator showing concentration of blistering agent	<range??>
gAgentType	integer; nerve agent type	
hAgentType	integer; blistering agent type	
sysStatus1	integer	
sysStatus2	integer	
gDose	double;	
hDose	double	
gHighestAgentType	integer	
hHighestAgentType	integer	
seconds	integer	
minutes	integer	
hours	integer	
dayOfMonth	integer	
month	integer	
year	integer	

FIG. 153

Properties	Description	Value
vcc	integer	
inletFanCurrent	integer	
recircFanCurrent	integer	
waterIntake	integer	
pressure	integer	
temperature	integer	
batteryVoltage	integer	
positiveHT	integer	
negativeHT	integer	
gNoise	integer	
hNoise	integer	
positiveDacOffset	integer; digital to analog conversion offset	
positiveDacScalar	integer	
negativeDacOffset	integer	
negativeDacScalar	integer	
dspParamCR	integer; digital signal processing parameter	
coronaControlModes	integer	
dspParamPC	integer	
dspParamDP	integer	
dspParamDN	integer	
dspParamGP	integer	
dspParamGN	integer	
coronaNumber	integer	
coronaFiringCount	integer	
operatingCycleCount	integer	
lowVxMode	integer	
cycleTime	integer	
gMobilityCal	integer	
hMobilityCal	integer	
negativeSpectrumData	integer[]	
positiveSpectrumData	integer[]	
Timestamp	string	
textBlock	string	

FIG. 154

Property name	Description	Value
Timestamp	string; generic property	1
RegisterValues	Long array; (need description)	
location	MessageType	LocationType
geoLocation	MessageType	GeoLocationType

FIG. 155

Property Name	Description	Value
Enabled	integer	1
SimulatorMode	integer	0
IPAddress	string	"64.210.18.21"
Port	integer	2101
PollingRate	double	1.0
MBAddress	integer	1
ModBusType	string	"MODBUS_RTU"
MBStartAddress	integer array [ ]	{100 200}
MBCount	integer	6
MBFunction	string	"READ_HOLDING_REGISTERS"
MBSlave	integer	1

FIG. 156

Property	Description	Value
Timestamp	string	string
gammaStr	string	string
gammaCount	Gamma dose rate: μSv/hr or mR/hr	double
neutronStr	string;	string
neutronCount	rate: counts/sec – displayed continuously	double
doseStr	string	string
doseAmount	double	double
numNuclidesFound	integer	integer
foundNuclides	string array	string []
foundNuclidesList	string	
location	string	LocationType
geoLocation	string	GeoLocationType

FIG. 157

Property name	Description	Value
Enabled	integer: Sensor proxy is enabled or not	1
SimulatorMode	integer: Specifies whether sensor adapter is processing real data or working in simulation mode	0 = getting real data 1 = simulation mode
PollingRate	string	
location	string	LocationType
geoLocation	string	GeoLocationType

FIG. 158

Property name	Description	Value
Enabled	Sensor proxy is enabled or not	0=disabled|1 = enabled|2=suspended
SensorName	Name used to address this sensor proxy for sending commands	
SimulatorMode	All (or most) sensor adapters have the SimulatorMode property.	0 – sensor adapter behaves as usual 1 – rather than trying to retrieve data from the real sensor, the sensor adapter simply publishes the default data message (according to the polling period).
remoteIPAddress	IP address and port specify the socket to which the client opens a port.	"nnn.nnn.nnn.nnn"
remoteIPPort		nnnn
PollingRate	Polling rate is in seconds	nn
DataMessageType	The data and alert message types must match to entries in the Message Types section of the SPM file.	
AlertMessageType		
NameOfDetectedGas	An array of name/value pairs.	None; does not need to be specified.

FIG. 159

Property name	Description	Value
Enabled	Sensor proxy is enabled or not	1
IPAddress	IP address of sensor	"11.22.33.212"
port	Port on sensor	nnnn
PollingRate	seconds	1.0
SimulatorMode	<need modes>	
AssayType	Trivial name of pathogen	"Anthrax"|"Ricin"|"SEB"
HazMat	<need explanation>	
Location	Location	Bldg name/st address
Geolocation	GPS Location	GPD coordinate

FIG. 160

Property name	Description	Value
Enabled	Sensor proxy is enabled or not	1
TrapPort	integer	
PollingRate	double	
ReadCommunityString	string	
WriteCommunityString	string	
NotificationCommunityString	string	
SPModel	string	

FIG. 161

Property name	Description	Value
Enabled	Sensor proxy is enabled or not	1
HTTPGetURL	The URL for the Voxeo service	session.voxeo.net/VoiceXML.start
NumberToDial	Phone number to dial	0123456789
tokenID	Token ID	provided by Voxeo service
messageToSend	Message to send over voice	
recvTimeout	HTTP Get timeout (higher value to allow for call response)	180

FIG. 162

Result Value	Description
No answer	The call was not answered
Busy	The line is busy
Call number is bad	The number provided was not a validly formatted number (stop that!)
Invalid token	The token provided was not a valid token.
Internal error	There was an error in the token initiation
Not specified	An error occurred with no specific reason given

FIG. 163

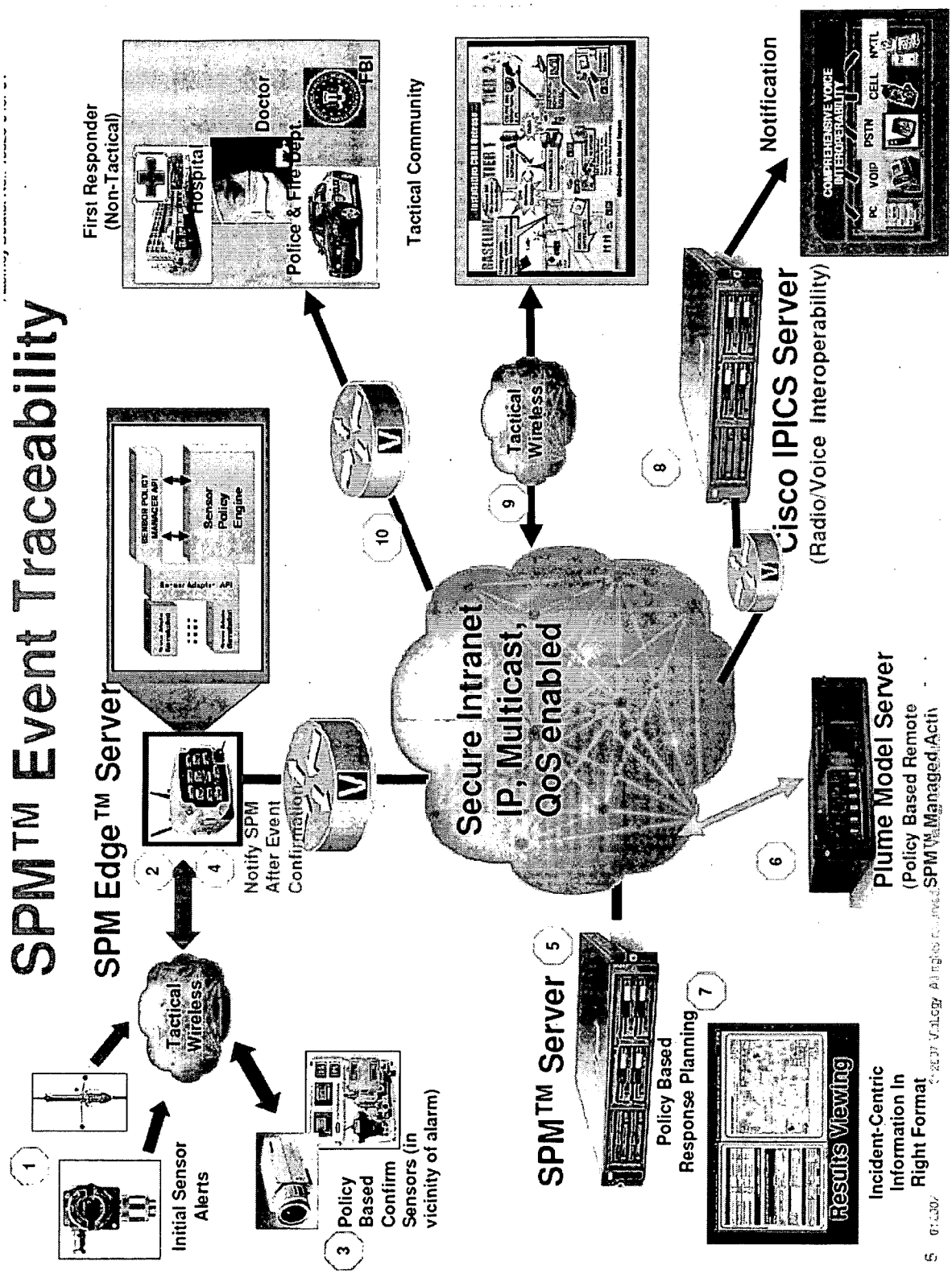

SPM™ Event Traceability
SPM Edge™ Server
Sensor Policy Engine
First Responder (Non-Tactical)
Hospital
Doctor
Police & Fire Dept.
FBI
Tactical Community
Tactical Wireless
Tactical Wireless
Notify SPM After Event Confirmation
Secure Intranet IP, Multicast, QoS enabled
Cisco IPICS Server
(Radio/Voice Interoperability)
Notification
COMPREHENSIVE VOICE INTEROPERABILITY
PC
VOIP
PSTN
CELL
NXTL
Plume Model Server
(Policy Based Remote
SPM™ Managed Acti
SPM™ Server
Policy Based
Response Planning
Results Viewing
Incident-Centric
Information In
Right Format
Initial Sensor
Alerts
Policy
Based
Confirm
Sensors (in
vicinity of alarm)
1
2
3
4
5
6
7
8
9
10

FIG. 164

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2008/072727

A. CLASSIFICATION OF SUBJECT MATTER

H04L 12/22(2006.01)i, H04L 12/12(2006.01)i

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)
IPC: H04L, G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched
Korean Utility models and applications for Utility models since 1975

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
WPI, eKIPASS(KIPO internal) & keywords: sensor, policy management

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X Y A	ZHOU YING et al. 'Mobile Agent-based Policy Management for wireless Sensor Networks' In: IEEE Wireless Communications, Networking and Mobile Computing, 2005. Proceedings. 2005 International Conference on Volume 2, 23-26 Sept. 2005 Page(s):1207 - 1210 Digital Object Identifier 10.1109/WCNM.2005.1544270 See figs. 1, 3, 4; sections Ⅰ, Ⅲ, Ⅳ. B-C.	4, 5, 7-26 1 2, 3, 6
X Y A	US 2005-256947 A1 (MURTHY V. DEVARAKONDA et al.) 17 November 2005 See abstract; figs. 1, 3; paragraphs [3], [5], [44], [47], [48], [51].	2, 3, 6, 27, 28 1 4, 5, 7-26
A	US 2007-27966 A1 (RAJIV SINGHAL et al.) 1 February 2007 See abstract; paragraphs [13], [15], [16], [19], [52]-[54], [65], [68], [69].	7, 15-22

☐ Further documents are listed in the continuation of Box C.

☒ See patent family 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 application or patent 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 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 MAY 2009 (15.05.2009)	**15 MAY 2009 (15.05.2009)**

Name and mailing address of the ISA/KR
Korean Intellectual Property Office
Government Complex-Daejeon, 139 Seonsa-ro, Seo-gu, Daejeon 302-701, Republic of Korea
Facsimile No. 82-42-472-7140

Authorized officer
YANG, CHAN HO
Telephone No. 82-42-481-5689

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.

PCT/US2008/072727

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2005-0256947 A1	17.11.2005	None	
US 2007-0027966 A1	01.02.2007	None	