Conceptual & Concrete Architectures of Open Network Management System (OpenNMS)

Analyzed by

BASEM SHIHADA

University of Waterloo
Dept. of Computer Science
200 University Avenue West
Waterloo, Ontario, Canada
(519) 8851211 ext. 5387

CS798 Final Project

2nd April 2002

Table of Contents

1 Abstract:

Developers of open source software require high-level component description, inter-component relation description and a detailed component functionality description. Most of the current open source implementations suffer form the availability of a comprehensive description of the software components architecture. This results a miss-concept in the architecture of the software or a future mistakes in determining how the software components communicate between each other. Furthermore, by the growth of the software, the developer becomes unable to identify or to analyze the software components relations, which will make the future modification sometimes very expensive. This results that some open source software goes to end. Within this project I have chosen an open source network management product called OpenNMS. This provides a conceptual and concrete architecture of OpenNMS. Also analyze the interconnected sub-systems and their components.

OpenNMS provides an open-source alternative to commercial proprietary network management products e.g. network management products from Cisco, Nortel, IBM or others. OpenNMS is written in JAVA and consist of 140,475 line of code. The scope of this project will cover the conceptual and concrete architecture of OpenNMS. On the other hand the project will focus on the software architecture issues and will not deal with any networking issue. Different architecture styles are expected to be seen within this product. Since this product has the ability to automatically detect and service any network, then its expected that there will be different layer styles applied. Since OpenNMS interacts with different other network services such as ICMP, SNMP (v1 for data collection), FTP, HTTP, SMTP, DNS, Router, Sybase (TCP), MySQL, Postgres ,Oracle (TCP) ,DHCP ,MSExchange, IMAP, and POP3, several software interfaces are built to organize object interactions [1].

The goal of this project is to make the reader become familiar with the basic functions of a network management system. Further more, analyze the architectural functionality of OpenNMS and build conceptual and concrete architectures to this network management product. By the end of this project you will be able to know the software major components “sub-systems”, the component relations and the whole functionality of the software architecture. In this project SNIFF+ has been used as a reverse engineering tool kit to construct the concrete architecture.

2 Introduction

Network management means different things depending where its used. In some cases, it involves a solitary network consultant monitoring network activity with an outdated protocol analyzer. In other cases, network management involves a distributed database, auto-polling of network devices, and high-end workstations generating real-time graphical views of network topology changes and traffic. In general, network management is a service that employs a variety of tools, applications, and devices to assist human network managers in monitoring and maintaining networks [2]. In this project, OpenNMS has been selected as an open source implementation of all mentioned network management issues.

2.1 Network Management

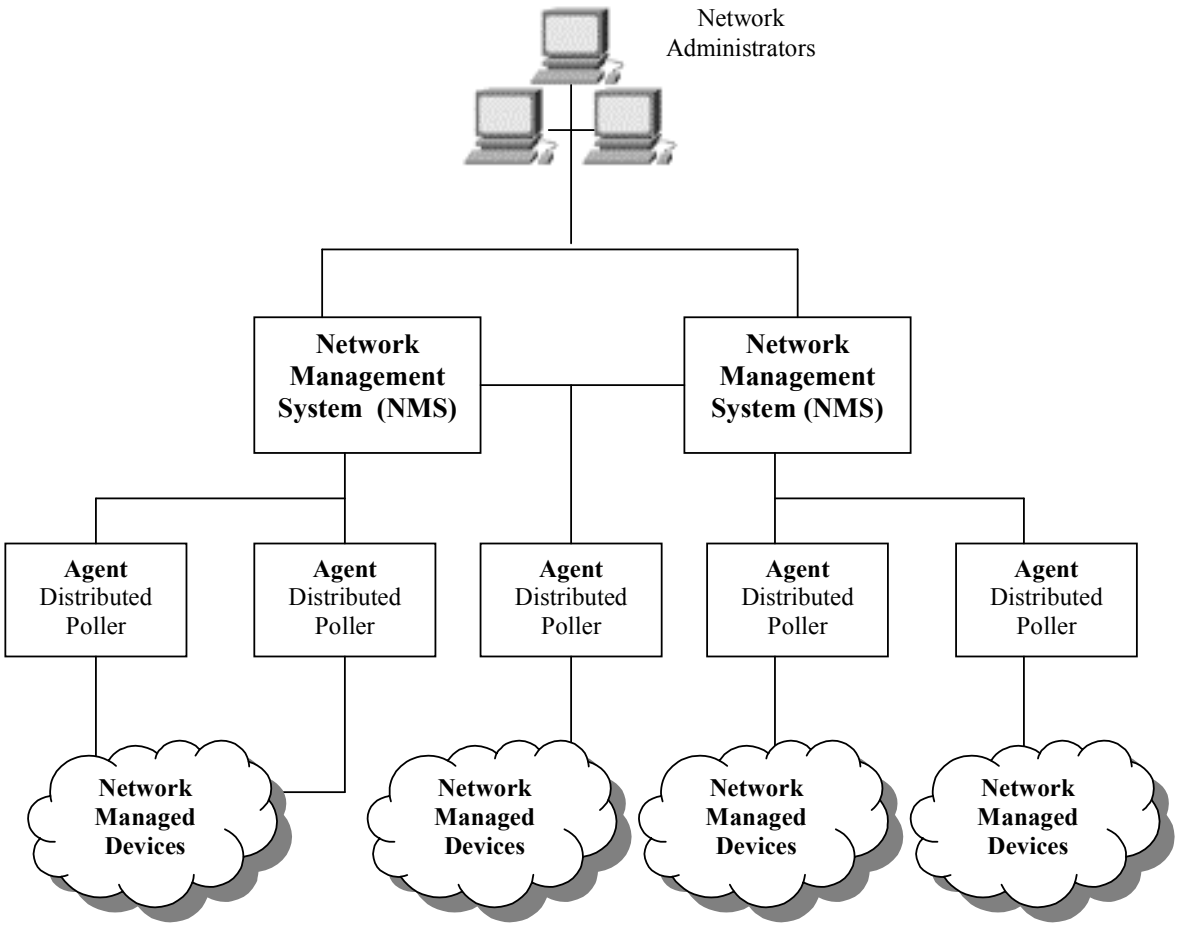
Most network management architectures use the same basic structure and set of relationships, figure 1 illustrates an abstract view of OpenNMS depending on the general view of network management architectures. End stations (managed devices), such as computer systems and other network devices, run software that enables them to send alerts when they recognize problems (for example, when one or more user-determined thresholds are exceeded). Upon receiving these alerts, management entities are programmed to react by executing one, several, or a group of actions, including operator notification, event logging, system shutdown, and automatic attempts at system repair [1,2].

Management entities also can poll end stations to check the values of certain variables. Polling can be automatic or user-initiated, but agents in the managed devices respond to all polls. Agents are software modules that first compile information about the managed devices in which they reside, then store this information in a management database, and finally provide it (proactively or reactively) to management entities within network management systems (NMSs) via a network management protocol. Well-known network management protocols include the Simple Network Management Protocol (SNMP) and Common Management Information Protocol (CMIP). Management proxies are entities that provide management information on behalf of other entities [2].

2.2 OpenNMS

OpenNMS provides an open-source alternative to company manufactured network management products. Also has the ability to change from traditional network management products in its focus on services versus infrastructure. OpenNMS's focused on how the network topology was configured, and then attempted to divine the status of services based upon the status of routers, switches and servers. In addition, OpenNMS focuses on the services themselves, such as e-mail, web servers and databases. By measuring the performance and availability of these services, one can more closely capture the experience of the end-users [1,2].

OpenNMS is written in JAVA and consist of 140,475 line of code. The configuration data is available directly via extensible markup language (XML) files. One of the most important features of this product is that its able to work in a distributed hieratical manner and can monitor several services such as, ICMP, SNMP, FTP, HTTP, SMTP, DNS, Router TCP, Sybase (TCP), MySQL, Postgres, Oracle (TCP), DHCP, MS Exchange (relies on banners for identification), IMAP, POP3, figure 1 shows how the system works in a distributed manner [1,2].



Abstract view of OpenNMS as SNMP architecture

2.3 Project Outline

The rest of the project is outlined as follows; section 3 analyzes OpenNMS depending on the conceptual view of the system architecture with a summary of the basic characteristics of each sub-system. It also addresses how dose the system components interact with each other. Some conceptual architecture diagrams have been drawn to show the system architecture. Chapter 4 covers a summary survey about a commercial reverse engineering toolkit called SNIFF+. This toolkit, which support Java, has been used to construct the system concrete architecture. Chapter 5 focuses on the resulted SNIFF+ concrete architecture and describes how the system components are related to

each other. This section also comments on why there was a gap between what has been built in the conceptual architecture and what has been found in the concrete one. Section 6 specifically illustrates the data terminology, which has been used through the project. Finally section 7 will summarizes the achieved goals of this project and some suggestions for future work. During the project several expected examples and scenarios have been give to illustrate the system behavior under different circumstances.

3 Conceptual Architecture

This section describes the internal OpenNMS conceptual architecture and explains the expected implemented service components. It also addresses and explains the system-to-system relation levels and component distribution. Furthermore it presents to the reader the OpenNMS conceptual architectural design. A description of the architecture is illustrated using what has been analyzed from the system illustration guides, system features, examples, and other product related information. The conceptual architecture will be related or based on the abstract network management product, which explained in the previous section.
From the conceptual architecture prospective, the system consists of several Modules; figure 2 visualize OpenNMS architecture:

3.1 Administrator Graphical Interface

This is the main OpenNMS graphical user command interface, which handles the user input commands and output reports, in other words it's the OpenNMS graphical command-processing unit. The user is able to do the monitoring, node controlling, data reporting, and many other tasks.

3.2 Web Server Interface

Provides convenience functions for web-server based interfaces. OpenNMS has different web functionalities to deal with group servlets. Web node management is done so that it stores node, interface, and service information. Users functionality, web authentication, web OpenNMS principal, and web graph are all works with others to maintain the full functionality of the OpenNMS web-based interfaces.

3.3 Event Control Unit

This component is the only and the main event-handling and management component, which manages the different events coming from the network and from the network administrators. Event handling listens for events from the network servicing, network monitoring, network configuration, and from other system components then processes and sends events to the Master Station (network administrator or SNMP components) when queried for.

3.4 SNMP

Within OpenNMS there should be a centralized management unit, which is simple network management protocol SNMP, or other management protocol, which dose the node-to-nod and node-to-agent management. For example, we define management session as a communication channel between the manager and a remote agent. A session encapsulates various parameters like community strings, protocol version, and packet encoding. Once a session is created the manager code can communicate with the remote agent by sending requests and waiting for responses.

3.5 Network Servicing

OpenNMS uses IP interfacing to represent the auto action execution for any existing application services within the network. When an event is received by this service that has one of either a notification, trouble ticket, or auto action then a process is launched to execute the appropriate commands.
There are several daemon programs to do specific service functionalities such as notify by the discovery process when a new node is discovered, services to be as a multiplexor for DHCP requests and responses, services for XML configuration functionalities, TCP Receiver and the UDP Receiver receive services, filtering services and others.

3.6 Network Monitoring

OpenNMS has defined several network interfaces for passing interfaces to a service monitor. There are many different types of network in use today including IPv4, IPv6, IPX, and others. To accommodate the possible differences OpenNMS provides the basic information that a monitor can use to determine the type of interface and its expected address type. In addition to providing typing and address information, the interface allows for the monitor to associate key-value pairs with an interface. This can be used to save state information between the various invocations if necessary.

3.7 Network Configuring

This component receives its activation commands from the main centralized SNMP unit. By executing some certain management command this component considered the last operation done to configure or manage a network. For example, the configuration management commands could be applied on different levels such as application level for example, accounting, DSL, cable, and access register. Also could be cover networking level for example, IP, MPLS, and Gateways. Or could be applied on the routing level for example, router configuration, switches, and hubs.

3.8 Network Device Interfaces

OpenNMS has all the required interfaces with different network protocols and services such as, TCP/IP, ICMP, FTP, HTTP, DNS, DHCP, MSExchange, IMAP, and POP3. This interface helps the monitoring unit to discover and work with different network protocols.

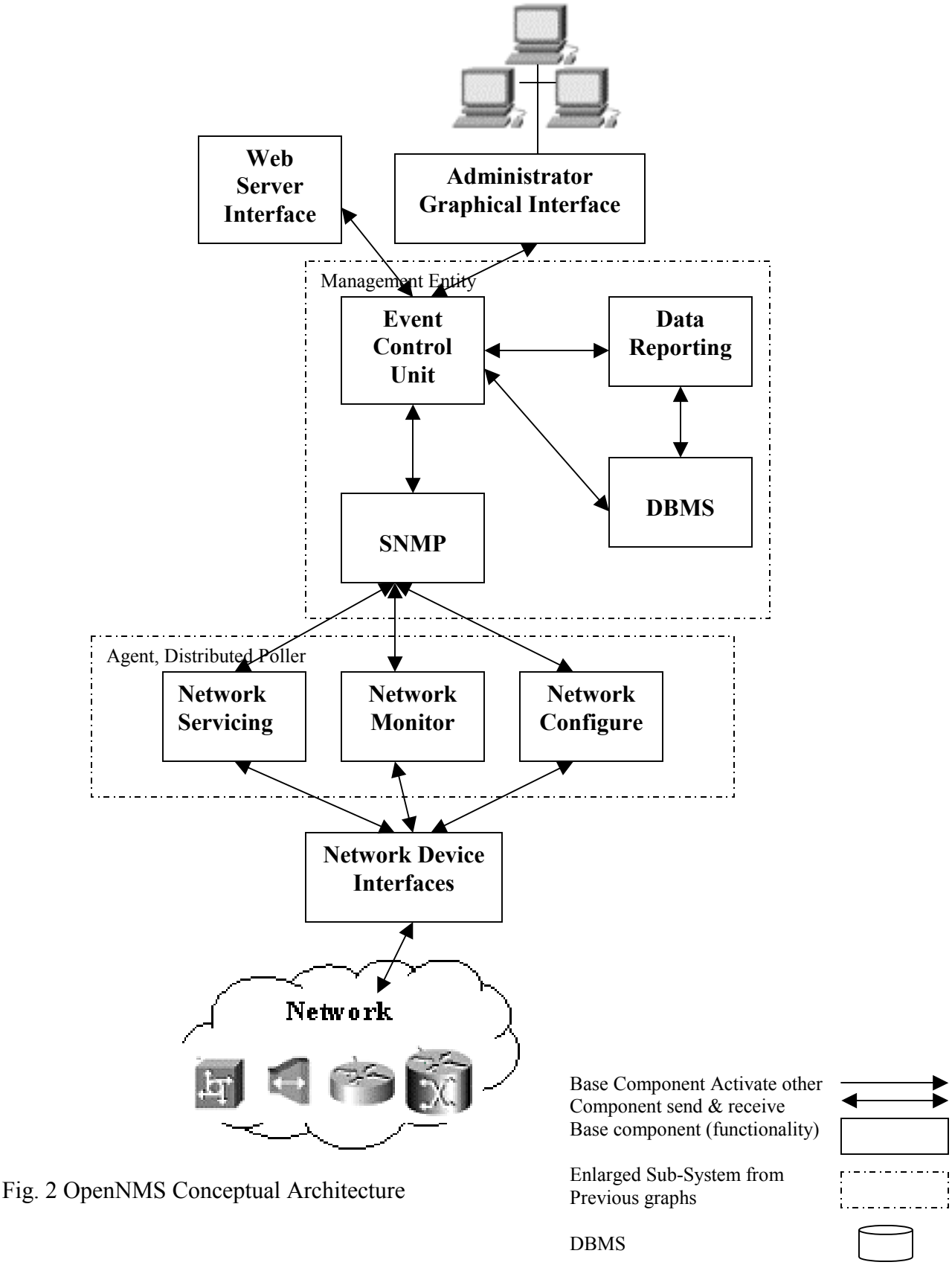


Fig. 2 OpenNMS Conceptual Architecture

3.9 Data Reporting

OpenNMS provides methods to get category-wise-reports for each category in the "views.xml". Filter is used during the process to get the IP interfaces for that category rule. Using the list of IP interfaces, node reports are created by referencing tables in the database. Finally reports are generated from the report nodes. There are some specific methods are implemented to make useful to the user when s/she wants reports.

3.10 DBMS

Within OpenNMS there is a collection and storage unit. This unit is a direct implementation of PostgreSQL. There are several functions used to store and retrieve the collected data. For example, functions used to calculate the downtime of all managed services on node between two dates, calculate the number of all managed services on node, calculate the percentage availability for all managed services on node between two dates, calculate the percentage availability of a (Node, IP Address, Service) tuple between two dates, and calculate the percentage availability for all managed services on an interface between two dates

Within this section a conceptual architecture is given and the next step is to try to compare the conceptual architecture to the concrete architecture. First, SNIFF+ as a reverse engineering toolkit, which support Java, has been used to extract the concert architecture. In the coming section a short description of SNIFF+ has been provided and the detailed concrete architecture is constructed. Comparing both the conceptual to the concrete architectures is one of the main interesting issues of this project.

4 SNIFF+ Toolkit

SNiFF+™ is a high-end source code analysis environment and SNiFF+ PRO is an open development environment for source code-centric application developers.
“There are many benefits of using SNIFF+ such as, first, increases productivity and quality by providing a comprehensive set of intuitive, interactive, and integrated code visualization and navigation tools. Second, enables teams to organize and manage code more efficiently. Third, automates the build process across multiple platforms via the SNiFF+ PRO's build solution. Forth, allows server side development from a single desktop through SNiFF+ PRO’s remote compile and debug feature. Fifth, supports many different development tools and integrates easily with a variety of popular text editors, compilers, debuggers and CM tools because of its open architecture. SNIFF+ works with any source cod written in C/C++, Java, Ada, Corba IDL, Fortran.” [3]

In the coming section, OpenNMS concrete architecture has been generated using SNIFF+. Each sub-system will be in a place of analysis for comparing them the components with the conceptual architecture or to see how it work and how it relates to others, also we will notice that there are some changed between what has been proposed in the conceptual architecture and the concrete architecture.

5 Concrete Architecture

This section analyzes each independent sub-system based on the concrete architecture generated from SNIFF+. There are many sub-systems, which all together perform certain layer functionality in OpenNMS. These functional components are grouped together depending on their base functionality provided in OpenNMS. Further more, depending on the common functionality studded in OpenNMS the functions are categorized in to several sub-systems. First, network discovery sub-system. Second, polling/monitoring sub-system. Third, event handling. Forth, event correlation. Fifth, notification. Sixth, performance data collection. Seventh, reporting. Eight, configuration management. Ninth the JoeSNMP Finally, independent components that is used to convert different XML files from version 1.1 to 2.0.
These sub-system and their entire components are first explained depending on their functionality then graphed together. Figure 3 visualizes OpenNMS concrete architecture.

5.1 Administrator Graphical Interface

This is the main OpenNMS graphical user command interface, which handles the user input commands and output reports, in other words it's the OpenNMS graphical command-processing unit. The user is able to do the monitoring, node controlling, data reporting, and many other tasks.

5.2 Web Server Interface

Provides convenience functions for web-server based interfaces. OpenNMS has different web functionalities to deal with add, delete, group, modify, rename, save and update group servlets. Web node management is done so that it stores node, interface, and service information. Users functionality such as add, delete, add new user Bean, containing data from the user info page, rename, update user, save user, remove duty from servlets schedule, save user in a servelets, or rename a user in a serveletes. Web authentication, web OpenNMS Principal, web graph are all works with others to maintain the full functionality of the OpenNMS web-based interfaces.

5.3 Event Control Unit

This system is considered the core internal functionality management for the whole system and consists of several components. The following is a description of this system and what components it consists of.

5.3.1 Event Handling

Event handling component is the core management component, which manages the different events coming from the network and from the network administrators. Event handling listens for events from the network discovery, network servicing, network monitoring, network configuration, and from other system components then processes and sends events to the Master Station (network administrator or SNMP components) when queried for.

Event Handler and Event broadcaster are both listen for incoming events, load info from the 'event.conf', add events to the database and send the events added to the database out via Java Message Service (JMS). It also maintains a service name to service ID mapping from the services table so as to prevent a database lookup for each incoming event.

5.3.2 Event Correlation

This component is used to represent the auto action execution service. When an event is received by this service that has one of either a notification, trouble ticket, or auto action then a process is launched to execute the appropriate commands.

5.3.3 Event schedule & archive

The events schedule & archive is responsible for archiving and removing events from the 'events' database table. The OutageManager receives events selectively and maintains a historical archive of each outage for all devices in the database

5.3.4 Event Notification

This component notify any discovery process when a new node is discovered it then polls for all the capabilities for this node and is responsible for loading the data collected into the database through the event handling component.
Notifd: (which is the system name inside the source code) is a sub-component, which used to represent the notification execution service. When an event is received by this service that has one of either a notification, trouble ticket, or auto action then a process is launched to execute the appropriate commands.

5.3.5 JoeSNMP

JoeSNMP stands for "Java SNMP". It is a completely "free" implementation of the SNMP protocol written entirely in Java. JoeSNMP, is based around the concept of an SNMP session. A session is a communication channel between the manager and a remote agent (Agent, Distributed Poller). A session encapsulates various parameters like community strings, protocol version, and packet encoding. Once a session is created the manager code can communicate with the remote agent by sending requests and waiting for responses through the event-handling component.

Each management application requires an object that implements the SNMP-Handler interface. The SNMP-Handler interface is responsible for processing received SNMP Protocol Data Units (PDU) on behalf of the application. If an error occurs with the session then the handler is informed of the error, not the function that created the session. Also a utility unit is provided to handle all the functionalities, which dose not has direct communication with JoeSNMP [4].

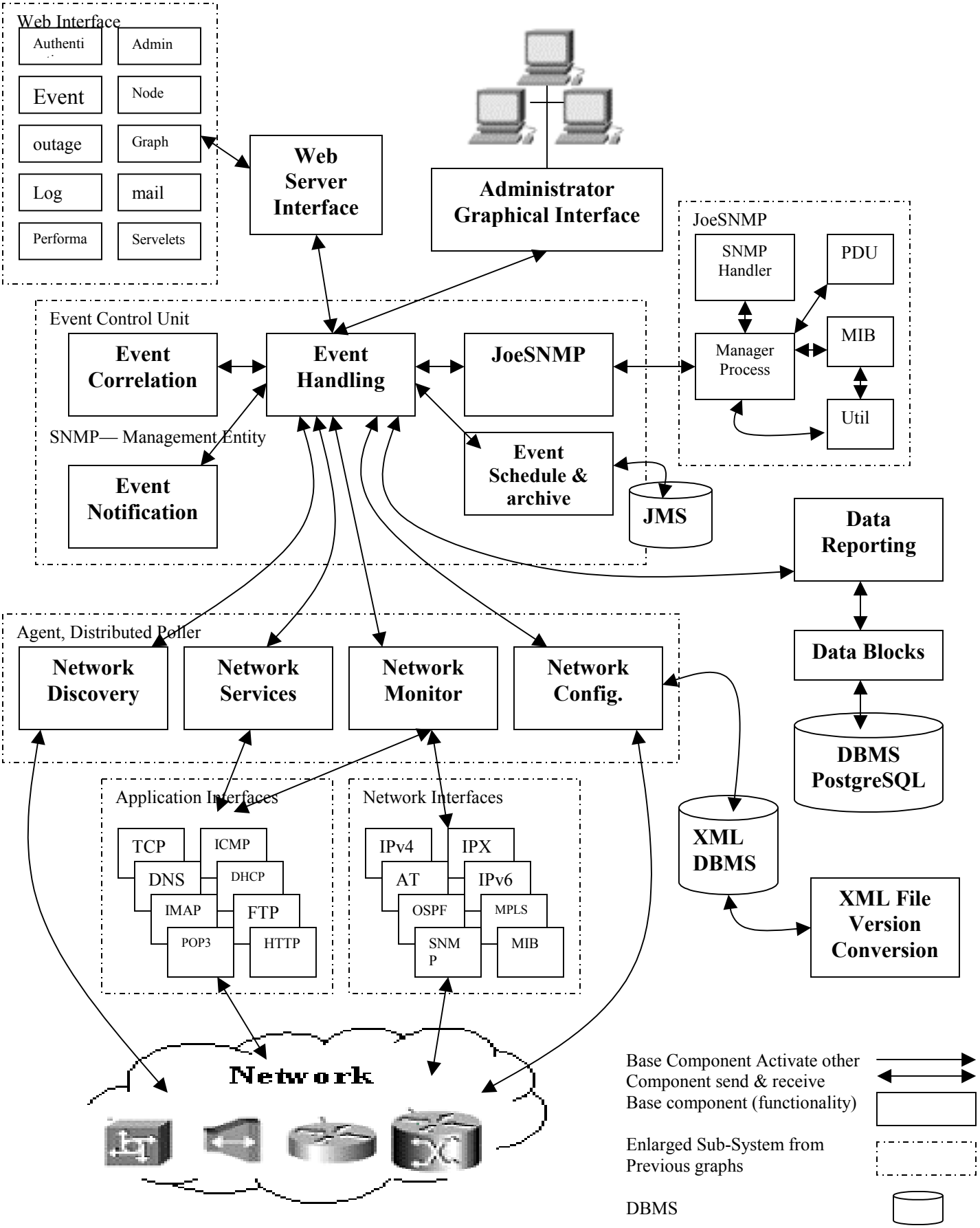


Fig. 3 OpenNMS Concrete Architecture

5.4 Agent, Distributed Poller

This system is considered the core low-level distributed network management agent. For OpenNMS to function and apply the management commands there are four implemented components, which are working on, different network layers to handle the different network stages. The following is a description of this system and what components it consists of.

5.4.1 Network Discovery

This component is the main interface to the OpenNMS network discovery service. Further more, it's considered the first activated component after the system launce to discover and analyze the network devices. The resultant data is returned and stored in the DBMS thought the event-handling component. The component implements the singleton design pattern, in that there is only one instance in any given network device. The service delays the reading of configuration information until the service is started.

5.5.2 Network Services

OpenNMS uses IP interfacing to represent the auto action execution for any existing application services within the network. This component is responsible of interacting with the current existing application services within the network. When an event is received by this service that has one of either a notification, trouble ticket, or auto action then a process is launched to execute the appropriate commands. Some of these services are FTP, SMTP, ICMP, POP, and Voice over IP.

5.5.3 Network Monitor

This component is considered the core functionality stage regarding network management process. It's an on demand-activated component, usually its activated immediately after the network discovery unit. This component consists of different functionalities such as, schedule: This class implements a simple scheduler to ensure the polling occurs at the expected intervals. RRD data source, Pop3 Monitor, Network Interface: This class is designed to be a well defined front for passing interfaces to a service monitor, including IPv4, IPv6, IPX, Apple Talk (AT) and others, MIB Threshold, MIB Object, LDAP Monitor, IMAP Monitor, ICMP Monitor, HTTP Monitor, FTP Monitor, DNS Monitor, DHCP Monitor, TCP Monitor, SNMP Monitor, SMTP Monitor, and SMB Monitor.

5.5.4 Network Config.

This component receives its activation commands from the main centralized JoeSNMP unit. By executing some certain management command this component considered the last operation done to configure or manage a network. For example, the configuration management commands could be applied on different levels such as application level for example, accounting, DSL, cable, and access register. Also could be cover networking level for example, IP, MPLS, and Gateways. Or could be applied on the routing level for example, router configuration, switches, and hubs.

5.6 Data Reporting

Data reporting component used to get category-wise-reports for each category in the "views.xml". Filter is used during the process to get the IP interfaces for that category rule. Using the list of IP interfaces, 'Report Node's are created by referencing tables in the database. Finally reports are generated from the report nodes.
The most important method getDataForView is activated whenever the network admin. wants certain reports. Data blocks are the defined structures for different data representation for reporting.

5.7 DBMS PostgreSQL

Postgresql is the used database management system. This database is considered the place where the collected network data is stored.
Structure Iplike is used to hold list of octets that are read from the match text. There are many sub-functions working over the data. The following is a summary list of them.

- GetManagedOutageForNodeInWindow: Is used to calculate the downtime of all managed services on node between two dates.
- GetManagedServiceCountForIntf: Is used to calculate the number of all managed services on node.
- GetManagedServiceCountForNode: Is used to calculate the number of all managed services on node. It returns their number.
- GetManagePercentAvailNodeWindow: is used to calculate the percentage availability for all managed services on node between two dates.
- GetManagedOutageForIntfInWindow: is used to calculate the downtime of all managed services on node between two dates.
- GetPercentAvailabilityInWindow: is used to calculate the percentage availability of a (Node, IP Address, Service) tuple between two dates.
- GetOutageTimeInWindow: is used to calculate the downtime of a (Node, IP Address, Service) tuple between two dates.
- GetManagePercentAvailIntfWindow: is used to calculate the percentage availability for all managed services on an interface between two dates.

5.9 XML File Version Conversion

This is an independent component used to covert XML files from version 1-1 to 2-0. The converted XML files are basically the files, which hold the network configuration management commands. Such as, views.xml, groups.xml, users.xml, snmp.xml to snmp-config.xml, services.xml/packages.xml to capsd-configuration.xml, packages.xml to capsd-configuration.xml, packages.xml to poller-configuration.xml, packages.xml packages.xml to poller-configuration.xml, packages.xml to discovery-configuration.xml packages.xml to discovery-configuration.xml.

6 SCENARIOS

OpenNMS tasks can be traced into two ways, the network-to-admin and admin-to-network.

6.1 Bottom to up:

Assume that the performance of the network and its response time or throughput starts to deteriorate as a result of increased levels of traffic in selected parts of the network [5]. Within the agent, distributed poller sub-system, first the network monitor component is going to be notified by the network interface component regarding the problem. Second, the information is going to be reported to the network administrator, goes to JoeSNMP and then stored in the DBMS.
If the administrator controls the system, the event handling component will wait until the network administrator send a management command. Otherwise the JoeSNMP unit will automatically read the contents of the management information base (MIB) component and apply the default management commands. These commands will be transferred to the network config component, which will apply it to the selected network through the network interfaces. Then the problem is solved.

During all the mentioned process stages, the system stores each thread in the event schedule & archive component with an assistance of the event correlation and event notification components.

6.2 Up to bottom:

Suppose that the network administrator wants to increase the network bandwidth to a selected part of the network [5]. The network administrator request some reports from the data reporting component. These reports will show the network status and the network component information, which were stored during the network functionality. The network administrator sends a certain management command to the event handling unit. The thread is created, scheduled and archived. Then passed to the JoeSNMP unit. Depending on the stored management information in MIB, the management information is prepared and sends to the network config component. Then, the network config components create the corresponding low-level management command and send it to the network via the network interface. Finally, the network monitor will read the new changes and return it to the network administrator via the event handling component.

As mentioned in 6.1, during every process stages the system stores each thread in the event schedule & archive component with an assistance of the event correlation and event notification components.

7 CONCLUSION

This project presented conceptual and concrete architectures for Open source Network Management System (OpenNMS). This project is the process of visualizing the interconnected software components functionality and management. OpenNMS is a Java based software system, which consist of 140,475 line of code. SNIFF+ has been used as a reverse engineering toolkit to construct the concrete architecture.

The primary contribution of this project is to visualize the internal software architecture of OpenNMS and analyze it from the conceptual and the concrete aspects. The goal is achieved by generating both conceptual and concrete architectures, which increases the level of system understanding and make it easier to find future errors or insert more levels of functionalities in their right places. The following goals were achieved in the design:

1. Analyzing a network based management systems.
2. Drawing or visualizing the system components conceptually.
3. Analyzing the low-level system functionality from the concrete architecture.
4. Have a chance to work on one of reverse engineering toolkits, SNIFF+
5. Discover how the network-based software is internally designated and function.

However, additional work needs to be done to study the full compoent-to-compoent and object-to-object effects and interactions on a distributed environment. And how can we draw architecture for OpenNMS putting in consideration that it could be an enterprise system.

8 ACRONYMS

The following table provides some repeated acronyms repeated through the project.

OpenNMS	Open Network Management System
ICMP	Internet Control Message Protocol
SNMP	Simple Network Management Protocol
FTP	File Transfer Protocol
HTTP	Hyper Text Transfer Protocol
SMTP	Simple Mail Transfer Protocol
DNS	Domain Name Server
TCP/IP	Transfer Communication Protocol
DHCP	Dynamic Host Configuration Protocol
CMIP	Common Management Information Protocol
XML	Extendable Markup Language
IMAP	Internet Message Access Protocol
POP3	point of presence
MIB	Management Information Base
JMS	Java Message Service
PDU	Protocol Data Unit
AT	Apple Talk
RRD	Relative Rate Data

JoeSNMP	Java Simple Network Management Protocol
SNIFF+	Reverse Engineering Toolkit

9 REFERENCES

[1] Open Network Management System main web page
http://www.opennms.com/

[2] Cisco documentation (SNMP)
http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/snmp.htm

[3] SNIFF+ Reverse engineering toolkit
http://www.windriver.com/products/html/sniff.html

[4] JoeSNMP free Java implemenation of SNMP
http://kde.planetmirror.com/pub/opennms/releases/joeSNMP/

[5] Application Scenarios
Fred Halsall, Data Communication, Computer Networks and open systems. Fourth edition, Addison-wesley 1997.