Design and Implementation of an XML-Based Management Agent

Mi-Jung Choi, Jung-Min Oh and James W. Hong

Dept. of Computer Science and Engineering, POSTECH, Korea
{mjchoi, meanie, jwkhong}@postech.ac.kr

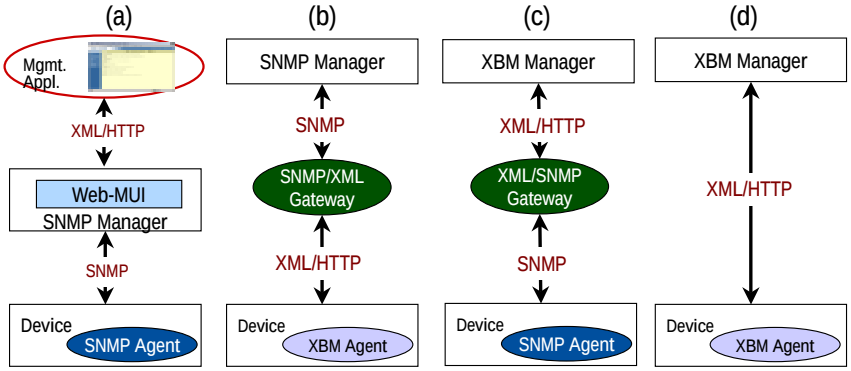
Abstract

As the Internet evolves, many network devices continue to emerge. Most of these network devices are basically equipped with an SNMP agent and an Embedded Web Server (EWS) as a management module. While both have drawbacks in network management, the drawback of SNMP concerns configurability and, with EWS, the absence of a management information model. To resolve these drawbacks, XML-based network management has been proposed as an alternative. To maximize the advantages of XML-based network management, such as defining a variety of management information by XML Schema and performing efficient configuration management, both the manager and the agent must be able to process XML documents. Although a few network device manufacturers such as Cisco and Juniper Networks have developed and equipped XML-based agents on their own devices, they merely concentrate on configuration management. In this paper, we propose the design and implementation of lightweight and efficient XML-Based Management (XBM) agent. It is small and portable enough to be embedded in any network device, and performs fault management, performance management, and configuration management. We also verify the effectiveness of our XBM agent by comparing the performance with an SNMP agent.

Keywords: XML, XML-Based Management, XML-Based Management (XBM) Agent, SNMP

Introduction (1)

❖ Combinations of Manager and Agent



❖ Goals

- Propose requirements and an architecture of an XML-Based Management (XBM) agent & Explain implementation detail
- Analyze & summarize the performance of XBM agent

The rapid pace of Internet evolution is currently witnessing the emergence of a variety of network devices. Most of these network devices are basically equipped with an SNMP [1] agent and an Embedded Web Server (EWS) [2] for system and network management. The SNMP management framework has been applied since 1988, and has some weaknesses especially related to configuration management and the application development process to manage huge networks [3]. EWS provides the Web user interface, but has difficulty in defining management information due to the absence of a management information model. Therefore, it is insufficient for central network management system, while it is adequate for providing management user interface to each device [4].

XML-based network management has been suggested as an alternative to solve the drawbacks of SNMP and EWS. An Extensible Markup Language (XML) [5] provides powerful modeling features for structured management information in network management using the XML Schema [6], and XML-based network management can transfer and configure a large amount of configuration data over HTTP. Also, the XML-based network management can be easily implemented using standard API and freely available XML software. The management information written in XML document can solve the drawback of EWS, the absence of a management information model.

As depicted in the above figure, four possible combinations between managers and agents can be considered for XML-based integrated network management. (a) shows the current, widely deployed SNMP-based network management, and (d) shows XML-based management using an XML-based manager and an XML-based agent. The gateways shown in (b) and (c) translate messages and operations between different management schemes, XML and SNMP. (d) is the most ideal framework to gain the maximum advantages of XML-based network management. That is, to utilize the benefit of XML-based management [3, 4], the agent as well as the manager must process XML.

The existing EWS can be extensible to an XML-Based Management (XBM) agent with additional XML processing module because XML can be transferred through HTTP. Today, the CPU power and memory size of network devices tends to be higher. However, XBM agent must be efficient and use as few resources as possible because the management module is regarded as an additional functionality to the fundamental functionalities of each device. In addition, it must guarantee stability and security in order not to interfere with the basic functions of each device.

A few device manufacturers, such as Cisco and Juniper Networks, have developed and equipped XBM agents on their devices, but unfortunately they merely concentrate on configuration management. The Cisco Networking Services (CNS) Configuration Agent [7] of Cisco and the JUNOScript [8] of Juniper Networks are examples of an XBM agent developed by a manufacturer. In this paper, we propose an XBM agent architecture by extending the existing EWS architecture. It is small and portable enough to be embedded in any network device, and perform fault management, performance management, and configuration management as well. We also explain the implementation detail and performance test results of our XBM agent.

Related Work (1)

- XML-based Network Management
  - Uses XML DTD or Schema for a management information model.
  - Exchanges management data in the form of XML documents.
  - Uses standard XML technologies for processing the data.
  - Web-based Integrated Management Architecture (WIMA) :J.P. Martin-Flatin, EPFL, 2000
  - XML-based Network Management (XNM) : H. T. Ju, POSTECH, 2001
  - IETF XML Configuration (XMLCONF) : apply XML technologies to configuration management of IP based network devices, June 2002
  - Integration SNMP with XML:
    - Translation Models: J.P. Martin-Flatin, EPFL, 2000
      - Model-level mapping & Metamodel-level mapping
    - SNMP MIB to XML Schema mapping: J. H. Yoon, POSTECH, 2001
      - Validated by developing an XML-based SNMP MIB browser.
    - Library to access SMI MIB (libsmi): Frank Strauss, 2000
    - XML/SNMP gateway: Y. J. Oh, POSTECH, 2002
      - Provide interaction translation methods

XML-based network management (XNM) utilizes a Document Type Definition (DTD) [5] or XML Schema [6] for management information modeling. It exchanges management data in the form of XML documents, and processes data by utilizing open standard XML technologies. XNM provides technical advantages. First, the XML Schema provides powerful modeling features for structured management information in network management. XML-based network management can transfer a large amount of data over HTTP and management data over HTTP can be compressed to reduce network traffic overhead.

J.P. Martin-Flatin proposed applying XML technologies to integrated management in his research on Web-based Integrated Network Management Architecture (WIMA) [9], which describes the advantages of XML/HTTP based communication and method of translation from SNMP MIB to XML. Our previous work [10] describes an XML-based network management architecture with EWS previously embedded in many existing devices. In this research, XNM extended the use of EWS for element management to network management. In May 2003, IETF has formed a working group on Network Configuration (netconf) [11] to apply XML technologies to configuration management of IP based network devices. They define the concepts and requirements for network configuration management and provide guidelines for the use of XML within IETF standards.

In addition, XML/SNMP gateway has been developed as an integration solution between XML-based management and the existing SNMP management. J.P. Martin-Flatin, proposed SNMP MIB to XML translation models, namely Model-level mapping and Metamodel-level mapping [9]. In Model-level mapping the DTD is specific to a particular SNMP MIB (set of MIB variables), and the XML elements and attributes in the DTD have the same names as the SNMP MIB variables. In Metamodel-level mapping the DTD is generic and identical for all SNMP MIBs. Strauss presented a library to access SMI MIB information, “libsmi” [12], which translates SNMP MIB to other languages, such as JAVA, CORBA, C, XML, etc. This library provides a tool for MIB dump (mibdump), which allows dumping the content of a MIB module into an XML document. Finally, our XML/SNMP gateway has achieved the specification translation [13] of SNMP SMI to XML Schema and interaction translation as well [14].

Related Work (2)

- Existing XBM Agents
  - Cisco's CNS Configuration Agent
    - Cisco Networking Services (CNS) Configuration Agents located on Cisco ISO network devices
    - Cooperates with Cisco Configuration Registrar, Web-based system for automatically distributing configuration files to Cisco IOS network devices
    - Uses its own XML parser to interpret the configuration data from the received configuration files
  - Juniper Network's JUNOScript
    - Allows client applications to access operational and configuration data using an XML-RPC
    - Defines the DTDs for the RPC messages between client applications and JUNOScript servers running on the devices
    - Delivers the request to the appropriate software modules within the device, encodes the response with JUNOScript tags, and returns the result to the client application

Currently, device vendors such as Cisco and Juniper Networks have developed their own XBM agent and equipped it on their devices. However, they merely concentrate on configuration management. In this section, we explain the Configuration Registrar of Cisco [7] and the JUNOScript of Juniper Networks [8]

The Cisco Configuration Registrar [7] is a Web-based system for automatically distributing configuration files to Cisco IOS network devices. It cooperates with Cisco Networking Services (CNS) Configuration Agents located on each device. When a Cisco device is connected to a network and runs, the Configuration Registrar delivers the initial configuration information to a CNS Configuration Agent. The Configuration Registrar uses HTTP to communicate with the agent, and transfers configuration data in XML. The Configuration Agent in the device uses its own XML parser to interpret the configuration data from the received configuration files. This CNS Configuration Agent is a restricted form of an XBM agent responsible for configuration management, which parses defined tags by Cisco.

Recently, Juniper Networks introduced JUNOScript [8] for their JUNOS network operating system. The JUNOScript is part of their XML-based network management effort. It adapted a simple model and is designed to minimize both development costs and the impact on the managed device. The JUNOScript allows client applications to access operational and configuration data using an XML-RPC. The JUNOScript defines the DTDs for the RPC messages between client applications and JUNOScript servers running on the devices. Client applications can request information by encoding the request with JUNOScript tags in the DTDs and sending it to the JUNOScript server. The JUNOScript server delivers the request to the appropriate software modules within the device, encodes the response with JUNOScript tags, and returns the result to the client application.

Requirements

- Basic functionality: exchanges management information in the form of an XML document through HTTP
  - EWS is a basic module for processing HTTP
  - An XML processing module: XML parser, XPath handler
  - Notification mechanism to send alarms to the manager
- Additional functionality
  - Scheduling methods for periodic monitoring data: agent sends periodic data to the manager by itself
- Non-functional requirements
  - Low resource requirements: must use as little RAM, ROM, and CPU as possible
  - High reliability: highly reliable like one of the embedded system components
  - High portability: portable on various RTOS and embedded systems
  - Security: limit access to sensitive information or configure & control

We present the requirements of the XBM agent that we must consider during development. The requirements are divided into two parts – functional and non-functional.

Network management using an Embedded Web Server (EWS) is usually applied to simple device management where the administrator connects to the EWS and manages the network device by exchanging management information through HTML/HTTP. HTML is merely a natural language with display logic for the Web interface. An operator can understand the meaning of management information by looking at the display of the Web browser. Nevertheless, it is almost impossible for computer software to understand management information by the interpretation of HTML. Therefore, configuration management information disseminated by EWS cannot be collected or processed by the manager application. Due to the absence of a centralized management capability, management functions through the EWS have a difficulty in processing high level network management such as data preservation, aggregation, and report generation. However, because XML allows us to define the user-customized tag, it solves the absence of management information modeling by replacing HTML with XML [10].

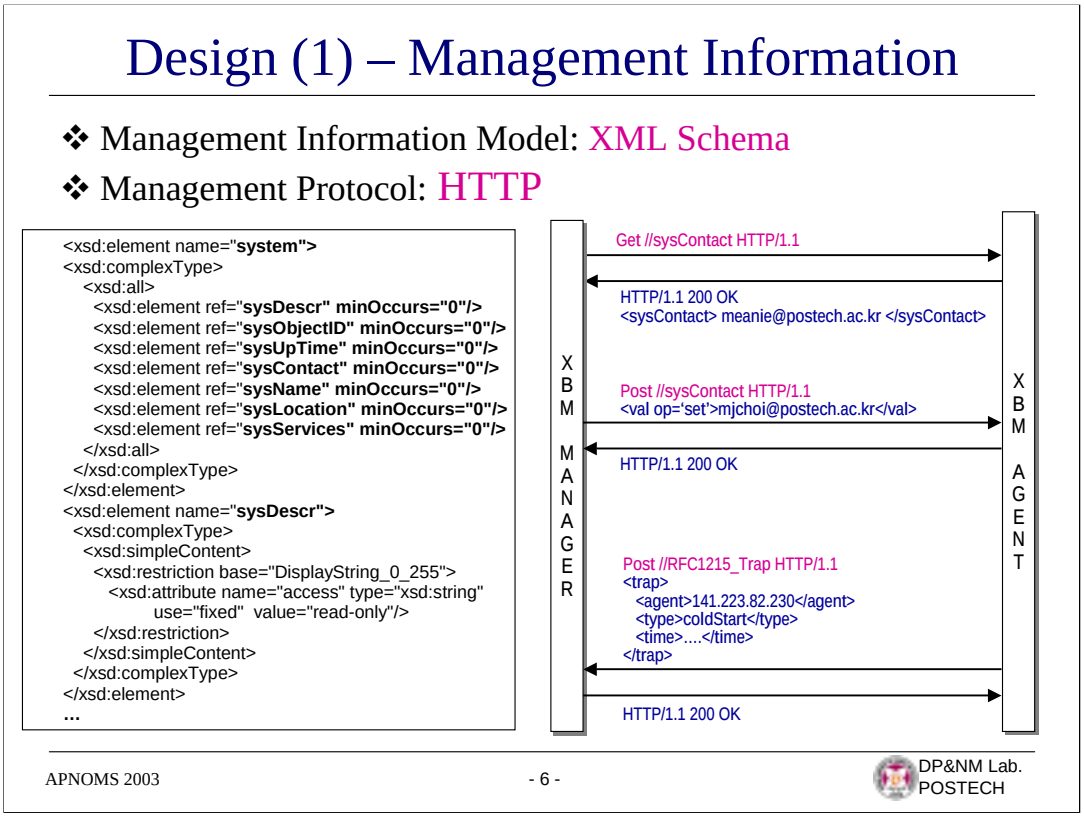
An XBM agent is embedded in network devices and exchanges management information defined in the XML document format with the manager through HTTP. Because the communication protocol is HTTP, it needs an EWS as a basic module. The additional module to the existing EWS is an XML processing module which processes the XML document. The XML processor module must parse the XML document including management information and access the specific part of the XML document by recognizing the XPath expression.

Basically, the request and response of the EWS is client-driven, where the client (or manager) sends a request to the server and the server responds. However, sometimes a network device needs to send a message to the manager without the request driven by the manager. For example, the agent in a network device must occasionally send a notification message to the manager, due to the error occurrence of network device status. That is, the XBM agent needs a notification mechanism to send an asynchronous message to the manager.

In addition, periodic monitoring in network management usually uses a polling technology in the case of SNMP. The manager requests management information periodically to the agent, and the agent responds. This polling mechanism is subject to heavy processing overhead when producing many requests for periodic data from agents if one manager oversees many agents. Moreover, a shorter monitoring period generates not only processing overhead but also heavy network traffic. To alleviate this problem, the agent might also send periodic data to the manager by itself without requests from the manager. For this architecture, the XBM agent needs a type of scheduler to decide the time to send the data. Therefore, the XBM agent including the scheduler reduces the processing overhead of the manager, and effectively decreases network traffic for data gathering.

Every network device has its own fundamental functionality to perform. Even though the performance of the network device hardware is improving, the CPU and memory resources are still scare. Therefore, an XBM agent must be as small and efficient as possible not to interfere with the main functionality of network devices. In addition, network devices require high reliability. As an embedded component of a network device, the XBM agent must also be highly reliable. Because it is a subordinate process, it must protect against the propagation of internal failure to the entire system at the very least.

Also, the network device varies and has several types of operation systems and processors. Therefore, the XBM agent must be portable so that it can run on a much broader range of embedded system environments. Finally, it must provide security in the network communication protocol and management access mechanism as well. Security is an important concern in network management, specifically that which involves equipment configuration or administration. The XBM agent must protect network device from being cracked by intentional network attacks. In short, the XBM agent must be small, efficient, portable, and secure.



We present the architecture of XBM agent for network management based on proposed requirements. First, we explain management information model and communication protocol to manage network devices. Next, we describe the XBM agent architecture.

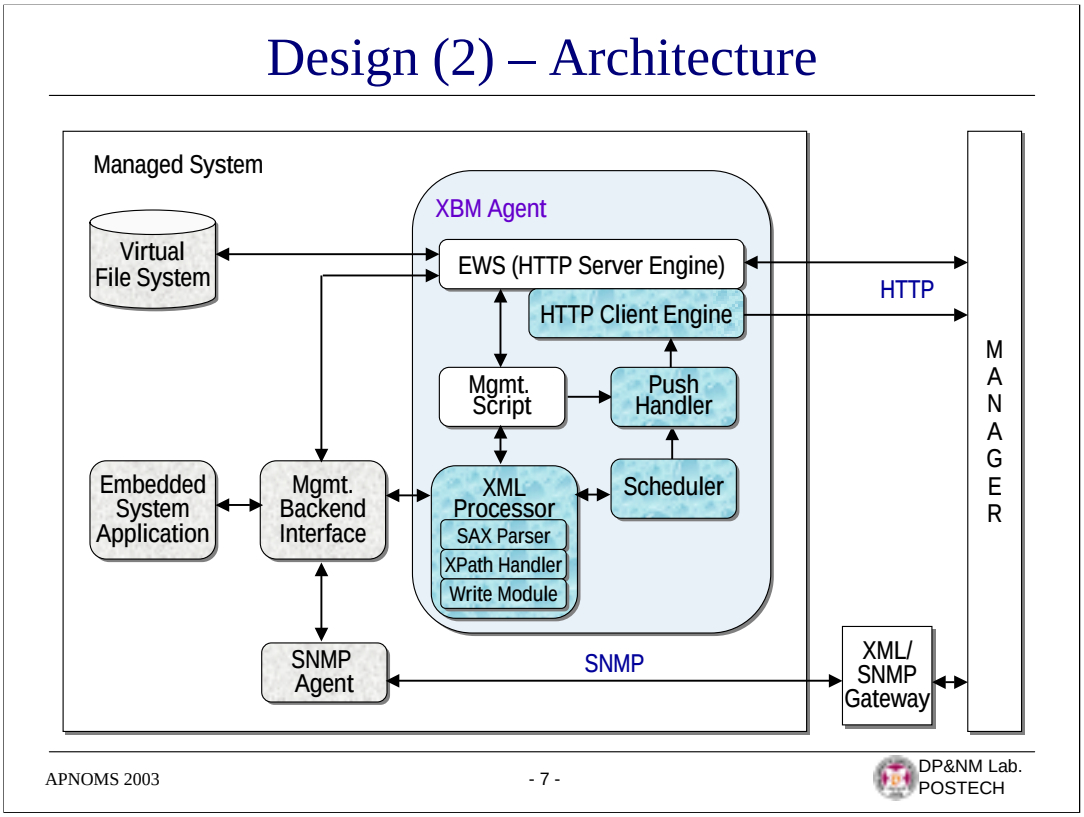
XML provides two fundamental approaches to define the XML documents structure: DTD and XML Schema. DTDs are used to specify a property for each element and a relationship between the elements. However, DTDs do not support rich information modeling, so a new modeling mechanism, the XML Schema, is proposed. XML Schemas represent various management information by adding new data types to a variety of data types (44 kinds of basic types), and defining the document structure of management information. Therefore, the XML Schema is more suitable to define management information. Basically, management information is SNMP MIB II that can be applied to all network devices. Each node of SNMP MIB converts into an element of the XML Schema: the name of object into XML tag, 'syntax' into the data type definition, and 'access' into the attribute for example. The left figure shows an example of an XML Schema in the system group of MIB II. This illustrates the XML Schema of the 'sysDescr' object of system group.

For information transmission, XML documents can be easily transferred by using an HTTP which is already equipped worldwide. The management information of XML document transformed through HTTP over TCP. TCP for transport protocol instead of UDP makes it possible to transmit reliable management data and large application-messages without limitations in message size.

The operations of HTTP are Get and Post. The HTTP Get operation gathers management information from the agent, and HTTP Post updates the management information. The response of HTTP Get is XML documents with management information, and the response of HTTP Post includes the 'HTTP/1.1 200 OK' message. XPath is used for addressing managed objects. A manager can query effectively about managed objects of agent through XPath. XPath expressions are formed using the element name, attributes and built-in functions. In the Get operation, one or more values can be retrieved depending on the parameters of XPath. Also, we can retrieve specific information with conditioning and filtering.

The right figure illustrates a data exchange example of Get, Set (Post), Trap between an XBM manager and an XBM agent. This shows a communication example of sysContact of system group in MIB II. Trap information is an example of coldStart of SNMPv1 trap.

The XBM agent sends a notification to the XBM manager by a push mechanism using a Post operation [9]. For supporting this mechanism, the XBM agent must include an HTTP client. Typically, some data needs to be cyclically monitored within a certain time period. The XBM manager sends subscription information to the XBM agent. After receiving the subscription information, the XBM agent schedules a series of message distributions and sends it to the subscriber at the scheduled time through a Push mechanism using a Post operation. The management data transferred periodically from the XBM agent to the XBM manager can reduce network overhead because it is not based on polling [9]. Also, XML-based network management performs the basic management functionality through HTTP operations: Get and Post, so it is not necessary to develop a new management protocol.



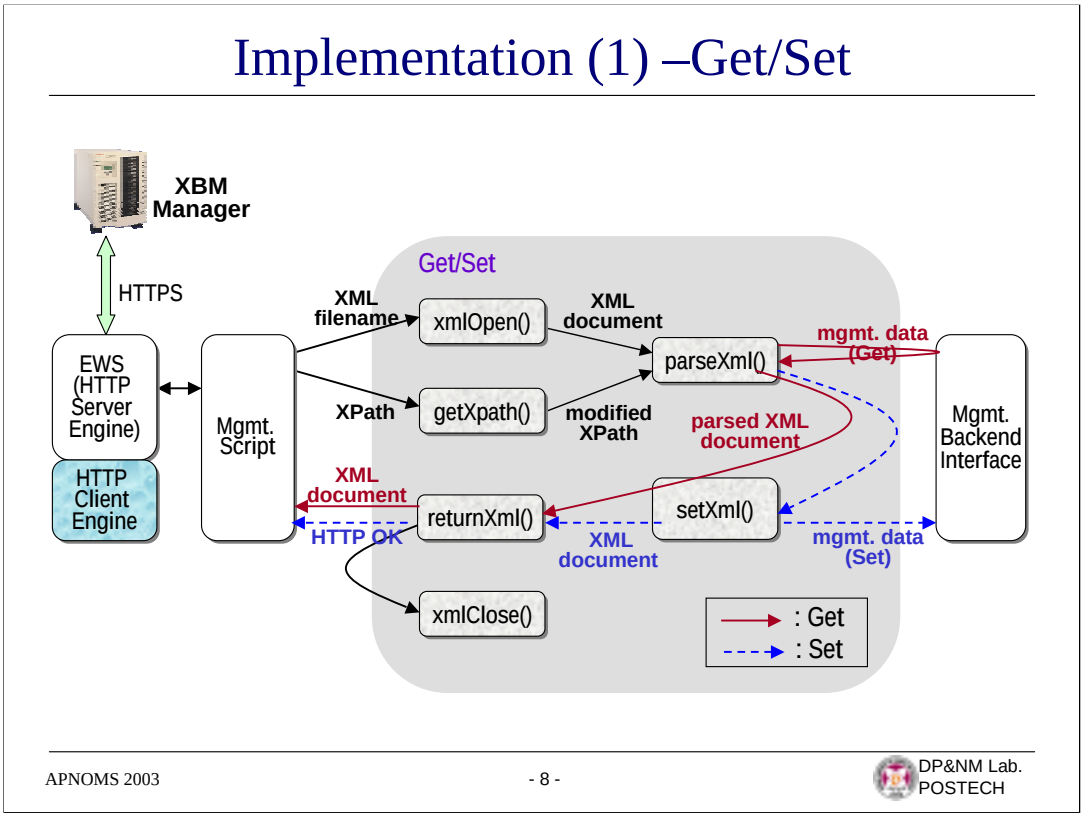
The XBM agent needs an XML processor as a basic module in addition to EWS. This figure illustrates the architecture of an XBM agent. The XBM agent includes an *Embedded Web Server (EWS)* [2] as a basic component. The components added to the *EWS* are the *XML Processor* having a *SAX Parser*, which is an XML Parser, the *Push Handler*, and the *HTTP Client Engine*. SAX parser does not support a write function; therefore, a *Write Module* is also necessary.

In previous work [4], we used DOM and XPath for handling the XML document in the agent as well as in the manager. DOM and XPath are effective to process XML documents for accessing and filtering. The device we used to embed the XBM agent was a Linux server, which posed no problems with the resource. However, supporting DOM and XPath requires extra memory resources in the device. To access a part of an XML document, the DOM tree of the whole XML document is loaded into the memory. This wastes CPU and memory resources in the device. Moreover, the code size of basic libraries widely used for supporting DOM and XPath is big. Therefore, it is not suitable to apply DOM in embedded systems with few resources.

The code size and executable memory size of SAX is smaller than DOM [15]. As SAX is an event-driven mechanism for accessing and processing XML documents, there is no need to load the entire XML tree to the memory. Therefore, SAX is much lighter than DOM from the perspective of functionalities and resources. Though SAX can be lighter in resource usage because SAX reads the XML document in sequential order and generates an event for a specific element, the processing time for accessing XML document of SAX is slower than that of DOM after the DOM tree is generated. However, the management information of most network devices is not immense and can be defined into several small XML documents containing mutually related management data. Therefore, the processing time of the SAX parser matters little. The access method of SAX is serial and read-only, so we add a *Write Module* as part of XML processor to provide a writing mechanism. Because we set a greater value on low resource requirements, we selected SAX instead of DOM. The *SAX Parser* parses the XML document, and selects the specified node when parsing, and reads management data. In order to send up-to-date information, the agent gathers information from the *Management Backend Interface*. The *Write Module* updates the node value for the selected node through the *Management Backend Interface* before replying to the manager.

To send a notification to the XBM manager, the XBM agent needs a *Push Handler* and *HTTP Client Engine* as well as an *XML Processor* module. Also, to send periodic management information to XBM manager at a fixed time with one schedule request, the XBM agent needs a *Scheduler*. The *HTTP Client Engine* delivers asynchronous messages to the XBM manager for reporting alarm and distributing management data according to the schedule. The *Scheduler* manages subscription information and the schedule for the distribution of the management information. Subscription information includes the subscriber's URL for receipt (subscriber information), managed object's XPath expression (management item), and schedule information containing the start time, the end time, and interval. The *Push Handler* receives the request from the *Scheduler* and sends the scheduled data to the manager through the *HTTP Client* at the scheduled time. Also, the *Push Handler* sends a notification generated in the agent, which is sent to the manager through the *HTTP Client*.

If an *SNMP Agent* is also available in the managed network device, the same *Management Backend Interface* can be used. This reduces memory usage. The XBM manager communicates with the *SNMP Agent* through the XML/SNMP gateway [13, 14].



We have implemented an XBM agent to manage an IP sharing device based on the XBM agent design presented in slide 9. Moreover, we focused on the implementation of an efficient and lightweight XBM agent by considering such requirements as low resource utility (CPU usage and memory size).

In addition, for portability to equip any type of embedded system, we used the C programming language throughout agent implementation and developed components per each module. From the security aspect, to control the access of management information, the access to the XBM agent is permitted through the authentication with ID and password in the initial contact. Also, we used the HTTPS protocol for secure communication between the XBM manager and agent.

The IP sharing device equipped with our XBM agent runs on an embedded Linux based on linux2.2.13-7 kernel using Motorola's MPC850DE processor with 16MB ROM. We used a powerpc-linux-gcc compiler.

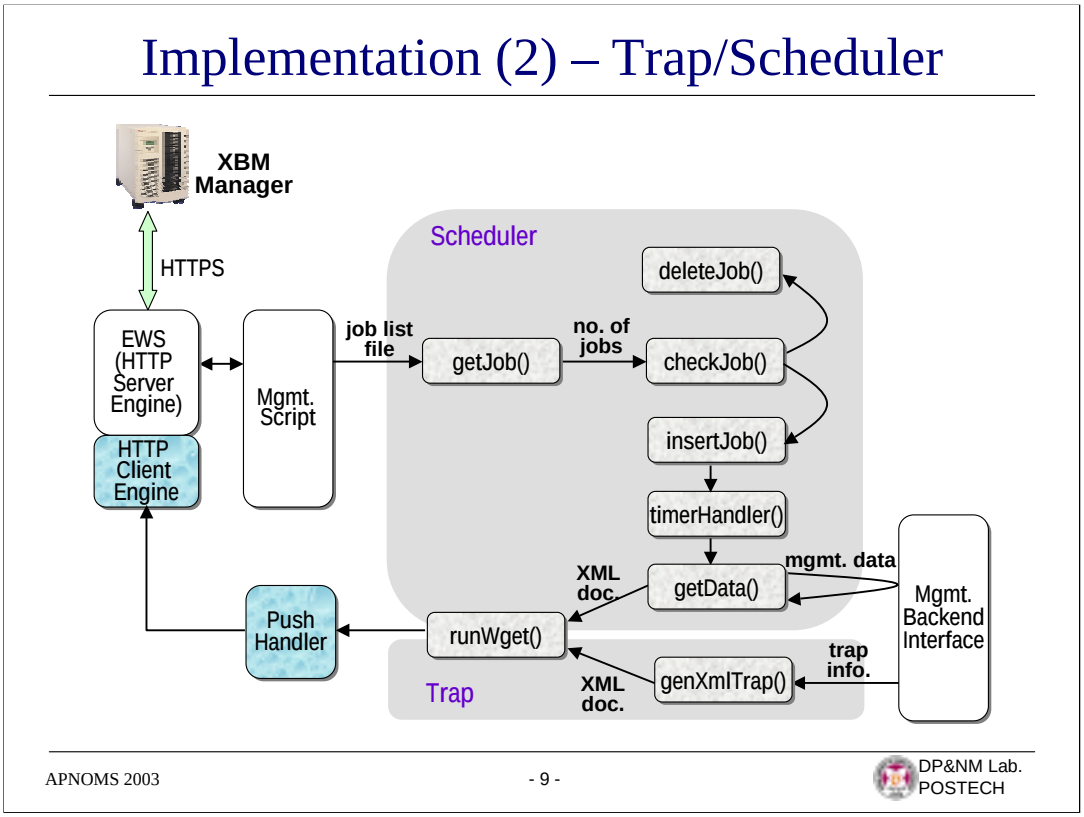
The main function of the XBM agent is to retrieve and update management information according to the manager's request of Get/Set operation. Also, the XBM agent delivers periodic monitoring data to the manager by performing the scheduler according to the manager's request of scheduling. Figure 5 illustrates the function call for the flow of Get/Set process of the XBM agent.

The process of the Get operation in XBM agent is as follows. The Get request from the manager calls the Get module through Mgmt. Script with two parameters: XML filename, and XPath expression. The value of XML filename is used as the parameter of the *xmlOpenI()* function for opening the appropriate XML document, the value of XPath is the parameter of the *getXpath()* function which returns the structure type of XPath to utilize in next phase. The XML document from *xmlOpen()* and XPath returned from *getXpath()* input into the *parseXml()* function as parameters. The *parseXml()* parses the XML document through the XPath grammar. The real management value is retrieved through the call of *Mgmt. Backend Interface*, then the XML document is updated with this value using the *setXml()* function. The parsed XML document is sent to *returmXml()* and backtracks to *Mgmt. Script* as string format. Finally, the *xmlClose()* function closes the previously opened XML document.

The process of Set operation is almost equal to the Get operation. The Set module opens the XML document and retrieves the specific XML document part by applying XPath expression as a Get operation, then modifies the XML document calling *setXml()*, and updates the real value through *Mgmt. Backend Interface*. Afterwards, it generates the XML document from the result of Set operation and calls *returnXml()* function. The returned result to *Mgmt. Script* is the response of Set operation, 'HTTP OK' message.

In this figure, the *getXpath()* function is the *XPath Handler* module processing XPath. Currently, XPath supports various syntaxes. However, if the XPath expression is complex, the processing time is slow [16], and the *XPath Handler* supporting full XPath grammar [17] is heavy. This does not meet the requirements of low resource utility. Therefore, we implemented a part of XPath grammar sufficient to accessing the XML document of management information in the XBM agent. Moreover, we implemented the *XPath Handler* to access the specific management information applying XPath expression during the parsing without loading the XML document into memory. Table 1 shows the XPath grammars which we implemented in our XBM agent. We implemented the XPath Handler considering the extensibility supporting more XPath syntaxes. It is desirable to extract management information using simple XPath expression considering the processing time.

In this figure, the *parseXml()* function, which reads XML document in sequential access and applies XPath expression, is the core module of *SAX Parser*. This parses the XML document from the root element to its child element, retrieves the element name and attributes, and compares the retrieved values to XPath expression processed by the *getXpath()* function, then if the comparison result is equal, the values are sent to the next function. The *setXml()* function is for the Write Module. In the case of a Set request, if the Set operation needs to update the value, this function modifies the XML document after parsing the document that applies the XPath expression.



This figure describes the process of the *Scheduler* for delivering periodic data to the manager at the scheduled time and Trap for sending a notification to the manager. If the *Scheduler* receives a scheduling request from the manager, it updates the job list file. The *Scheduler* processes the job with pthreads. In the initial start, the *Scheduler* runs the *getJob()* function, and this function reads the job list file and retrieves the job contents.

As mentioned in the scheduler part in the slide 9, the job list file includes subscriber information, the management item, and schedule information containing start time, end time, and interval. The *checkJob()* function receives the number of jobs and the job list from *getJob()*, and compares the existing pthreads information to new job list. According to the comparison results, the *checkJob()* generates the new job thread calling *insertJob()*, or destroys the existing job thread calling *deleteJob()*. The generated thread having the same time interval as the parameter checks the schedule calling *timerHandler()* and runs the *runWget()* function at the scheduled time. The *runWget()* function calls the *Push Handler*, then the processed management information is sent to the manager through an *HTTP Client Engine* called Wget. A notification from the *Mgmt. Backend Interface* is sent to *genXmlTrap()* function in the Trap module. This function generates an XML document containing the trap information and calls the *runWget()* function. Trap information is delivered to the manager by this mechanism.

Performance Test (1)

- Verify the performance of XBM agent by comparing it with the SNMP agent on the same IP sharing device
- SNMP agent extends the Net-SNMP and supports only SNMPv1
- CPU load, run-time memory usage, and executable code size

Agent	CPU load	Run-time memory usage	Executable code size
SNMP (Net-SNMP)	**17 %**	**600 KB**	**400 KB**
XBM	**20 %**	**700 KB**	**550 KB**

< Resource Utility of SNMP and XBM Agent >

We verify the performance of our XBM agent by comparing it with the SNMP agent on the same IP sharing device. We compare resource utilities, such as CPU load, run-time memory size, and executable code size. We also compare the generated network traffic of each agent between the manager and the agent. Moreover, we measure how much network traffic is reduced by a scheduling and push mechanism for periodic monitoring. Finally, the processing time of our XBM agent against the traditional SNMP agent is measured with the response time upon Get/Set requests between the manager and the agent.

This table compares the SNMP agent and the XBM agent in terms of CPU load, run-time memory usage, and executable code size. This information is discovered by the Linux command such as top, cpuload, etc., and the status data in proc directory generated during the run-time of the process daemon. Both the XBM agent and the SNMP agent provide SNMP MIB II information. The SNMP agent extends the Net-SNMP and supports only SNMPv1. Currently, most network management systems (NMS) use the GetNext operation of SNMPv1. The users of NMS want to know the comparison of SNMPv1 agent. Therefore, we test the performance of the SNMPv1 agent.

From the above table, we can determine that our XBM agent does not take more resources than the SNMP agent. While the XML parser is generally considered to consume much resources and is insufficient in application to the embedded systems, our XBM agent is lightweight enough to equip in network devices and perform network management functionality.

Performance Test (2)

❖ Network traffic (MIB II – system, interfaces group)

Management property	Get request message (bytes)		Get response message (bytes)	
	SNMPv1	XBM	SNMPv1	XBM
sysDescr	82	238	145	240
sysContact	82	240	103	190
system Group	572	241	722	624
inOctets (2 interfaces)	169	240	175	252
outOctets (2 interfaces)	169	241	176	256
interfaces Group	3720	241	3818	1654

< Message Size of Get >

This table shows the network traffic of each agent between SNMP agent and manager, and between XBM agent and manager in terms of system group and interface group of MIB II information. We captured packets and those sizes between each manager and agent by network traffic monitoring tool, Ethereal.

In the case of requests for one object in this table, we can easily determine the smaller size of the Get request message and response message of the SNMP agent because it is aimed to serve the network management protocol. However, the XBM agent uses the HTTP protocol, so increases network traffic for each information access over SNMP. Conversely, for a grouped request, the XBM agent produces less network traffic than the SNMP agent because the SNMP agent requests several GetNext operations and receives responses for every node, while the XBM agent retrieves the whole system group or interfaces group information in one request using an HTTP message. Moreover, the difference is outstanding in the case of the interfaces group because it includes more managed objects. Most cases of Get or Set operations for management information request multiple data at once. Therefore, the XBM agent produces less network traffic than the SNMP agent in this case.

The response time of the Get/Set request is measured by the time() function call from the request initiation to response reception. From the response time, the SNMP agent takes an average 40 ms and XBM agent average 60ms for accessing each leaf node in system group in MIB II. The SNMP agent takes 160 ms, while XBM agent 180ms for retrieving the system group. In the case of in/out octet value in the interfaces group, the SNMP agent takes about 80ms and the XBM agent 110 ms. For the interfaces group, the SNMP agent takes 700 ms, while the XBM agent 760 ms. The response time of the Set request is almost the same as that of the Get request in both the SNMP agent and XBM agent. These data do not show a significant difference in the response time between them. Therefore, the XBM agent achieves a good performance enough for processing XML documents. As a result, the XBM agent is small and efficient as the SNMP agent in terms of resource utilization and processing time. In addition, the XBM agent gives outstanding network traffic reduction to access much information in a limited time.

Regarding in/out octets to monitor periodic network traffic for devices with two interfaces, the SNMP creates a request message of approximately 169 bytes to each in/out octet information and a response message of approximately 175 bytes. However, the XBM agent receives request message of 600 bytes to set schedule information in the initial time and sends only message of 252 bytes periodically to the manager. If the period is shorter and the manager has many devices to monitor, the scheduler mechanism in the XBM agent achieves a more effective network traffic reduction. The manager can reduce processing overhead to generate Get request messages for periodic monitoring.

Concluding Remarks

- Presented the design and implementation of an XBM agent
- Verified the functionality of our XBM agent by applying it to the IP sharing device
- Showed low resource usage and less network overhead from the performance test results compared with the SNMP agent

- Future work
  - Augment additional parser functions according to management information complexity
  - Enhance SAX Parser and XPath Handler by surveying the necessary XPath expressions
  - Evaluate the performance of each module in the XBM agent, and improve the response time
  - Identify the scalability of the XBM manager when it communicates with many XBM agents

In this paper, we presented the design and implementation of an XBM agent which uses XML/HTTP to communicate with the manager to take maximum advantage of XML-based network management. We have focused on the stability and the efficiency of resource utilization, so that it can be embedded and run in a network device.

We have verified the functionality of our XBM agent by applying it to the IP gateway (a commercial Internet sharing device), and showed low resource usage and less network overhead from the performance test results. This result showed almost the same performance and resource usage compared with the existing management paradigm, SNMP agent. In other words, we successfully verified that our XBM agent is designed and implemented as efficient and lightweight enough to decrease the overhead of XML processing and be embedded in any network device. Moreover, the XBM agent is more efficient than the SNMP agent in terms of network traffic.

Our future work in XBM agent development is to augment additional parser functions according to management information complexity, enhance SAX Parser and XPath Handler by surveying the necessary XPath expressions. We will evaluate the performance of each module in the XBM agent, and improve the response time. We intend to identify the scalability of the XBM manager when it communicates with many XBM agents simultaneously and compare it with that of SNMP-based management.

References

[1] J. Case, M. Fedor, M. Schoffstall, and J. Davin (Eds.), “A Simple Network Management Protocol (SNMP)”, RFC 1157, IETF, May 1990.

[2] M. J. Choi, H. T. Ju, H. J. Cha, S. H. Kim, and J. W. Hong, “An Efficient and Lightweight Embedded Web Server for Web-based Network Element Management”, Proc. IEEE/IFIP Network Operations and Management Symposium (NOMS 2000), Hawaii, USA, April 2000, pp. 187~200.

[3] F. Straus, and T. Klie, “Towards XML Oriented Internet Management”, Proc. IFIP/IEEE International Symposium on Integrated Network Management (IM 2003), Colorado Springs, USA, March 2003, pp.505~518.

[4] H. T. Ju, “Embedded Web Server Architecture for Web-based Element and Network Management”, Ph.D. Thesis, POSTECH, February 2002.

[5] Tim Bray, Jean Paoli and C. M. Sperberg-McQueen, “Extensible Markup Language (XML) 1.0”, W3 Recommendation REC-xml-19980210, February 1998.

[6] W3C, “XML Schema Part 0,1,2”, W3 Consortium Recommendation, May 2001.

[7] Cisco Systems, Cisco Configuration Registrar, http://www.cisco.com/univercd/cc/td/doc/product/rtrmgmt/ie2100/cnfg_reg/index.htm.

[8] P. Shafer and R. Enns, “JUNOScript: An XML-based Network Management API”, http://www.ietf.org/internet-drafts/draft-shafer-js-xml-api-00.txt, August 27, 2002.

[9] J.P. Martin-Flatin. “Web-Based Management of IP Networks and Systems”, Ph.D. Thesis, Swiss Federal Institute of Technology (EPFL), October 2000.

[10] H. T. Ju, M. J. Choi, S. H. Han, Y. J. Oh, J. H. Yoon, H. J. Lee, and J. W. Hong, “An Embedded Web Server Architecture for XML-based Network Management”, Proc. IEEE/IFIP Network Operations and Management Symposium (NOMS 2002), Florence, Italy, April 2002, pp.1~14.

[11] IETF, “Network Configuration (netconf)”, http://www.ietf.org/html.charters/netconf-charter.html

[12] Frank Strauss, “A Library to Access SMI MIB Information”, http://www.ibr.cs.tu-bs.de/projects /libsmi/.

[13] J. H. Yoon, H. T. Ju, and J. W. Hong, “Development of SNMP-XML Gateway for XML-based Integrated Network Management”, Accepted to appear in the International Journal of Network Management (IJNM), 2003.

[14] Y. J. Oh, H. T. Ju, M. J. Choi, J. W. Hong, “Interaction Translation Methods for XML/SNMP Gateway”, Proc. DSOM 2002, Montreal Canada, October 2002, pp. 54~65.

[15] Devsphere, “XML Parsing Benchmark”, http://www.devsphere.com/xml/benchmark/index.html.

[16] Georg Gottlob, Christoph Koch, and Reinhard Pichler. “XPath Query Evaluation: Improving Time and Space Efficiency”, Proc. 19th International Conference on Data Engineering (ICDE 2003), Bangalore, India, March 2003.

[17] ZVON Org, “XPath Tutorial”, http://www.zvon.org/xxl/XPathTutorial/General/examples.html.