Table of Contents

List of Figures

List of Tables

1. Introduction

SNMP [5] is the most widely used method for network management on the Internet. However, SNMP is insufficient to manage huge and continuously expanding networks because of constraints in scalability and efficiency. An XML-based network management [2, 3, 4] has been proposed to complement SNMP-based network management. XML-based network management has many advantages, as follows:

- The XML Schema [7] provides powerful and extensible modeling features for structured management information.
- The XML-based network management can transfer management data in an efficient and reliable manner using widely deployed protocols, such as HTTP [22].
- The management processes for analyzing, storing, and presenting management information can be easily implemented using XML-related technologies.
- SOAP [16] with WSDL [17] and UDDI [18] can be used to implement Web Services for powerful high-level management operations.

As depicted in Figure 1, four possible combinations between managers and agents can be considered for XML-based integrated network management [4]. Figure 1 (a) shows the current, widely deployed SNMP-based network management, and Figure 1 (d) shows an XML-based management using XML-based manager and XML-based agent. The gateways shown in Figure 1 (b) and Figure 1 (c) translate messages and operations between different management schemes, XML and SNMP. Figure 1 (d) is the most ideal framework to gain the maximum advantages of XML-based network management. However, most network devices currently deployed are basically equipped only with SNMP agents, and this architecture is hardly applicable to the current situation. Therefore, the XML/SNMP gateway is needed for the XML-based manager to manage SNMP-enabled network devices as well as devices equipped with XML-based agents. Figure 1 (c) shows the most practical management framework using the XML/SNMP gateway, which translates and relays messages between an XML-based manager and an SNMP agent.

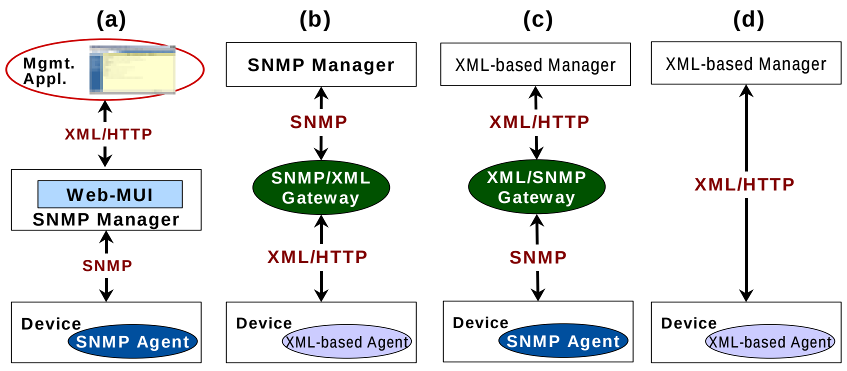


Figure 1. Interaction Combinations of Managers and Agents

The XML/SNMP gateway must provide both specification translation and interaction translation between the two different management domains. Recent research on SMI to XML Schema translation [1, 3, 26] provides the foundation for the XML/SNMP gateway. However, no concrete or standardized interaction translation method for the gateway exists yet. Most XML-based applications implement various communication methods between peer systems because XML does not specify any standard communication protocol. Therefore, research on concrete and verified methods to efficiently exchange messages between XML and SNMP is necessary.

In this thesis, we propose three methods for interaction translation in the XML/SNMP gateway for integrated network management. These translation methods enable different implementations of the XML-based manager to communicate with SNMP agents in a standardized manner. First, we propose an XML parser-based translation. The gateway uses the XML document structure and the XML parser interfaces to translate management information between XML and SNMP. We analyze the functions and meanings of the XML parser interfaces on management information, and translate the interfaces into appropriate SNMP operations. We also propose an HTTP-based translation using XPath [13], XQuery [14], and XUpdate [15], which enables us to easily define a detailed request message over HTTP. This method improves efficiency in XML/HTTP communication, which is the most common in the exchange of XML documents. Finally, we apply Simple Object Access Protocol (SOAP) [16], which is accepted as a standard protocol for XML, and propose a SOAP-based translation between the XML-based manager and the XML/SNMP gateway. The gateway advertises its translation services to the manager using SOAP RPC. These three methods are considered to cover overall operation schemes for processing and exchanging XML-encoded information between managers and agents. We have implemented an XML/SNMP gateway for our XML-based Global Element Management System (XGEMS) [4] which manages one or more types of network elements scattered throughout the world. In this thesis, we also present the implementation details and the results of performance analysis.

The remainder of this thesis is organized as follows. Section 2 presents an overview of XML technologies, and introduces related work on XML-based network management and XML-SNMP integration. Section 3 describes the architecture of the gateway and the details of proposed interaction translation methods for the XML/SNMP gateway. Implementation details are described in Section 4, and performance analysis results are presented in Section 5. Finally, we summarize our work and discuss directions for future research in Section 6.

2. Related Work

In this section, we first explain XML related technologies that are applicable to network management. We also introduce related work on XML-based network management performed by others. Finally, we describe recent research on XML-SNMP integration.

2.1 XML Related Technologies

- **DTD and XML Schema**: XML has two fundamental approaches to define the XML document structure: Document Type Definition (DTD) and XML Schema [7]. The DTD is used to specify a content model for each element. The content description is part of the element declaration in the DTD, and specifies the order and quantity of elements that can be contained within the element being declared. That is, the DTD is used to specify a property for each element in addition to the relationship between the elements. However, because the DTD does not support a complex information model, another modeling mechanism, the XML Schema, was proposed. The XML Schema substantially revised and extended the capabilities found in XML DTDs. The XML Schema is based on XML, so it can be parsed and manipulated in exactly the same manner as the XML documents through the standard API. The XML Schema supports a variety of data types (44 kinds of basic types), while the DTD treats all data as strings or enumerated strings. The XML Schema also allows inheritance relationships between elements and supports namespace integration.

- **XSL and XSLT**: Extensible Stylesheet Language (XSL) [8] is a mark-up language designed for illustrating the method to display XML documents on the Web. XML documents describe only the contents and the structure of the contents. An XSL stylesheet specifies the presentation of a class of XML documents by describing how an instance of the class is transformed into an XML document that uses a formatting vocabulary. That is, XSL enables XML to separate contents from presentation. XSL consists of two parts: a language for transforming XML documents, and an XML vocabulary for specifying formatting semantics. The style sheet technology to transform documents is XSL Transformation (XSLT) [9], which is a subset of XSL technology that fully supports the transformation of an XML document from one format into another, such as HTML or another custom XML document type. The reason for publishing the XSLT specification separately from XSL is that XML documents can be displayed, providing an easy display format for end users by transforming the XML documents without formatting semantics.

- **DOM and SAX**: The Document Object Model (DOM) [10] is a platform- and language-independent interface that allows programs and scripts to dynamically access and update the content, structure, and style of documents. The DOM is an API for valid HTML and well-formed XML documents. The Simple API for XML (SAX) [12] is an event-driven and serial-access mechanism for accessing XML documents. While a DOM parser parses the XML document and creates a DOM tree, keeping the entire structure in memory at the same time, SAX reads the XML document in sequential order and generates an event for a specific element. Therefore, if the application calls for sequential access to XML documents, SAX can be much faster than other methods without requiring much system overhead. However, it does not provide the hierarchical information that a DOM parser provides. A SAX parser generates events such as the start of an element and the end of an element, while accessing the XML document. By capturing the event, applications can process operations, such as gaining the name and attribute of the element.

- **XPath**: XPath [13] is a language used to identify particular sections of an XML document. XPath has a compact, non-XML syntax to be used within URIs

[21] and XML attribute values, and operates on the abstract, logical structure of an XML document. Each node in an XML document is indicated by its position, type, and content using XPath.

- **XQuery and XUpdate**: XQuery [14], a query language for XML, is designed to be broadly applicable across all types of XML data sources, such as structured and semi-structured documents, relational databases, and object repositories. XQuery uses XPath for path expression. Further, XQuery provides such features as filtering documents, joining across multiple data sources, and grouping the contents. XUpdate [15] is an update language, which provides open and flexible update facilities to insert, update, and delete data in XML documents. The XUpdate language is expressed as a well-formed XML document, and uses XPath for selecting elements and conditional processing.

- **SOAP**: Simple Object Access Protocol (SOAP) [16] is a lightweight protocol for exchanging information in a distributed environment. It is an XML-based protocol that consists of three parts: an envelope that defines a framework for describing the contents of a message and how to process it, a set of encoding rules for expressing instances of application-defined data types, and a convention for representing remote procedure calls and responses. SOAP defines the use of XML and HTTP or SMTP to access services, objects, and servers in a platform- and language-independent manner.

- **WSDL**: Web Services Description Language (WSDL) [17] is an XML-based language used to define Web Services and describe how to access them. A WSDL document is a collection of one or more service definitions. The document contains a root XML element named *definitions*, which contains the service definitions. The *definitions* element can also contain an optional *targetNamespace* attribute, which specifies the URI associated with the service definitions. WSDL defines services as collections of network endpoints. These endpoints are defined as ports in WSDL. WSDL separates the service ports and their associated messages from the network protocol binding. The combination of a binding and a network address results in a *port*, and a *service* is defined as a collection of those ports.

2.2 XML-based Network Management

In this section, we introduce related work on XML-based network management. First, we describe research work on management architectures. We also present standardization efforts within the DMTF and the IETF, and introduce recent industry efforts in XML-based network management.

2.2.1 Management Architecture

- **Web-based Integration Management Architecture**: J.P. Martin-Flatin formulated the concept of using XML for integrated management in his research on Web-based Integrated network Management Architecture (WIMA) [3]. WIMA provides a way to exchange management information between a manager and an agent through HTTP. HTTP messages are structured with a MIME multipart. Each MIME part can be an XML document, a binary file, BER-encoded SNMP data, etc. By separating the communication and information models, WIMA allows management applications to transfer SNMP, CIM, or other management data. A WIMA-based research prototype, JAva MAnagement Platform (JAMAP) [3], implemented push-based network management using Java technology.
- **XML-based Network Management Architecture**: We proposed an XML-based network management (XNM) using EWS (Embedded Web Server) in our previous research [2]. In this work, we extended the use of EWS for element management to network management. XNM uses XML to transfer management information over HTTP between an agent and a manager. XNM also uses DOM

for the representation of and access to management data.

2.2.2 Standard Activities

- **WBEM**: Web-Based Enterprise Management (WBEM) [38] is an initiative of the Distributed Management Task Force (DMTF), and includes a set of technologies that enables interoperable management of an enterprise. WBEM consists of a Common Information Model (CIM) [39], a DTD for the representation of the CIM in XML [40] and a specification for CIM operations over HTTP [41]. CIM provides a comprehensive object-oriented information model and CIM schemas are implemented not only for managing servers but also for network resources, such as switches and routers. WBEM is currently being updated to include emerging standards, such as SOAP. DMTF is collaborating with OASIS [42] to sponsor a new management protocol technical committee and to develop open industry standard management protocols.
- **IETF XML Configuration BOF**: In the 54th IETF meeting in July 2002, a BOF session concerned with XML configuration (XMLCONF) was held. This BOF discussed the requirements for a network configuration management, and how the existing XML technologies, namely SOAP, WBEM, SyncML [43] and JUNOScript [44] could be used to meet those requirements. There are some Internet-Drafts [45, 46, 47] that present basic concepts and the requirements for XML network configuration and provide guidelines for the use of XML within IETF standards protocols.

2.2.3 Industry Effort

- **Juniper Networks' JUNOScript**: Recently, Juniper Networks introduced JUNOScript for their JUNOS network operating system. The JUNOScript is part of their XML-based network management effort and uses a simple model, designed to minimize both the implementation 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.

- **Cisco's Configuration Registrar**: The Cisco Configuration Registrar [48] is a Web-based system for automatically distributing configuration files to Cisco IOS network devices. The Configuration Registrar works in conjunction with the Cisco Networking Services (CNS) Configuration Agents located at each device. The Configuration Registrar delivers the initial configuration to Cisco devices when starting up on the network for the first time. 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.

2.3 XML-SNMP Integration

In this section, we describe our previous work on XML-SNMP integration and recent research performed by others in this area.

2.3.1 Specification Translation

- **J.P. Martin Flatin's Management Information Model**: J.P. Martin-Flatin proposed SNMP MIB to XML translation models, namely Model-level mapping and Metamodel-level mapping [3]. 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.

- **F. Strauss's libsmi**: F. Strauss presented a library to access SMI MIB information, "libsmi" [26], 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.

- **Our Work on MIB to XML Translation**: In our previous work on the XML/SNMP Gateway, we developed an SNMP MIB to XML translation algorithm, and also implemented an SNMP MIB to XML translator using this algorithm [1]. For validation of the algorithm, we implemented an XML-based SNMP MIB browser using this SNMP MIB to XML Translator.

2.3.2 XML/SNMP Gateway

- **F. Strauss's XML/SNMP Gateway**: Recently, F. Strauss implemented an SNMP-to-XML gateway [25] using mibdump [26]. The gateway works as follows. When a MIB module to be dumped is passed to mibdump, an SNMP session is initiated. Next, sequences of SNMP GetNext operations are issued to retrieve all objects of the MIB from the agent. Mibdump collects the retrieved data, and the contents of these data are dumped in the form of an appropriate XML document with respect to the predefined XML Schema. However, this gateway provides no other granularity than the level of MIB modules: neither single objects nor tables of a MIB.

- **Avaya Labs Research**: Avaya Labs. is currently developing an XML-based management interface for SNMP enabled devices [27]. The prototype system consists of three parts:

- A tool for automatic generation of XML Schema definition based on an SNMP SMI information module.
- An XML-RPC [30] based messaging protocol for retrieving and modifying MIB information in SNMP enabled devices. The messaging protocol defines the XML Schema for a set of query commands (GET, SET, LIST, CREATE, DELETE) and identifies MIB variables using XPATH-based identifiers.
- An adapter for retrieving and modifying device information in the form of XML data based on the information in the device's MIB.

They have already implemented a tool for mapping SNMP SMI information modules to XML Schema. This tool is an extension of a previously implemented tool for converting SNMP SMI to CORBA-IDL [28]. They are currently implementing the XML document adapter for SNMP MIB modules using net-snmp [29] and XML-RPC libraries [30].

The XML-SNMP integration work by F. Strauss and Avaya Labs has not yet produced any concrete or detailed mechanism for interaction translation of the XML/SNMP gateway. This thesis focuses on the interaction translation of the gateway for high-level operations in XML-based integrated network management.

3. Interaction Translation Methods for XML/SNMP Gateway

In this section, we present an architecture of the XML/SNMP gateway and describe the three methods proposed for interaction translation in the gateway.

3.1 Architecture of XML/SNMP Gateway

An XML/SNMP gateway provides a method to manage network devices equipped with SNMP agents in an XML-based integrated network management. The gateway translates and relays management information between the XML-based manager and the SNMP agent. The XML/SNMP gateway must provide a specification translation and an interaction translation. The gateway converts a MIB definition into an XML Schema definition, and which is used for validation of each instance of the XML document. Also, the gateway translates the XML operation from the XML-based manager into an SNMP operation.

Figure 2 illustrates the architecture of an XML/SNMP gateway. An XML document of a MIB in the gateway is created by a specification translator, called a MIB to XML translator, which translates the MIB definition into an XML Schema and an XML document [1]. This translation result is the basis for the interaction translation. The Request Handler analyzes the request from the XML-based manager and invokes the corresponding method of the XML Parser. The XML Parser retrieves OIDs from the XML document and delivers them to the SNMP Stack. The SNMP Stack then sends an SNMP request to the SNMP Agent, and returns its response to the XML Parser. The XML Parser returns newly updated XML content to the Request Handler, which then delivers the result to the manager. The Trap Handler receives SNMP trap messages from the agent, and fills out the XML document with the trap information. The XML-encoded trap information is sent to the Trap Reporter. The Trap Reporter then sends the notification message to the manager.

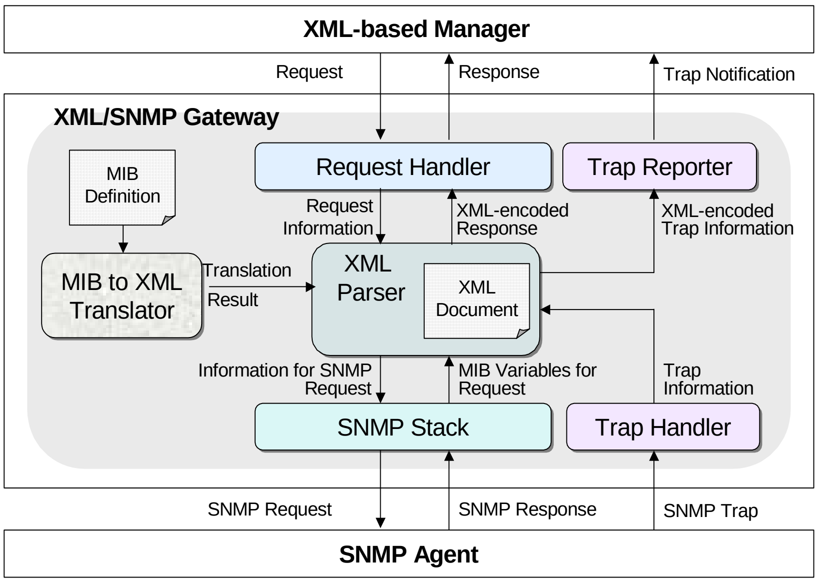


Figure 2. Architecture of XML/SNMP Gateway

3.2 Interaction Translation Methods

In this section, we describe the details of the three methods proposed for interaction translation in the XML/SNMP gateway.

3.2.1 XML Parser-based Translation

The XML/SNMP gateway transfers management information to the XML-based manager, converting SNMP MIB from the SNMP Agent into an XML document. Basically, the gateway manipulates the XML structure for information translation and intermediate storage for management data using an XML parser.

As explained in Section 2.1, there are currently two popular paradigms for processing XML, which are DOM [10] and SAX [12]. DOM builds a complete object representation of the XML document in memory. This can be memory-intensive for large documents. Only after the document is completely parsed can the application manipulate the data. SAX operates at one level lower. Rather than actually constructing a model in memory, it informs the application of the start and end of elements through events.

In our approach to the XML parser-based translation, we first consider the DOM-based translation because a DOM parser provides a random access using an element name or an XPath expression and various manipulations of an XML document, whereas SAX does not support any modification of the XML document data. In this method, the standard DOM interface is translated into an SNMP operation. This method is very useful in the case of an internal gateway, which is integrated within a manager system, and can be the basis for HTTP- and SOAP-based translation methods. The manager can directly access management information in the DOM tree using the DOM API provided by the gateway. Below, we describe the behavior and the additional meaning of the DOM interfaces in the gateway, and show the translation between DOM interfaces to SNMP operations.

Table 1 shows the mapping of DOM interfaces to SNMP operations. It includes fundamental interfaces described in the DOM Level 2 core specification [10]. The attributes of the DOM interfaces, such as *Node* and *Text*, are mapped to MIB nodes as the result of a specification translation, and the methods are translated into operations, involving SNMP operations. Interfaces excluded in this mapping have no special meaning in this translation method. First, a request for the *NodeValue*, which is an attribute of the *Node* interface, is translated into an SNMP GET request, then the current *NodeValue* is updated with a new value through the SNMP GET. In the case of a modification of the *NodeValue* from the manager, this is translated into an SNMP SET request after comparing the new value with the current value. Further, modification of the *NodeValue* of the trap nodes caused by an SNMP Trap is translated into a notification message to the manager through the pre-defined DOM event handler.

Interfaces	Methods / Attributes	Translation Results
Document	*Document*	• A MIB module
	GetElements ByTagName ()	• Returns a list of node with a specified node name in the entire MIB module
Node	*Node*	• A node with type ELEMENT_NODE: translated into a MIB node • A node with type TEXT_NOE: translated into a value of a MIB leaf node
	NodeName	• The name of a node
	NodeValue	• Retrieval from the manager: translated into an SNMP GET request • Modification from the manager: translated into an SNMP SET request • Modification on *trap* nodes: translated into a notification message through DOM event handler
	ChildNodes / NextSibling	• Request for *ChildNodes* or *NextSibling*: translated into the SNMP GETNEXT request, and returns its newly updated value(s)
Text	*Text*	• The value of a MIB leaf node

	AppendData()
	InsertData()
		• First, retrieves the current value through the SNMP GET request
		• Next, it inserts or appends a specified value to the current value
		• Finally, translated into the SNMP SET request with the modified value

Table 1. Mapping of the DOM Interfaces to SNMP Operations

As a result of specification translation, a DOM tree in the gateway consists of elements mapped to MIB nodes. The value of a leaf node in the MIB is stored in the text node of an element in the DOM. The value of the text node is initialized with a null value. Next, the manager accesses the text node: that is, the manager requests the value of *Element::firstChild* or *Element::nextSibling*. Afterwards, the gateway sends an SNMP GET request and updates a node value with the returned data. In the case of a table object, which has variable numbers of instances, an element for the table object in the DOM consists of a list of child elements with the same number of the instances. These children are created and deleted dynamically as the instances of the table object are changed. Using the SNMP GetBulk operation, we can also improve the consistency of MIB table data and reduce the number of SNMP requests resulting in the efficiency improvement of the gateway.

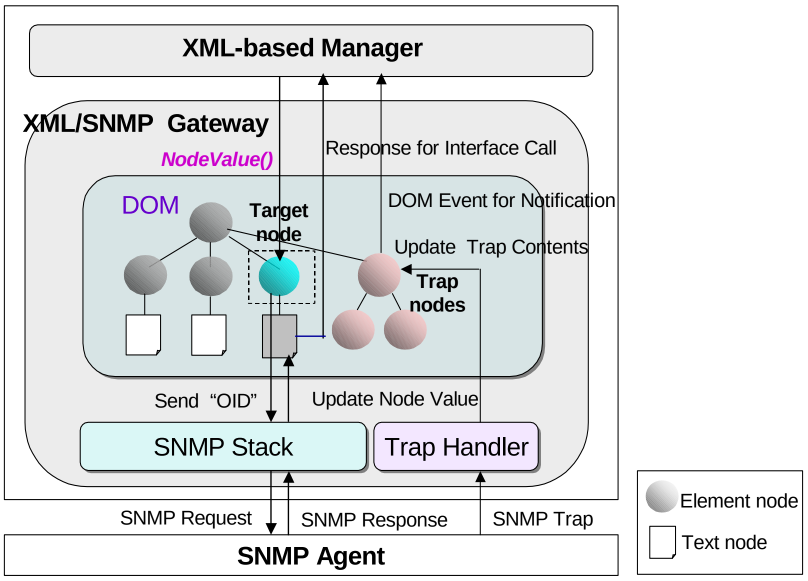


Figure 3. Interaction between XML-based Manager and Gateway using DOM

Figure 3 illustrates how the manager retrieves a MIB value using DOM API, which involves an SNMP operation. A DOM interface call from the XML-based manager is translated into an SNMP GET or SET request and returns a node value from the SNMP response to the manager after filling out the DOM content. In the case of notification from the agent, the Trap Handler receives an SNMP Trap and updates trap nodes in the DOM. This modification causes the invocation of a predefined event handler. The event handler then sends a notification message to the manager. This event mechanism is provided by standard DOM event interfaces.

Using the additional DOM interfaces, such as the Traversal and Range Interfaces, the manager and the gateway can also easily implement various and sophisticated functions. The DOM Traversal Interfaces provide different logical views using a filtering interface, and the DOM Range Interfaces provide methods to access and manipulate the document tree within a specified range [11]. The DOM-based translation method can extend its functionality through these additional features of DOM interfaces, and the XML/SNMP gateway can easily integrate legacy SNMP agents into XML-based management through the translation between DOM interface-based requests to SNMP requests and vice versa.

3.2.2 HTTP-based Translation

In this section, we describe the HTTP-based translation method. The HTTP [22] is a generic stateless protocol, which can be used for many applications in a standardized manner through extension of its request methods, error codes and headers. In this method, the XML/SNMP gateway translates an HTTP request from the XML-based manager to an appropriate SNMP request according to the parameters of an HTTP request message. An HTTP GET request is used for retrieving management data and a POST is for updating the data. We extend an HTTP message with XPath [13], XQuery [14] and XUpdate [15]. This method provides efficient ways to retrieve MIB objects in XML/HTTP communication. XPath, XQuery and XUpdate expressions are delivered as a parameter in URI of HTTP GET and message content of HTTP POST.

Figure 4 illustrates the translation of HTTP requests into SNMP requests using XPath, XQuery and XUpdate. In Figure 4, the DOM parser is used for processing an XML document. The SAX parser can also be applied here instead of the DOM parser. The Request Handler receives and parses an input argument in the HTTP request, and delivers the XPath/XUpdate expression to the XPath/XUpdate Handler. This handler analyzes the expression and retrieves a list of target nodes, specified by the expression, using the DOM interface. The interface call is translated into the SNMP request in the same manner as the XML parser-based translation method. For notification delivery, the HTTP Client in the gateway sends an asynchronous event message to the HTTP Server in the manager. This event message is generated by the DOM event handler, as mentioned in Section 3.2.1.

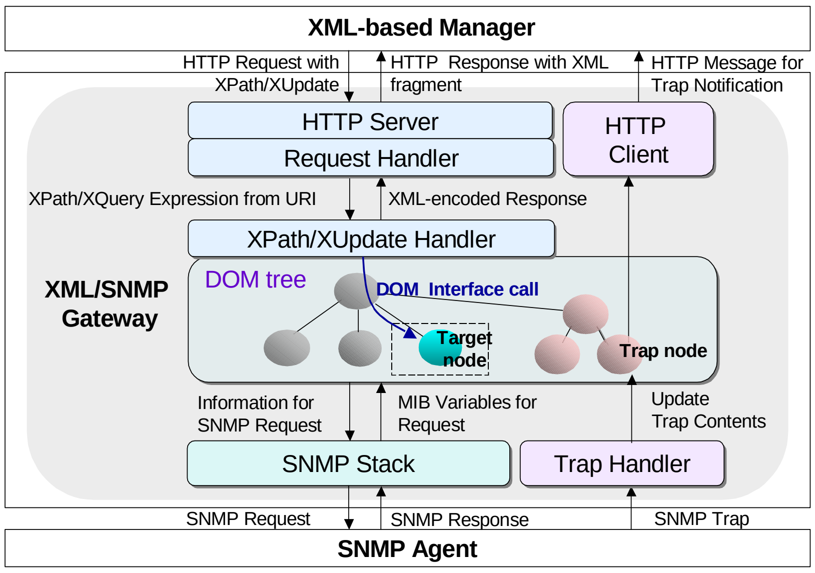
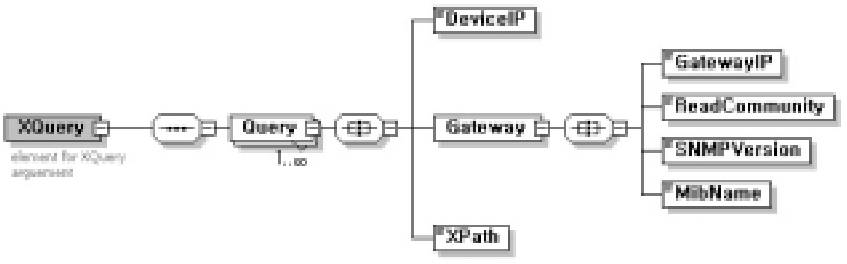
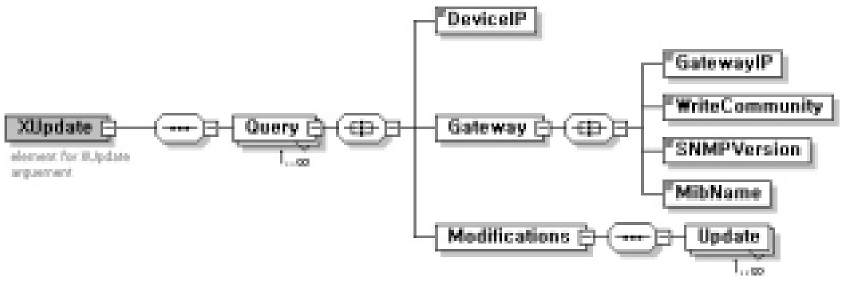


Figure 4. Interaction Translation of HTTP Request to SNMP Request

When a manager requests specific management information, an expression for addressing the managed objects is essential in the request message. We use XPath and XQuery to indicate a target object. XPath is a standard for addressing particular parts of an XML document and provides a rich addressing mechanism for an efficient and effective query on management information.



(a) *XQuery* parameter format



(b) *XUpdate* parameter format

Figure 5. XML Schema for *XQuery* and *XUpdate* Format in HTTP Request Parameter

We first define an XML Schema for a GET/POST message in HTTP communication between the manager and the gateway. The manager retrieves management information using an HTTP GET request with a parameter named *XQuery*, which describes the detailed request. Figure 5 shows the XML Schema diagrams for *XQuery* and *XUpdate* parameters using an XML editor, XML Spy [36]. The XML Schema for *XQuery* includes definitions of elements, which are *DeviceIP* for device identification, *XPath* for addressing portions of management information, and several elements for SNMP communications. These XML

Schemas are also applied to the parameter format of the SOAP operations, such as *get* and *set* explained in Section 3.2.3. The manager sends HTTP POST requests to insert, delete, and update management information. An HTTP POST message contains request details in its message body. For HTTP POST message, we define a parameter named *XUpdate* using the XUpdate [15] extension. XPath expressions are used in *XUpdate* parameter as an attribute of *Update* element to specify target nodes for each update. A *Modifications* element has one or more *Update* elements, and thus multiple modifications can be processed in one HTTP request from the manager. An asynchronous trap message from an SNMP agent is sent through the HTTP POST message via the XML/SNMP gateway. The gateway generates an XML document, which contains the trap message, and delivers it to the manager using the HTTP POST message.

Example of Using XPath
http://hostname:8080/gateway?XQuery=<XQuery><Query>
<DeviceIP>141.223.82.72</DeviceIP>
<Gateway><GatewayIP>141.223.82.56</GatewayIP>
<ReadCommunity>public</ReadCommunity><SNMPVersion>1</SNMPVersion>
<MibName>RFC1213-MIB</MibName></Gateway>
<XPath>device[@type="server"]</XPath></Query>
<Query> … </Query><XQuery>
Example of Using XQuery
<result> { Let $t := input() //ifTable/ifEntry/ ifType[contains(./text(), "6")]
RETURN
<totalInOutOctets count="{count($t) }"><in> { sum($t/ifInOctets/text()) } </in>
<out> { sum($t/ifOutOctets/text()) } </out></totalInOutOctets> } </result>
Example of Using XUpdate
<XUpdate><Query><DeviceIP>141.223.82.72</DeviceIP>
<Gateway><GatewayIP>141.223.82.56</GatewayIP>
<WriteCommunity>media</ WriteCommunity ><SNMPVersion>1</SNMPVersion>
<MibName>RFC1213-MIB</MibName></Gateway><Modifications>
<Update select="//sysConact">admin</Update>
<Update>…</Update></Modifications></Query><Query> … </Query></XUpdate>

Table 2. Examples of using XPath, XQuery and XUpdate in HTTP Request

Table 2 shows examples of using XPath, XQuery and XUpdate in HTTP requests. XQuery provides a powerful and structured facility. XQuery uses XPath as a subset and can easily express a complicated query. For example, when a manager retrieves the total number of in/out octets of a network interface with type "Ethernet" in the MIB-II *interfaces* table, the manager must send a number of requests to retrieve all instances of the objects in the table and then calculate the total number of octets from the returned values. The second example in Table 2 simplifies this request applying XQuery. XQuery also provides such features as filtering a document to produce a table of contents, joining across multiple data sources, grouping and aggregating the contents, and querying based on sequential relationships in documents. The XML-based manager can reduce the number of requests and data transfer for further processing, such as complicated statistical analysis. Thus, it can improve management efficiency by adopting XPath and XQuery in a message format.

3.2.3 SOAP-based Translation

As mentioned in Section 2.1, SOAP [16] is a protocol for exchanging XML-based messages over HTTP or SMTP. At the basic functionality level, SOAP can be used as a simple messaging protocol and can also be extended to an RPC protocol. In this section, we describe an interaction translation method based on a SOAP-based communication.

An XML-based manager exchanges an XML-encoded message with communication peers, such as an XML-based agent and an XML/SNMP gateway. This means that a request message from the manager and a response message from the gateway are all formatted as an XML document. SOAP provides a standard method to transfer XML-encoded messages over HTTP between the manager and the gateway.

We define several SOAP RPC messages between an XML-based manager and an XML/SNMP gateway using WSDL [17]. As explained in Section 2.1, WSDL is an XML-based language used to define Web Services and describe how to access them, and SOAP is becoming the de facto standard protocol for Web Services. Therefore, we apply the WSDL Schema for service description instead of a proprietary XML structure using an XML Schema. The manager finds a definition of a method to invoke using this WSDL document. As described in Table 3 and Table 4, WSDL defines the main elements as follows:

- *message*: An abstract, typed definition of the data being communicated
- *operation*: An abstract description of an action supported by the service
- *portType*: An abstract set of operations supported by one or more endpoints
- *binding*: A concrete protocol and data format specification for a particular port type
- *port*: A single endpoint defined as a combination of a binding and a network address
- *service*: A collection of related endpoints

The WSDL document in Table 3 describes the *get* and *set* operations published by the gateway as a service named *SoapInterfaceService*, and the WSDL in Table 4 shows the definition of *trap* notification published by the manager in order to receive notification message from the gateway.

get and *set* Operations

```
<wsdl:definitions (namespace declarations) >
  <wsdl:message name="setRequest">
    <wsdl:part name="param" type="xsd:string" /></wsdl:message>
  <wsdl:message name="setResponse">
    <wsdl:part name="setReturn" type="xsd:string" /></wsdl:message>
  <wsdl:message name="getRequest">
    <wsdl:part name="param" type="xsd:string" /></wsdl:message>
  <wsdl:message name="getResponse">
    <wsdl:part name="getReturn" type="xsd:string" /></wsdl:message>
  <wsdl:portType name="SoapInterface">
    <wsdl:operation name="get" parameterOrder="param">
     <wsdl:input message="intf:getRequest" name="getRequest" />
     <wsdl:output message="intf:getResponse" name="getResponse" /></wsdl:operation>
    <wsdl:operation name="set" parameterOrder="param">
     <wsdl:input message="intf:setRequest" name="setRequest" />
     <wsdl:output message="intf:setResponse" name="setResponse" /></wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="SoapInterfaceSoapBinding" type="intf:SoapInterface">
    <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="get"><wsdlsoap:operation soapAction="" />
      <wsdl:input name="getRequest">
        <wsdlsoap:body encodingStyle=http://schemas.xmlsoap.org/soap/encoding/
        namespace="http://hostname:8080/axis/SoapInterface.jws" use="encoded" />
      </wsdl:input>
      <wsdl:output name="getResponse">
        <wsdlsoap:body encodingStyle=http://schemas.xmlsoap.org/soap/encoding/
        namespace="http://hostname:8080/axis/SoapInterface.jws" use="encoded" />
      </wsdl:output></wsdl:operation>
    <wsdl:operation name="set"><wsdlsoap:operation soapAction="" />
      <wsdl:input name="setRequest">
        <wsdlsoap:body encodingStyle=http://schemas.xmlsoap.org/soap/encoding/
        namespace="http://hostname:8080/axis/SoapInterface.jws" use="encoded" />
      </wsdl:input>
      <wsdl:output name="setResponse">
        <wsdlsoap:body encodingStyle=http://schemas.xmlsoap.org/soap/encoding/
        namespace="http://hostname:8080/axis/SoapInterface.jws" use="encoded" />
      </wsdl:output></wsdl:operation></wsdl:binding>
  <wsdl:service name="SoapInterfaceService">
  <wsdl:port binding="intf:SoapInterfaceSoapBinding" name="SoapInterface">
    <wsdlsoap:address location="http://hostname:8080/axis/SoapInterface.jws" />
  </wsdl:port></wsdl:service></wsdl:definitions>
```

Table 3. WSDL Definition of *get* and *set* Operations

trap Notification
<wsdl:definitions (*namespace declarations)* >
<wsdl:message name="trapRequest"><wsdl:part name="param" type="xsd:string" />
</wsdl:message>
<wsdl:message name="trapResponse"><wsdl:part name="trapReturn" type="xsd:boolean" />
</wsdl:message>
<wsdl:portType name="SoapInterface"><wsdl:operation name="trap" parameterOrder="param">
<wsdl:input name="trapRequest" message="intf:trapRequest" />
<wsdl:output name="trapResponse" message="intf:trapResponse" /></wsdl:operation>
</wsdl:portType>
<wsdl:binding name="SoapInterfaceSoapBinding" type="intf:SoapInterface">
<wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
<wsdl:operation name="trap"><wsdlsoap:operation soapAction="" />
<wsdl:input name="trapRequest">
<wsdlsoap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://hostname:8080/axis/SoapInterface.jws" /></wsdl:input>
<wsdl:output name="trapResponse">
<wsdlsoap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://hostname:8080/axis/SoapInterface.jws" /></wsdl:output>
</wsdl:operation></wsdl:binding>
<wsdl:service name="SoapInterfaceService">
<wsdl:port name="SoapInterface" binding="intf:SoapInterfaceSoapBinding">
<wsdlsoap:address location="http://hostname:8080/axis/SoapInterface.jws" /></wsdl:port>
</wsdl:service></wsdl:definitions>

Table 4. WSDL Definition of *trap* notification

Guidelines for designing and implementing Web Services recommends the use of a course-grained RPC [19]. That is, more coarse-grained service operations reduce network overhead and improve performance, though they are less flexible. We define services into three big operations, namely *get*, *set*, and *trap*. Each operation has a parameter which is tagged with *param* and a return value, which is also defined as an XML element between the manager and the gateway. Besides the essential elements, such as *get*, *set* and *trap*, the SNMP GetBulk operation or other complex types of requests can be defined by extending the *get* or *set* operation with XPath or XQuery. As described in Section 3.2.2, XPath, XQuery and XUpdate provide an efficient way of indicating the managed objects to be retrieved. XPath, XQuery and XUpdate can also be applied in the SOAP request message as a parameter of each method.

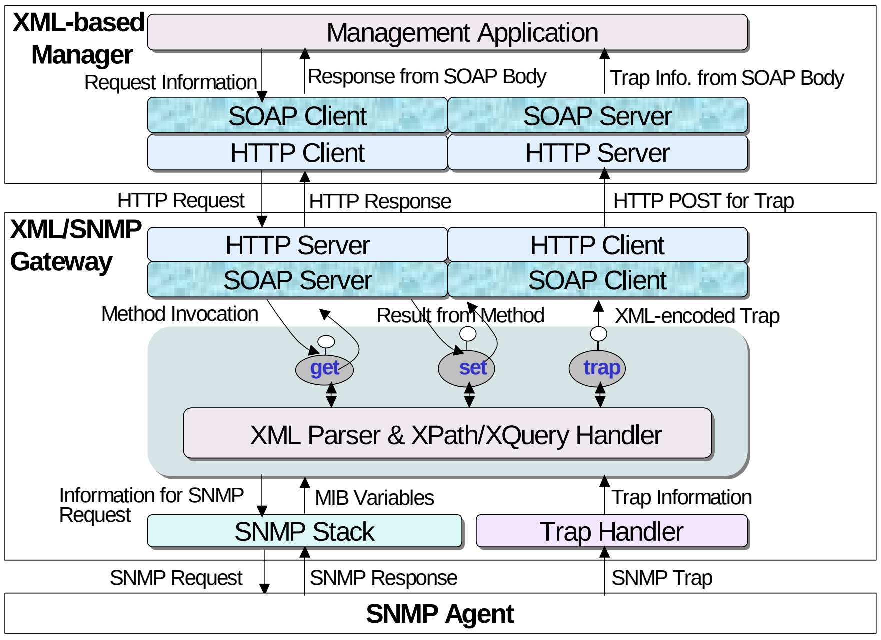


Figure 6. SOAP-based Architecture of Manager and Gateway

Figure 6 illustrates a SOAP-based architecture of XML-based manager and the gateway. In this architecture, the SOAP Client in the manager generates an XML-encoded SOAP request, such as *get* and *set*. Then the HTTP Client sends the HTTP POST request including the SOAP request in its body to the HTTP Server in the gateway. The SOAP Server parses the HTTP message into a properly formatted RPC call and invokes an appropriate method published by the gateway. This SOAP Server receives the result of the method and generates a well-formed SOAP response message. The response message backtracks to the SOAP Client in the manager. Finally, the manager application receives the results of the method invocation. For notification delivery, the SOAP Client in the gateway sends an asynchronous event message to the SOAP Server in the manager. This event message is generated in the same way as the XML parser- and HTTP-based translation methods. Figure 7 shows the sequence of the SOAP message exchange between the manager and the gateway.

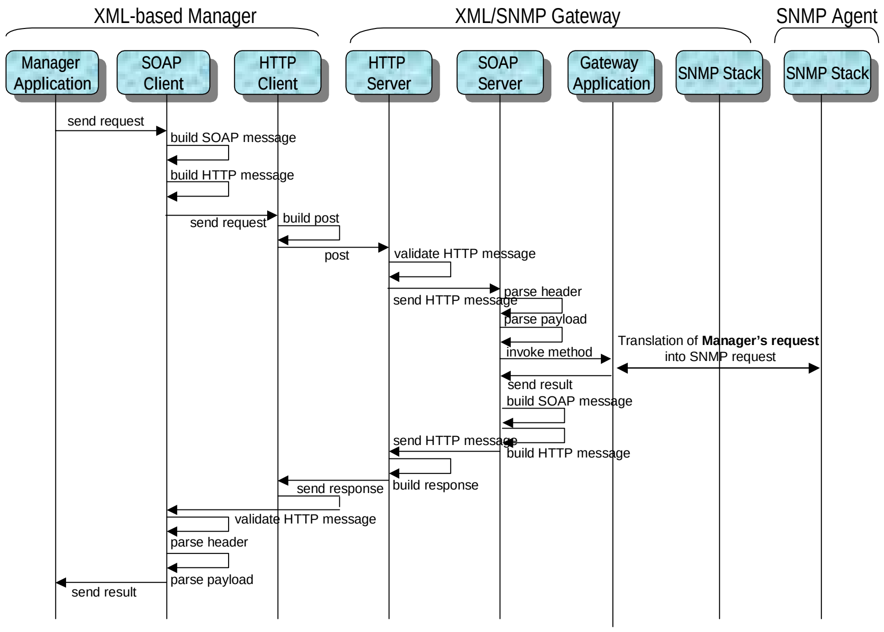


Figure 7. A Sequence Diagram of Manager-Agent Interaction using SOAP

It is very common to exchange XML encoded data over HTTP. However, SOAP provides a better solution than a proprietary XML/HTTP because SOAP is an open standard with a growing body of developers and vendors supporting it. SOAP defines a standard vocabulary and a structure for messages between communication peers, which eliminates the overhead of parsing and processing messages by a proprietary method in managers and gateways. Further, the WSDL Schema provides self-documenting features of defining data and interface type like IDL in CORBA and DCOM, and provides the basis of an RPC mechanism. One disadvantage of SOAP is the overhead in converting the native format of application data into an XML-based SOAP message. However, this overhead is eliminated in this approach because the XML/SNMP gateway already deals with XML documents as a communication message.

3.3 Analysis of the Proposed Methods

In this section, we discuss the advantages and disadvantages of the proposed methods. Table 5 summarizes them.

Methods	Advantages	Disadvantages
XML Parser-based Translation	• Can be the basis for the other translation methods • No need for a request handler between internal gateway and manager	• Imposes a burden on the manager by invoking a series of interfaces for request processing in an appropriate order
HTTP-based Translation	• Simple to implement in XML/HTTP • Provides an efficient mechanism for querying managed objects	• No standard in the use of URI string as a request specifier • Need of XPath, XQuery and XUpdate parsers
SOAP-based Translation	• Simple to implement over HTTP • Inherits advantages in the HTTP-based translation • Provides a standard method to implement an RPC	• Needs a SOAP server and client • Overhead of packaging SOAP messages

Table 5. Advantages and Disadvantages of the Interaction Translation Methods

As the first approach to interaction translation in the XML/SNMP gateway, we have focused on the DOM interface in the XML parser-based translation. The gateway uses the DOM structure and its interfaces to translate management information between XML and SNMP. This DOM level translation provides a method to directly access management information through the standard DOM interfaces for the XML-based manager. This method can be the basis for the other two translation methods. The XML parser-based method is targeted especially for an internal gateway, which is integrated within the manager system. In an internal gateway, the manager's request using the DOM interface is directly translated into an SNMP operation, so there is no need for a request handler between the internal gateway and the manager. However, this imposes a burden on the manager by calling a series of interfaces for processing requests.

Next, the HTTP-based translation method can be easily implemented in XML over HTTP, applying XPath, XQuery and XUpdate. This method also provides an efficient and effective communication between a manager and a gateway by reducing the number of requests and resulting in less data transfer. However, in order to support XPath, XQuery and XUpdate, both the manager and the gateway must include an XPath, XQuery and XUpdate enabled modules, and which are supported by commonly used XML parsers [31, 37].

Finally, we proposed the interaction translation based on the SOAP-based communication. In this method, an operation of a service defined in a WSDL is translated into a specific method served by the gateway. This translation method has the advantages of the HTTP-based method because SOAP is based on HTTP, and a SOAP message can also contain XPath, XQuery and XUpdate expressions in a parameter of each message. Using SOAP, the gateway can receive requests from and send responses to the XML-based manager in a standardized way and eliminate the overhead of parsing and processing messages by a proprietary method on XML/HTTP. The manager also has the advantage of remote access to gateway codes without knowing how to package up an XML message and make an HTTP request. Finally, the gateway can communicate with any type of manager, which can understand the interface definition in the XML Schema or WSDL, and which can generate SOAP RPC messages from the definition. However, the main drawback of SOAP is the parsing overhead involved in processing the XML-encoded SOAP requests compared to other equivalent RPCs. One of the early objections to the use of SOAP has been its performance, especially when used as an alternative to CORBA and COM. However, recent versions of SOAP implementations have only a slightly lower performance than RMI or IIOP [20]. This is mainly due to two factors: the use of SAX rather than

DOM to parse the messages and optimization lessons learned from earlier product releases. The use of SAX parsers has increased throughput, reduced memory overhead, and improved scalability.

4. Implementation

We have implemented an XML/SNMP gateway for our XML-based Global Element Management System (XGEMS) [4] which manages one or more types of network elements scattered possibly throughout the world. In this section, we explain the implementation details of the XML/SNMP gateway.

The XML/SNMP gateway has been implemented on a Linux server. We used the Apache Tomcat 4.0 for the Web server and Servlet engine. We used Xerces 1.4.4 [31] for an XML parser, Xalan 2.4.0 [32] for an XPath/XSLT processor. We also used Innovation's HTTP Client V0.3-3 [34] for the HTTP Client and OpenNMS's joeSNMP 0.2.6 [35] for the SNMP Handler and the Trap Handler. These are all Java-based.

We first implemented the XML parser-based translation method using the Xerces DOM parser. This translation method is the basis for the HTTP- and SOAP-based translation methods. In this method, the gateway generates and exposes a simple interface class for the manager to access this XML parser-based translation. The manager is implemented as a Java bean running with the gateway in the same process. It uses the gateway as API calls and can share a consistent DOM tree with the gateway.

The HTTP-based translation has been implemented and validated for our XGEMS. XPath and XUpdate expressions are applied to HTTP request messages, and all arguments for the requests are all XML-encoded. The HTML specifications [23] recommend that the HTTP GET method should be basically used for just retrieving data, whereas POST is recommended for storing or updating data, ordering a product, or sending E-mail. As the specifications recommend, we used the HTTP GET method for retrieving MIB information and the HTTP POST for updating MIB and sending traps. In the case of a lengthy parameter, GET may cause practical problems with implementations which cannot handle such long URIs. According to the official statement by Microsoft published in February 2000, the maximum URL length is 2,083 characters in Internet Explorer [24]. To begin to solve this limitation, we first replaced all GET requests with POST, and the performance was degraded as the overall processing time of each request increased. The details of the experiment are described in Section 5.2. Then, we decided to use HTTP GET for simple retrieval operations and POST for relatively long modification requests. We are now replacing the relatively wordy XML Schema for *XQuery* and *XUpdate* parameters with a concise XML Schema to complement the GET method and are considering a mechanism to divide a request into multiple ones when the length of a GET request message exceeds the URI length limitation.

For the SOAP-based translation, we used Apache Axis 1.0 for the SOAP engine, which is the SOAP 1.1 and WSDL 1.1 compliant. We deployed the existing translation functions into Web Services using a Java Web Service (JWS) [33] provided by Axis. This deployment method automatically generates a WSDL file and proxy/skeleton codes for the SOAP RPCs. In order to enable the existing Java class as a Web Service, we simply copy the Java file into the Axis Web application, using the extension ".jws" instead of ".java" We deployed the main class namely *SoapInterface* into Web Services as an interface to the SOAP-based XML/SNMP gateway. We installed the Axis SOAP engine both on the manager and the gateway. The SOAP Client of XGEMS can access the deployed services of the gateway by setting the SOAP endpoint to "http://hostname:8080/axis/SoapInterface.jws".

5. Performance Analysis

Using the implemented gateway, we have implemented and validated our proposed three translation methods of the gateway. We also evaluated the latency performance, which is the roundtrip time for sending and receiving messages between the manager and the gateway. We measured the roundtrip time to retrieve the whole MIB-II values using iterative SNMP GetNext requests. In this section, we describe the design of the performance experiment and present its results.

5.1 Experimental Design

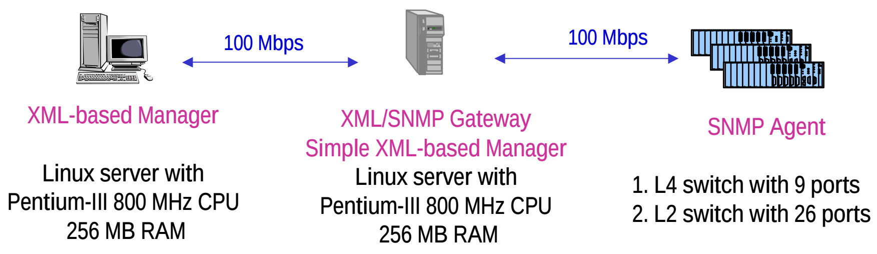


Figure 8. Performance Testbed

As described in Figure 8, we performed the test with the XML-based manager, gateway, and managed devices joined by a 100 Mbps LAN. For the gateway machine, we used a Linux server with a Pentium-III 800 MHz CPU and 256 MB RAM. For the manager, we also used a Linux server with a Pentium-III 800 MHz CPU and 256 MB RAM. In order to measure the performance of the XML parser-based method in an internal gateway, we implemented a simple XML-based manager as a Java bean within the gateway system. We performed the experiment with two managed devices with a different number of Ethernet interfaces. The managed devices are an L4 switch with 9 ports (Device 1), and an L2 switch with 26 ports (Device 2) within the same domain. The MIB-II interface table size increases as the number of ports increases, and thus the XML document size for MIB-II in each device also increases.

As illustrated in Figure 9, we categorized the test into four from T1 to T4, as follows: T1 is the latency between the SNMP stacks on the gateway and the SNMP agent; T2 is the latency of the DOM-based translation method; T3 is the roundtrip time for the HTTP-based translation; and T4 is for the SOAP-based translation. These mean the roundtrip time for retrieving whole MIB-II data from the managed devices. These results also show the overhead of each translation method compared with the latency of the SNMP communication between the gateway and the agent. *System.currentTimeMillis()* in JDK 1.4.1 is used for time measurement in the test. We also measured the setup time for each communication method, which means the time between when an HTTP or a SOAP request was initiated in the manager and when the gateway was actually contacted.

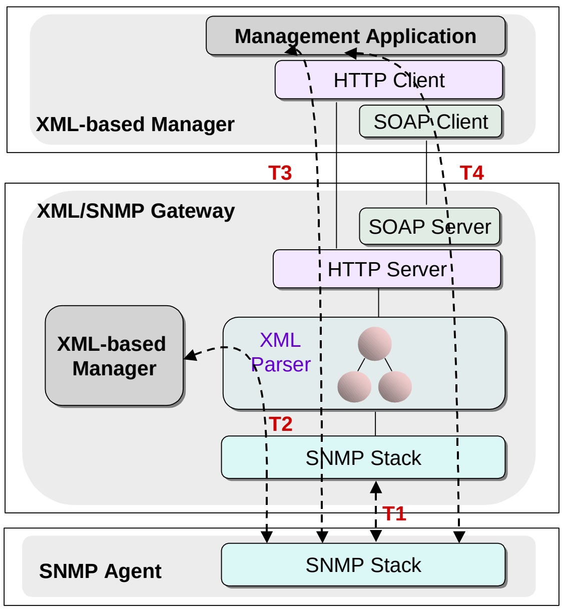


Figure 9. Latency Performance Measurements in XML/SNMP Gateway

In order to validate our proposed translation methods and provide guidelines for the implementation of the translation methods producing better performance, we also evaluated the performance according to the variations of the details applied to each translation method. First, we evaluated an overhead in parsing various XPath expressions which are commonly used in the proposed translation methods. Second, we compared the performance of the HTTP-based translation method using HTTP GET and HTTP POST. Finally, we considered the design features which impact performance of the gateway in SOAP-based translation. When examining the SOAP-based architecture, an efficient use of SOAP methods should be considered. Therefore, we tested the latency of several variations of message definition for the *get* operation.

5.2 Experimental Results and Analysis

Table 6 compares the latency performance of the XML parser-, HTTP- and SOAP-based translation methods. These results do not include the processing time for User Interface and the connection setup time for HTTP or SOAP. In these tests, the HTTP connection setup time is about 0.1 milliseconds. Therefore, this setup time is negligible for overall latency performance. Apache Axis took roughly 15 to 20 milliseconds for the SOAP connection setup. In addition, the processing time of specification translation and the loading time of the converted XML Schema document took about 535 milliseconds.

We first presented the latency between the SNMP stacks in the gateway and managed devices. The SNMP communication for retrieving MIB-II took about 1307 milliseconds for Device 1 and about 4283 milliseconds for Device 2. The number in parentheses represents the overhead for each translation method in percentage. As predicted, the XML parser-based interaction method added a little processing overhead to direct SNMP request, and the SOAP-based interaction method took much more processing overhead than other methods. However, each translation method adds approximately the same amount of overhead time to the basic SNMP processing time though each T1 value increases in proportion to the size of the MIB data. This additional translation overhead takes small portion of the overall processing time in the gateway.

Device (MIB size) / Method	Device 1 (28 KB)	Device 2 (54 KB)
SNMP Stack: T1 (ms)	1307.1	4283.6
XML Parser-based Translation: T2 (ms)	1360.6 (4.1 %)	4317.6 (0.8 %)
HTTP-based Translation: T3 (ms)	1419.1 (8.6 %)	4418.8 (3.2 %)
SOAP-based Translation: T4 (ms)	1613.3 (23.4 %)	4922.2 (14.9 %)

Table 6. Comparison on the Latency Performance of the Interaction Translation Methods

Next, we evaluated the overhead of parsing various XPath expressions in XML parser-based translation. We categorized XPath expressions into four types, expressions A to D, and measured the latency using sample expressions of those types. Table 7 shows the latency for retrieving MIB-II *interfaces* data using these four types of XPath expressions. In the cases of expression A and B, the XPath parsing overheads are quite small and negligible for the overall processing time. Most processing time in expression A is consumed for retrieving whole *ifTable* data from the SNMP agent. Expression B also consumed most of the processing time for retrieving *ifInOctets* and *ifOutOctets*, and its overall overhead is much smaller than the overhead in expression A because the retrieved data size decreased from 12 KB to 2 KB. However, the overhead for parsing an XPath increased considerably in the case for expression C though the expression specifies the same MIB objects as expression B. The retrieved data through expression C is exactly the same as the one in expression B, but the XPath parsing overhead in expression C causes an increase in overall overhead to about 200%.

In the case of expression D, the overhead for parsing an XPath is more serious. In order to filter the data with the condition that the value of *ifType* is 6 (ethernet-csmacd), the gateway must retrieve all of *ifType*, *ifInOctets* and *ifOutOctets*, and return *ifInOctets* and *ifOutOctets* with *ifType* 6. Thus, expression D especially took much time for parsing the complicated XPath expression.

Therefore, one must carefully consider which XPath expression to select for a particular operation though XPath provides expressions for exactly specifying target objects to retrieve and avoids for the gateway delivering unnecessary data to the manager. The XML/SNMP gateway must provide acceptable performance when concurrent calls from managers are made. Therefore, it is more desirable to delegate these overheads of XPath parsing to the managers provided that most manager systems have sufficient system resources. Also, it can be considered that the XML/SNMP gateway supports limited XPath expressions and time-consuming XPath parsing is performed in each manager for reducing the overhead of the gateway.

Type	A	B	C	D
Example Expression	//ifTable	//ifInOctets | //ifOutOctets	//ifType/following-sibling::ifInOctets | //ifType/following-sibling::ifOutOctets	//ifType[.='6']/following-sibling::ifInOctets | //ifType[.='6']/following-sibling::ifOutOctets
Retrieved Data Size (KB)	12.4	2.1	2.1	1.8
Overhead of XPath Parsing (ms)	0.5	0.6	108.2	1432.2
Overall Overhead (ms)	353.8	103.5	212.3	1682.8

Table 7. Comparison on the Parsing Overhead of XPath Expressions

Table 8 shows the comparison on the performance of HTTP-based translation method using HTTP GET and POST. According to the results, the

HTTP POST method shows slightly lower performance than the GET method, and its overhead increases as the overall processing time increases. In the case of retrieving whole MIB-II data using HTTP POST, the results increased by 21.6 milliseconds for Device 1 and 204.6 milliseconds for Device 2 compared to the result of using HTTP GET presented in Table 6 as T3. As we explained the implementation details of the HTTP-based translation in Section 4, it is more desirable to use the HTTP GET method for most retrieval operations of which the message length is within the limit of GET. As seen in these test results, the HTTP GET method generally shows a better performance than POST. Because of this performance problem, recent SOAP implementations also support the HTTP GET binding as well as POST.

Device (MIB data) / Method	Device 1		Device 2	
	system group	MIB-II	*system* group	MIB-II
HTTP GET (ms)	96.5	1419.1	94.9	4418.8
HTTP POST (ms)	112.2	1440.7	101.2	4623.4

Table 8. Performance Comparison of HTTP GET and POST in HTTP-based Translation Method

Finally, Table 9 summarizes the test result on the effects of the variations in the number of parameters and the number of method calls of a SOAP message on the overall SOAP performance. A SOAP service can consist of any number of methods, and several method calls can be combined into a single method call. This improves the SOAP server's performance by reducing the overhead associated with requests processing. The client also benefits since it is more efficient to send and receive a block of data at once resulting in the reduced latency of the network. However, it could be worth dividing a method into individual methods if the same subset of the method is frequently used in the SOAP service.

We first performed a test with the *get* operation with a different number of parameters. The first three columns in Table 9 show the latency using the *get* operation for MIB-II retrieval from Device 1. The result in the first column of the three is exactly the same as T4 for Device 1 in Table 6. We modified the WSDL definition presented in Section 3.2.3 and defined different types of *get* operation dividing the parameter, namely *param*, into two and six parameters. As shown in Table 9, the latency increases as the number of parameters increases from one to six. In Section 3.2.2, we defined and applied the *XQuery* and *XUpdate* parameters in the HTTP-based translation. We also used *XQuery* and *XUpdate* parameters as the only parameter in *get* and *set* operations. Instead of defining multiple parameters for a method, we defined one parameter which is encoded with several meaningful tags which can be parsed into several arguments, and thus reduced the overhead of parsing multiple parameters in the SOAP server of the gateway.

Type / Latency	*Get* operation using one method with			*Get* operation using	
	1 param (type A)	2 params (type B)	6 params (type C)	One method (type D)	Two methods (type E)
Device 1 (ms)	1613.3	1676.6	1908.8	1613.3	1743.5
Device 2 (ms)	4922.2	4993.1	5298.4	4922.2	5063.3

Table 9. Latency of Several Variations of SOAP Get Operation

We also performed a test with two types of *get* operations: one includes one SOAP method call and the other includes two SOAP method calls. Type D is the same as type A using one method with one parameter. Type D using a single big method is useful for the manager which usually connects to the gateway, retrieves bulk data in one request and disconnects. By contrast, type E using multiple methods with separate functions is efficient for the manager which retrieves data in sequential requests with one connection. In general, type E is scalable and efficient because the repeated function in type D is performed once in type E using the separate method. However, we applied type D because the manager can retrieve various portions of MIB using iterative XPath expressions in one request, and type D provides better performance.

In summary, we have presented the effects of several variations of the proposed methods on the overall performance of the gateway. First, the XPath parsing overhead considerably affects the overall overhead of the gateway, and the use of alternative expressions or the split of the parsing overhead between the gateway and the manager increases the performance of the gateway. Second, the latency of the HTTP GET method is smaller than the HTTP POST latency. Finally, one SOAP method call with a single parameter generally produces better performance compared to more than one method call, or one with multiple parameters for the same operation. We reduced the interaction translation overheads applying these test results and validated that our XML/SMMP gateway results in a better performance.

6. Conclusions and Future Work

In this thesis, we have proposed three interaction translation methods between an XML-based manager and the XML/SNMP gateway. First, we proposed an XML parser-based translation method, which enables the manager to directly access the internal gateway using DOM or SAX interfaces in order to exchange management information with the SNMP agent. In HTTP-based translation, we defined a parameter for an HTTP request using XPath, XQuery, and XUpdate. XPath, XQuery, and XUpdate can be easily applied to HTTP messages in order to express the location path of target objects and a query language, and to describe multiple modification operations in a single expression. This method improves efficiency in XML/HTTP communication, which is the most common in the exchange of XML documents. We also proposed a SOAP-based translation method. Using SOAP, the gateway provides a flexible and standardized method for interaction with an XML-based manager in a distributed environment. The gateway can communicate with any type of XML-based manager, which can understand the interface definition in WSDL, and which can generate a SOAP message from the interface definition.

We have implemented and validated the XML/SNMP gateway for our XML-based Global Element Management System (XGEMS) using all three of the proposed methods. The HTTP- and SOAP-based translation methods are based on the XML parser-based translation. We have also performed several experiments for the latency performance of the three interaction translation methods and validated the proposed architecture. We evaluated the performance according to the variations of the details applied to each translation method. First, we evaluated an overhead in parsing various XPath expressions. Second, we compared the performance of the HTTP-based translation method using HTTP GET and HTTP POST. Finally, we tested the latency of several variations of message definition for a SOAP operation. Through the experiments, we found the effects of these variations on the overall performance of the gateway and validated that our XML/SMMP gateway provides efficient translation facilities in the integrated network management.

We are currently improving the efficiency and performance of the gateway using the SAX parser instead of the DOM parser and also plan to concentrate on the scalability of the gateway. We are interested in implementing Web Services using UDDI integration with our current implementation. Implementing the CORBA interface for the CORBA-based manager system and WBEM [38] implementation using CIM to XML mapping [40] and CIM operations over HTTP [41] are also considered to be valuable for future research.

References

[1] 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, 2002.

[2] 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. of the IEEE/IFIP Network Operations and Management Symposium (NOMS 2002), Florence, Italy, April 2002, pp. 5-18.

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

[4] M. J. Choi, Y. J. Oh, H. T. Ju and J. W. Hong, " XML-based Integrated Network Management for IP-based Networks ", Submitted to IEEE Network Special Issue on Network Management of Multi-service, Multimedia, IP-based Networks, October 2002.

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

[6] W3C, "Extensible Markup Language (XML) 1.0", W3C Recommendation, October 2000.

[7] W3C, "XML Schema: Part 0,1,2", W3C Recommendation, May 2001.

[8] W3C, "Extensible Stylesheet Language (XSL) Version 1.0", W3C Recommendation, October 2001.

[9] W3C, "XSL Transformations (XSLT) Version 1.0", W3C Recommendation, November 1999.

[10] W3C, "Document Object Model (DOM) Level 2 Core Specification", W3C Recommendation, November 2000.

[11] W3C, "Document Object Model (DOM) Level 2 Traversal and Ranges Specification", W3C Recommendation, November 2000.

[12] W3C, "Simple API for XML Version 2.0", WC3 Recommendation, November 1999.

[13] W3C, "XML Path Language (XPath) Version 2.0", W3C Working Draft, April 2002.

[14] W3C, "XQuery 1.0: An XML Query Language", W3C Working Draft, April 2002.

[15] XML:DB, "XUpdate", Working Draft, http://www.xmldb.org/xupdate/xupdate-wd.html, September 2000.

[16] W3C, "SOAP Version 1.2", W3C Working Draft, December 2001.

[17] W3C, "Web Services Description Language (WSDL) Version 1.2" W3C Working Draft, July 2002.

[18] OASIS, "Universal Description, Discovery and Integration (UDDI)", http://www.uddi.org/.

[19] Sun Microsystems Inc., "Using Web Services Effectively", 2002.

[20] P. J. Murray, "SOAP Status Report", http://www.capescience.com/articles/content/soapstatus.pdf, 2001.

[21] T. Berners-Lee, R. Fielding, and L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax", IETF RFC 2396, August 1998.

[22] R. Fielding, J. Gettys, J. Mogul, H. Frystyk Nielsen, L. Masinter, P. Leach and T. Berners-Lee, "Hypertext Transfer Protocol - HTTP/1.1", RFC 2616, IETF HTTP WG, June 1999.

[23] D. Raggett, A. Le Hors, I. Jacobs, "HTML 4.01 Specification", IETF HTML WG, http://www.w3.org/TR/html401, December 1999.

[24] Miscrosoft Corporation, "INFO: Maximum URL Length Is 2,083 Characters in Internet Explorer (Q208427)", September 2001.

[25] F. Straus, and T. Klie, "Towards XML Oriented Internet Management", Accepted to the 8th IFIP/IEEE International Symposium on Integrated Network Management (IM 2003), Colorado Springs, US, March 2003.

[26] Frank Strauss, "A Library to Access SMI MIB Information",

http://www.ibr.cs.tu-bs.de/projects/libsmi/.

[27] Avaya Labs., "XML based Management Interface for SNMP Enabled Devices", http://www.research.avayalabs.com/user/mazum/Projects/XML/.

[28] S. Mazumdar, "CORBA/SNMP Gateway", http://www1.bell-labs.com/project/CorbaSnmp/.

[29] NET-SNMP, http://net-snmp.sourceforge.net/.

[30] First Peer, Inc., "XML-RPC for C and C++", http://xmlrpc-c.sourceforge.net/.

[31] Apache XML project, "Xerces Java parser", http://xml.apache.org/xerces-j/.

[32] Aache XML project, "Xalan Java", http://xml.apache.org/xalan-j/.

[33] Aache XML project, "Axis", http://xml.apache.org/axis/.

[34] Innovation., "HTTPClient Version 0.3-3", http://www.innovation.ch/java/HTTPClient/.

[35] OpenNMS, "joeSNMP", http://www.opennms.org/mailman/listinfo/joesnmp/.

[36] Altova, "XML Spy", http://www.xmlspy.com/.

[37] Gnome project, "The XML C library: libxml", http://xmlsoft.org/.

[38] WBEM, "WBEM Initiative", http://www.dmtf.org/wbem/.

[39] DMTF, "Common Information Model (CIM)", http://www.dmtf.org/standards/standard_cim.php.

[40] DMTF, "Specification for the Representation of CIM in XML Version 2.0", DMTF Specification, July 1999.

[41] DMTF, "Specification for CIM Operations over HTTP Version 1.0", DMTF Specification, August 1999.

[42] Organization for the Advancement of Structured Information Standards, http://www.oasis-open.org/.

[43] SyncML Initiative, http://www.syncml.org/.

[44] 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 2002.

[45] M. Wasserman, "Concepts and Requirements for XML Network

Configuration", Internet-Draft, http://www.ietf.org/internet-drafts/draft-wasserman-xmlconf-req-00.txt, June 2002.

[46] T. Goddard, "Towards XML Based Management and Configuration", http://www.ietf.org/internet-drafts/draft-goddard-xmlconf-survey-00.txt, June 2000.

[47] S. Hollenbeck, et. al, "Guidelines for the Use of XML within IETF Protocols", http://www.ietf.org/internet-drafts/draft-hollenbeck-ietf-xml-guidelines-06.txt, August 2002.

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

요 약

최근 XML 기반의 네트워크 관리가 기존의 SNMP 기반의 네트워크 관리가 가지는 확장성, 효율성 등의 한계를 극복하기 위한 대안으로서 등장하였다. 그러나 아직까지 기존의 대부분의 네트워크 장비에 내장되어있는 SNMP 에이전트를 그대로 활용하여 네트워크를 통합 관리할 수 있는 방법을 제시하고 있지 못하다. 이러한 XML 기반의 통합 네트워크 관리 시스템을 위하여 XML 기반의 매니저 시스템과 SNMP 에이전트와의 연동을 제공하는 XML/SNMP 게이트웨이가 반드시 필요하다. XML/SNMP 게이트웨이는 이종의 두 프로토콜 사이에서 규약 변환(Specification Translation)과 함께 상호 작용에 대한 변환(Interaction Translation)을 모두 제공할 수 있어야 한다. 최근 이러한 XML/SNMP 게이트웨이의 구현에 있어서 기반이 되는 규약 변환, 즉 SNMP SMI (MIB)를 XML로 변환하는 연구들이 이루어지고 있다. 그러나 XML/SNMP 게이트웨이가 실제로 XML 기반의 매니저 시스템과 SNMP 에이전트 사이에서 구체적인 요청을 처리하고 그 결과를 반환하는데 필요한 상호 작용 변환에 대한 연구는 부재한 실정이다.

본 논문에서는 XML/SNMP 게이트웨이의 상호 작용 변환을 위한 세 가지 방법, 즉 XML 파서 기반의 변환 방법, HTTP 기반의 변환 방법 그리고 SOAP 기반의 변환 방법을 제시하고, 각 방법들의 장단점을 비교, 분석하였다. 먼저 게이트웨이가 관리 정보를 XML로 변환하기 위하여 사용하고 있는 XML 파서 기반의 변환 방법을 제시하였다. 이 변환 방법에서는 게이트웨이가 사용하는 XML 파서의 기능과 의미를 분석하여 이를 적절한 SNMP 오퍼레이션으로 변환한다. 이 변환을 통하여 매니저 시스템은 게이트웨이의 XML 파서를 통하여 관리 정보를 직접 접근할 수 있다. 이 방법은 특히 게이트웨이가 매니저 시스템 안에 통합되어 있는 형태인 내부 게이트웨이(Internal Gateway)의 경우 매우 유용하며, 또한 나머지 두 변환 방법인 HTTP 기반의 변환 방법과 SOAP 기반의 변환 방법을 위한 기반이 될 수 있다. 두 번째로 HTTP 기반의 변

환 방법에서는 현재 XML 문서의 전달을 위하여 가장 널리 사용되고 있는 HTTP 상에서 XPath, XQuery 그리고 XUpdate 표현을 사용하여 상세한 요청 메시지를 전달하고 HTTP 통신의 효율성을 높일 수 있는 방법을 제시하였다. 마지막으로, 최근 XML을 위한 새로운 표준 프로토콜로서 등장한 SOAP을 적용하여 게이트웨이가 SOAP RPC 메커니즘을 통해 분산된 매니저 시스템들에게 변환 기능을 제공할 수 있는 방법을 제시하였다.

우리는 XGEMS(XML-based Global Element Management System)에 대하여 우리가 제안한 세가지 상호 작용 변환 방법을 적용하여 XML/SNMP 게이트웨이를 구현하였으며, 이를 바탕으로 본 논문에서 제안한 세 가지 상호 작용 변환 방법의 성능을 분석하고, 다양한 구현 형태에 따른 성능 차이를 비교하였다. 먼저 각 관리 장비로부터 관리 정보를 가져오는 데에 소요되는 시간을 비교한 결과, 기본적으로 SNMP 스택을 통해 관리 정보를 가져오는데 소요되는 시간과 비교하여 각 변환 방법별로 변환에 따른 오버헤드의 증가가 있었으나 전체적인 처리 시간에 대하여 큰 비율을 차지하지 않았고, 또한 관리 정보의 사이즈 증가에 따른 각 변환 방법의 오버헤드의 증가율은 점차 감소하였다. 이러한 결과에 비추어 볼 때, 세가지 변환 방법에 따른 오버헤드가 전체적인 변환 기능 수행에서 큰 문제가 되지 않는 것으로 판단할 수 있다. 다음으로 각 변환 방법에 대한 다양한 구현 형태별 성능 분석 결과에 따르면, 우선 각 방법에서 공통적으로 사용되고 있는 XPath의 경우, 표현 양식에 따라 XML 파서에 의한 구문 분석 상의 오버헤드 차이로 인하여 전체적인 처리 시간에 상당한 영향을 가져옴을 알 수 있었다. 따라서 같은 요청을 처리하기 위하여 비교적 처리 시간이 짧은 XPath 표현을 사용하는 것이 매우 중요하며, 일반적으로 매니저 시스템의 실행 환경이 좋은 것을 고려할 때, XPath에 의한 최종적인 데이터 필터링 기능을 매니저에서 처리하도록 함으로써 전체적인 게이트웨이의 오버헤드를 줄이는 방안을 고려할 필요가 있다. 두 번째로 HTTP 기반의 변환

방법에서 HTTP GET 메소드를 사용하는 방법과 HTTP POST 메소드를 사용하는 방법에 따른 처리 속도를 비교한 결과, 예상하는 바와 같이 GET 메소드를 사용하는 경우 좋은 성능을 보였다. 마지막으로 SOAP 기반의 변환 방법에서 특정 SOAP 오퍼레이션에 대하여 하나의 메소드와 여러 개의 메소드를 사용한 방법에 따른 처리 속도를 비교하였으며, 또한 하나의 메소드에서 하나의 인자를 사용하는 경우와 여러 개의 인자를 사용하는 경우에서 각각의 처리 속도를 비교하였다. 이러한 실험 결과를 바탕으로 각 변환 방법의 상세 구현 내용에 따른 게이트웨이의 성능 변화를 확인하였으며, 본 논문에서 제안하는 XML/SNMP 게이트웨이가 XML 기반의 통합 관리에서 효율적으로 변환 기능을 제공할 수 있음을 보였다.

우리는 현재 게이트웨이의 성능 및 효율성 향상을 위하여 기존의 DOM 파서 대신 SAX 파서를 적용한 게이트웨이 시스템을 구현하고 있다. 또한 현재 SOAP 기반의 구현 시스템을 기반으로 하여 UDDI 통합을 통한 Web Services 아키텍쳐 구현 및 게이트웨이의 확장성 향상에 대하여 연구 중이다. 앞으로 CORBA기반의 관리 시스템과 연동하기 위한 CORBA 인터페이스를 제공하는 방안에 대하여 연구할 계획이며, CIM-XML 변환 및 CIM 오퍼레이션에 대한 HTTP 변환에 대한 기존 연구를 바탕으로 WBEM 아키텍쳐를 적용하는 방안에 대해서도 고려하고 있다.