95.495 Final Report

THE JXTA SERVICE IS MANAGED WITHIN SNMP AGENTX

Qiang Luo
277918

Supervisor: Tony White
August 12th, 2004
Acknowledgements

I would like to thank Dr. Tony White, of the School of Computer Science at Carleton University, for his assistance and guidance in the preparation of this project and report.
# Index

Abstract .......................................................................................................................................................... 6  

1 Introduction ........................................................................................................................................ 7  

2 Background ......................................................................................................................................... 9  
  2.1 SNMP ............................................................................................................................................. 9  
    2.1.1 Overview ................................................................................................................................. 9  
    2.1.2 SNMP Messages and Features ............................................................................................... 11  
    2.1.3 SNMP Packet Format .............................................................................................................. 12  
    2.1.4 SNMP Communication Layers ............................................................................................ 13  
    2.1.5 SNMP Versions ...................................................................................................................... 14  
  2.2 AGENTX ...................................................................................................................................... 15  
    2.2.1 AgentX Architecture and Component Responsibilities ..................................................... 16  
    2.2.2 PDU Description .................................................................................................................... 18  
    2.2.3 AgentX Communication Layers .......................................................................................... 20  
    2.2.4 AgentX Features ................................................................................................................... 20  
  2.3 JXTA .......................................................................................................................................... 21  
    2.3.1 JXTA Architecture and Component .................................................................................... 22  
    2.3.2 JXTA Protocols ..................................................................................................................... 23  

3 System introduction ......................................................................................................................... 25  
  3.1 J.AgentX Introduction .................................................................................................................... 25  
  3.2 J.AgentX Architecture ................................................................................................................... 26  
  3.3 Process ........................................................................................................................................ 27  
    3.3.1 Processing AgentX Administrative Messages .................................................................... 28  
      3.3.1.1 Open a session ................................................................................................................. 28  
      3.3.1.2 Closing a Session .......................................................................................................... 28  
      3.3.1.3 Registering Subtree ...................................................................................................... 28  
      3.3.1.4 Unregistering Subtree ................................................................................................... 29  
    3.3.2 Processing Received SNMP Protocol Messages ............................................................... 29  
      3.3.2.1 Get and Get-Next ......................................................................................................... 29  
      3.3.2.2 Set ................................................................................................................................. 31  

4 Project Design and Implementation .............................................................................................. 33
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Objective</td>
<td>33</td>
</tr>
<tr>
<td>4.2</td>
<td>Design Pattern</td>
<td>33</td>
</tr>
<tr>
<td>4.3</td>
<td>Scenarios</td>
<td>35</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Initializing the JXTA Service</td>
<td>35</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Closing the JXTA Service</td>
<td>36</td>
</tr>
<tr>
<td>4.3.3</td>
<td>Running the JXTA Service</td>
<td>37</td>
</tr>
<tr>
<td>4.3.4</td>
<td>Modifying the Result of Running the JXTA Service</td>
<td>37</td>
</tr>
<tr>
<td>4.3.5</td>
<td>Starting the Manager-Side Interface and the Subagent-Side Interface</td>
<td>38</td>
</tr>
<tr>
<td>4.4</td>
<td>Implementation</td>
<td>39</td>
</tr>
<tr>
<td>4.4.1</td>
<td>Manager-Side Application</td>
<td>40</td>
</tr>
<tr>
<td>4.4.2</td>
<td>Subagent Side</td>
<td>43</td>
</tr>
<tr>
<td>4.4.3</td>
<td>Master Agent Side</td>
<td>45</td>
</tr>
<tr>
<td>4.4.4</td>
<td>Additional Issue: the Procedure of Adding a New Functionality</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td>Managing a JXTA Service with J.AgentX</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Implementation Testing</td>
<td>46</td>
</tr>
<tr>
<td>5.1</td>
<td>Initializing the Manager, the Master Agent and the Subagent</td>
<td>46</td>
</tr>
<tr>
<td>5.2</td>
<td>Invoking the Discovery Service</td>
<td>48</td>
</tr>
<tr>
<td>5.3</td>
<td>Closing the Discovery Service</td>
<td>50</td>
</tr>
<tr>
<td>5.4</td>
<td>Modifying the Result of the JXTA Discovery Service</td>
<td>51</td>
</tr>
<tr>
<td>6</td>
<td>Encountered Problems</td>
<td>53</td>
</tr>
<tr>
<td>7</td>
<td>Summary</td>
<td>54</td>
</tr>
<tr>
<td>8</td>
<td>Reference</td>
<td>55</td>
</tr>
</tbody>
</table>
Figure List

Figure 1 Network Management System Architecture ................................................................. 10
Figure 2 MIB Tree ......................................................................................................................... 10
Figure 3 SNMP Message Format .................................................................................................. 12
Figure 4 SNMP Communication Layer [1] .................................................................................. 13
Figure 5 AgentX Architecture ...................................................................................................... 17
Figure 6 AgentX PDU ................................................................................................................... 18
Figure 7 JXTA Architecture [5] .................................................................................................. 22
Figure 8 J.AgentX Architecture [7] ............................................................................................. 26
Figure 9 GET/GETNEXT Process ............................................................................................... 30
Figure 10 Success Running JXTA Service ..................................................................................... 36
Figure 11 Closing the JXTA Service ............................................................................................ 36
Figure 12 Modify the JXTA Service's Result Successfully ............................................................ 37
Figure 13 the Process of Getting the Master Agent's IP Address ................................................. 38
Figure 14 Manager-Side GUI ....................................................................................................... 42
Figure 15 Master Agent's Starting Window .................................................................................. 47
Figure 16 Subagent's Status after Registration ......................................................................... 47
Figure 17 Master Agent's Status after Registration .................................................................... 47
Figure 18 Manager GUI's Status after Connecting the Master Agent ......................................... 48
Figure 19 Subagent Status in Get Process .................................................................................... 49
Figure 20 Master Agent's Status in Get Process ......................................................................... 49
Figure 21 Peer Info (displayed in the manager interface) ............................................................. 49
Figure 22 Closing Discovery Service .......................................................................................... 50
Figure 23 Subagent Status in Set Process .................................................................................... 51
Figure 24 Master Agent Status in Set Process ............................................................................. 51
Figure 25 Modified Value after the Set Process ......................................................................... 52
Abstract

While the network is being dramatically developed in the recent decades, the issues related to the network management have been becoming increasingly important. Simple Network Management Protocol offers the de facto standard for the network management. This paper introduces a manner in which a manager at the network management station can manages the Peer-to-Peer JXTA services with SNMP Agent Extensibility (AgentX) protocol.
1 Introduction

Simple Network Management Protocol (SNMP) is a communication protocol that became an industry standard for managing and controlling networks. With SNMP a network manager can execute applications that monitor and control network nodes, managed devices that contain the SNMP agents. An agent is a network-management software module that resides in a managed device and has local knowledge of management information and translates that information into a form compatible with SNMP. With the network development, more functionality is expected to be provided by agents.

SNMP Agent Extensibility (AgentX) protocol is the first IETF (Internet Engineering Task Force) standard-track specification for extensible SNMP agents. It provides a standard solution for the agent extensibility problem. Based on AgentX protocol the multiple functionalities can be developed at SNMP agents efficiently.

JXTA is a set of open, generalized peer-to-peer (P2P) protocols that allow any connected device on the network - from cell phone to PDA, from PC to server- to communicate and collaborate as peers. JXTA services are the common or desirable functionalities in the P2P environment, such as search, index and discovery.

This paper introduces a manner in which the JXTA services can be managed by a network manager under the SNMP AgentX environment. Section 2 gives the
background information about protocols, SNMP, AgentX and JXTA. Section 3 introduces J.AgentX, an implementation of AgentX protocol. Section 4 discusses the project design and the implementation. Section 5 demonstrates the implementation testing. Section 6 talks about the encountered problem in the project implementation. Section 7 is the summary.
2 Background

2.1 SNMP

2.1.1 Overview

SNMP is a standard protocol used in the network management. Since its creation in 1988 as a short-term solution to manage nodes, such as routers, printers and computers in the growing Internet and other attached networks, SNMP is flexible enough to describe many things that are required by the network management and achieves the wide acceptance in the industry. Because of these advantages, many network managers have believed that SNMP should be used for all network monitoring applications.

Simple Network Management Protocol (SNMP) is based on the manager/agent model of the network management system. Network management system contains two primary elements: a manager and agents, as shown Figure 1. SNMP messages are communicated between managers and agents in the process of accessing the managed objects. The Manager is the console through which the network administrator performs network management functions. Agents are the entities that interface to the actual device being managed that contain managed objects. These objects are arranged in a virtual information database, MIB (Management Information Base). Every managed device keeps its MIB which serves as a data dictionary or code book and is used to produce and interpret SNMP messages.
Figure 1 Network Management System Architecture

Figure 2 MIB Tree
The MIB is organized in a tree structure with individual variables, such as node status or node description, being represented as leaves on the branches. A long numeric tag or object identifier (OID) is used to represent each variable uniquely in the MIB and in SNMP messages. For example, as shown in the Figure 2 the Internet standard MIB is represented by the object identifier 1.3.6.1.2.1. It also can be expressed as iso.org.dod.internet.mgmt.mib.

2.1.2 SNMP Messages and Features

To achieve its goal of being simple, SNMP includes a limited set of management commands and responses. SNMP uses five basic messages (GET, GET-NEXT, GET-RESPONSE, SET, and TRAP) (SNMP version 1) to communicate between the manager and the agent. The management system issues Get, GetNext and Set messages to retrieve the value of single or multiple object variables or to establish the value of a single variable. The managed agent sends a Response message to complete the Get, GetNext or Set. The TRAP message allows the agent to spontaneously inform the manager of an 'important' event, such as threshold that exceeds a predetermined value and failed authentication. Because the TRAP message is the only message capable of being initiated by an agent, it can notify the SNMP manager as soon as an alarm condition occurs, instead of waiting for the SNMP manager to ask.

The other simplifying factor of SNMP is built on a connectionless communication link. It is considered to be robust because of the independence of the managers
from the agents, e.g. if an agent fails, the manager will continue to function, or vice versa.

2.1.3 SNMP Packet Format

SNMP is packet oriented and the SNMP v1 packets (Protocol Data Units or PDUs) used in the communication between the manager and the agent. The variable bindings as a field of PDU are transferred during the communication. Each variable binding contains an identifier, a type and a value (if a Set or Response). The agent checks each identifier against its MIB to determine whether the object is managed and writeable (if processing a Set). The manager uses its MIB to display the readable name of the variable and sometimes interpret its value. The organization of the SNMP message is as shown in the Figure 3.

<table>
<thead>
<tr>
<th>Variable bindings</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SNMP PDU</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDU TYPE</td>
</tr>
<tr>
<td>REQUEST ID</td>
</tr>
<tr>
<td>ERROR STATUS</td>
</tr>
<tr>
<td>ERROR INDEX</td>
</tr>
<tr>
<td>VARIABLE BINDINGS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SNMP MESSAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>VERSION</td>
</tr>
<tr>
<td>COMMUNITY</td>
</tr>
<tr>
<td>SNMP PDU</td>
</tr>
</tbody>
</table>

Figure 3 SNMP Message Format

The following descriptions summarize the fields illustrated in the Figure 3.

- PDU type—specifies the type of PDU transmitted.
- Request ID—associates SNMP requests with responses.
- Error status—indicates one of a number of errors and error types. Only the response operation sets this field. Other operations set this field to zero.
- Error index—associates an error with a particular object instance. Only the response operation sets this field. Other operations set this field to zero.
- Variable bindings—serve as the data field of the SNMPv1 PDU. Each variable binding associates a particular object instance with its current value (with the exception of Get and GetNext requests, for which the value is ignored).

2.1.4 SNMP Communication Layers

SNMP message is wrapped in the User Datagram Protocol (UDP), which in turn is wrapped in the Internet Protocol (IP). These are based on a four-layer model developed by the Department of Defense, USA.

![Figure 4  SNMP Communication Layer](image)
SNMP resides in what is called the Application layer, UDP resides in the Transport layer and IP resides in the Internet layer. The fourth layer is the Network Interface layer where the assembled packet is actually interfaced to some kind of transport media. Figure 4 illustrates the organization of layers and the path how the message traverses layers.

2.1.5 SNMP Versions

There are three SNMP versions, SNMPv1, SNMPv2 and SNMPv3. As mentioned above, the operators defined by SNMPv1 are get, get-next, get-response, set-request, and trap. SNMPv1 describes the encapsulation of SNMPv1 PDUs in SNMP messages between SNMP entities and distinguishes between application entities and protocol entities, introduces the concept of an authentication service supporting one or more authentication schemes and access control based on a concept called an SNMP MIB view which controls the access level of each managed object, such as read only or read\write. However SNMPv1’s authentication scheme is based on community strings. It is a known fundamental weakness in the network management.

SNMPv2 provides several advantages over SNMPv1, such as expanded data types, improved efficiency and performance (adding Get-Bulk command), richer error handling and improved sets (row creation and deletion). However, SNMPv2 doesn’t meet the goals to provide the provision of security and administration delivering so-called "commercial grade" security with authentication, privacy,
authorization and access control, suitable remote configuration and
administration capabilities for these features.

SNMPv3 is derived from and builds upon both SNMPv1 and SNMPv2. It
addresses the deficiencies in SNMPv2 relating to security and administration.
The new features of SNMPv3 include authentication and privacy, authorization
and access control, naming of entities, people and policies, usernames and key
management, notification destinations, proxy relationships and remotely
configurable via SNMP operations. [9]

2.2 AGENTX

AgentX is the first IETF standard-track specification for extensible SNMP agents.
It overcomes a very real need to dynamically extend the managed objects in a
managed device on the network.

Before AgentX is published, users have to either use non-standard solutions or to
run multiple SNMP agents on different UDP ports, probably using proxies to
access them via a standard UDP port. Both approaches are not ideal solutions.
With lack of a standard the implementers may have to support several different
subagent environments (APIs) in order to support different target platforms. It can
also become quite cumbersome to configure subagents and the master agent on
a particular managed node. Proxies are not transparent for the management
station and thus require manager more intelligence work. In addition, running the
full-featured SNMP agents on a device usually costs more resources.
The AgentX protocol provides a standard solution to the problems in developing extensible SNMP agents. Master agents and subagents can be independently developed upon the AgentX protocol so that they are able to interoperate at the protocol level. Vendors can continue to differentiate their products in all other respects.

2.2.1 AgentX Architecture and Component Responsibilities

Within the AgentX framework, an agent is defined to consist of a master agent and zero or more processing entities called subagents.

The responsibilities of a master agent include accepting AgentX session establishment requests from subagents, accepting registration of MIB regions by subagents, sending and accepting SNMP protocol messages on the agent's specified transport addresses, providing instrumentation for the MIB objects defined in RFC 1907 and for any MIB objects relevant to any administrative framework it supports, sending and receiving AgentX protocol messages to access management information based on the current registry of MIB regions and forwarding notifications on behalf of subagents.

The responsibilities of a subagent include initiating AgentX sessions and registering MIB regions with the master agent, instantiating managed objects, binding OIDs within its registered MIB regions to actual variables, performing management operations on variables and initiating notifications. [3]
The AgentX MIB allows managers to identify the number of subagent sessions that are open with the master agent and allows the master agent to identify the MIB regions or MIB objects that a subagent implements. Each subagent registers a MIB subtree with the master agent. Requests for objects residing in a registered MIB subtree are passed from the master to the subagent agent by using the AgentX protocol. The subagent passes the results of an SNMP query back to the master agent by using the AgentX protocol.

![AgentX Architecture](image)

*Figure 5 AgentX Architecture*

*Error! Reference source not found.* illustrates the process when an extensible SNMP agent receives a SNMP message. When the AgentX master agent receives a request from a manager at the network management station via SNMP, it finds the subagent(s) responsible for the requested MIB region and dispatches appropriate AgentX requests to the subagent(s) via AgentX. Based
on the receiving message from the master agent, the subagent looks up its MIB, perform management operations and send response to the master agent via AgentX. After receiving the response from the subagent, the master agent forwards the response to the manager via SNMP.

2.2.2 PDU Description

Each AgentX PDU includes three main parts, header, context and searchrange. Figure 6 illustrates each field of AgentX PDU.

<table>
<thead>
<tr>
<th>VERSION</th>
<th>TYPE</th>
<th>FLAGS</th>
<th>RESERVED</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SESSIONID</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRANSACTION ID</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PACKET ID</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PAYLOAD LENGTH</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CONTEXT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(OPTIONAL)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OBJECT 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>START OF RANGE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OBJECT 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>END OF RANGE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OBJECT N</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>START OF RANGE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OBJECT N</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>END OF RANGE</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Figure 6 AgentX PDU*
The header contains 20 bytes. The first 32 bits of the header contain the protocol version number (currently 1), the PDU type, an 8 bit flags field and an 8 bit reserved field. Only first five bits of the flag field are currently in use and among them Bit 3 represents NON_DEFAULT_CONTEXT. If this bit is set the recipient is reminded that an octet string (Context field in the Figure 6) follows immediately after the header. Bit 4 of the flag field represents NETWORK_BYTE_ORDER and is used by the subagent to communicate its platforms' native byte ordering to the master agent.

The remaining 16 bytes are organized into four 32-bit words. These fields are sessionID, transactionID, and packetID and payload length. The sessionID identifies a session created by the subagent. The transactionID refers to the SNMP request that this AgentX PDU is servicing. The packetID is used to pair AgentX request and response PDUs. The payload length is simply the length of the remainder of the PDU, in bytes.

The Context part is optional. As mentioned above, it is only included within AgentX PDUs when the NON_DEFAULT_CONTEXT bit is set. An optional context field may be present in the agentx-Register-, UnRegister-, AddAgentCaps-, RemoveAgentCaps-, Get-, GetNext-, IndexAllocate-, IndexDeallocate-, Notify-, TestSet-, and Ping- PDUs.

A SearchRange consists of two Object Identifiers. In its communication with a subagent, the master agent uses a SearchRange to identify a requested variable.
binding. The first Object Identifier in a SearchRange indicates the beginning of the range. The second object identifier indicates the end of the range.

2.2.3 AgentX Communication Layers

Master agent and all subagents can reside on the same host. In such cases AgentX is more a form of inter-process communication than a traditional communications protocol.

The master agent and the subagent can be located in the different physical devices too. In such cases, the AgentX message is wrapped through the same processing steps as the SNMP message. AgentX message can be wrapped in the User Datagram Protocol (UDP) or TCP, which in turn is wrapped in the Internet Protocol (IP).

AgentX resides in the Application layer, UDP or TCP resides in the Transport layer and IP resides in the Internet layer. The fourth layer is the Network Interface layer where the assembled packet is actually interfaced to some kind of transport media.

2.2.4 AgentX Features

The labor of AgentX is divided rationally between the master agent and the subagent. The master agent is concerned with SNMP protocol operations and the translations to and from AgentX protocol operations needed to carry them out but typically has little or no direct access to management information; subagents,
which are "shielded" from the SNMP protocol messages processed by the master agent, are concerned with management instrumentation.

The internal operations of AgentX are invisible to an SNMP entity operating in a manager role. From a manager's point of view, an extensible agent behaves exactly as would a non-extensible (monolithic) agent that has the access to the same management instrumentation.

2.3 JXTA

JXTA is a set of open, generalized peer-to-peer (P2P) protocols that allow any connected device on the network to communicate and collaborate as peers. It provides a platform with the basic P2P network's functions, such as discovery of peers, searching, and file or data transfer.

The JXTA protocols are designed for developing P2P applications and to be independent of programming languages and transport protocols. They can be implemented with the different languages and can be implemented on top of the different transport protocols. They standardize the manner in which peers:

- Discover each other
- Self-organize into peer groups
- Advertise and discover network services
- Communicate with each other
- Monitor each other
In addition, JXTA technology is designed to be accessible by any device connected on the network, not just for PCs. [5]

2.3.1 JXTA Architecture and Component

![JXTA Architecture Diagram]

The Project JXTA software architecture is divided into three layers, as shown in Figure 7. The core layer provides the elements that are absolutely essential to every P2P solution. The services layer provides network services that are desirable but not necessarily a part of every P2P solution. The applications layer builds on the capabilities of the services layer to provide the common P2P application, such as instant messaging.
The main components of JXTA consist of peers, peer groups, advertisements, pipes and messages. Peers are a series of interconnected nodes on the network and can self-organize into peer groups, which provide a common set of services. Advertisements are XML documents used by JXTA peers to advertise their services. The JXTA peers use pipes to send messages to one another. Pipes are bound to specific endpoints, such as a TCP port and associated IP address. Messages are simple XML documents whose envelope contains routing, digest and credential information. [5]

2.3.2 JXTA Protocols

JXTA consists of six JXTA protocols which are used by JXTA peers for fulfilling the different JXTA functionalities.

- **Peer Discovery Protocol (PDP)** — used by peers to advertise their own resources and discover resources from other peers.
- **Peer Information Protocol (PIP)** — used by peers to obtain status information from other peers.
- **Peer Resolver Protocol (PRP)** — enables peers to send a generic query to one or more peers and receive a response (or multiple responses) to the query.
- **Pipe Binding Protocol (PBP)** — used by peers to establish a virtual communication channel, or pipe, between one or more peers.
- **Endpoint Routing Protocol (ERP)** — used by peers to find routes (paths) to destination ports on other peers.
• *Rendezvous Protocol (RVP)* — used by simple peers to connect to rendezvous peers and rendezvous peers propagate messages to other peers on behalf the simple peers. [5]
3 System introduction

3.1 J.AgentX Introduction

J.AgentX, a Java-based toolkit for dynamic extension of SNMP agents, is developed by the Department Engineering Information, University of Coimbra, Portugal. This toolkit is developed based on SNMPv1 and provides an easy way to dynamically supply SNMP-based interfaces to new management services.

J.AgentX is the first Java-based implementation of the AgentX standard and fully compliant with the AgentX standard. It is a lightweight toolkit that is fully dedicated to agent extension and designed to fulfill the following features for providing SNMP-compliant network management services.

- Simplicity: as simple to use as possible.
- Portability: 100% Java-based and system independent.
- Interoperability: provide full interoperability with other AgentX implementations.
- Flexibility: able to adjust to different environments and different application fields. This means, for instance, the possibility of mapping AgentX into new communication protocols and the possibility of maintaining simultaneous support for several protocols.
• Easy service development: technical details of SNMP and AgentX are truly hidden below high-level programming APIs, focusing the programmer's attention on the services being developed. [7]

3.2 J.AgentX Architecture

J.AgentX is instituted by a set of modules which are divided into two groups, master agent and subagent, as shown in Figure 8. Each module is implemented to achieve the different functionalities based on AgentX protocol.

![J.AgentX Architecture](image)

*Figure 8 J.AgentX Architecture [7]*
Master Agent Side:

- SNMP Engine module: is responsible for the communication with the Manager at the network management station, receiving SNMP request message from the manager and sending the response to the manager.
- Parse/Forward module: is responsible to receive and parse requests from the SNMP Engine and forward parsed requests to the AgentX Engine
- Get AgentX module: is responsible to receive and parse responses from the AgentX Engine and forward responses to the SNMP Engine.
- AgentX Engine: is responsible to communicate with the subagents.

Subagent Side:

- Subagent: is responsible to initialize the registration and invoke the management application based on the request message from master agent and send the response to the master agent.

3.3 Process

The main processes of J.AgentX toolkit can be broadly categorized into two groups:

1. Processing AgentX administrative messages (e.g., connection requests from a subagent to a master agent); and
2. Processing SNMP messages (the coordinated actions of a master agent and one or more subagents in processing, for example, a received SNMP GetRequest-PDU).
3.3.1 Processing AgentX Administrative Messages

3.3.1.1 Open a session

An agentx-Open-PDU is created by a subagent and sent to a master agent. If the master agent is unable to open an AgentX session for any reason, it may refuse the session establishment request, sending the agentx-Response-PDU, which indicates that the open action is failed, to the subagent. Otherwise, the master agent assigns a sessionID, which should be unique among all existing open sessions, to the new session and sends agentx-Response-PDU, which contains new sessionID, to the subagent.

3.3.1.2 Closing a Session

When the master agent receives an agentx-Close-PDU, if sessionID field in the PDU does not correspond to a currently established session with this subagent, the agentx-Response-PDU, which indicates that session is not open, is sent in reply. Otherwise, the master agent closes the AgentX session. The action includes that the session row is removed; the connection row is removed if no other session uses this connection and the registrations are cleaned. No error response is sent to the subagent.

3.3.1.3 Registering Subtree

When the master agent receives an agentx-Register-PDU, if sessionID field in the PDU does not correspond to a currently established session with this subagent, the agentx-Response-PDU, which indicates that session is not open, is sent in reply. Otherwise, the region is to be logically registered. If this is a
duplicate registration that would result in a duplicate region with the same priority and within the same context as that of a current registration, the agentx-Response-PDU is returned to indicate that duplicate registration occurs, and the requested registration fails. Otherwise, the no Error agentx-Response-PDU is returned.

3.3.1.4 Unregistering Subtree

When the master agent receives an agentx-Unregister-PDU, if the sessionID, the region, the priority, and the indicated context do not match an existing registration made during this session, the agentx-Response-PDU which indicate error occurs is returned. Otherwise, the no-error agentx-Response-PDU is sent in reply and the previous registration is removed.

3.3.2 Processing Received SNMP Protocol Messages

3.3.2.1 Get and Get-Next

The AgentX Get and GetNext PDUs are the primary means of servicing the SNMP requests of the same name. Based on the receiving SNMP request the master agent find appropriate registered subtree and issues one or more AgentX PDUs and generates a unique transaction ID to identify the SNMP request. Each AgentX PDU is dispatched to the subagent who registered the subtree selected.

Figure 9 illustrates the process when a Get or GetNext SNMP message is received by J.AgentX.
When a Get or Get-Next SNMP message is received by SNMP Engine, a transaction based on the SNMP message is created and conveyed to Parse/Forward module.

*Figure 9 GET/GETNEXT Process*

Parse/Forward module is responsible to split the request and dispatch each managed object request to the corresponding subagent if necessary. It checks whether the requested managed objects are contained in its master MIB. If yes, the SNMP response with the value of managed object is sent back to the manager. If no, Parse/Forward module checks whether the requested managed objects are contained within the currently registered subtrees. If no, no-such-object SNMP message is replied. If yes, the subagents responsible to the requested managed objects are found, AgentX-Get-PDUs or AgentX-GetNext-PDUs are created and sent to the subagents through AgentX Engine.
When the requested message is received, the subagent looks up its MIB. If management application is needed to be invoked, the application is invoked, the value of required managed object is updated and the updated value is retrieved to be sent back to the AgentX Engine of the master agent. If no management application is needed to be invoked, the value of the required managed object is retrieved from its MIB directly and is sent back to AgentX Engine. When any exception occurs in the process, an error message is sent back to the AgentX Engine.

When the response from the subagent is received by the AgentX Engine, it is forwarded to the Get AgentX module. A SNMP Response is created by the Get AgentX module is sent back to the manager through SNMP Engine.

3.3.2.2 Set

J.AgentX implements SNMP Sets command through the use of the AgentX TestSet, CommitSet, UndoSet and CleanupSet PDUs. The two-phase commits is designed to provide atomic SNMP Sets across multiple subagents with four types of PDU.

At the first stage of SNMP Set processing the master agent sends one or more AgentX TestSet PDUs to the subagents specified (Manager -> SNMP Engine -> Parse/Forward ->AgentX Engine ->Subagent). The subagents extract the values and test whether the Set operation would succeed. Since a single TestSet PDU
may include several variables to set, each one must be checked for validity. If all variables may be set, then a response is sent back to the master that the processing for the SNMP Set operation may proceed (Subagent->AgentX Engine -> Get Response). If one of the variables cannot be set, then an error is returned to the master indicating the offending variable.

The next stage of SNMP Set processing depends on whether all TestSet PDUs were successful. If all TestSet requests were successful then the master agents send a set of CommitSet PDUs, which informs each subagent that they should actually commit the changes to memory(Get Response -> AgentX Engine -> Subagent -> AgentX Engine ->Get Response). CommitSet operations should almost always succeed. However, there is still a chance that they may fail. In the event that a CommitSet operation fails, the master agent should send an UndoSet PDU to undo the changes made by the previous CommitSet operation. If a TestSet fails or a CommitSet succeeds, the master agent follows up with a CleanupSet PDU (Get Response -> Subagent) indicating the end of SNMP Set processing and response is sent back to the manager (Get Response -> SNMP Engine -> Manager).
4 Project Design and Implementation

4.1 Objective

The objective of this project is a JXTA service located at a managed device on the network can be managed by a manager at the network management station. The management includes features as follows:

- A manager can issue a message requiring the client to start the JXTA service.
- A client, the owner of the managed device, has the authority deciding whether to accept starting the JXTA service or not.
- If the client agrees to start the JXTA service, a manager can totally control the JXTA services invoked.
- A manager can modify the result of the JXTA service if necessary.

4.2 Design Pattern

The master agent/subagent architecture model is selected to achieve that the agent-side JXTA service is managed by a manager at manager-side and SNMP AgentX protocol is used in the communication between the manager and the agent. The Master Agent acts as the primary interface between the network manager and the subagents. The request from a manager, such as starting the JXTA service, running the JXTA service, stopping the JXTA service and modifying the result of the JXTA service, are wrapped into a SNMP GET or SET command and put into the SNMP request message. The request message is parsed by the master agent. If necessary, the SNMP requests are broken into
multiple requests by the master agent based on the manageability provided by each subagent and the requests are distributed to the appropriate subagents.

Subagents are responsible to manage their MIBs and the management applications that operate the JXTA services resided at subagents. When a request running the JXTA service is received from the master agent, the JXTA service is invoked and its MIB is updated based on the result of running the JXTA service. Then the response is retrieved from its MIB and sent back to the master agent.

After collecting all the responses from each subagent, the master agent wraps the final response into the SNMP Get-Response and sends it to the network manager on behalf of subagents.

There are two main benefits to select the SNMP AgentX protocol in the project. One is that SNMP protocol has been widely accepted in the network management field. It has already proved being robust and simple to be used in the network management activities. In addition, many SNMP protocol implementations have been well developed by vendors and organizations. So the main work of the project can be built upon one of existing SNMP AgentX implementations and be focused on developing the functionalities of sending request message and receiving response message at the manager side and functionalities of managing the JXTA services at agent side.
The other benefit is that only the master agent interacts with the manager at the network management station. The master agent and its subagents are viewed as a monolithic entity by the manager. The manager is not required understanding how to activate a subagent. It reduces the intelligent work of a manager.

4.3 Scenarios

There are five main scenarios in the process of managing the JXTA services.

4.3.1 Initializing the JXTA Service

When a manager at the manager side sends, a running JXTA-service request wrapped into the SNMP GET command, to the master agent at the agent side, the master agent forwards the request to the appropriate subagent at the agent side. The subagent notifies the client making a decision whether to start the JXTA service or not. If not, a response, which indicates that the request running JXTA service is rejected by the client, is forwarded to the master agent by the subagent. The master agent wraps the rejecting response into SNMP GET-RESPONSE and sends it back to the manager. If yes, the subagent is responsible to initialize the JXTA service and the result of running the JXTA is forwarded to the master agent. The master agent wraps the result into SNMP GET-RESPONSE and sends it back to the manager. Figure 10 illustrates the Starting-JXTA-Service request is executed successfully.
4.3.2 Closing the JXTA Service

When the manager at the manager side sends, a closing JXTA-service request wrapped into the SNMP GET command, to the master agent at client side, the master agent forwards the request to the appropriate subagent at the client side. If the JXTA service has been already started, the subagent stops the JXTA service and forward closing response to the master agent, the master agent wraps the response into the SNMP GET-RESPONSE message and sends the message back to the manager. If the JXTA service is not started, a notify response is replied. Figure 11 illustrates the Closing-JXTA-Service request is executed when the JXTA service has been started.
4.3.3 Running the JXTA Service

This scenario is almost same as the scenario of initializing the JXTA service. The only difference between two scenarios is that the client is not involved in the process of running the JXTA service which has already been started. The manager can invoke the JXTA service directly.

4.3.4 Modifying the Result of Running the JXTA Service

Because the result of running the JXTA service is stored in the subagent’s MIB, it can be modified by the manager with issuing SNMP Set command. When a manager at the manager side wraps the message which requires modifying the result of the JXTA service into the SNMP SET command and sends it to the

*Figure 12 Modify the JXTA Service’s Result Successfully*
agent at the client side. The process at agent side is divided into two stages and TestSet, CommitSet, UndoSet and CleanupSet PDUs are used to execute the SNMP set command. Figure 12 Illustrates the request of modifying the result of running the JXTA service is executed successfully.

### 4.3.5 Starting the Manager-Side Interface and the Subagent-Side Interface

When the manager-side interface and the subagent-side interface are initialized, the IP address of the master agent is automatically obtained by two interfaces. The process is as shown in Figure 13.

![Figure 13 the Process of Getting the Master Agent's IP Address](image-url)
4.4 Implementation

J.AgentX toolkit is selected as the platform of the project. As mentioned in the above design section, since the J.AgentX is one of well developed SNMP AgentX protocol implementations, the project’s implementation built upon J.AgentX toolkit is focused on three parts:

1. Application at the manager side: responsible to set up a managed object list which contains the managed objects; send the SNMP requests and receive SNMP response; automatically obtain the IP address of the master agent by using the JXTA group service.

2. Application at the subagent side: responsible to start a session and register a subtree with the master agent; set up and manipulate a subagent MIB and manage the JXTA service; automatically obtain the IP address of the master agent by using the JXTA group service.

3. Application at the master agent side: responsible to publish an advertisement including its IP addresses.

The main purpose of the project is to express how the JXTA service can be managed with SNMP AgentX protocol. The implementation of the project is concentrated on managing one of the primary JXTA services, Discovery Service, whose functionality is sending discovery query in a JXTA group and catch the responses returned from the other group peers. The response includes the peer name, the peer ID and peer description of a replying peer. Same as the discovery service, other JXTA services can be managed by the manager with
The procedure of managing a JXTA service by a manager with J.AgentX will be introduced at the end of this section. Prior to it, the implementation of project is introduced.

### 4.4.1 Manager-Side Application

Four main classes, `Manager_TaskGUI`, `Manager_Engine`, `ManagedObject` and `ManagedObjectList`, are implemented at the manager side.

1. **ManagedObject**:

   The instances of the `ManagedObject` class are the elements in the `ManagedObjectList`. Four instance variables in this class are as follows:

   - `vb`: a Varbind object which includes the type of the value of a managed object, the object ID and the value of the managed object.
   - `name`: a string variable used to describe the managed object.
   - `isGet`: a boolean variable used to indicate whether the managed object is readable
   - `isSet`: a boolean variable used to indicate whether the managed object is writeable.

2. **ManagedObjectList**:

   A new vector is created by the `ManagedObjectList`. All the managed objects are added into this vector at the beginning of running the manager-side application.
3. **ManagerEngine**

This class is responsible to create communication interface for sending and receiving the SNMP message, to create UDP data which wraps SNMP request message, to parse UDP data which wraps SNMP GET-Response message. A new thread is created by **ManagerEngine** which is responsible to catch the SNMP response from agents. The main instance methods in this class are as follows:

- **startEngine**: to load **ManagedObjectList** which contains all managed objects, to initiate a thread to catch the response from agents.
- **createUDP**: based on the type of a SNMP command, GET, GETNEXT and SET requests are wrapped into the appropriate UDP data.
- **run**: responsible to catch and parse the response from agents, update the message content in the manager side interface with the parsed response.
- **send**: responsible to send request to agents.

4. **ManagerTaskGUI**

This class implements an interface between a manager and the manager-side application. A manager can issue SNMP requests through the interface and the parsed SNMP response can be display in the message
area of the interface. In addition the IP address of the master agent can be obtained automatically by its instance method, `receiveMasterAddress`. The components of the interface are as shown in Figure 14:

- **Host field:** The IP address of the master agent is filled in this field.
- **Port field:** The port number (Port 161 is assigned to sending and receiving SNMP in default) used to listen SNMP request message by the master agent is filled in this field.
- **OID box:** Each element in this Combo Box represents the object identifier of a managed object.

![Manager-Side GUI](image)

*Figure 14 Manager-Side GUI*

- **Value field:** The content in this field is used as the set value of the managed object in the SNMP SET command.
• **Get button**: used to invoke the SNMP GET command.
• **Get_Next button**: used to invoke the SNMP GET_NEXT command.
• **Set button**: used to invoke the SNMP SET command.
• **Clear button**: used to clean the message area and value field.
• **Message area**: used to display the parsed SNMP response.
• **Quit Menu Item**: used to exit the manager-side application.
• **Help Menu Item**: used to help user to manipulate the interface.

### 4.4.2 Subagent Side

Two classes, *MySubagent* and *MyDiscoveryService*, are implemented at the subagent side.

1. **MyDiscoveryService**:

This class is implemented to manage the JXTA Discovery Service, to send a discovery query in the JXTA group and to receive discovery response by implementing DiscoveryListener interface. A new thread is created in this class to continually send the discovery query and some methods are used to stop the thread when the subagent receives the request closing the discovery service. The main instance methods in this class are as follows:

- **MyDiscoveryService**: the constructor of the class in which a standard dialogue is popped out. The client can make decision whether to accept starting the JXTA service request or not.
• \textit{createJXTA}: instantiate the JXTA group and get the discovery service from the group.

• \textit{startJXTAService}: the new thread is created and started in this method.

• \textit{run}: start the discovery listener and continually send the discovery query message.

• \textit{discoveryEvent}: responsible to catch discovery response from the other peers in the group and find the IP address of the master agent.

• \textit{closeService}: responsible to stop sending discovery query and stop the discovery listener.

2. \textit{MySubagent}:

This subagent opens a session with the master agent, registers a subtree and adds two varbinds to its MIB. One is used to invoke the JXTA service and the other is used to stop the JXTA service. The main methods in this class are as follows:

• \textit{getPeerInfo}: used to invoke the JXTA discovery service and result is stored into the MIB.

• \textit{setPeerInfo}: used to set the value of the managed object in the MIB.

• \textit{closeDiscovery}: used to close the discovery service.
• main: is responsible to open a session, register a subtree and add varbinds to the MIB bind three above methods to the appropriated varbinds and get the IP address of the master agent.

### 4.4.3 Master Agent Side

The `SNMPAgentXd` class is implemented at the master agent side. This class is responsible to instantiate the master agent and continually publishes the peer advertisement in the JXTA group. The description field of the advertisement contains the IP address of the master agent. So other peers in the group, including the subagents and the manager at the network management station, can get its IP address through the group discovery service.

### 4.4.4 Additional Issue: the Procedure of Adding a New Functionality

**Managing a JXTA Service with J.AgentX**

J.AgentX provides a manner to invoke a management application through the SNMP GET or SET request. If a new JXTA service is required to be managed by the manager at the network management station, a new varbind object should be added into both the ManageObjectList at the manager side and the subagent MIB. One of two methods in the Subagent class of J.AgentX API, `assignMethodGetToVarbind` and `assignMethodSetToVarbind`, is used to bind the JXTA service to the new varbind in the subagent MIB. The selection of the method depends on the manner in which the JXTA service is invoked by using either the SNMP Get command or the SNMP Set command. After the JXTA service is bound to the varbind, the manager at the network station is able to
manage the JXTA service through issuing the SNMP request to this managed object. When the subagent receives the SNMP request to the managed object in its MIB, the subagent invokes the JXTA service bound to this managed object.

5 Implementation Testing

5.1 Initializing the Manager, the Master Agent and the Subagent
The manager, the master agent and the subagent are instantiated in three different machines. The instantiating order is the master agent, the subagent and the manager.

*Figure 15* illustrates that after the master agent is instantiated, the instance of the master agent loads the configure file, publishes the peer advertisement continually in the peer group and wait the subagents and the manager to connect to it.

After the instantiation, the subagent uses the address gotten through the JXTA discovery service to open a session and register the subtree with the master agent. *Figure 16* and *Figure 17* illustrate that the master agent and the subagent are ready to process the requests from the manager after the registration action is successfully executed.

After the instantiation, a user graph interface is popped out at the manager's machine. The master agent's IP address is shown on the host field of the
interface in Figure 18. It means that a master agent at this address is ready to receive the SNMP message.

Figure 15 Master Agent's Starting Window

Figure 16 Subagent's Status after Registration

Figure 17 Master Agent's Status after Registration
5.2 Invoking the Discovery Service

When the item “1.3.6.1.4.1.1331.11.2.1.0” on the OID field of the manager interface is selected and Get Button is clicked, the request to invoke the JXTA discovery service is wrapped into the SNMP Get Request to the managed object whose object identifier is “1.3.6.1.4.1.1331.11.2.1.0”. The request is sent to the master agent.

A dialogue window is popped out at the subagent machine. The client makes the decision whether to accept the request or not.

If the client accepts the request, the JXTA discovery service is executed and the status of the subagent and the master agent in the Get process are shown in

Figure 18 Manager GUI's Status after Connecting the Master Agent
**Figure 19 Subagent Status in Get Process**

```
Sending a Discovery Message
Get a Discovery Response [1 elements] from peer: unknown
Peer name = jxta
Peer ID = urn:jxta:uuid-596162616462614672746150325833CC56789CE912854996DA
ECB6997791403
Description = 134.117.28.62
```

**Figure 20 Master Agent's Status in Get Process**

```
Publishing an Advertisement of IP address
- SNMP Get Request -
Publishing an Advertisement of IP address
Publishing an Advertisement of IP address
Publishing an Advertisement of IP address
- Response arrived -
Response: From SubAgent's Session Id #1
SNMP: Sending GET_RESPONSE(V1, C:public, RID:1, ES:NO_ERROR, EI:0, V2List:Oid:1.3.6.1.4.1.1331.11.2.1.0, OctetString: Peer name = BART
Peer ID = urn:jxta:uuid-596162616462614672746150325833CC56789CE912854996DA
```

**Figure 21 Peer Info (displayed in the manager interface)**

```
SNMP_Agents

Device address: 134.117.28.62
Device port: 161

OID: 1.3.6.1.4.1.1331.11.2.1.0

Responses:
134.117.28.62
StartJxta Discovery Service
1.3.6.1.4.1.1331.11.2.1.0
Peer name = BART
Peer ID = urn:jxta:uuid-596162616462614672746150325833CC56789CE912854996DA
```

49
Figure 19 and Figure 20. Figure 21 illustrates that the peer information is displayed in the manager interface after running the JXTA discovery service.

After the JXTA service is initialized to operate, when the Get button is clicked again, the dialogue window at the subagent machine is not popped out and the JXTA service is directly controlled by the manager.

5.3 Closing the Discovery Service
When the item “1.3.6.1.4.1.1331.11.2.2.0” on the OID field of the manager interface is selected and Get Button is clicked, the discovery service is stopped. The closing indication is displayed in the manager interface as shown in Figure 22. If the manager want to execute the discovery service again, the permission of the client is required again.

![Figure 22 Closing Discovery Service](image-url)
5.4 Modifying the Result of the JXTA Discovery Service

When the item “1.3.6.1.4.1.1331.11.2.1.0” on the OID field of the manager interface is selected and Set Button is clicked, the request to modify the result of running the JXTA discovery service is wrapped into the SNMP Set Request to the managed object whose object identifier is “1.3.6.1.4.1.1331.11.2.1.0”. The request is sent to the master agent. The Set value is the string on the Set Value field of the manager interface.

The statuses of the master agent and subagent in the Set process are shown in Figure 23 and Figure 24. Figure 25 displays the modified value returned by the agent.

![Figure 23 Subagent Status in Set Process](image)

![Figure 24 Master Agent Status in Set Process](image)
Figure 25  Modified Value after the Set Process
6 Encountered Problems

The implementation of project is developed on top of the J.AgentX API and the JXTA API. During the implementation two problems related to the J.AgentX API and JXTA API are encountered. The first encountered problem comes across when implementing the closing the discovery service. One of the predefining features of the project is that the discovery service can be stopped by the manager immediately after the closing request is sent. But the transaction in the J.AgentX toolkit is designed to be carried out atomically, the previous request process can not be interrupted by the closing request.

The second problem is encountered when implementing the feature of automatically finding the IP address of master agent by using the JXTA group discovery service. If the subagent and master agent are located at the same machine, the invocation of the built-in method in the JXTA library used to initializing the JXTA platform triggers the port binding exception. The reason is that if two JXTA platforms are run at the same machine, one port is bound by two applications so that the port binding confliction occurs.
7 Summary

The SNMP protocol provides a simple method to meet the basic management requirement on the network. The AgentX protocol provides a standard solution to the problems in developing extensible SNMP agents so that the agents can be developed more efficiently and more functionality of agents can be expected. The JXTA protocols standardize the manner in the development of primary P2P functionalities. Those three technologies can be applied together not only to increase the range of network management system but also to utilize the network resource more reasonably so that the network can be managed more efficiently.
8 Reference


3. **RFC 2257 - Agent Extensibility (AgentX) Protocol Version 1**
   http://www.faqs.org/rfcs/rfc2257.html

4. *An Overview of the AgentX Protocol*
   http://www.simple-times.org/pub/simple-times/issues/6-1.html

5. **Project JXTA v2.0: Java™ Programmer’s Guide**
   http://www.jxta.org/docs/JxtaProgGuide_v2.pdf


10. API downloads address
    - **J.Agentx**: http://eden.dei.uc.pt/agentx/
    - **JXTA**: http://www.jxta.org/