UPPSALA
UNIVERSITET

IT 14 038
Examensarbete 15 hp
Juni 2014

Implementing A Network Monitoring Feature In A Multipurpose Device Control Application

Joseph Lundström

Institutionen för informationsteknologi
Department of Information Technology

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Implementing A Network Monitoring Feature In A Multipurpose Device Control Application

Joseph Lundström

As the number of network connected devices grow, the need for monitoring and managing these devices increase as well. This paper looks at different methods of network monitoring, and which one is the most suitable for a network-enabled control system. The research compares the strengths and weaknesses of primarily CORBA, NetConf and SNMP. Relevant research will be analyzed to determine which protocol is the most extensible, most manageable and which one is the best performance-wise. The paper shows that SNMP is faster than NetConf in most cases when the number of monitored objects are less than 500. It also shows that SNMP uses less traffic than CORBA when a single object is retrieved. The paper concludes that SNMP is the most suitable for a network-enabled control system and presents one possible implementation of this.

Handledare: Martin Ansrud
Ämnesgranskare: Arnold Pears
Examinator: Olle Gällmo
IT 14 038
Tryckt av: Reprocentralen ITC

Contents

1 Introduction

Monitoring a network can be a daunting task. A typical business network can be expected to have hundreds, if not thousands, of network-enabled device. It is not uncommon that the majority of these devices need support and maintenance. Physically running around a building complex making sure that the devices are running correctly is impractical in terms of investment in time and money. Network monitoring provides a labour and cost effective solution, and systems that provide effective monitoring are important.

This report describes the creation of a network monitoring feature which has been implemented as an extension to an already existing Java-application named *iCtrl*[1].

The application iCtrl is used to control video and audio devices from a computer with a touch screen. The computer which the application is running on is almost exclusively plugged into a network, which is why some kind of network monitoring is desirable. The rare occasions where the computer is not plugged into a network include situations when the client does not have a network architecture to support it.

When designing the new application a several features were identified as important by the client. The network monitoring extension should be written in Java, and it should be tailored to easily work with iCtrl. The network monitoring extension should also be easily extended as more factors that need to be monitored might be added to iCtrl as it is still in development.

1.1 iCtrl

The application iCtrl was created by Martin Arnsrud at ATEA Sverige AB. The application is in its entirety written in Java. The application is used together with a computer preferably with a touch-screen running some version of Linux. The purpose is to be able to control different kind of devices, for example projectors, video-conference units and sound processors, from an easy-to-use touch interface. The application together with a computer is used in numerous conference rooms, lecture rooms and other establishments. This means that there are a lot of different configurations of iCtrl running in different locations (every location has its own "version" of iCtrl, but they are usually very similar). The application was created to give a more customizable option to similar applications already on the market. The most notable competitors are Crestron[2], Extron[3] and AMX[4].

[1] http://www.atea.se/it-infrastruktur/it-loesningar/samverkan/moetesrum/telemedicin/ictrl/
[2] www.crestron.com
[3] www.extron.com
[4] www.amx.com

1.2 Network Management

One can define network management as A. Clemm does: *"Network management refers to the activities, methods, procedures, and tools that pertain to the operation, administration, maintenance, and provisioning of networked systems."* [4]. Clemm explains that *operation* deals with keeping the network up and running, that *administration* involves keeping track of the resources in the network. Furthermore, *maintenance* is performing repairs and upgrades, and lastly *provisioning* involves configuring resources in the network (for example setting up a voice service for a customer).

1.2.1 Approaches

F. Luminati describes in his seminar paper that there are two different ways to perform the management task [5]. One is remotely invoking the operations directly on managed objects (known as *management by remote invocation*). The second one sends management instructions/logic to be executed locally on the managed devices (known as *management by delegation*). The second one can either be implemented as a "one-hop" procedure, which can be supported by the manager-agent model. The other way is to have the logic move from device to device.

As an example we can look at the most common protocol for network management SNMP. This protocol uses the manager-agent model, which is a form of management by delegation. In comparison, CORBA[9] uses management by remote invocation.

1.3 SNMP

SNMP stands for Simple Network Management Protocol. The first version of the SNMP protocol was first created in 1988. Today, almost all network-enabled devices have some kind of SNMP-agent in them. SNMP was designed to minimize the complexity of the agents, be extensible and be as independent of the implementation of particular hosts or gateways [6]. The result of this is that strengths of SNMP is the simplicity, interoperability and low footprint on agents. SNMP uses UDP[5], which means that the management applications are in full control of the retransmission strategy. F. Luminati states that these properties are the reason for SNMP's success [5].

SNMP is divided into two major systems, agents and the NMS (network management system). The agents run on the managed devices while the NMS queries the agents to get information about the devices. The queries include a dot-separated number - or as it is usually called - OID (**o**bject **id**entifier), which is global. This means that, for example, if you query *1.3.6.1.2.1.1* you will always get "System Info", independent of what device you query. In other words, the every OID of a device is something that can be monitored. To keep things from getting too confusing, every OID also has a textual representation which is defined in a MIB (Management Information Base). The MIB is a readable form of the monitorable objects on a device. An agent can implement as many MIBs as it

[5]http://tools.ietf.org/html/rfc768

wants, but the basic MIB called MIB-II is implemented by all agents. Here, the most basic set of variables such as system location are defined. It is possible to apply for a custom range of OIDs. This means that if there is a need to monitor objects not included in standard MIBs, it is also possible to define a completely new one.

There are two ways for the agent and the NMS to communicate with each other. The one that is used more frequently is that the NMS will poll (called queries) information from the agents to get updates of the state of the devices. However, if something critical happens the agents can actually send information directly to the NMS (called a trap). The device being monitored will have an SNMP-agent running. The agent will collect information about the device for the NMS.

SNMP has been criticized for its lack of security[1]. In version 1, only community names are used to define groups with different read and write rights. Not before version 3 encryption was used to securely transmit queries. Another widely known disadvantage of SNMP is the lack of transaction control. This means that configuration messages are not guaranteed to arrive in the right order, or at all.

1.3.1 Advantages and Disadvantages of SNMP

SNMP is the most common network monitoring protocol. Almost every device has an agent. The advantages of SNMP can be summarized as:

- The agent has a very small footprint and uses very small amount of resources.
- The information model is very extensible.
- Very low amount of network traffic because of UDP.

At the same time, SNMP is not perfect, here are the disadvantages of SNMP.

- No real security in version 1 and 2.
- No transaction control.
- Because of UDP, agents will not get any confirmation when it sends a trap to the NMS.

1.4 Alternatives to SNMP

Even though SNMP is the most widespread and common protocol for network managing, we will look at a few alternatives to SNMP to see if they are as viable for network managing.

1.4.1 NETCONF

NetConf, or the Network Configuration protocal, was an attempt to produce a protocol to accomplish what SNMP fails to do (transaction support and security). It came to be

in early 2000s and was created by IETF [5]. NetConf also uses the agent-manager model, using XML-encoded Remote Procedure Call (XML-RPC) for communication between the two. The agent will hold three different configurations at any time. One is the running one, the configuration which is used by the agent at a specific time. Another is the candidate one, a configuration which can be edited and manipulated without changing the running state of the agent directly. The last one is a start-up configuration for initial settings.

1.4.2 CORBA

CORBA is not strictly for network management purposes, but instead has a more general purpose. Fundamentally, CORBA is a way to make method-calls between different platforms, either in the same address space or remotely [5]. CORBA is an example of *management by remote invocation* mentioned in 1.2.1, where abstract objects are defined for each device. To support new functionality and extensibility, it is possible to add new abstract objects or modify existing ones. This kind of architecture is good for scalability, as it can deal with very big networks [5]. CORBA also provides interfaces that are standard and open, which means that information can be exchanged between devices of different vendors. Because of the nature of CORBA, a server and a client can be written in different languages but still communicate. So why has not CORBA become a more popular choice for network management than it is? Mostly it is because it does not support bulk data retrieval and is, in comparison to other solutions, resource hungry.

1.5 SNMP In Comparison With Alternatives

One of the biggest attractions of NetConf is the use of XML. However, this also means that it has a significant larger overhead than SNMP. In [8] a head-to-head performance comparison between SNMP and NetConf is presented. The comparison concludes that at a lower number of managed objects, SNMP was faster. When the amount of managed object exceeded 500, NetConf starts to be more efficient. What is more important is that NetConf is able to configure up to 100,000 managed objects in a single transaction, while SNMP's *best case* is 2779 transactions for the same number of objects. However, SNMP is still more widely used and a more common standard.

Just like NetConf does seem to be more efficient than SNMP in some cases, CORBA does the same. In a paper by Qiang Gu and Alan Marshall [7] they are put to the test. One noticeable difference between the two is that CORBA uses TCP while SNMP uses UDP. Generally, when the messages are small UDP is faster and uses less resources. If the messages are larger, TCP will actually be faster as it provides flow control and is efficient at segmenting the messages. It might not be surprising that when a single object is retrieved SNMP fares much better. By just looking at the traffic volume CORBA's 332 bytes surpasses the 71 bytes of SNMP by a lot. This is mostly because of the TCP connection overhead. Because of the TCP connection there is also a big latency difference between COBRA and SNMP. CORBA's latency lands on 6.3ms while SNMP has only 0.62ms. However, when retrieving tables of data, CORBA stands out significantly. This

is because CORBA has a function of returning an entire table, but SNMP, as mentioned before cannot handle such transactions and then instead has to return every row individually. In the paper by Q. Gu and A. Marshall [7] they test this by retrieving 200 rows. CORBA uses only 488 bytes while SNMP (version 2) uses almost 10 times more, specifically 4702 bytes. Looking at latency, CORBA wins here as well if there are more rows than 20 being retrieved. There is only an increase from 12ms to 50ms when retrieving 1 and 2000 rows respectively, using CORBA. Retrieving the same from SNMP resulted in a delay of 1.4ms and 4472ms, respectively.

1.6 Network Monitoring with iCtrl

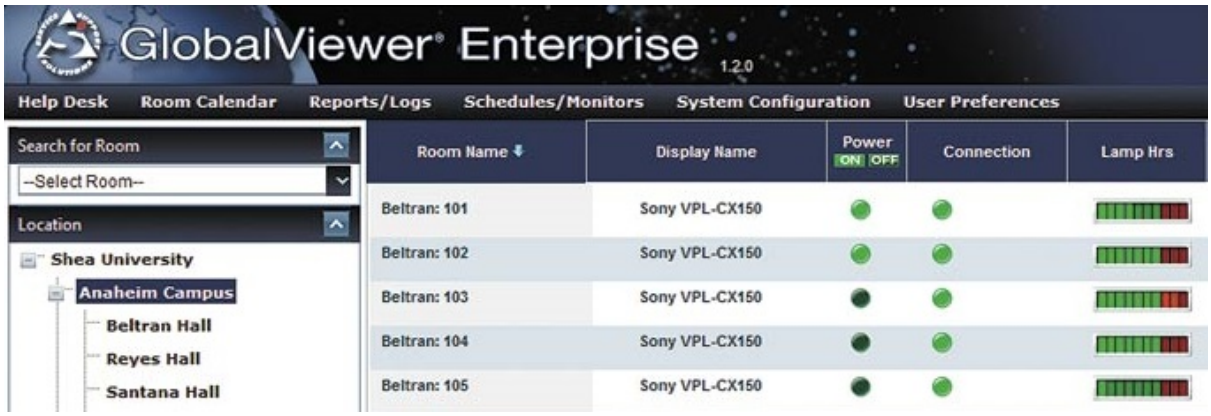

Figure 1: Extract of GlobalViewer by Extron Electronics

To really compete with other control system solutions, iCtrl needs some kind of network monitoring. By looking at two of the most notable competitors, Crestron and Extron, we can see that they both have applications for monitoring the control systems remotely. Crestron uses SNMP together with their proprietary application called Fusion RV. Extron has a similar setup with SNMP and also a proprietary application called GlobalViewer. One of the most notable features of these two is the ability to monitor not only the control system, but an entire *room*. This means that if for example a projector is somehow connected to the control system (either through some kind of Ethernet connection or over RS232), information about the projector's status is reported by the control system to the monitoring application. The same goes for all the equipment in the room. This kind of monitoring has a few advantages. One is that if the equipment is not on the network, or if they do not have any SNMP-agent (or other) of themselves they can still be monitored indirectly. Another advantage is that an entire room can be monitored as a single *object* in the monitoring application. This could be less complex and easier for the end-user to understand.

1.7 Problem Description

The problem is to find a way to implement network monitoring into an already-existing system. Furthermore, since iCtrl is used in a lot of different environments, and configura-

tions are made to the application depending on which devices needs to be controlled, the chosen network monitoring protocol needs to be easily configured in a similar way. As iCtrl is still in development, a complete implementation of a network monitoring solution is impossible. Therefore, the goal of this project is to choose a protocol and start an implementation, with good extensibility in mind.

1.7.1 Summary of Requirements

These are the requirements for the project:

- Written in Java.
- Good extensibility.
- Be able to monitor arbitrary objects, should be easy to add new ones.
- Be able to monitor devices indirectly through the control system (as described in 1.6).
- Be able to be run on slower machines.
- Be able to choose which objects should be monitored for every iCtrl configuration.

2 Method

2.1 Developing the Network Monitoring Application

The waterfall model[3] was used while developing the network monitoring application. This model was chosen mainly because of the size of the project does not merit using more complex models. The waterfall model is easy and effective when requirements are well known.

2.1.1 Requirements Phase

In the waterfall model the requirements phase is conducted in the beginning, and the beginning only. Because of this there needs to be good requirements before the next phase starts. ATEA was required to be a part of this phase, so they could express any specific requests of the monitoring application. Research on the competition was also conducted. It was determined what features they supply in their applications and what needs they meet. The type of hardware iCtrl runs on was also considered. This was especially important when choosing a protocol, as the protocols varied quite significantly in footprint and resource usage.

2.1.2 Design Specification Phase

In this phase one of the most important decisions was made, namely, which protocol should be used. In the previous section, SNMP and two alternatives were presented. Except these, there is also the option of creating something completely proprietary. To determine which of these are the most suitable research about what should be monitored in iCtrl has been conducted. Furthermore, it was determined how important the different aspects of the different protocols are for iCtrl. There was also some considerations about which of the protocols are the best to implement in Java. As mentioned in the previous phase, there was also considerations to what the hardware iCtrl usually runs on can manage. The overall architecture was also determined as well.

2.1.3 Implementation Phase

In the waterfall model, implementation is its own phase, where all the coding takes place. In this phase coding was done in Java, using the design specification as base. JavaDoc will be used as primarily documentation and as further development is likely the code is easy to follow and as much as possible completely decoupled.

2.1.4 Testing and Evaluation Phase

Testing was conducted using four different test methods. These are unit testing, integration testing, system testing and acceptance testing[10]. During the evaluation the

requirements were revisited, to make sure the solution solves the needs of the application. The competitiveness of our proposed system was also evaluated.

3 Design and Implementation

3.1 Choosing a protocol

By looking at the requirements the protocol that suits iCtrl the best is SNMP. SNMP has qualities that match every requirement. SNMP can be (and already is) written in Java. Some of the key features of SNMP are easy on resources and good extensibility which also match the requirements perfectly.

But before we get ahead of ourselves, we should look at the disadvantages of SNMP and see if those are acceptable in this application. The disadvantages are:

- No real security in version 1 and 2.
- No transaction control.
- Because of UDP, agents will not get any confirmation when it sends a trap to the NMS.

The first one is arguably the worst trade-off. However, security is seldom a problem with these kind of applications, as they are often-times not the target of attacks since the information that can be gathered from them is limited to see things like if a projector is on or off. The second one is not much of a problem either, as precise timing is not important when the purpose of the application is to control other devices. If the video-conference unit turns on before the projector, or vice-versa, does not matter. The last one is also not much of an issue as information will be continually polled from the devices, so if a trap is missed, the next poll will get the information needed.

3.2 Designing SNMP for iCtrl

First and foremost, some devices the iCtrl-application controls are not always network-enabled (if this is the case, they are usually controlled by iCtrl through the RS232 protocol), and some of them that are does not always have SNMP. Although, monitoring these devices is still desirable, and since iCtrl does have some protocol of speaking to these devices, iCtrl can *answer* SNMP-requests on their behalf.

A good example of this is a projector which is controlled over RS232. It is very often desirable to have information about how many hours a projector has had its lamp on, giving some idea when to change the lamp before it breaks. This information can be retrieved over RS232, and iCtrl already does this, which is why it is an appropriate idea to use that information to be able to monitor the projector over SNMP, even if it is not directly network-enabled.

Since every configuration of iCtrl is different, the SNMP agent must also have this option. If the configuration for iCtrl adds a projector, it should be equally easy to add this to the SNMP agent, and it should be monitored right away.

Before you can start to define your own MIB for SNMP, you need to have an OID registered. After registered you get an OID with the form *1.3.6.1.4.1.<your-number>*. The textual representation of this is *iso.org.dod.internet.private.enterprises.<your-company>*. A number was registered for ATEA (41761), so now any OID with *1.3.6.1.4.1.41761* as prefix can be used by ATEA and more importantly, for this thesis.

3.3 MIB

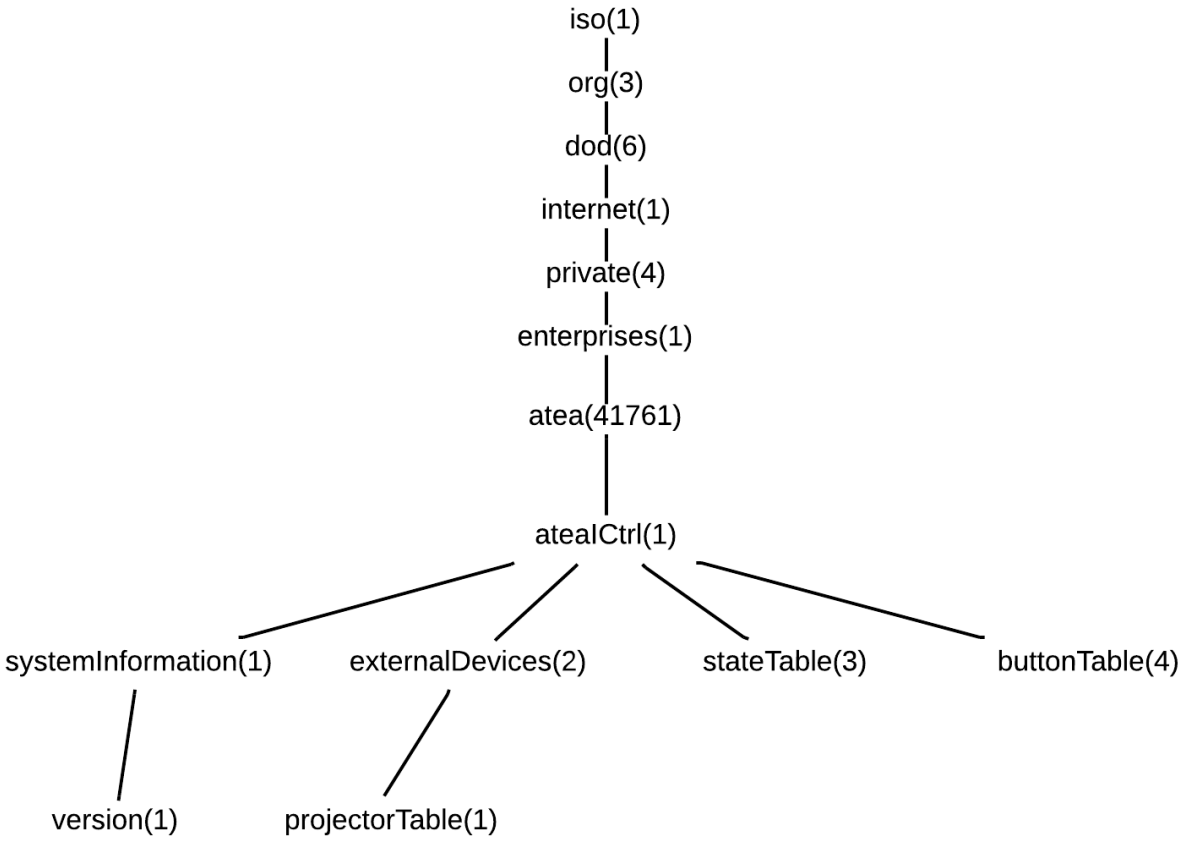

Figure 2: Top-level MIB. In reality there are more leaves, but this figure represents the basic structure.

The MIB was one of the more crucial parts of this thesis, as the result would have to be very extensible and cover most possible additions later. After the SNMP-agent is rolled out in real situations, one cannot simply rewrite the MIB since it would thwart backwards compatibility. The figure 2 shows the tree-structure of the written MIB. The MIB is done in such a way that adding more devices and/or variables is trivial (and therefore only a few examples are in the resulting MIB).

System Information
System Information is stored under *1.3.6.1.4.1.41761.1.1*. Here, variables such as iCtrl-version (shown in figure) and system on/off state are stored.

External Devices
Each external device will go under OID *1.3.6.1.4.1.41761.1.2*. There is an example of the external device projector, which is represented as a table which more variables can be added to (now there is only the projector ip, name and lamp-hours. Every device should be represented as a table, since there is the possibility of having more than one of each device.

State
There is another important variable which is called state. This is used to monitor how long some state of the system is being used. Take for example a conference-room where you have a video-conference system and you would like to know how often it is used. Then the iCtrl-programming could create a new state in the state-table called "video-conferencing" and start some timer whenever the video-conference is used through the iCtrl touch-interface.

Buttons
It can be important to know how often some buttons are being pressed to answer some questions about further development (is that button really important? is that button pressed practically every time the system is used?). This is why the MIB has a table of buttons and how many times they have been pressed.

3.4 Implementation

Listing 1: An snmpwalk-command done on a computer running running iCtrl with the SNMP-agent.

```
> snmpwalk -v2c -M+/usr/share/mibs/atea
-m+ATEA-MIB -c public 127.0.0.1 1.3.6.1.4.1.41761

> ATEA-MIB::systemName.0 = STRING: "An Example iCtrl-System"
> ATEA-MIB::systemOn.0 = INTEGER: 1
> ATEA-MIB::systemIp.0 = IpAddress: 127.0.2.1
> ATEA-MIB::projectorName.1 = STRING: Example Projector
> ATEA-MIB::projectorIp.1 = IpAddress: 127.0.0.1
> ATEA-MIB::projectorLampHour.1 = Gauge32: 50
> ATEA-MIB::buttonName.1 = STRING: Example Button 1
> ATEA-MIB::buttonName.2 = STRING: Example Button 2
> ATEA-MIB::buttonPresses.1 = Gauge32: 4
> ATEA-MIB::buttonPresses.2 = Gauge32: 10
> ATEA-MIB::stateName.1 = STRING: Running SNMP-test
> ATEA-MIB::stateTime.1 = Timeticks: (110565) 0:18:25.65
```

Listing 2: An snmpwalk-command done on a computer running running iCtrl with the SNMP-agent. Here, the entire OID to each variable is shown.

```
> snmpwalk -v2c -M+/usr/share/mibs/atea
-m+ATEA-MIB -c public -On 127.0.0.1 1.3.6.1.4.1.41761

> .1.3.6.1.4.1.41761.1.1.1.0 = STRING: "An Example iCtrl-System"
> .1.3.6.1.4.1.41761.1.1.2.0 = INTEGER: 1
> .1.3.6.1.4.1.41761.1.1.3.0 = IpAddress: 127.0.2.1
> .1.3.6.1.4.1.41761.1.2.1.1.1.1 = STRING: Example Projector
> .1.3.6.1.4.1.41761.1.2.1.1.2.1 = IpAddress: 127.0.0.1
> .1.3.6.1.4.1.41761.1.2.1.1.3.1 = Gauge32: 50
> .1.3.6.1.4.1.41761.1.2.3.1.1.1 = STRING: Example Button 1
> .1.3.6.1.4.1.41761.1.2.3.1.1.2 = STRING: Example Button 2
> .1.3.6.1.4.1.41761.1.2.3.1.2.1 = Gauge32: 4
> .1.3.6.1.4.1.41761.1.2.3.1.2.2 = Gauge32: 10
> .1.3.6.1.4.1.41761.1.3.1.1.1 = STRING: Running SNMP-test
> .1.3.6.1.4.1.41761.1.3.1.2.1 = Timeticks: (42175) 0:07:01.75
```

Listing 3: An snmpwalk-command done on a computer running running iCtrl with the SNMP-agent. Here, the entire textual path to each variable is shown (all of these are prefixed with *.iso.org.dod.internet.private.enterprises.atea.ateaICtrl* but it has been left out for brevity).

```
> snmpwalk -v2c -M+/usr/share/mibs/atea
-m+ATEA-MIB -c public -Of 127.0.0.1 1.3.6.1.4.1.41761

> .iCtrlSystemInformation.systemName.0 = STRING: "An Example iCtrl-System"
> .iCtrlSystemInformation.systemOn.0 = INTEGER: 1
> .iCtrlSystemInformation.systemIp.0 = IpAddress: 127.0.2.1
> .iCtrlExternalDevices.projectorTable.
projectorEntry.projectorName.1 = STRING: Example Projector
> .iCtrlExternalDevices.projectorTable.
projectorEntry.projectorIp.1 = IpAddress: 127.0.0.1
> .iCtrlExternalDevices.projectorTable.
```

```
projectorEntry.projectorLampHour.1 = Gauge32: 50
> .buttonTable.buttonEntry.buttonName.1 = STRING: Example Button 1
> .buttonTable.buttonEntry.buttonName.2 = STRING: Example Button 2
> .buttonTable.buttonEntry.buttonPresses.1 = Gauge32: 4
> .buttonTable.buttonEntry.buttonPresses.2 = Gauge32: 10
> .stateTable.stateEntry.stateName.1 = STRING: Running SNMP-test
> .stateTable.stateEntry.stateTime.1 = Timeticks: (110982) 0:18:29.82
```

The implementation is as follows.

- An open-source third-party Java library called *snmp4j* [2] has been used for the Java-implementation.
- Using this library, an agent has been created, which provides easy configuration for some selected monitored variables.
- The code is done is such a way that, when something new that should be monitored, it will be easy to add it to do implementation.

There had to be some differences as how the SNMP-commands were handled in the snmp4j-library. To achieve this, some methods had to be overridden and implemented in such a way that when for example an snmpget-command was received, if iCtrl already had the information it needed, that value should just be returned. This feature was added because often-times iCtrl polls information continuously from the devices it controls, and as long as that information is saved somewhere the snmpget can just return the latest information iCtrl pulled.

The listing 1 is an example of an SNMP-walk command done on a machine running the iCtrl-application with the new SNMP-extension. Also listings 2 and 3 shows the same command, but represented with full names and full OIDs, respectively.

The agent is done in such a way that first, when the agent is added to an iCtrl-application, it is an empty shell. Nothing is being monitored until something is added to the agent. For example, if you want to monitor a projector you have to call *addProjector(SnmpProjector p)*. After that is done the projector will be monitored. When the agent gets a request about information of that projector, the agent will ask projector p for the information.

Another thing that had to be implemented was that some information (for example button presses) would be lost if the iCtrl-application is shut down. Because of this an XML-document is used to save such information.

4 Testing and Evaluation

According to tests conducted, the code is fully working. But evaluation must be done to see if the requirements are met. Let us revisit the requirements.

4.1 Architecture and implementation

Written in Java
The code has been entirely written in Java. One could argue that the MIB has not been written in Java, but this is impossible.

Good extensibility
As all basic forms of SNMP-types already have code, extending is merely defining new OID and calling code with that OID in the Java-application. However, since some things that might come up when iCtrl continues to develop it is impossible to know if this goal has been reached fully.

4.2 Flexibility and scalability

Be able to monitor arbitrary objects
This somewhat follows from the good extensibility. There are no restrictions in SNMP on what kind of objects can be monitored.

Be able to monitor devices indirectly through the control system
There is no limitations that the information retrieved must be about the control system. As long as the control system does have the requested information an object can be defined to retrieve that information, regardless if it is information about the control system itself or something else.

Be able to be run on slower machines
This is not an issue as SNMP is the least resource hungry of the alternatives, and generally has a very small footprint.

Be able to choose which objects should be monitored for every iCtrl configuration
The code has been written in such a way that the agent starts as an empty shell. For every iCtrl configuration, you are able to add the objects you want into the agent as described in 3.4.

Should be working with current version of iCtrl
All testing has been done with the latest version of iCtrl. It will also be easy to develop iCtrl alongside this agent, as the agent does not invoke any method calls to any of the iCtrl libraries, except the agent itself and the added objects as as described in 3.4.

5 Conclusion

An agent has been created and the requirements have been met. However, two things in particular were omitted due to time constraints. As of now, the agent uses version two of SNMP. This, as described in 1.3, is not the latest version and lacks good security. An obvious improvement would be to add this to the agent, preferably by being able to choose which version to use in each configuration (for example if an iCtrl-system is to be used in some high-security building, using version 3 might be a must). Another thing that is not yet implemented is the usage of traps. There is no way to send traps from the current version of the agent. If traps were implemented, they would been used to send errors directly to the NMS. How important these traps are depends on how often the NMS requests information. If information is pulled from the agent in fairly frequently the errors will be detected anyway.

Intellectual Property

We certify that the material in this report is solely produced by its authors, except where otherwise indicated and clearly referenced.

References

[1] Douglas R. Mauro, Kevin J. Schmidt, *Essential SNMP* (2nd edition), O'Reilly, 2005.

[2] SNMP4J [Internet] [cited 2013 Jun 20]. Available from: http://www.snmp4j.org/

[3] P. Kruchten, *Going Over the Waterfall with the RUP*, IBM Corporation, 2004.

[4] A. Clemm, *Network Management Fundamentals*, Cisco Press, 2006.

[5] F. Luminati, *Alternatives to SNMP and Challenges in Management Protocols*, presented at University of Zurich Department of Informatics (IFI), 2013.

[6] J. Schonwalder, A. Pras, J.-P. Martin-Flatin, *On the future of Internet management technologies*, Communications Magazine, IEEE , vol.41, no.10, pp.90,97, Oct 2003 Technologies.

[7] Q. Gu, A. Marshall, *Network Management Performance Analysis And Scalability Tests: SNMP vs. CORBA*, Advanced Telecommunications System Laboratory, School of Electrical & Electronic Engineering, The Queen's Universite of Belfast, 2004.

[8] B. Hedstrom, A. Watwe, S. Sakthidharan, *Protocol Efficiencies of NETCONF versus SNMP for Configuration Management Functions*, MSc Thesis, University of Colorado, 2011.

[9] Randy Otte, Paul Patrick, Mark Roy, *Understanding CORBA (Common Object Request Broker Architecture)*, Prentice-Hall, Inc, 1996.

[10] John Watkins, Simon Mills, *Testing It: An Off-the-Shelf Software Testing Process*, Cambridge University Press, 2010.