Embedded System Management using WBEM

Michael Hutter, Alexander Szekely, and Johannes Wolkerstorfer
Institute for Applied Information Processing and Communications (IAIK)
Graz University of Technology, Inffeldgasse 16a, 8010 Graz, Austria
email:{Michael.Hutter,Alexander.Szekely,Johannes.Wolkerstorfer}@iaik.tugraz.at

***Abstract*—Web-based management solutions have become an increasingly important and promising approach especially for small and embedded environments. This article presents the design and implementation of an embedded system that leverages the Web-based Enterprise Management (WBEM) solution. WBEM has been designed to manage large heterogeneous environments but has not yet been deployed on small and embedded devices. First, we evaluate existing WBEM implementations due to its resource requirements. Second, we describe the design of an embedded network device that has been realized on a system-on-chip prototyping platform. A small-footprint WBEM server has been integrated that requires less than 900 kB of non-volatile memory. We provide performance measurements of our solution and compare the results with other Web-based management approaches. They show that WBEM is suitable to run on such resource-constraint devices and to be applicable in practice.**

I. Introduction

With the inexorable growth of computer networks and Internet connectivity, management issues of distributed systems have become a topic of increasing interest. Network devices not only implement their own management tools and protocols but they may also be deployed on multiple-vendor platforms and heterogeneous environments. Reliable administration and configuration has thus become a very time-consuming and exhausting task. At the same time, networked devices have moved from standalone to Internet-enabled devices that can be accessed simply by common Web browsers. This allows easy administration of the device and offers additional services such as remote notification and error handling over the Internet. Nevertheless, in order to provide comprehensive end-to-end capabilities for embedded devices, different design issues have to be observed such as memory consumption and processing-power constraints.

One approach to meet the management demands is to use the Web-Based Enterprise Management (WBEM) [1] solution. It is a set of standards that has been developed by the Distributed Management Task Force (DMTF). The main goals of this initiative have been to provide interoperability among multiple-vendor management solutions. WBEM is closely related to the Common Information Model (CIM), which describes the managed system in a fully object-oriented approach. WBEM/CIM is supported by many large players of the computer industry like Microsoft, Intel, and Compaq. The Windows Management Instrumentation (WMI) interface of the Microsoft Windows operating system is an example of a WBEM implementation. In particular, WBEM involves the integration of a Web-based server that communicates with a client. It is capable of receiving and sending XML-encoded messages that are sent over the Hyper-Text Transport Protocol (HTTP) [2]. A WBEM server has to deal with several issues such as HTTP and XML coding, CIM schema repository, security features like Secure Shell (SSH) and authentication, and a CIM Object Manager (CIMOM) that is used to handle client requests and server responses. These issues certainly need an amount of resources and make an integration into embedded devices difficult.

Embedded WBEM servers differ from conventional general-purpose Web servers in several ways: They have to be light-weight in terms of CPU usage and have to do without fast processors. They still have to work within limited resources like memory and battery lifetime. Embedded WBEM servers are used to provide information about the system such as working status and configuration settings as well as offer the opportunity of command control and monitoring functions. Essentially, there are many application scenarios for embedded servers. They are integrated into mobile devices to provide different Web services such as news, emails or chat rooms. They can also be found in the domotic field where home appliances are connected to each other to provide greater safety and more convenience to home users. Embedded servers offer interoperability and allow unique view of all involved devices there. Another example of integrated Web servers can be found in embedded routers and switches that are distributed over networks. Figure 1 shows a typical example where embedded IPsec gateways constitute a secure tunnel over the Internet. The gateways establish secure peer-to-peer connections creating Virtual Private Networks (VPN) and therefore have to be configured with common security policies.

In this article, we present the practical results of our research to develop such an embedded network-security gateway that implements a light-weight WBEM server as a management solution. Although WBEM is considered to be sufficiently complex for many embedded systems, we show that WBEM is indeed applicable to resource-constraint environments. Firstly, we realized an embedded IPsec gateway that runs on a system-on-chip prototyping platform including an Field-Programmable Gate Array (FPGA). Our solution makes use of Quantum-Key Distribution (QKD) in combination with a quantum-enabled version of Internet Key Exchange (QIKE) to generate IPsec key material. As an operating system, we have used embedded Linux that runs on top of the FPGA. In addition to the device implementation, we have written a

 DOI: http://dx.doi.org/10.1109/INM.2009.5188840

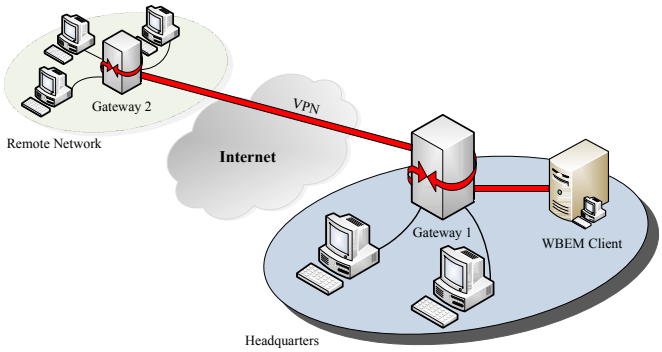

Fig. 1. Management of an IPsec-enabled Virtual Private Network (VPN).

customized client application to prove the concept of efficient management and distribution of security policies in multiple networked IPsec-gateway instances. Finally, we present the results of performance measurements on our implementation and compare them with other existing Web-based management solutions. To our knowledge this is the first article that covers the practical integration and analysis of the WBEM approach into such an embedded platform.

The remainder of this article is structured as follows: Section II provides an overview of different management solutions and gives an overview about related works on that topic. In Section III, WBEM is described in detail. Section IV deals with the resource requirements of common WBEM implementations. In Section V, the target platform is introduced. Furthermore, the management requirements and the practical implementations of the managed resources are described. In Section VI, we discuss the results of the performance of our implementation and point out other non-WBEM solutions that have been realized on embedded devices. Conclusions are given in Section VII.

II. Management Solutions and Related Work

Many network-management solutions exist that range from simple command-line tools to high-end and sophisticated management systems. One of the simplest approaches in this context is remote login to the managed device using Command Line Interfaces (CLI). Classical protocols like *Telnet* and *Rlogin* offer this kind of access [3]. Configuration information can be changed by issuing commands to the target device or by editing configuration files. Status information can be obtained in the same way. Nevertheless, remote login suffers from many problems like security and the individual configuration of network devices, which together lead to higher maintenance costs. It is obvious that carrying out such basic approaches in huge enterprises is largely infeasible when regarding actual management fundamentals such as fault detection, accounting, monitoring, or security [4].

A widely used management solution is the Simple Network Management Protocol (SNMP) [5]. Since the late 1980s, the protocol has gained lots of popularity, not only for device configuration but even more for network monitoring. While users had expressed major concerns with older versions of SNMP due to the lack of effective security, SNMPv3 became accepted after it had been fully standardized by the Internet Engineering Task Force (IETF). In competition with the design of the SNMP protocol, there exists the Common Management Information Protocol (CMIP) [6]. It provides enhanced features compared to SNMP regarding security, the handling of tasks, or the detection of errors. However, CMIP was designed for large-scale provider networks like those used in the telecommunication industry and makes an integration to tiny devices difficult due to its complex and resource-intensive behavior.

Besides traditional protocols like SNMP and CMIP, Web-based solutions have become a more interesting and promising network-management approach especially in view of embedded devices. Indeed, the Web browser has become the de-facto standard client that provides an interface for many applications. The management of network devices involves the integration of a tiny Web server that allows the administration over the Internet. Common solutions are Web services described using the Simple Object Access Protocol (SOAP) [7], Universal Description Discovery and Integration (UDDI) [8], or Web Service Definition Language (WSDL) [9]. They typically rely on the open and standardized Extensible Markup Language (XML) protocol. Other Web-based approaches are Netconf [10], which is an IETF Working Group chartered protocol; the Java Management Extension (JMX) [11], which is promoted by Sun Microsystems; and WBEM, which is addressed in this article and further described in Section III.

Many articles can be found in literature that focus on network-management solutions. J. Schönwälder et al. [12], A. Corrente et al. [13], X. Du et al. [14], and C. Pattinson [15] reported traffic analysis of SNMP implementations as well as performance analysis of SNMP over TLS or SSH. There are also many articles which concentrate on implementations of Web services. A great deal of research has been done in this field and reported by A. Pras et al. [16], L. Z. Granville [17], and G. Pavlou [18].

In view of embedded environments, many articles propose tiny Web servers that implement specific Web services. D. Schall et al. [19] implemented a SOAP-based Web service that has been integrated into a mobile device. They compared the performance of C++ and Java implementations and performed round-trip delay measurements. In [20], L. Pham et al. demonstrated a realization and performance analysis of a SOAP server that runs on an iPAQ (HP 3870) and two mobile devices (Ericson P900 and Siemens S55). G. B. Machado et al. [21] integrated a SOAP-based server onto an ARM microcontroller and H.-T. Ju et al. [22] embedded a typical HTTP Web server into a MPC 860 32-bit microcontroller. C. Kalbfleisch et al. [23] discussed the usefulness of SNMP in a real-time environment. They examined performance characteristics of SNMP using a Unix-like real-time operating system.

There are only a few articles that discuss WBEM implementations and its performances. In addition, there have so far been no articles that focus on a WBEM implementation

on an embedded device. S.-J. Lee et al. [24] have analyzed the requirements for managing ubiquitous computing servers based on WBEM technology. They have performed benchmark tests of WBEM-server implementations on a general-purpose PC (Pentium IV 1.6 GHz). A more detailed evaluation of the same WBEM implementations is reported by S.-M. Yoo et al. in [25].

III. Web-Based Enterprise Management

WBEM is composed of several standards. One of these standards is the Common Information Model (CIM), which is used to model the resources that have to be managed. CIM is based on an object-oriented model and can be structured into three layers: the core model, the common model, and the extension schemas. The core model represents the root of the CIM object hierarchy and is used to describe basic elements of the overall management system. The core model is expanded by the common model, which is tied to particular management areas. These areas are technology-independent such as systems, users, policies, network, events, devices, databases, or applications. However, the core model and the common model are referred to as the CIM schema that is constantly developed and updated by the DMTF. Extension schemas, in contrast, are technology-dependent extensions of the common model.

Another specification in WBEM is CIM-XML. This standard is used for exchanging CIM information between the WBEM server and the client. First, the CIM information is encoded using XML. Finally, the XML-encoded message is transmitted over HTTP. There are also other standards like the CIM Query Language (CQL), which is used to define specific query operations, or the WBEM-server discovery feature using the Server Location Protocol (SLP) standard.

The central unit of a WBEM server is the CIM Object Manager (CIMOM) running on the device to be managed. It is responsible for handling the CIM operations that are sent by clients. Dependent on the CIM operation, the CIMOM typically invokes the appropriate provider, which is responsible for the resource a client has requested. In particular, providers are libraries that implement the functionality to retrieve and set information of different kinds of system resources. The provider passes this information to the CIMOM, which transmits the information back to the client using CIM-XML.

IV. Resource Requirements of WBEM/CIM Servers

Realizing a management system on a constrained embedded system requires a thorough analysis of whether the provided resources are sufficient for the smooth operation of the device. There exist a wide range of WBEM/CIM-server implementations, but not all of them fulfill the requirements of embedded systems. OS-dependent WBEM/CIM servers are integrated into the operating system like WMI into Microsoft Windows and WBEM Services into Sun Microsystems Solaris. In contrast, OS-independent WBEM/CIM servers are, for example, the Tivoli suite from IBM, the CiscoWorks from Cisco, and the WBEM Solutions from Hewlett Packard.

TABLE I
Resource requirements of OpenWBEM, OpenPegasus, and SFCB

	OpenWBEM [1] **[kB]**		**OpenPegasus** [26] **[kB]**		**SFCB** [27] **[kB]**	
	After start	**After enum**	**After start**	**After enum**	**After start**	**After enum**
Total RSS	4 496	5 388	5 328	13 468	1 896	2 728
Private RSS	3 668	4 440	4 484	10 492	792	1 612
Shared RSS	828	948	844	2 976	1 104	1 116
Shared RSS system libs	540	632	572	632	536	536
Effective RSS	3 956	4 756	4 756	12 836	1 360	2 192
Static disk space	50 900		9 800		1 900	

Not commercial but rather open-source implementations are OpenWBEM, OpenPegasus, and the Small Footprint CIM Broker (SFCB). However, due to the fact that our prototyping platform uses Linux as an operating system, we have focused on implementations that are implemented in C or C++. Moreover, we have concentrated on open-source solutions. Hence, three WBEM/CIM implementations (OpenWBEM [1], OpenPegasus [26], and SFCB [27]) are shortlisted and are analyzed in the following.

We have determined the static disk-space usage as well as the dynamic run-time RAM usage of the WBEM/CIM implementations in order to characterize their memory requirements. However, the evaluation of consumed run-time RAM of a running Linux program is a challenging task. In general, there exist tools like *ps*, *top*, or *pmap* that can be used to determine the memory usage of a running process. Among other information, they provide the Virtual Segment Size (VSZ) and the Resident Segment Size (RSS). The VSZ represents the overall address space of a process. This can be memory that is mapped to physical RAM, or memory that has been swapped out to disk space, or even memory that has not been used at all. It defines the address space that the process can use without producing a segmentation fault. The RSS value, in contrast, represents the memory pages that are mapped to physical RAM. Nevertheless, RSS neither includes memory information of swapped pages nor does it take care of memory pages that are shared between processes. Both VSZ and RSS do not provide an accurate description of consumed run-time memory.

In order to provide a good approximation about the run-time memory consumption of the three WBEM/CIM implementations, we have used the *smaps* information that is located in the virtual */proc* filesystem of Linux. The *smaps* information shows the private RSS that represents the resident memory-pages that are only used by the process itself. Moreover, it shows the shared RSS that represents the resident memory-pages that are shared with other processes running on the machine. For each WBEM/CIM implementation, we have determined the following information: the total size of the resident memory (RSS), the private and shared RSS, the

size of shared system-libraries that constitute to the shared RSS, and the effective RSS. The effective RSS is made by subtracting the shared system-libraries from the private RSS. The reason for this is that there exist system libraries that have already been mapped into physical RAM and are only duplicated during *copy-on-write*. For that matter, the effective RSS can be considered as a good measure for the real run-time memory consumption of processes provided that no memory-page swapping occurs during evaluation. It has therefore been used in our analysis.

The results of our measurements are shown in Table I. All tests were performed on a dual-core PC (2.8 GHz, 1 GB memory) running 32-bit Ubuntu Linux. All WBEM/CIM implementations hold the same number of CIM classes, the same CIM schema (v2.9.0), and the same provider libraries (SBLIM Base OS-Instrumentation). Two measurements have been performed for each WBEM/CIM implementation. First, the run-time memory consumption has been determined right after the startup of the server. Second, the same measurements have been performed after repeating a CIM operation 10 times in order to evaluate the resource requirements of each server at run time. The *enumerateInstances* operation has been called from the Linux_OperatingSystem class. The first line shows the total resident memory of the corresponding WBEM/CIM implementations. The second and third lines represent the private and shared RSS. The fourth line represents the memory consumption of shared system libraries like *libc.so*, *libpthread.so*, *ld.so*, or *libdl.so*. The fifth line constitutes the effective RSS.

In the last line of Table I, the static disk space of the three WBEM/CIM implementations is shown. These values have been obtained from standard installation after deleting unimportant files like manuals, documentations, unused binary files, includes, and test files. The disk size of the provider libraries and the disk size of the CIM repository have been omitted as the size heavily depends on the number of supported CIM classes.

The results of our measurements show that SFCB provides the best performance both in static memory and run-time memory consumption. Note that these measurements only give a rough insight into the overall memory consumption of the underlying WBEM/CIM implementations. The results highly depend on the used operating system, the used compiler, and the enabled features like Secure Socket Layer (SSL) or Service Location Protocol (SLP). However, it is possible to further reduce the memory footprint of each server. For OpenPegasus, there exists a Pegasus Enhancement Proposal (PEP) [26] focusing on that topic. For SFCB, there exists a manual describing further steps for shrinking the server to a smaller footprint. [27].

In light of the fact that SFCB has been especially designed for resource-constrained environments and that it provides the best performance in our memory profiling, we decided to use it in our embedded IPsec-gateway implementation. SFCB supports Common Manageability Programming Interface (CMPI), SSL for secure configuration management, basic and client certificate-based authentication, HTTP chunking, indications for event handling, and an efficient binary repository that can be further compressed to reduce the memory footprint.

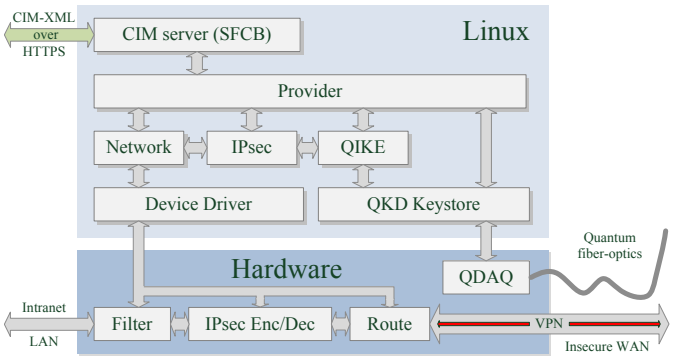

Fig. 2. System-on-chip architecture of our prototyping platform

V. Running WBEM on a System-on-Chip Prototyping Platform

For our implementation, we have used the ML410 system-on-chip prototyping platform from Xilinx. The main component of this board is a Virtex-4 FX60 Field-Programmable Gate Array (FPGA). FPGAs are integrated circuits with programmable logic cells and configurable interconnects that can be used to implement arbitrary hardware functions. The Virtex-4, in particular, provides an embedded PowerPC 405 processor and multiple Ethernet Media-Access Controllers (MAC) as hard macros. These MACs support Gigabit Ethernet. The PowerPC processor is clocked at 300 MHz and the rest of the board at 100 MHz. The FPGA provides 58 880 configurable logic cells that are used to accommodate the hardware needed to implement the quantum-enabled IPsec functionality. Next to the FPGA, a DDR-SDRAM memory of 64 MB, two Gigabit Ethernet Physical Layer (PHY) transceivers, and a Compact Flash controller are available. We have used a Compact Flash card to configure the FPGA after power-up and to initialize the root RAM file system. The Compact Flash card is also used to save persistent configuration information and logging data, in addition.

In Figure 2, the overall system architecture of our WBEM-enabled prototyping platform is shown. The system architecture is separated into a software and a hardware part. Several components like the whole IPsec functionality are implemented in hardware in order to achieve Gigabit throughput rates. Moreover, packet filtering, encryption, decryption, IP routing, and the Quantum Data Acquisition (QDAQ), which is used to get IPsec keys from a quantum optical channel, are implemented in hardware. The software part uses Linux 2.6 as an operating system running on the PowerPC processor. SFCB has been integrated into Linux and is used to manage several services. On the one hand, there are providers for managing software components like the Quantum Internet Key Exchange (QIKE) or the Quantum Key Distribution (QKD) keystore. On the other hand, there are providers for managing hardware

modules. These providers use device drivers to configure and monitor hardware settings.

A. Management Requirements

Regarding manageability, our IPsec-gateway prototype has to meet numerous requirements: First, several providers are needed, which are used to manage the network device. It should be able to configure network settings, IPsec policies, and QKD and QIKE settings. Moreover, the system status and performance should be made available by the providers. Second, the handling of errors and failures must be provided in order to meet reliability demands. Therefore, configuration settings have to be stored and reloaded after a reboot of the system. Administrators should be able to get warnings and error messages by subscribing to CIM process indications. Third, a client application is needed that provides the ability to manage distributed gateways by setting different IPsec policies and network configurations and by monitoring system states, logs, faults, and warnings. The client should be able to connect to the gateways over a secure tunnel and should be authenticated before login.

B. Management Implementation

In order to guarantee interoperability between general-purpose clients and individual management implementations, CIM providers have to abide by a standardized schema. The CIM schema offers a common management representation for multiple WBEM/CIM solutions without knowing proprietary enterprise provider implementations. In order to meet all requirements to manage our embedded IPsec-gateway prototype, we have defined several models that fairly correspond to the model in the CIM schema v2.9.0. Essentially, the management of our prototype is composed of 59 CIM classes that are located in six different provider libraries. These are providers for networking, IPsec, QKD, QIKE, operating system, and error handling using syslog. The networking provider is used to manage resources like network interfaces, Ethernet ports, and IP-routing tables. For the routing table configuration, for example, instances of the CIM Network-Model like *CIM_NextHopIPRoute* and *CIM_RemoteServiceAccessPoint* have been created. Associations, which actually link instances of objects, have been used to establish a relationship between the routing entries and their gateways, interfaces, and the underlying host system. These are, for example, *CIM_AssociatedNextHop*, *CIM_RouteUsesEndpoint*, and *CIM_HostedRoute*. A cut-out of the network schema is given in Figure 3. For the network providers, 17 classes have been created of which 9 contain association properties and 3 support method invocations (see Table II for an overview of all implemented classes and their static disk-space consumption).

In order to realize IPsec policy configurations, we have created instances of the classes *CIM_Service*, *CIM_Settings*, and *CIM_Configuration*. The *IPsec_Service* instance allows the starting, stopping, and flushing of the Security Policy Database (SPD) that is maintained by Linux. One *IPsec_Setting* instance

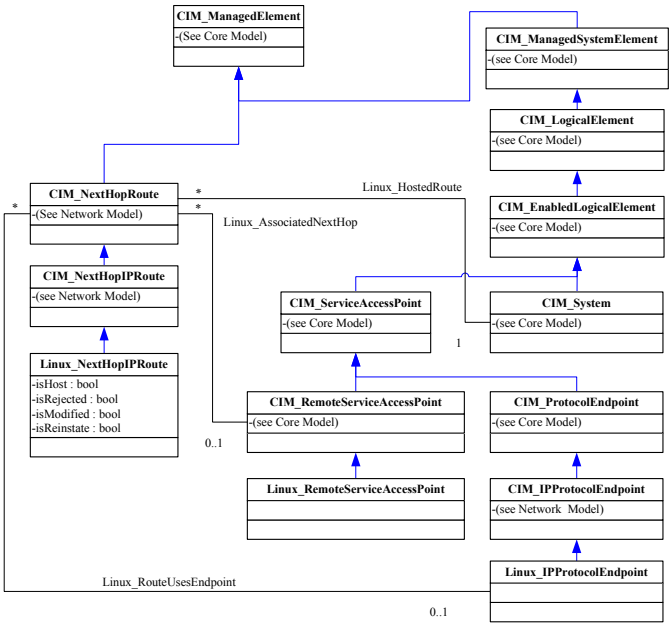

Fig. 3. Cut-out of the network CIM-schema constitution

represents one policy entry in the database that can be written into the kernel by simply running the *setkey* command. The parameters are thereby constructed by *IPsec_Setting* instances. The *IPsec_Configuration* instance stores information about the configuration-file location. The IPsec provider includes 3 classes whereas the *IPsec_Service* class additionally contains method properties.

The QKD and QIKE providers are used for the quantum-specific key distribution and key exchange between IPsec gateways. They allow the modification, addition, and deletion of different system configurations such as QKD key rate, authentication, and security association negotiation. They hold 25 classes. 3 support method invocations and 2 support event handling using indications. All settings are represented by instances of the *CIM_Setting* class. Instances of *CIM_Service* allow the starting and stopping of the QKD and QIKE services. The configuration data is represented by *CIM_Configuration* instances.

TABLE II
OVERVIEW OF THE IMPLEMENTED CIM PROVIDERS

	Instance	**Association**	**Method**	**Indication**	**Static disk space [kB]**
Network	17	9	3	0	312
IPsec	3	0	1	0	97
QKD	8	0	2	1	180
QIKE	17	0	1	1	368
OS	12	5	2	1	215
Syslog	2	1	0	0	30
Summary	59	15	9	3	1 202

For the operating-system management and the syslog-

message handling, we have made use of existing providers available from the SBLIM project [27]. The classes provide access to running processes, system parameters, statistical data, and syslog messages. Instances of the classes *CIM_ComputerSystem*, *CIM_UnixProcess*, *CIM_Processor*, *CIM_Card*, *CIM_OperatingSystem*, and *CIM_StatisticalData* are used. For our prototype, we have compiled 14 classes including 6 association classes, 2 classes that allow method invocations (starting and stopping of processes), and 1 that allows the indication of system parameters such as CPU speed and current memory consumption.

The error handling of our prototype has been realized as follows: First, all configuration data and settings of the providers are actually stored in files that are located in non-volatile memory (Compact Flash card). After an intentional or unintentional reboot of the system, all stored information is reloaded from these files. This enables data persistency and allows the recovery of stored configuration settings. Second, as soon as failures occur during run time, processes generate error messages that are reported to syslog. These messages are also stored on the Compact Flash card. Next to this, the messages are forwarded to a specific UDP Unix socket (port 49251). If WBEM clients have subscribed for indications, a separated thread is forked by the corresponding indication provider. This thread listens to the respective UDP port where the syslog messages are sent to. In the case of an error, indication instances, which are derived by the *CIM_ProcessIndication* class, are created and sent to the CIM server that again sends them to the client application over HTTPS (see Figure 4).

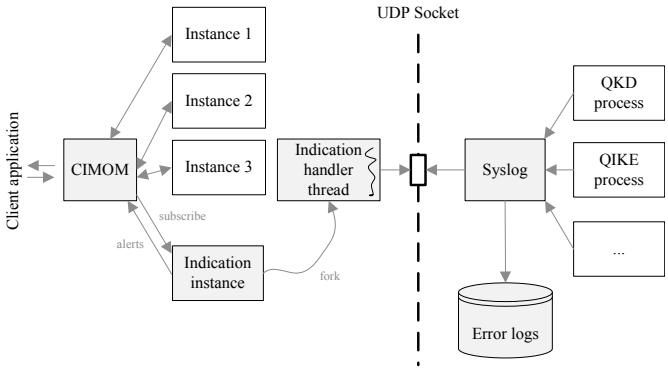

Fig. 4. Handling of errors

A graphical management interface has been implemented that is able to control IPsec-gateway instances. In Figure 5, a snapshot of the client is shown. The client allows to change the parameters of network interfaces (IP address, subnet mask, interface, and status), Ethernet ports (MAC address), routing table (destination and gateway addresses, subnet masks, interfaces, and different flags such as dynamic or modified routes), IPsec entries (source and destination addresses, ports, directions, protocols, modes, and tunnel sources and destinations), QKD and QIKE settings (quantum acquisition, cascade, confirmation, and privacy amplification), syslog messages and operating system handling (running processes, program execution, and memory consumption). The communication between the client and the IPsec gateways is done via SSL and basic authentication is used for authentication.

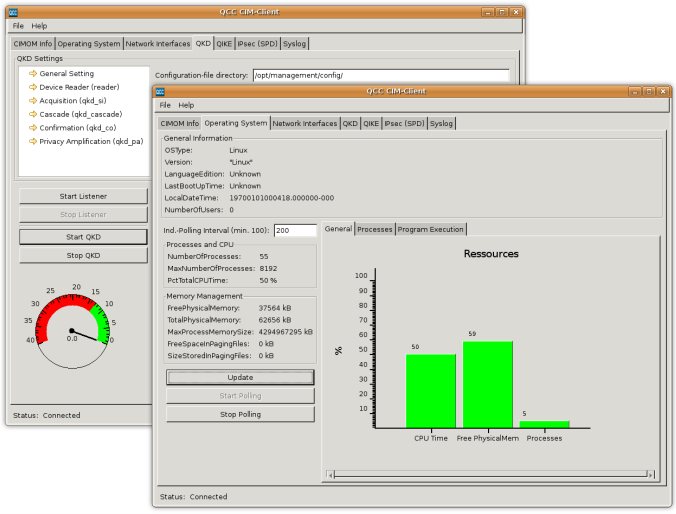

Fig. 5. Graphical Management Interface

VI. Performance Measurements

This section presents results of performed profiling of packet latency and CPU utilization of our WBEM/CIM-enabled prototype. The goal of the performance measurement has been to determine the characteristics of the embedded WBEM/CIM server. Furthermore, the efficiency and realizability of WBEM/CIM on these kinds of devices is treated.

Basically, SFCB has been designed to be highly modular so that it is possible to adjust the functionality that is definitely required. Thus, we have deleted unnecessary files, providers, and libraries, which are not used during operation. Furthermore, a special class provider of SFCB has been used that reduces the class-repository size by caching and compressing the data by a factor of about 5. Several compiler options have been set such as disabling the debug support or passing the object-file optimization flag (-Os) to the compiler. By applying these settings, we have been able to reduce the disk-space to the minimum of 899 kB.

In order to evaluate the run-time performance of our WBEM/CIM-enabled prototype, we have performed two different measurements: First, the running time (latency) of common CIM operations has been measured between our prototype and a general-purpose computer. The general-purpose computer has been programmed to send CIM messages to our WBEM/CIM-enabled prototype, to receive the CIM-message responses, and to analyze their latency. Both devices are connected by Gigabit Ethernet. The time between the first byte sent and the last byte received has been measured. Furthermore, each measurement has been performed several times in order to minimize outlier influences. Second, the same measurements have been performed on an SFCB instance that has been started on another general-purpose computer further

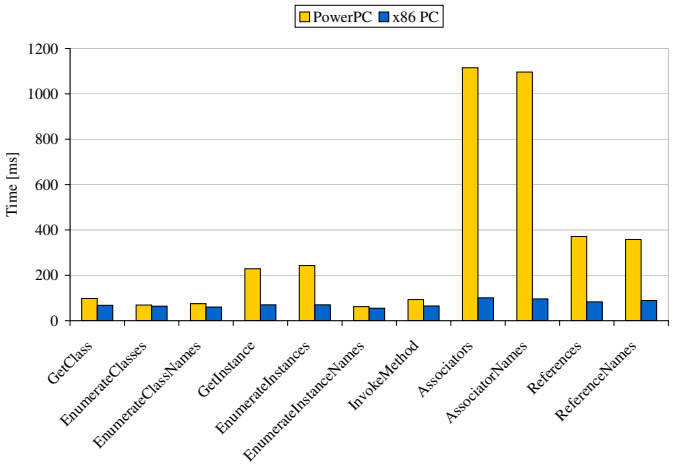

Fig. 6. Result of the measured running time of common CIM operations

TABLE III
RESULTS OF THE CPU-TIME PROFILING

	PowerPC			x86 PC		
	user [%]	**kernel [%]**	**idle [%]**	**user [%]**	**kernel [%]**	**idle [%]**
idle	2	1	97	2	1	97
enumerateClasses	37	59	4	47	9	44
enumerateClassNames	40	34	26	48	7	45
enumerateInstances	27	65	8	49	9	42
enumerateInstanceNames	24	35	41	48	9	43
getClass	43	29	28	46	7	47
getInstance	25	66	9	49	8	43
invokeMethod	22	57	21	49	6	45
associators	35	62	3	42	15	43
associatorNames	36	61	3	41	13	46
references	43	52	5	45	14	41
referenceNames	41	53	6	44	13	43

denoted as x86 PC. The used x86 PC has 1 GB of RAM and it offers a dual-core processor operating at a frequency of 2.8 GHz.

In Figure 6, the results of the running-time measurements of the CIM operations are shown. Both the running time between our prototype and the personal computer, and the running time between the x86 PC and the general-purpose computer are given. It shows that the running time of many CIM operations is nearly the same for both our prototype and the x86 PC. For our prototype, the associators or associatorNames operation, in contrast, leads to a much higher latency. This can be explained by the fact that many CPU cycles have to be spent for performing these operations and that our prototype only has a 300 MHz CPU compared to our dual-core x86 PC operating at 2.8 GHz.

In Table III, the performance of our prototype and the x86 PC is given. The Linux tool *top* has been used to measure the CPU load of the WBEM/CIM server. Therefore, each CIM operation has been executed in a loop while the needed CPU cycles have been recorded. The CPU cycles are further separated into cycles that have been required for processes operating in the user mode and for processes operating in the kernel mode. The idle-CPU value represents the percentage of time when the CPU has been waiting for network I/O requests. The results show that our prototype needs many more CPU cycles for processes that belong to the kernel. This is due to the inefficient network driver of our prototype. Nevertheless, for some CIM operations, some idle cycles have been left that emphasize the overall performance and show that the bottleneck is not the WBEM/CIM processing.

Similar performances have been obtained by non-WBEM solutions reported by, for example, G. B. Machado et al. [21]. As already described in Section II, they have developed a SOAP-based Web service using an ARM microcontroller. The embedded server has a total size of 65 kB and has four simple calculation operations implemented. These operations were adding, subtraction, multiplication, and division. The measured round-trip time for a 464 Bytes packet was approximately 156 ms. L. Pham et al. [20] also have performed performance measurements on three independent mobile devices that also implemented SOAP-based Web servers. The servers have less than 100 kB of static memory and about 300 kB of maximum run-time RAM. Their measured round-trip time was between 220 ms and 1 250 ms for a handheld device (PDA) connected to a PC via USB connection and between 310 ms and 26 560 ms for two different mobile phones communicating over a public GPRS-network provider. G. Pavlou et al. [28] have compared Web-service solutions with SNMP and Common Object Request Broker Architecture (CORBA). The memory footprint of their SNMP solution was about 1 980 kB, the CORBA platform needs around 10 000 kB, and their Web-service prototype needs about 4 000 kB. Of course, the size of the prototypes depends on the implemented functionality. The response times of their Web-service solutions are comparable to our WBEM prototype solution which lies around 100 ms.

However, a fair-minded comparison of those management solutions is largely infeasible. This is due to the fact that each solution offers different functionalities. WBEM has been designed for large-scale enterprise networks and therefore supports much richer features than customized Web-service solutions, which implement functionality for individual needs. Nevertheless, since WBEM and Web services are based on a very similar technology, a comparison of their performances becomes reasonable.

VII. CONCLUSIONS

This article presents the design and implementation of an embedded IPsec-gateway prototype that uses WBEM as a management solution. A small-footprint WBEM/CIM server has been used that needs less than 900 kB of non-volatile memory. It has been realized on a Xilinx Virtex-4 FPGA prototyping-platform that integrates a 300 MHz PowerPC. We have implemented several system providers and a client application which allows an easy way of embedded device management. Furthermore, we have provided memory and performance measurements of our implementation which consumes 2 MB of memory and answers most management

requests in about 100 ms.

During our experiments, it has been shown that the complexity and effort of managing our prototyping platform using WBEM has become relatively high due to the multitude of different CIM classes. The navigation through these elements, the configuration, and associations between various classes make it a real challenge for design engineers and network administrators. Although this complexity and effort may somewhere preclude the usage of WBEM in large-scale networks, it has still proven to be applicable in constraint platforms and that it can be integrated into networked embedded devices such as IPsec gateways that need to be managed.

Acknowledgment

The development and prototype implementation of the system has been performed within the project *QCC – Quantum Cryptography on Chip* (810195/1522) funded by FIT-IT System-on-Chip programme of the Austrian bm:vit ministry and the project *C@R – Collaboration at Rural* (IST-FP6-34921) funded by the European Commission.

References

[1] DMTF, "Web-Based Enterprise Management (WBEM)," http://www.dmtf.org/standards/wbem.

[2] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "RFC 2616: Hypertext Transfer Protocol – HTTP/1.1," Jun. 1999. [Online]. Available: http://www.ietf.org/rfc/rfc2616.txt

[3] J. Postel and J. K. Reynolds, "RFC 854: Telnet Protocol Specification," May 1983. [Online]. Available: ftp://ftp.math.utah.edu/pub/rfc/rfc854.txt

[4] I. T. U. (ITU), "ITU-T Recommendation M.3400 (2000): TMN Management Functions," February 2000.

[5] J. Case, M. Fedor, M. Schoffstall, and J. Davin, "RFC 1157: Simple Network Management Protocol (SNMP)," http://www.ietf.org/rfc/rfc1157.txt, May 1990.

[6] International Organization for Standardization (ISO), "ISO/IEC 9596-1: Information Technology OSI — Common Management Information Protocol (CMIP)," 1991.

[7] W3C, "SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)," http://www.w3.org/TR/soap12-part1/, 2007.

[8] Luc Clement and Andrew Hately and Claus von Riegen and Tony Rogers, "UDDI Spec Technical Committee Draft," http://www.uddi.org/pubs/uddi_v3.htm, 2004.

[9] W3C, "Web Services Description Language (WSDL) 1.1," http://www.w3.org/TR/wsdl, 2001.

[10] IETF Working Group, "Network Configuration (netconf)," http://www.ietf.org/html.charters/netconf-charter.html, 2008.

[11] Sun Microsystems, "Java Management Extensions (JMX) Technology," http://java.sun.com/javase/technologies/core/mntr-mgmt/javamanagement/, 2008.

[12] J. Schönwälder, A. Pras, M. Harvan, J. Schippers, and R. van de Meent, "SNMP Traffic Analysis: Approaches, Tools, and First Results," in *International Symposium on Integrated Network Management – IM, Munich, Germany, May 21-25*. IEEE, May 2007, pp. 323–332.

[13] A. Corente and L. Tura, "Security performance analysis of SNMPv3 with respect to SNMPv2c," in *Network Operations and Management Symposium, 2004. NOMS 2004. IEEE/IFIP*, vol. 1, April 2004, pp. 729–742.

[14] X. Du, M. Shayman, and M. Rozenblit, "Implementation and performance analysis of SNMP on a TLS/TCP base," in *International Symposium on Integrated Network Management Proceedings, Seattle, WA, USA, 14-18 Mai, 2001*. IEEE, Mai 2001, pp. 453–466.

[15] C. Pattinson, "A Study of the Behaviour of the Simple Network Management Protocol," in *Proceeding of 12th International Workshop on Distributed Systems: Operations & Management (DSOM?01), Nancy, France, 15-17 October 2001*. IEEE, October 2001.

[16] A. Pras, T. Drevers, R. van de Meent, and D. Quartel, "Comparing the Performance of SNMP and Web Services-Based Management," in *IEEE electronic Transactions on Network and Service Management*, vol. 1, no. 2, 2004.

[17] T. Fioreze, L. Z. Granville, M. J. B. Almeida, and L. R. Tarouco, "Comparing Web services with SNMP in a management by delegation environment," in *9th IFIP/IEEE International Symposium on Integrated Network Management, 15-19 May 2005, Nice, France.* IEEE, Mai 2005, pp. 601–614.

[18] G. Pavlou, P. Flegkas, S. Gouveris, and A. Liotta, "On Management Technologies and the Potential of Web Services," in *IEEE Communications Magazine*, vol. 42, no. 7. IEEE, July 2004, pp. 58–66.

[19] D. Schall, M. Aiello, and S. Dustdar, "Web Services on Embedded Devices," in *International Journal of Web Information Systems 2006*, vol. 2, no. 1, 2006, pp. 45–50.

[20] L. Pham and G. Gehlen, "Realization and Performance Analysis of a SOAP Server for Mobile Devices," in *11th European Wireless Conference, Nicosia, Cyprus, April*, vol. 2. VDE Verlag, April 2005, pp. 791–797.

[21] G. B. Machado, F. Siqueira, R. Mittmann, and C. A. V. e Vieira, "Embedded Systems Integration Using Web Services," in *Proceedings of the International Conference on Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies*. IEEE, April 2006, pp. 18–18.

[22] H.-T. Ju, M.-Y. Choi, and J. W. Hong, "An Efficient and Lightweight Embedded Web Server for Web-based Network Element Management," in *10th International Journal of Network Management*, vol. 10. John Wiley & Sons, Inc, September 2000, pp. 261–275.

[23] C. Kalbfleisch, S. Hunt, K. Low, and D. Mathieson, "Network Management of Real-Time Embedded Processors," in *Particle Accelerator Conference, Washington, DC, USA*. IEEE, 1993, pp. 1823–1825.

[24] S.-J. Lee, M.-J. Choi, S.-M. Yoo, J. W. Hong, H.-N. Cho, C.-W. Ahn, and S.-I. Jung, "Design of a WBEM-based Management System for Ubiquitous Computing Servers," Academic Alliance Paper Competition for 2004, Philadelphia, USA, September 2004.

[25] S.-M. Yoo, J. W.-K. Hong, J.-G. Park, C.-W. Ahn, and S.-W. Kim, "Performance Evaluation of WBEM Implementations," in *KNOM Review, Network Operations and Management, Pohang, Korea*, vol. Vol. 8, no. No. 2, February 2006, pp. 7–13.

[26] OpenPegasus, "OpenPegasus Website," http://www.openpegasus.org.

[27] Standards Based Linux Instrumentation, "Small Footprint CIM Broker (SFCB) Website," http://sblim.wiki.sourceforge.net/Sfcb.

[28] G. Pavlou, P. Flegkas, and S. Gouveris, "Performance Evaluation of Web Services as Management Technology," http://www.ibr.cs.tu-bs.de/projects/nmrg/meetings/2004/bremen/, January 2004, 15th Network Management Research Group (NMRG) Meeting.