Transparent, Messaging-Free Management Model For Programmable, Intelligent Network Elements

Optimum Communications Services, Inc.
info@ocsipholding.com

ABSTRACT

This paper presents a direct network element (NE) configuration and status file transfer routine based model for managing programmable NEs, avoiding complexity and restrictions of intermediate messaging and transaction protocols of conventional network management systems (NMS). The routine of transferring binary network management data files between NMS and NEs enables logical decoupling of the NMS server side and the NE side network management actions, allowing the NMS server software applications, the NE programmable hardware functionality and the NMS communications network to be implemented without dependencies from each others, as well as flexibly changing any of these elements without impacting others. The presented generic NMS architecture thus is well suited for managing programmable NEs of arbitrary functionalities. Moreover, programmable hardware enables cost-effective implementation of NEs that can be designed to be fully self-operating at any given application (due to not having to be over-functionalized for a variety of applications, as in the case of hardwired logic), thus allowing dynamic network operation under static management configuration. Accordingly, a highly reliable, scalable and flexible network and management system architecture is produced with a core NMS routine that operates effectively without a need for exception handling.

Categories and Subject Descriptors

D.3.3 [**Programming Languages**]: C.2.3 [**Network Operations**]: Network management.

General Terms

Design, Economics, Management, Performance, Reliability, Security, Standardization.

Keywords

Automated, transparent NMS, autonomous NEs.

1. INTRODUCTION

Traditional NMS (e.g. SNMP, TL1, CMIP) rely on messaging for transactions and communications between system elements. Conventional NMS transactions are normally event-triggered and can occur unpredictably, e.g. based on dynamic network events. For instance, network defect activations and deactivations cause NMS messages. Likewise, even basic NMS operations, such as accessing a NE status or control parameter conventionally involve messaging and command transactions, requiring protocol, message, language or data format conversions. Such traditional NMS have inherent problems:

- Performance degrades when NMS capabilities are most needed i.e. during bursts of messaging triggering events such as network failures.
- Components are often technology or implementation dependent, complicating system integration via requiring various stages of middleware, resulting in lost NMS transparency and reduced flexibility.
- Transactions are protocol, language or format specific, requiring protocol conversion agents etc., further complicating NMS implementations and reducing transparency and flexibility.
- Since NMS operations are based on predefined, technology-specific fixed set of commands or methods, functionality supportable through conventional NMS is restricted to a subset supported by each component.

Thus, even with their exhaustive implementational complexity, conventional NMS techniques are usually inefficient and restrictive in operation. To address these challenges, a streamlined NMS and programmable, self-operating NE architecture is presented, providing transparent and flexible network management with architecturally improved scalability, reliability and performance independent of system load.

2. TRANSPARENT NMS MODEL

2.1 Streamlined NMS for Programmable NEs

Programmability of NEs, i.e., the shift from discrete functionality NEs to non-discrete or even dynamic functionality NE, is commonly assumed to complicate the network management. However, this paper presents a model for actually simplifying NMS as well as NE software (SW), enabled in part by programmable NE hardware (HW).

Key differentiating attributes of such NMS architecture optimized for programmable, intelligent (i.e. autonomous) NEs include:

- Arbitrary variety of programmable NE functionalities and management thereof supported through generic NMS and NE SW;

- Dynamic network operation realized under static network management configuration;
- Simple, reliable, scalable and transparent NMS-NE file transfer routine based operation, with user (or client application) interface side as well as NE side actions occurring as automatic consequences of the contents of binary files transferred between NMS server and NEs, without a need for the NMS or NE SW to interpret contents of such files.

Fig. 1 below illustrates the presented NMS model, providing logical separation between independently occurring functional routines of 1) network management data (NMD) file transfer between NMS file server and the programmable NEs; 2) NE-side actions performed autonomously by (programmable HW of) the target NEs based on contents of the binary NMD files copied from NMS server; and 3) NMS-side actions occurring automatically based on contents of NMD files copied from the NEs.

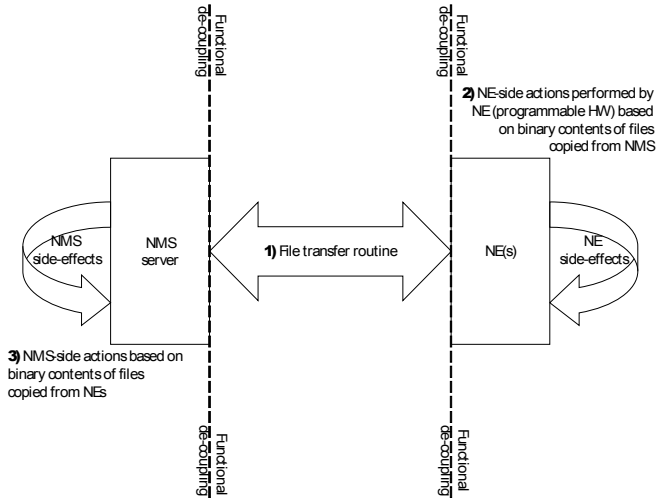

Figure 1. File transfer routine based NMS model.

Uniquely, compared to conventional messaging based NMS technologies, with the transparent NMS model illustrated in Fig. 1, the intended network management actions associated with the NMD files transferred from NMS server to the NEs occur as direct consequences of the NEs storing these files at their local memories, with such NE-side actions carried out automatically by the NE programmable HW, without a need for either the NMS or NE SW to process the contents of the NMD files. Likewise, the NMS GUI automatically displays current network status based on the NMD files transferred from the NEs to NMS file server, and performs any user notification of reportable network events as a direct consequence of values of pre-defined bit locations (e.g. NE alarm status bits) in the NMD files, again without a need for the NE SW to process contents of any NE status files, or for any event-based messaging or other exception case handling by the NE or NMS SW.

Accordingly, the transparent binary file transfer routine based NMS model per Fig. 1 allows unrestricted network management operations based on simple, repeating transfer of NMD files between NMS server(s) and programmable NEs. The presented transparent, messaging-free NMS model involves concepts initially published in [1].

2.2 Architecture Overview

Fig. 2 below illustrates the presented direct binary file transfer routine based NMS communications architecture.

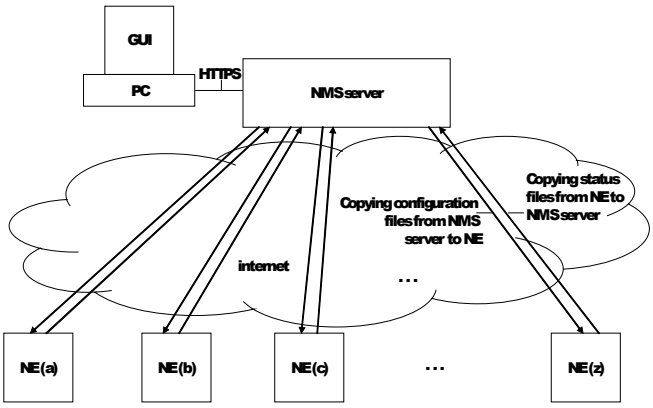

Figure 2. File transfer routine based transparent NMS architecture for programmable NEs.

Fig. 2 presents an overview of functional architecture of the presented network management process. At high-level, the NMS, via a set of automatic routines, transfers binary network management data (NMD) files between an NMS file server and a set of programmable NEs, while network management actions occur as automatic consequences of the contents of the NMD files.

The NMS process of Fig. 2 is based on the following mutually asynchronous and conceptually decoupled sub-processes (see also Fig. 1):

1) A set of automatic file transfer routines transfers NMD files between the NMS server and the NEs;

2) The NEs perform on their end the appropriate NMS actions associated with the NMD files;

3) The NMS GUI and any application software act on the NMD files at the NMS server, to perform the NMS transactions on its end.

The sub-process 1) is based on a secure Network File System (NFS), with the NMS server providing NFS server and the NEs NFS client functionalities. This sub-process further comprises the below two independent NMD file transfer routines that the NEs repeat periodically, e.g. every 1 or 10 seconds:

a) The NFS clients of the NEs look for and copy their associated NMD program and control files, referred to as NE configuration files, from NE-specific configuration file directories at the NFS server, over a network to their local memories.

b) In addition, the NFS clients at the NEs copy contents of their status register segments within their local memories via a type of NMD file referred to as NE status file over a network to their associated NE status directories at the NMS server.

The NMS communications network between the NMS file server and the NEs can be for instance the Internet.

The sub-process 2) is performed by the NE HW, e.g. per [2], automatically based on the binary contents of the NE control files, normally without further involvement by either NMS or NE SW. An exception to that is a case when an NE control file contents contain such a value in a particular NE control register, referred to as the reboot control register, that is intended to cause the NE SW to reboot, in which case the NE SW will do a reboot of the type indicated by the reboot control register value. Aside this reboot exception, the NE HW automatically, without SW involvement, completes the network management actions indicated by the contents of new NE control files copied to the control register segment in its local memory space. The NE also copies to its program memory segment within its local memory space any new program files from its associated directory at the NMS file server designated for program files for that destination NE. The program memory segment of an NE comprises multiple directories to allow storing multiple NE program files, and the value of the NE reboot control register indicates both whether the NE is to reboot, and the directory (in the program memory segment of the NE) for the program files with which to reboot. In addition, the NE HW automatically maintains and updates a set of NE status parameters in its status register memory space, and the NE SW reads the contents of this status register segment in the NE memory space to a NE status file that the NE copies to an appropriate directory at the NMS server designated for status files from that source NE. The NE copies also the contents of its control registers via its NE status file back to the NMS server, allowing the user to verify the actual values of also the NE control registers via GUI. Hence, the phrase NE status file herein refers to the contents of both the NE control and status registers, collectively referred to as NE device registers.

The sub-process 3) is performed by the NMS GUI SW via providing access in a human understood format for the system user to the NMD files at the NE-specific directories at the NMS file server. This sub-process involves read access to the status register values within the NE status files, write and read access to control register values within the NE control files, and producing NE program files to appropriate folders at the file server. Moreover, the NMS GUI displays notifications of significant events in the network such as NE alarm activations according to principles per [2] and [3].

It is seen from the above discussion that the three main sub-processes of the NMS process are mutually decoupled, other than through the contents of the NMD files (transferred between NMS server and NEs) that indicate the intended actions to be performed at the other end of the system to complete any given network management operation.

Compared against conventional messaging and command based NMS techniques, this decoupling between the functional elements of the network management system yields several architectural benefits, including:

- Reliability and scalability: High load of NMS or network event activity on one element of the system does not negatively interfere with other elements. For instance, while e.g. the NMS server is heavily loaded during for instance a network service contract testing period when the NE control parameters are changed rapidly for test case purposes, the file transfer routine, the SW of the NEs, and even the HW of those NEs not under the test, are not at all impacted. Likewise, a heavy load of e.g. network defect activations and de-activations at a given NE does not impact the NMD file transfer routines, the other NEs, the NMS server or GUI SW; instead, e.g. per [2] and [3], just a single NE alarm notification is generated at the NMS GUI when a previously defect-free NE enters a defected state. (Note that 'NE' here can refer also to e.g. group of nodes, sub-networks, contract network groups etc. hierarchically.) As a consequence, such NMS is highly reliable and scalable, providing predictable, steady performance under any load of NMS and network event activities.

- Transparency: The architecture (per Fig:s 1 and 2) provides transparent NMS communications all the way from the NMS GUI to the NE HW device registers and back, without intermediate messaging protocol conversion or command translation agents etc. non-transparent middleware common with traditional NMS communications techniques. Accordingly, this NMS model inherently enables a more intuitive and flexible network management, by allowing direct access to the NE parameters of interest via an intuitive and transparent GUI, without requiring the network operator's personnel to know about or deal with the details of any intervening messaging protocols, command language syntaxes etc. such technicalities.

- Versatility and extensibility with simple core functionality: The NMS is flexible regarding any changes needed to the implementation of either the NE, NMS server, GUI etc. elements of it, as well as any changes to the network through which the NMS and NEs transfer files, or to the way the GUI and the NMS file server communicate. Consequently, any of these system elements can change without the need to redesign the rest of the NMS system.

The full NMS further comprises PC(s) or terminals hosting the NMS GUI application, e.g. HTML based web browser. In such a system implementation, the GUI connects to the NMS server over a secure HTTP connection. Regarding Fig. 2, note that there is no implied limit to the number of NEs supported by this network alarm monitoring system, but that instead this system architecture supports an arbitrary number of NEs, and that there can equally well be multiple physical NMS server and user computers as well as multiple concurrent NMS GUI applications.

2.3 NMS Server Information Architecture

Fig. 3 is a diagram of a logical directory structure at the NMS file server, in the context of Fig. 2.

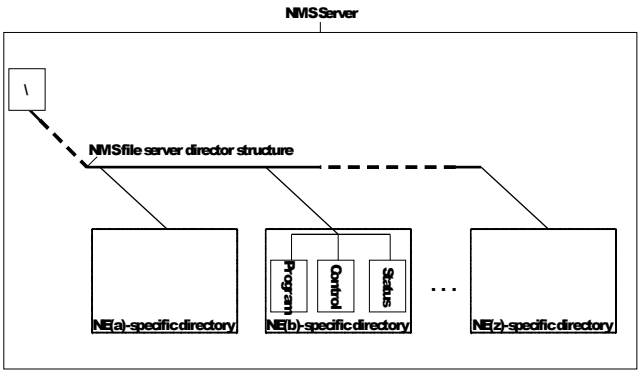

Figure 3. NMS server information architecture for the transparent, file transfer based NMS model.

Fig. 3 illustrates a logical directory structure at the NMS file server for storing NMD files for a set of NEs managed through the NMS. Below the file server directory root, there is a directory structure holding a set of NE-specific directories. Each of the directories stores NMD files for its associated NE. Each directory comprises subdirectories for holding program, control and status files, respectively, of the NE associated with the directory.

Operation of the NMS file server in a process of configuring and monitoring a given NE is based on the below principles:

1) A system user, e.g., a network operator staff member, and/or client application software produces desired types of NE program and control files for a NE, using the NMS GUI client and related server SW, into the program and control file directories associated with the NE at the NMS file server.

2) The NE, via a repeating routine, for instance every ten seconds, looks for and copies these files from its associated directories at the NMS server to their appropriate locations within the local memory space of the NE. The NE will consequently autonomously complete on its end the NMS operations indicated via each new NE configuration file.

3) The NE, also via a repeating routine performed e.g. once every second, copies the contents of its device status registers via its NE status file to the status folder at its directory at the NMS server. The NMS SW will consequently display NE status data, along with a new NE alarm notification as necessary, to the user via the GUI, based on the contents of the latest NE status file at its associated directory at the NMS server, for instance utilizing the network alarm monitoring principles per [2] and [3].

The management process for a group of NEs is based on simply repeating, or executing in parallel, the management process of a single NE described above. Copying of files between directories at the NMS server and the NEs is based on secure NFS, e.g. NFS version 4. Also, each NE HW unit is identified by its unique NE ID number configured at the factory on a non-volatile memory, e.g. flash drive, for each NE unit, and the names of the NE-specific directories at the file server include the NE ID of their related NEs, based on which each NE knows to access its appropriate directory at the NMS server.

The need for identifiers for source, destination, message or transaction is avoided for NMS communications between the NMS server and the NEs, in part via the use of NE-specific directories at the NMS file server for storing the NMD files associated with each one of the NEs. Note that transaction, source, destination etc. identifiers are usually necessary with customary NMS communications schemes, per the common messaging protocols (e.g. SNMP, CMIP, TL1 etc.), requiring related NMS messaging protocol processing to be performed by conventional NMS and NE devices, thus making the traditional network management systems implementation, operation and administration more complicated and less flexible compared to the plain binary NMD file transfer based NMS communication presented herein.

Besides its more straightforward and transparent implementation and more flexible and intuitive operation, benefits of the presented NMS model enabled via the NE-specific NMD file folders include elimination of NMS communications overhead that is needed with conventional NMS messaging protocols, and the clarity and intuitiveness of the NMS file server structure based on a repeated set of similar NE-specific directories for the set of NMD files per each NE.

2.4 NE Information Architecture

Fig. 4 is a diagram of a logical structure of a local memory space of an NE (in the context of Fig. 2).

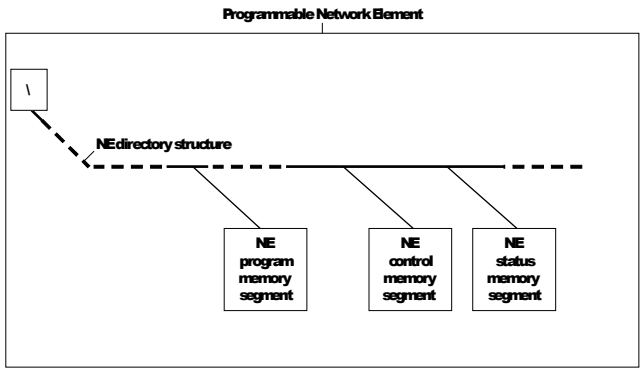

Figure 4. Information architecture of a programmable NE managed through the transparent NMS model.

Fig. 4 illustrates a logical structure of the local memory space of each NE of the network management system per Fig. 2. The embedded memory space of the NE comprises a program memory segment, a control register segment and a status register segment. NEs can have various other memory segments, e.g. RAM, in addition to the three segments shown in Fig. 4, and there can be as many layers of hierarchy of NE logical directory structure below its root as desired, as well as the shown memory segments can have sub-directories. Reference specifications for the embedded system architecture of an example of an autonomous, programmable NE, including NE device register descriptions with related application notes, are provided in [2].

The NE memory space is organized as a logical directory structure, with the NE program memory segment forming a logical subdirectory at the NE for holding the NE program files, and the NE control register segment and the NE status register segment each forming binary files under the NE logical directory structure. The NE may comprise a HW unit with an embedded microprocessor and a set of embedded memories organized from the NE SW perspective as a continuous directory structure.

In an example NE per [2], the NE program memory directory is a flash drive, and the NE control and status files are predefined address ranges within the embedded memory space of the NE microprocessor containing the NE device control and status registers, respectively. Furthermore, the NE device registers are implemented within a programmable logic device that is configured, at least in part, via the NE program files stored at the directory. The programmable hardware logic of such autonomous NEs completes on the NE side the network management operations indicated via each new NE control file, as well as produces and keeps updated a predefined set of NE status parameters on the NE status file. Such NEs are capable of operating autonomously and dynamically, even with NE program and control files that are static for a duration of a network service contract that a given set of NEs are deployed for.

The HW of a programmable NE implementation comprises, besides the embedded microprocessor and its memories, a programmable logic device (PLD, or FPGA i.e. Field Programmable Gate Array) within which, both the microprocessor as well as the HW logic, including the NE device registers, can be included. With such re-programmable NE HW, the NE program files in the flash drive comprise both a binary file for configuring the programmable HW logic of the NE (its PLD/FPGA), as well as a binary executable program for the NE microprocessor.

The NE SW executes periodically, e.g. once every ten seconds, a repeating routine comprising the below steps:

1) The NFS client of the NE looks for and copies to its local memory segments new NE program and control files from its associated directories at the NFS server of the NMS server computer;

2) The NE HW automatically completes the NMS actions indicated via new NE control files;

3) The NE NFS client copies its status file to its associated directory at the NFS server.

While the step 2) generally is performed by the NE HW, the NE SW however checks the value of a particular address in the control register segment referred to as the reboot control register after it has copied a new NE control file from the NMS server. In case that the reboot control register was set to a value indicating NE reboot action, the NE SW will perform the type of NE reboot specified by the value of the reboot control register. However, as a general rule, i.e., in cases that the reboot control register was not set in its active value, the NE HW will complete all the network management operations indicated by each new NE control file automatically without any SW involvement.

Benefits of this NE functional architecture include that the NE SW does not need to process the NE control or status files or to perform related consecutive actions (other than checking the reboot command register in the NE control files and rebooting the NE as necessary). Consequently, the processing load for the NE SW is significantly reduced through elimination of intermediary processing stages of conventional NMS communications, while the entire system operation is made faster, transparent and more predictable and reliable via programmable HW based automation.

2.5 Autonomous, Programmable NEs

It is finally noted that the remote re-programmability of NE HW logic functionality, enabled by PLD technology and the NMS concepts such as the ones discussed herein, enables cost-efficiently designing and implementing highly intelligent NE HW capable of dynamic operation without a need for any SW involvement after initial configuration for a given application.

The high level of intelligence in such fully-programmable NE HW is made economical due to that, unlike with non-programmable HW where the hardware logic typically needs to be designed for a variety of different potential applications and operating modes, with NE HW whose logic can be reprogrammed for a variety of operating applications, it is expedient to use a generic physical HW design with a PLD (such as high capacity FPGA) as the differentiating component, to be programmed with single-application logic loads.

That way, any individual logic design can often actually become less complex, while achieving higher level of system intelligence and performance with plain hardware logic operation than what would conventionally be realizable with combinations of HW and various layers of SW. Furthermore, such fully HW-logic automated, self-operating NEs can achieve truly realtime dynamic operation, synchronously even over wide area network distances among a number of independently timed nodes, and even down to individual network data plane bit timeslot granularity.

Reference system specifications for such autonomous, programmable NEs, for example applications of transport network throughput maximization based on realtime data load variations between a number of routing nodes, and routing, switching and forwarding table free packet delivery network model, are provided in [4] and [5], and [6] and [7], respectively.

3. CONCLUSIONS

The presented NMS model provides direct, binary file transfer based network management communication that avoids the complexity and restrictions of intermediate messaging protocols or transaction languages and conversions thereof. Since the entire NMS operation is based on a repeating file transfer routine without exception cases, any dynamics of NMS application software, NE hardware and NMS communications are effectively decoupled from each others. Thus, unlike with conventional, event-triggered NMS, the presented NMS is inherently highly reliable, such that its performance does not degrade during periods of high load of network event activity. This NMS functional architecture is particularly effective with programmable, intelligent NE HW capable of autonomous operation without SW involvement i.e. under static NMS configuration. Such messaging-free NMS communications architecture provides transparent, file transfer routine based highly automated, scalable and reliable operation, with both NMS server as well as NE side actions occurring as direct consequences of the contents of the binary files transferred between NMS and NEs, without any intermediate processing, data format conversions etc. As a result, the presented NMS model, based on generic NMS and NE SW acting as transparent binary file management and transfer agents, enable managing programmable NEs with unlimited range of functionalities.

4. REFERENCES

[1] Sandstrom, M. 2006. U.S. Patent Application #11/566,178. Direct Binary File Transfer Based Network Management System Free of Messaging, Commands and Data Format Conversions.

[2] Sandstrom, M. 2006. U.S. Provisional Patent Application #60/866,208. Intelligent, Self-Operating Network Element.

[3] Sandstrom, M. 2006. U.S. Patent Application #11/563,079. Intelligent Network Alarm Status Monitoring.

[4] Sandstrom, M. 2002. U.S. Patent #7,333,511. Dynamically Channelizable Packet Transport Network.

[5] Sandstrom, M. 2009. U.S. Patent Application #12/363,667. Network Data Transport Multiplexer Bus With Global And Local Optimization Of Capacity Allocation.

[6] Sandstrom, M. 2002. US Patent #7,254,138. Transparent, Look-up-free Packet Forwarding Method for Optimizing Global Network Throughput Based on Real-time Route Status.

[7] Sandstrom, M. 2009. U.S. Patent Application #12/390,387. Packet-Layer Transparent Packet-Switching Network