OWL-Based Node Capability Parameter Configuration

Patcharee Thongtra, Finn Arve Aagesen, Kornschnok Dittawit

OWL-based Node Capability Parameter Configuration

Patcharee Thongtra, Finn Arve Aagesen, Kornschnok Dittawit

Department of Telematics
Norwegian University of Science and Technology (NTNU)
N7491 Trondheim, Norway
{patt, finnarve, kornschd}@item.ntnu.no

Abstract. Node capability parameter configuration is the validation and settings of node capability parameter values according to node capability parameter configuration specification (CapSpc). A node capability is a property of a node required as basis for service implementation. This paper presents a Node Capability Parameter Configuration System (CapCon). Node Capability Ontology (CapOnt) is the basis for CapCon. This paper has focus on CapCon in network management. CapSpc specifies required types, parameters, and parameter values for the node capabilities. OWL and OWL/XDD are used to represent the ontology concepts. The NETCONF framework is applied for the network management functionality. A prototype implementation and a case study including experimental results are presented.

Keywords: Network Management, NETCONF, Capability Configuration.

1 Introduction

Network management has been a topic of study for some decades. Some important network management frameworks are SNMP (Simple Network Management Protocol) [1], WBEM (Web-Based Enterprise Management) [2], WS-Management (Web Services Management) [3] and NETCONF (Network Configuration Protocol) [4]. In this paper, we apply the SNMP concept MIB as a generic concept for the system of managed objects.

A *node capability* is generally defined as a property of a *node* applied as basis for service implementation. *Node capability parameter configuration* is the validation and settings of node capability parameter values according to a *node capability parameter configuration specification* (CapSpc). CapSpc specifies required types, parameters, and parameter values for the node capabilities of all nodes in a domain. Existing network management frameworks as mentioned above do not provide functionality for automatic node capability parameter configuration.

This paper presents a *Node Capability Parameter Configuration System (CapCon).* CapCon identifies which nodes are capable/incapable to meet, or have already met, the requirements of CapSpc. CapCon can set the node capability parameter values on the nodes that can meet the requirements, while generating a report containing node

capability types installation instruction for the nodes incapable of satisfying the requirements.

Ontology is a formal and explicit specification of shared concepts [5-6]. Node Capability Ontology (CapOnt) enables the definitions of non-rule-based and rule-based concepts. CapOnt is the basis for the definition of CapSpc. Some MIB concepts will be a subset of CapOnt, but represented differently. There is accordingly a need to transform between CapOnt and MIB. CapCon can be integrated with various network management frameworks. In this paper NETCONF protocol with NETCONF agents are applied. NETCONF was designed for managing configuration data. NETCONF, however, does not provide functionality to efficiently manage configuration data for a set of nodes.

The rest of the paper is organized as follows. Section 2 presents an overview of NETCONF. Functionalities and system architecture of CapCon are described in Section 3. Section 4 presents CapOnt, comprising the concepts, the representation and storage of the ontology concepts, as well as the transformation between the CapOnt concepts and YANG-based concepts. Section 5 presents a prototype implementation of CapCon. The CapOnt node capabilities and the NETCONF MIB considered are based on the SNMP IF-MIB [7]. NETCONF agent is based on the YUMA toolkit [8]. The functionality of YUMA agent was extended to support IF-MIB YANG module [9] and also to enable registration and de-registration of nodes. Section 6 presents related work and Section 7 gives summary and conclusions.

2 NETCONF Management Framework

NETCONF management framework follows the manager/agent model. Managed objects, however, are divided into configuration data and state data. *Configuration data* is writable and is required for transforming a device from its initial default state into the desired state. *State data* is read-only status information and statistics. In the following short descriptions of NETCONF capability, NETCONF configuration datastore, NETCONF protocol, and YANG are given.

NETCONF capability is a set of functionalities that are implemented by both manager and agent. The NETCONF capabilities are declared in <hello> messages sent by each peer during the session initialization. The base NETCONF capability is the minimum set of functionalities each peer must implement. In addition to the base NETCONF capability, there is a set of standardized NETCONF capabilities which the NETCONF agent may support, e.g., :candidate and :notification. The :candidate NETCONF capability indicates that the agent supports the candidate configuration datastore (see below). The :notification NETCONF capability indicates that the agent supports the basic notification delivery mechanisms.

NETCONF configuration datastore is a complete set of configuration data. NETCONF has *running*, *startup*, and *candidate* configuration datastore. *Running* configuration datastore represents the currently active configuration, while *startup* configuration datastore is the configuration that will be used during the next startup.

Candidate configuration datastore represents a configuration that may be prepared and later committed to become the new running configuration datastore. Only the running configuration datastore is present in the *base* NETCONF capability. The other configuration datastores can be defined by additional NETCONF capabilities.

NETCONF protocol provides a simple RPC mechanism to install, manipulate, and delete configuration data. It has four layers: *content*, *operation*, *RPC*, and *transport*. *Content* layer represents the managed objects. *Operation* layer defines operations that can be invoked as RPC methods such as *get* operation to retrieve the managed object instances and *edit-config* operation to modify a configuration datastore. *RPC* layer provides a transport-independent framing mechanism. Several transport protocols can be used, e.g. SOAP, SSL, and BEEP.

NETCONF MIB concepts are represented by YANG [10], but are still partly based on SNMP MIB. YANG-based MIB is structured into modules. Objects in each module are modeled as YANG nodes in a hierarchical tree. YANG nodes are of four types: *container*, *list*, *leaf* and *leaf-list*. A *container* node as well as a *list* node is an internal YANG node that has no value apart from a set of child nodes. A *leaf* node as well as a *leaf-list* node has a value but no child nodes. A container node and a leaf node have at most one instance. A list node and a leaf-list node may have multiple instances. YANG modules supported by a NETCONF agent are indicated via the <hello> messages. An MIB compiler called libsmi [11] has been implemented to translate SMIv2 MIB modules to YANG modules. As an example of a SNMP-based *NETCONF MIB*, a part of *IF-MIB YANG module (NETCONF IF-MIB)* translated from SNMP IF-MIB [7] is presented in Fig. 1.

```
module IF-MIB {namespace "urn:ietf:params:xml:ns:yang:smiv2:IF-MIB"; prefix "if-mib";
container interfaces {
leaf ifNumber {type int32; config false; .. }
list ifEntry {key "ifIndex"; leaf ifIndex {type if-mib:InterfaceIndex; .. }
leaf ifType {type ianaiftype-mib:IANAifType; config false; .. }
leaf ifSpeed {type yang:gauge32; config false; .. }
leaf ifAdminStatus { type enumeration {enum up {value 1;} enum down {value 2;} enum
testing {value 3;}} config true; .. } .. }}} .. }
```

Fig. 1. A part of IF-MIB YANG module.

3 Functionalities and System Architecture

CapCon architecture as illustrated in Fig. 2 has three functionality components, two repositories and NETCONF agents. The functionality components are Node Capability Administrator *(NCA)*, Node Capability Monitor *(NCM*) and Node Capability Parameter Modifier *(NCD). NCA* is concerned with the registration and de-registration of nodes and their node capabilities. *NCM* monitors node liveness and node capability- types, parameters, and parameter values. *NCD* is responsible for the

validation and settings of node capability parameter values according to CapSpc. The repositories are *Node Capability Type Repository* (NCRep) and *Inherent Node Capability Repository* (INRep). NCRep stores the ontology concepts. INRep stores the existence and liveness of nodes as well as inherent node capabilities. *Inherent node capabilities* are observed node capability *instances*. The NETCONF agent registers and de-registers with NCA using <register> and <deregister> messages. Both messages contain the node IP address.

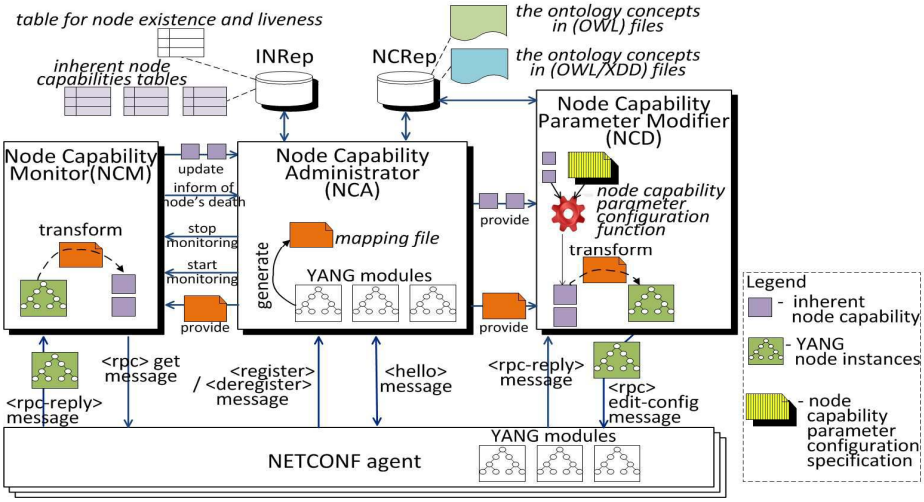

Fig. 2. CapCon Architecture.

NCA creates INRep. First, NCA creates tables for storing the inherent node capabilities and a table for storing node existence and liveness in INRep. Then, if NCA gets a <register> message from a NETCONF agent, it will record the node's existence. If NCA gets a <hello> message from a NETCONF agent, it will parse the message and identify which YANG modules are supported. NCA will load these YANG modules into its memory and create a mapping file for the transformation between CapOnt and MIB. This mapping file will be sent to NCM and NCD. Also, NCA will send a monitoring request to NCM. If NCA gets a <deregister> message from a NETCONF agent, it will remove the node existence record and send a request to NCM to stop monitoring the node liveness and the inherent node capabilities. In addition, NCA provides the current view of inherent node capabilities in INRep to the other functionality components.

NCM gets monitoring requests from NCA and regularly sends an <rpc> message containing the NETCONF get operation to the agent. Then, if NCM gets an <rpc-reply> message with YANG node instances, it will inform NCA that the node is alive. Also, NCM will use the mapping file to transform the YANG node instances to the inherent node capabilities, and update NCA of such inherent node capabilities. If NCM does not get a <rpc-reply> message for a certain period, it will inform NCA that the node is dead.

NCD regularly identifies which registered nodes are capable/incapable of meeting, or have already met, the requirements of node capability types, parameters and

parameter values. A node having the required types, parameters and parameter values have already met the requirements; a node having only the required types and parameters are capable of meeting the requirements. On the other hand, a node that lacks the required types and parameters are incapable of meeting the requirements. NCD sets the node capability parameter values for those nodes that can meet the requirements, while generating a report for further installation of the required node capability types for those previously incapable of satisfying the requirements. NCD operations are based on Equivalent Transformation (ET) [12]. CapSpc, CapOnt and the inherent node capabilities are inputs. Outputs are one or more of nodes' IP address and the NETCONF edit-config operations with node capability parameter instances and required values. NCD will use the mapping file to transform the node capability parameter instances to YANG node instances, and send the <rpc> messages to the corresponding agents. Each of these messages contains the NETCONF edit-config operation with the YANG node instance.

4 Node Capability Ontology

This section defines CapOnt concept structure as well as the language representation and storage of these concepts. The projection of MIB concepts into CapOnt as well as the transformation between CapOnt and MIB concepts is also considered.

4.1 Upper Node Capability Ontology

The generic structure of CapOnt is illustrated in Fig. 3. This generic concept structure is denoted as *Upper Node Capability Ontology*. *Inference* parameters define logical relations to other node capability types. Parameters can be defined by *rules*. The relations between the node, node capability types and node capability parameters are referred to as: *ownership relation* between the node and node capability type, *constitution relation* between two node capability types; and *description relation* between the node capability- type and parameter. A *service management function* is an action with constraints and is defined by a rule. For a specific system, specific CapOnt concepts are defined.

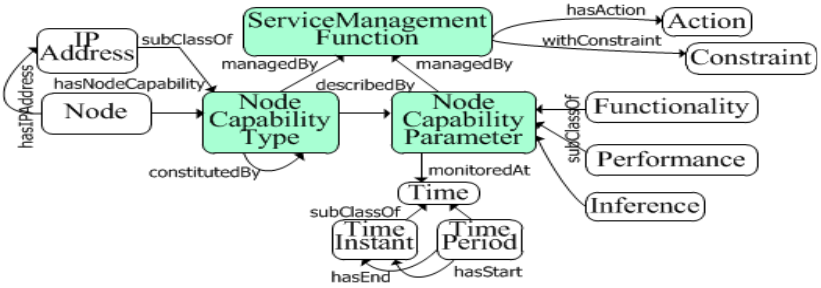

Fig. 3. Upper Node Capability Ontology.

4.2 MIB Concepts in CapOnt

NETCONF MIB concepts are represented by YANG, but are partly based on SNMP MIB. SNMP MIB objects will in CapOnt be represented as node capability types and non-rule-based parameters. SNMP MIBs are structured in a logical tree. SNMP tables are defined by managed objects on 3 MIB node levels. The top level node is here denoted as a *table root node*, the second level node as a *table access node*, and the lowest level nodes as *columnar leaf nodes*.

To create CapOnt from a SNMP MIB group, we consider three cases: 1) The inner node X is neither a table root node nor a table access node, 2) The leaf node Y is not a columnar leaf node, and 3) The inner node Z is a table root node with a table access node A and columnar leaf nodes {Ci}. For Case 1) the pair of connecting inner nodes Xi and Xj is projected to two node capability types (Xi and Xj) with a constitution relation from Xi to Xj. In Case 1) the pair of inner node X and leaf node Y is projected to a node capability type X, a non-rule-based parameter Y, and a description relation from X to Y. In Case 3) the pair of inner node X and table root node Z with a table access node A and columnar leaf nodes {Ci} is projected to two node capability types (X and A), non-rule-based parameters {Ci}, and there is a constitution relation from X to A, and description relations from A to each Ci.

4.3 Representation and Storage of CapOnt

Non-rule-based concepts are expressed by OWL [13]. OWL is a standard Web ontology language, which provides a rich set of constructors to define concepts. However, OWL is limited when it comes to describe rule-based concepts. So, *rule-based concepts* are expressed by OWL/XDD [14]. OWL/XDD extends ordinary XML-based elements by incorporation of variables and rule-based concepts. A rule-based concept in CapOnt is expressed by an OWL/XDD XML clause of the form:

$$H \rightarrow B_1, .., B_m, \{C_1, .., C_n\} \quad (1)$$

where $m, n \geq 0$, H and B_i are XML expressions, and each of C_i is a pre-defined XML condition on the clause. H is called the *head* of the clause while the set of B_i and C_i is the *body* of the clause. An XML expression is an XML-based element or document embedding zero or more variables. The upper ontology concepts as well as the CapOnt concepts for the prototype framework described in Sec.5 are given at http://tapas.item.ntnu.no/wiki/index.php/CapOnt.

CapSpc is based on CapOnt concepts. CapSpc is expressed by one or more OWL/XDD XML clauses, where XML expressions of these clauses are instances of the CapOnt concepts with variables.

The ontology concepts are stored in NCRep. The inherent node capabilities are *instances* of the *CapOnt non-rule-based concepts*. They are stored in a relational database, and are inputs for dynamically setting variables in the CapOnt rule-based concepts with suitable values. NCA is responsible for creating tables in the database. These tables denoted as *inherent node capabilities tables* are generated from the *representation* of the *CapOnt non-rule-based concepts*, using the *rules i)-iv)* below.

i). An *OWL Class class_x* will be mapped to a *Table class_x*, if it has either one of the properties: hasNodeCapability, constitutedBy, or describedBy. The Table class_x is assigned an auto-numbered Primary key named ID and a TIMESTAMP Column named *MonitoredAt* for recording the row-creation/update time. The Table "Node" is also assigned a Column IPAddress.
ii). For Class(*class_x* .* restriction(*property_i* allValuesFrom(unionOf(*set_of_classes*))) .*); *class_y* ∈ set_of_classes, *Foreign key class_x_ID* will be generated in *Table class_y* referring to the Primary key in *Table class_x*, if property_i ≡ hasNodeCapability || constitutedBy.
iii). For Class(*class_x* .* restriction(*property_i* allValuesFrom(unionOf(*set_of_classes*))) .*); *class_y* ∈ set_of_classes, *Column class_y* will be generated in *Table class_x*, if property_i ≡ describedBy.
iv). For Class(*class_y* .* restriction(*property_i* allValuesFrom(*primitive_datatype*)) .*), *primitive_datatype* will be converted to an SQL primitive datatype and become the *Column class_y* datatype, if property_i ≡ hasValue.

The expression (2), used in the rule ii) – iv), expresses a restriction of an OWL Class *class_x* on *property_i*, in which the property's range is restricted to *property_range_j*.

Class(*class_x* .* restriction(*property_i* allValuesFrom(*property_range_j*)) .*); (**2**)
property_range_j ≡ unionOf(*set_of_ classes*) || *primitive_datatype*

The node *instances* and the node capability type *instances* are stored in the *Tables* generated from the rule i). The non-rule-based parameter *instances* are stored in the *Columns* generated from the rule iii).

4.4 Transformation between CapOnt and NETCONF MIB

NCA is responsible for creating a *mapping file* to be used for the transformation between *CapOnt* and the corresponding YANG nodes in NETCONF MIB. A capability type, which is constituted by at least another node capability type, is mapped to a YANG *container* node. A capability type, which is not constituted by others, is mapped to a YANG *container* node or a YANG *list* node. A non-rule-based parameter is mapped to a YANG *leaf* node.

A YANG module defines a tree of YANG nodes. A *path* defines the YANG node's position relative to the root. NCA assigns a path to all YANG nodes, defined as a sequence of YANG nodes' types and names, separated by slashes, starting from the root to the YANG node assigned the path. NCA finds the corresponding YANG node for each of the node capability types and non-rule-based parameters by matching the ontology concept's name and the YANG node's name. When SNMP MIB managed objects are the basis of CapOnt concepts, there is a one-to-one mapping between these concepts and the YANG nodes. Otherwise, other logics for matching are required. A set of pairs of an OWL Class, expressing a node capability type or a non-rule-based parameter, and a YANG node's path is then generated and stored in the mapping file. With respect to the IF-MIB YANG module example in Fig. 1, an example pair for the parameter *ifAdminStatus* and the corresponding leaf node *ifAdminStatus*, locating

under the list node *ifEntry* and the container node *interfaces* is {<Class rdf:ID="ifAdminStatus">, container interfaces/list ifEntry/leaf ifAdminStatus}.

5 A Prototype Node Capability Parameter Configuration System

5.1 General

The system consists of a load balancer and a cluster of web servers as illustrated in Fig. 4. The load balancer captures the HTTP-based user requests and forwards them to connected web servers on a round robin basis. The number of connected web servers during an interval is dynamic and depends on the total number of user requests during the previous interval. CapCon connects and disconnects web servers to/from the load balancer. Web servers will go into hibernation for energy conservation in case of an inactive time period, i.e., no incoming user requests forwarded from the load balancer or user requests have already been responded.

A prototype is implemented using JAVA programming language. NCA, NCM, and NCD are implemented based on TAPAS platform [15], which enables NCA, NCM, and NCD to be instantiated and executed in different nodes. NCRep is a physical directory, while INRep is realized by a PostgreSQL database. The NETCONF MIB considered is NETCONF IF-MIB [9]. NETCONF agent is based on YUMA toolkit [8]. YUMA agent functionalities have been extended to support NETCONF IF-MIB and also to provide registration and de-registration of nodes as explained in Section 3.

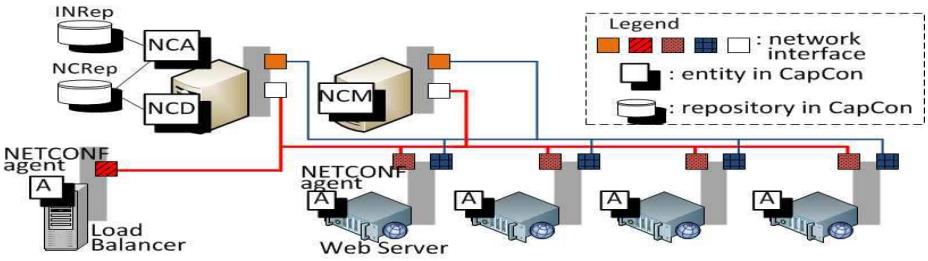

Fig. 4. A web-based application example.

5.2 Node Capability Ontology Concepts

The node capability types and non-rule-based parameters in CapOnt are projected from the *interfaces MIB group* in *SNMP IF-MIB* [7]. In this MIB group, there is a managed object *ifNumber* and a table *ifTable*. *ifNumber* specifies the number of network interfaces. For *ifTable*, *ifEntry* is the conceptual row representing a particular network interface. There are 22 columnar objects of which 18 have Status = current. As examples *ifIndex* has values unique for each network interface, *ifInOctets* gives the total number of octets received on the network interface, and *ifSpeed* defines the

maximum bandwidth of the interface in bit/second. The object *ifAdminStatus* defines the state of the interface which can either be 'up(1)', 'down(2)', or 'testing(3)'.

From the interfaces MIB group, CapOnt has two node capability types and 19 non-rule-based parameters. The node capability types are *interfaces* and *ifEntry*. The non-rule-based parameters are *ifNumber* and all of those 18 mentioned columnar managed objects. These concepts' instances are stored in the inherent node capabilities tables generated by using the rules in Section 4.3. CapOnt has also a rule-based parameter *ifInUtilization* and two service management functions. *ifInUtilization* is the inbound bandwidth utilization of a network interface in percentage. The service management functions' actions are setting a network interface's state. In this paper, the actions are specialized as the NETCONF edit-config operation to set the parameter *ifAdminStatus* value.

The CapOnt non-rule-based concepts in an OWL file as well as the rule-based ifInUtilization parameter and service management functions in OWL/XDD XML clauses are presented at http://tapas.item.ntnu.no/wiki/index.php/CapOnt.

5.3 Node Capability Configuration Specification Concepts

CapSpc for the CapCon prototype is expressed in Fig. 5. It is expresses that "A node with a 100Mbps network interface acting as the load balancer is required. Four nodes with two network interfaces acting as the web servers are required. The first network interface of the web server that *ifIndex* = 1 is required for the connection with CapCon. This network interface's state is always 'up'. The second network interface of the web server that *ifIndex* = 2 is required for the connection with the load balancer, and its state will be changed dynamically.

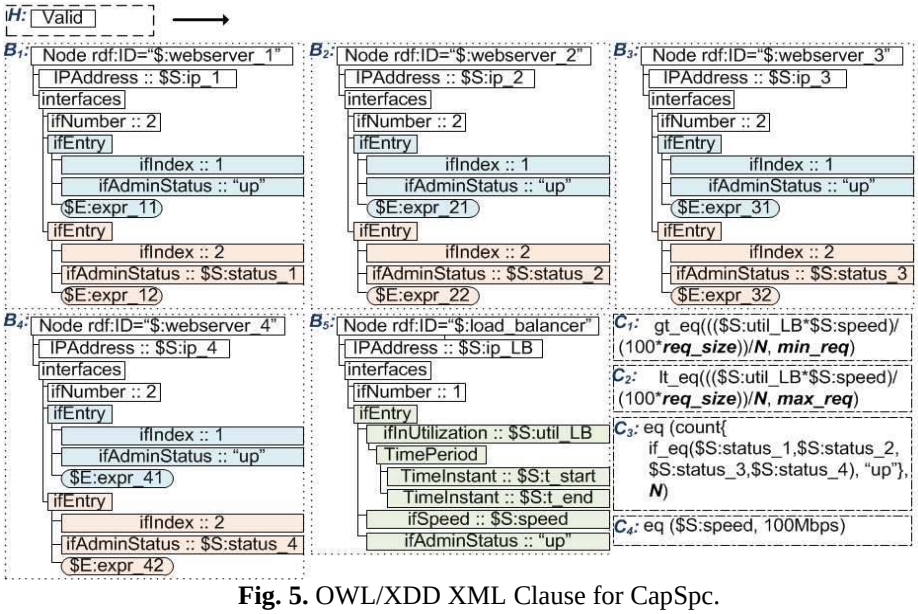

Fig. 5. OWL/XDD XML Clause for CapSpc.

$S:util_LB is the value of the ifInUtilization of the load balancer during the previous interval, between *$S:t_start* and *$S:t_end* second. The total number of user requests during the previous interval can be calculated from: ($S:util_LB*$S:speed)/(100*req_size), where *req_size* is the user request size in bit, and *$S:*speed is the maximum bandwidth of the load balancer (100Mbps). *N* is the appropriate number of web servers connecting with the load balancer during an interval. This means the number of user requests per web server must be between *min_req* and *max_req*, where *min_req* is the estimated *minimum concurrent user requests* per second that a web server should process, and *max_req* is the *maximum concurrent user requests* per second that a web server can process."

5.4 Experimental Results

Two sets of experiments (I, II) have been conducted to illustrate the result of automatic parameter values setting. In this case study, the parameter considered is ifAdminStatus. The evaluation is based on the level of energy conservation achieved from the disconnection of web servers. In both experiments, there is a node with a 100Mbps network interface being the load balancer, four nodes with two network interfaces being the web servers. A *period of inactivity* before hibernate mode of the web servers becomes activated is two minutes. A *cost unit* determines the amount of energy usage: 1 cost unit during normal mode and 0.3 cost units during hibernation mode. The time interval for the polling of NCM as well as the identification and value settings in NCD is 30 seconds. The controlled variables in the CapSpc in Section 5.3 are set as req_size = 500 bytes, min_req = 500 req/sec, and max_req = 1000 req/sec. In these experiments inbound traffic of the load balancer is controlled, and is generated only from the user requests.

In experiment I, the user requests were randomly generated at the rate between 1000-4000 req/sec. In experiment II, the user requests were generated according to the time of day: 2001-4000 req/sec during 08:00-22:00 and 1000-2000 req/sec during 22:00-08:00. During the execution, the web servers were capable to meet, or had already met, the requirement. Based on the CapSpc, the CapOnt concepts, and the inherent node capabilities from NCA, the node capability parameter configuration function in NCD calculated the ifInUtilization value of the load balancer, and validated if the number of connecting web servers was the same as required. When it was not, this function returned one or more of randomly selected web servers' IP address and the NETCONF edit-config operations to NCD. Each of the operations contains the parameter ifAdminStatus instance and required value ('up'/'down'). NCD sent the <rpc> messages to the corresponding agents, and consequently such web servers were re-connected/disconnected.

The experiments have been carried out with the same total number of user requests for both experiments. The results from both experiments show that three out of four web servers were disconnected and entered hibernating mode. Table 1 shows the number of times each web server was disconnected and hibernated. It also shows that the time percentage that a web server in experiment II enters hibernation after getting disconnected is more than its counterpart in experiment I, since the user requests rate

in experiment II is more certain. We can indicate that the usage of CapCon for the "time-based" user request rate results in higher energy conservation.

Table 1. Experimental results.

	Experiment I	Experiment II
Server_1: Disconneted / Hibernated	258/32	276/122
Server_2: Disconneted / Hibernated	39/11	20/19
Server_3: Disconneted / Hibernated	13/8	5/5
Conserved energy (% of saving cost unit per day)	16.09%	41.58%

6 Related Work

Two aspects regarding system architecture and information model are considered in this paper. Most of the recent works related to NETCONF-based configuration management systems, however, focus only on one of these aspects. Some examples are found in [16-19]. The work in [16-17] focuses on the system architecture aspect, [18] mainly considers the NETCONF MIB represented by YANG, and both aspects are discussed in [19]. Cui et al. [16] present a detailed design of NETCONF manager and agent. A proxy configuration file is used to specify the agents in the entire managed devices. In CapCon, the NETCONF agents are able to register nodes and the node capabilities without the need of configuration file. Liu et al. [17] improve traditional network management system by adding a simple judging function to dynamically decide whether the NETCONF protocol is supported on the managed devices. If the NETCONF protocol is supported, NETCONF operations are used. Otherwise, SNMP operations are used. Nataf and Festor [18] integrate the NETCONF MIB represented by YANG into the NETCONF agent from the ENSUITE open source framework, and implement a browser to retrieve and edit the configuration data. Elbadawi and Yu [19] present the design and implementation of a configuration validation system using Erlang programming language. Comparing with CapCon, the system provides the validation without the parameter value settings.

7 Conclusions

Node Capability Parameter Configuration System (CapCon) which enables automatic node capability parameter configuration is presented. Two aspects including centralized architecture and Node Capability Ontology (CapOnt) are discussed in detail. The CapOnt concept representations and the transformation between CapOnt and SNMP-based NETCONF MIB are presented. In the case study, the prototype web-based application integrated with CapCon has been implemented. The automatic parameter value setting by CapCon dynamically adapts the number of active web servers.

CapCon provides flexibility in terms of the applied frameworks. The CapOnt concepts as well as the CapSpc concepts can be added, modified and removed during

the system runtime. CapCon also enables automatic discovery of nodes and their node capabilities. In this paper, CapCon is integrated with NETCONF. The flexible nature of the system enables integration with various network management frameworks. However, the future work can be focused on developing a decentralized architecture of CapCon.

References

1. Subramanian, M.: Network Management - Principles and Practice. Addison-Wesley, 2000.
2. DMTF, Web-Based Enterprise Management, http://dmtf.org/standards/wbem
3. DMTF, Web Services Management, http://dmtf.org/standards/wsman
4. Enns, R.: NETCONF Configuration Protocol, IETF RFC 4741, Dec 2006.
5. Studer, R., Benjamins, V. R., Fensel, D.: Knowledge Engineering: principles and methods. In: Data & Knowledge Engineering, vol. 25, pp. 161-197, 1998.
6. Thongtra, P., Aagesen, F. A.: Capability Ontology in Adaptable Service System Framework. In: Proc. of 5th Int. Multi-Conference on Computing in the Global Information Technology, Spain, Sep 2010.
7. McCloghrie, K., Kastenholz, F.: The Interfaces Group MIB, IETF RFC 2863, Jun 2000.
8. YUMA - YANG-based Unified Modular Automation toolkit for the NETCONF protocol, http://www.NETCONFcentral.org/yuma
9. IF-MIB YANG module, http://www.netconfcentral.org/modulereport/IF-MIB
10. Bjorklund, M.: YANG - A Data Modelling Language for the Network Configuration Protocol (NETCONF), IETF RFC 6020, Oct 2010.
11. libsmi - A Library to Access SMI MIB Information, http://svn.ibr.cs.tu-bs.de/projects/libsmi
12. Akama, K., Shimitsu, T., Miyamoto, E.: Solving Problems by Equivalent Transformation of Declarative Programs. In: Journal of the Japanese Society of Artificial Intelligence, vol. 13, pp. 944-952, 1998.
13. W3C, OWL Web Ontology Language Overview, 2004, http://www.w3.org/TR/owl-features/
14. Wuwonse, V., Yoshikawa, M.: Towards a language for metadata schemas for interoperability. In: Proc. of 4th Int. Conf. on Dublin Core and Metadata Applications, China, 2004.
15. TAPAS Platform - A support system for deployment, execution and management of service systems defined by the TAPAS architecture concepts, http://tapas.item.ntnu.no/wiki/index.php/TAPAS_Platform
16. Cui, J., Jia, K., Wu, L., Chen, C., Lai, M.: The Design of the Network Configuration Management based on NETCONF protocol. In: Proc. of Int. Conference on Applied Informatics and Communication (ICAIC 2011), Xi'an, China, Aug 2011.
17. Liu, L., Xiao, D., Dong, B., Shen, Q.: Implementation of the management of SNMP/NETCONF network devices for the next generation NMS. In: Proc. of 2nd Int. Conference on Electrical and Control Engineering (ICECE 2011), Yichang, China, Sep 2011.
18. Nataf, E., Festor, O.: End-to-end YANG-based Configuration Management. In: Proc. of 12th IEEE/IFIP Network Operations and Management Symposium (NOMS 2010), Osaka, Japan, Apr 2010.
19. Elbadawi, K., Yu, J.: High Level Abstraction Modeling for Network Configuration Validation. In: Proc. of IEEE Global Telecommunications Conference (GLOBECOM 2010), Miami, USA, Dec 2010.