Ontology-based Capability Self-Configuration

Patcharee Thongtra, Finn Arve Aagesen and Natenapa Sriharee

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

Abstract—**Capability configuration management is the allocation, re-allocation and de-allocation of capabilities. A capability is here defined as an inherent physical property of a network node which is a basis for implementing networked services. Capability configuration management requires a specification of the capability configuration management process. This paper proposes a framework for capability configuration process specification (CCPS) production and execution. CCPS is a set of actions with related parameters and is used to configure the capability objects of the network nodes prior to service deployment and instantiation. A CCPS is based on Web services.**

The production of CCPS is based on capability requirements defined by the networked services to be deployed and instantiated. The framework has an ontological reasoning ability. A case study to configure a VLAN connection using switches is also provided.

I. Introduction

Networked service systems are considered. Services are realized by service components, which by their inter-working provide a service in the role of a service provider to a service user. Service components are executed as software components in nodes, which are physical processing units such as servers, routers, switches, PCs and mobile phones.

A service system can be defined as the handling of the abstractions of the service component models of a service system through the service system life-cycle phases. The service life-cycle phases can be structures into

- service engineering,
- service configuration management and
- service discovery management.

Service engineering is the definition and implementation of the service component models. *Service configuration management* consists of capability configuration management, service deployment and service instantiation. *Service discovery management* is the process of exposing the services to the public, finding the services that satisfy the functional and QoS requirements, and accessing the services.

A capability [1] is an inherent property of a network node and is used as a basis to implement services. Capability configuration management comprises capability initialization, capability re-initialization and capability allocation adaptation. Capability initialization is the allocation and the arrangement of the capabilities for the service components to be distributed and instantiated. Capability re-initialization is the re-distribution and re-instantiation of the service components and capabilities when the instantiated service systems are unable to adapt satisfactorily as well as when capabilities are changed. Lastly, capability allocation adaptation is the monitoring of the performance of the executing service system, the re-allocation and the re-arrangement of the capabilities within the executing service systems.

Capability configuration management is a vital part of service configuration management and precedes service deployment and service instantiation. *Service deployment* is the introduction of new services and service components into the network nodes, while *service instantiation* is the creation of instances of the service components and making them ready for usage. In this paper, capability initialization comprises

- the production of a capability configuration process specification (CCPS) as well as
- the configuration of the capability objects of the processing nodes.

In this paper we propose a capability self configuration framework based on capability ontology. In the paper we are focusing on capability initialization. The framework is defined by

- information model,
- organization model and
- communication model

The information model defines the capability ontology, the capability requirement as well as CCPS. The capability ontology provides a formal way to describe and reason the semantics and axioms of the capability-related information. The capability ontology model is represented by OWL (Ontology Web Language) [2]. It expresses general capabilities of the network nodes and capability requirements of specific service system. Both are inputs to the CCPS framework. CCPS is here a set of SOAP (Simple Object Access Protocol) messages encapsulating the capability-related information to configure the capability objects of the network nodes.

The organization model defines the active components that participate in the capability configuration management; it has information model objects as input and output.

Lastly, the communication model describes SOAP messages to communicate between the active components and the network nodes. The communication model is based on WS-Management [3] which promotes the vendorindependent network management protocol by using Web services technology.

The framework described in this paper has three contributions as follows: First, the framework can automatically generate CCPS. Second, with the capability ontology the framework has a potential to reason the capability-related information in the service system. Reasoning ability is needed for a self-management service system. Third, action for the configuration is extensible. Based on Web services the network node can publish new services for the configuration without the reengineering of the framework.

The paper is organized as follows: Section II discusses related work, Section III explains the information model, and Section IV presents the organization model and the active components residing in the model. Section V describes the communication model. Section VI provides a case study of capability configuration for a VLAN (Virtual Local Area Network) connection. A summary and conclusion are provided in Section VII.

II. Related work

There have been several studies related to self-configuration networks, even though most of them have not considered a unify framework with service self-configuration management ability. With such ability the framework should be able to handle network configuration management, service deployment and service instantiation. SELFCONF [4] is an architecture for the self-configuration of networks in response to changed configuration policies. A DEN-ng specification [5] has been used as the information model and LDAP protocol as the communication model. Our work can include DEN-ng as part of the information model as well. NESTOR [6] has been demonstrated also in the automation of network configuration. The NESTOR automates configuration management by means of policy scripts with access to and manipulation of respective network nodes.

With the employment of ontology that is similar to our work, Glasner and Sreedhar [7] proposed to use ontology and OWL to model and represent the configuration of an Apache server. Based on their model the information can be validated. However, the model is not applicable for networked service systems in general. Moreover, the configuration management process for IP network nodes can be modeled by using OWL, SWRL and OWL-S, as presented in [8].

III. Information Model

Several ontologies have been proposed for use as network management information models, such as DEN-ng and CIM [9]. However, those ontologies can represent only the network nodes and their properties; they lack a representation of the relationships between them. Thus, it is difficult to reason the information.

In this paper we follow the approach of using ontology to represent the capability-related information, but we also pay attention to the relationships and their semantics. Our ontology model, *capability ontology model*, implements the capability concept. We select OWL as the representation language for the capability ontology since it has a rich set of operators for describing complex constraints.

A. *Capability Ontology*

A capability of a network node is described by inherent properties. Those properties represent the capabilities in terms of functional capability, resources capability and data capability with the detailed description as follows:

- Functional capability represents pure software or combined software/hardware components used for performing particular tasks which they can be measured in terms of the status.
- Resources capability represents hardware components with finite capacity. For example, processing units, storage units as well as communication units are the hardware components in which finite capacity can be measured in terms of the value.
- Data capability represents the data including the interpretation, validity and life span of which depend on the context of use.

Additionally these capabilities are defined according to the capability attributes: *negotiability*, *arrangement*, *compatibility* and *reducibility*.

With the capability attributes and their logical characteristics, such as inherited, transitive and symmetric, the capability ontology expresses the relationships and constraints between the network nodes and their properties well. Figure 1 represents the capability ontology model.

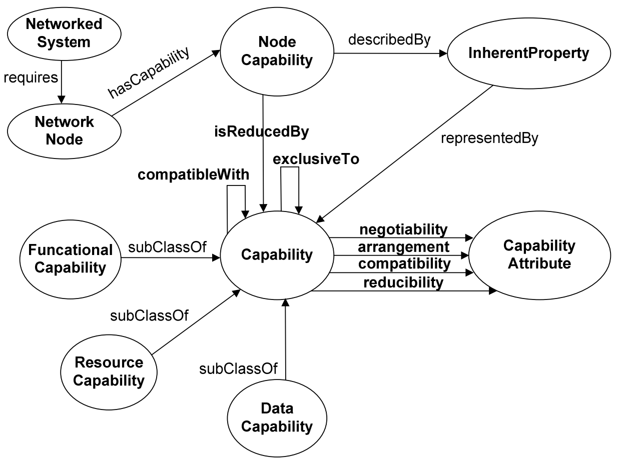

Figure 1: Capability Ontology Model.

Negotiability attribute represents whether it is possible to negotiate on a capability. The attribute is either *negotiable* or *non-negotiable*. If a capability is negotiable, there is a range of acceptable characteristics. For example, transmission channel capacity required by a text messaging application is negotiable (acceptance rate between 128 – 384 kbps), while access right data is non-negotiable.

The attribute negotiable has an inherited characteristic. If a network node capability is negotiable, then its inherent capabilities are negotiable. For example, the capability of a Web server providing normal HTTP pages is negotiable. Thus, the server's transmission channel capacity is negotiable.

Arrangement attribute represents whether a capability is limited for a special purpose (e.g., group of consumers) or it is for general use. The attribute is either *exclusive* or *shared*. For example, transmission channel and Web server process are by nature shared resources, but a password is by nature exclusive information.

Compatibility attribute represents whether a capability is definitely unique or it can be substituted by other capabilities. The attribute is either *compatible* or *incompatible*. For example, JRE (Java Runtime Environment) v.1.4.2 is compatible with JRE v.1.4.7, but not the other way round.

The attribute compatible has a transitive characteristic, while the attribute incompatible has a symmetric characteristic.

Reducibility attribute represents whether a capability is reduced proportional to the consumer (e.g. the user or other capabilities). The attribute is either *reducible* or *irreducible*. For example, bandwidth is decreased by the number of users multiplied by capacity consumed per user, while a Web server running process is non-reducible.

The attribute reducible has an inherited characteristic as well as a transitive characteristic. For example, the capability of a switch node is reducible; its memory is reduced by data in routing table.

B. *Capability Requirement*

The capability description describes the capability of the network node in general without specific capacity and status. The capability requirement states that which classes of the network nodes are needed in the service systems including the number of each class. It utilizes the capability description and specifies required capacity and status.

Table 1 represents an example of a VLAN streaming service system that consists of client switch and server switch. Both client switch and server switch are subclasses of a network node. The client switch capability is negotiable. The server switch is described by these capabilities: Interface, IOS (Internetwork operating system), NVRAM (Non-volatile random access memory), etc. Interface is a functional capability which is measured by its status. IOS is also a functional capability. IOS is compatible with the IOS itself. NVRAM is a reducible resource capability which has finite memory size that can be measured in terms of megabyte units. In addition, it contains administrator password that is an exclusive data capability.

Table 1: An example of the capability description for VLAN streaming service system.

```
<owl:Class rdf:ID=”VLAN_Streaming_ServSystem”>
  <rdfs:subClassOf
    rdf:resource=”#NetworkedSystem”/>
  <requires rdf:resource=”#ServerSwitch”/>
  <requires rdf:resource=”#ClientSwitch”/>
</owl:Class>
<owl:Class rdf:ID=”ClientSwitch”>
  <rdfs:subClassOf rdf:resource=”#NetworkNode”/>
  <hasCapability
rdf:resource=”#ClientSwitchCapability”/>
</owl:Class>
<owl:Class rdf:ID=”ClientSwitchCapability”>
  <negotiability rdf:resource=”#Negotiable”/>
</owl:Class>

<owl:Class rdf:ID=”ServerSwitch”>
  <rdfs:subClassOf rdf:resource=”#NetworkNode”/>
  <hasCapability
rdf:resource=”#ServerSwitchCapability”/>
</owl:Class>
<owl:Class rdf:ID=”ServerSwitchCapability”>
  <describedBy rdf:resource=”#Interface”>
  <describedBy rdf:resource=”#IOS”>
  <describedBy rdf:resource=”#NVRAM”>
  ...
</owl:Class>
```

```
<owl:Class rdf:ID=”Interface”>
  <rdfs:subClassOf
rdf:resource=”#FunctionalCapability”/>
  <hasStatus rdf:resource=”#InterfaceStatus”/>
</owl:Class>
<owl:Class rdf:ID=”IOS”>
  <rdfs:subClassOf
rdf:resource=”#FunctionalCapability”/>
  <compatibility rdf:resource=”#Compatible”>
  <compatibleWith rdf:resource=”#IOS”/>
</owl:Class>
<owl:Class rdf:ID=”NVRAM”>
  <rdfs:subClassOf
rdf:resource=”#ResourceCapability”>
  <hasFiniteCapacity rdf:resource=”#MemorySize”/>
  <hasFiniteCapacityUnit ref:resource=”#MB”/>
  <reducibility rdf:resource=”#Reducible”/>
</owl:Class>

<owl:Class rdf:ID=”AdministratorPassword”>
  <rdfs:subClassOf
    rdf:resource=”#DataCapability”>
  <arrangement rdf:resource=”#Exclusive”/>
</owl:Class>
```

From the capability description, the capability requirement can be defined further. Table 2 represents the capability requirement for VLAN streaming service system that requires two client switches and a server switch. The server switch requires at least 512MB NVRAM, installs IOS_12.1.8a, and implements VTP (Virtual Trunking Protocol). The requirement defines compatibility between two versions of acceptable IOS; it defines IOS_12.1.8a and IOS_12.1.9 to be compatible. Also, it describes an arrangement between VTP and VLAN information that VTP_1 is exclusive to VLAN_Savanne_Room (VLAN name “savanne_room” and no. 5).

Table 2: The capability requirement for VLAN streaming service system in NTNU room.

```
<VLAN_Streaming_ServSystem
rdf:ID=”NTNU_Savanne_Room”>
  <requires>
    <ClientSwitch rdf:ID=”Client_Switch_1”/>
  </requires>
  <requires>
    <ClientSwitch rdf:ID=”Client_Switch_2”/>
  </requires>
  <requires rdf:resource=”#Server_Switch_1”/>
</VLAN_Streaming_ServSystem>
<ServerSwitch rdf:ID=”Server_Switch_1”>
  <hasCapability
  rdf:resource=”#Server_Switch_Cap_1”/>
</ServerSwitch>
<ServerSwitchCapability
rdf:ID=”Server_Switch_Cap_1”>
  <describedBy rdf:resource=”#NVRAM_1”/>
  <describedBy rdf:resource=”#IOS_12.1.8a”/>
  <describedBy rdf:resource=”#VTP_1”/>
  ...
</ServerSwitchCapability>
<NVRAM rdf:ID=”NVRAM_1”>
  <hasFiniteCapacity
  rdf:resource=”#MemorySize_1”/>
</NVRAM>
<MemorySize
rdf:ID=”MemorySize_1”>512</MemorySize>
<IOS rdf:ID=”IOS_12.1.8a”>
  <compatibleWith rdf:resource=”#IOS_12.1.9”/>
</IOS>
<VTP rdf:ID=”VTP_1”>
  <exclusiveTo
  rdf:resource=”#VLAN_Savanne_Room”/>
</VTP>
<VLAN_Info rdf:ID=”VLAN_Savanne_Room”>
  <name>savanne_room</name>
  <number>5</number>
</VLAN_Info>
```

C. Capability Configuration Process Specification (CCPS)

CCPS is an XML-based specification composed of SOAP messages. The messages are sent to services implemented on the network nodes. The services are to get, update, create, delete and subscribe the capability of the network nodes. The capabilities are realized by the capabilities objects.

A message contains the configuration action (get, update, create, delete and subscribe) and the configuration action parameters. The configuration action and parameters are enclosed by the element *Action* and *ActionParams* respectively, while the configuration result is identified by the element *ActionResponse*. (Each message is detailed in Section V.) The configuration action parameter is likely a capability class (defined by the capability description). Table 3 illustrates the schema for CCPS.

Table 3: Schema for CCPS.

```
<xs:schema targetNamespace=
”http://schemas.tapas.item.ntnu.no/Capability”
xmlns:xs=”http://www.w3.org/2001/XMLSchema”
xmlns:ws=”http://schemas.xmlsoap.org/ws/2005/06/m
anagement”
xmlns:env=”http://schemas.xmlsoap.org/soap/envelo
pe”
xmlns:cap=”http://schemas.tapas.item.ntnu.no/Capa
bility”>
  <xs:import
namespace=”http://schemas.xmlsoap.org/ws/2005/06/
management” schemaLocation=”wsmanagement.xsd”/>
  <xs:import
namespace=”http://schemas.xmlsoap.org/soap/envelo
pe” schemaLocation=”soap.xsd”/>

  <xs:element name=”CCPS” type=”CCPS_Type”/>
  <xs:complexType name=”CCPS_Type”>
    <xs:sequence>
      <xs:element ref=”networkSystemID”/>
      <xs:element ref=”env:Envelop” minoccurs=1/>
    </xs:sequence>
  </xs:complexType>

  <xs:element name=”networkSystemID”
type=”String”/>
  <xs:complexType name=”Header”>
    ...
    <xs:element ref=”wsa:Action”/>
    <xs:element ref=”ActionParams”/>
  </xs:complexType>

  <xs:complexType name=”Body”>
    ...
    <xs:element ref=”ActionResponse”/>
  </xs:complexType>

  <xs:element name=ActionParams
type=”ActionParams_Type”/>

  <xs:complexType name=”ActionParams_Type”>
    <xs:sequence minoccurs=1>
      <xs:element ref=”ActionParam”/>
      <xs:element ref=”ActionValue”/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name=”ActionParam” type=”String”/>
  <xs:element name=”ActionValue” type=”String”/>

  <xs:element name=ActionResponse type=”String”/>
</xs:schema>
```

IV. Organization Model

The organization model is layered into three levels; *User Interface Layer*, *Computing Layer* and *Network Nodes Layer*, as shown in Figure 2. The computing layer consists of three components; *Capability Manager (CM)*, *Query Manager (QM)* and *Capability Object Configurator (COC)*, one repository; *Capability Ontology Repository (CORep)* and Web services registry; *Capability Web Services Registry (CWSReg)*. CORep stores the capability description, the capability requirement and the capability instance. CWSReg is where the network nodes register their Web services for the capability configuration and notifications.

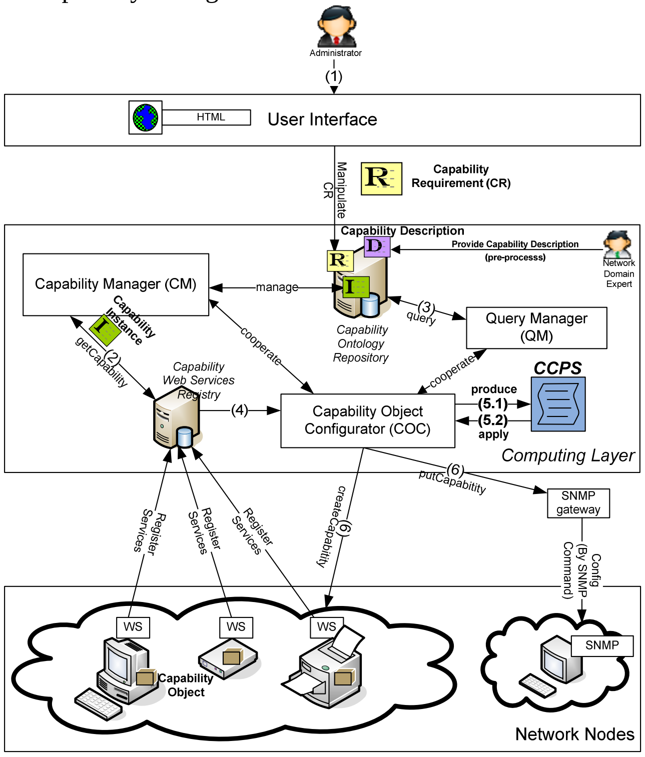

Figure 2: The organization model.

First, the administrator inputs, queries and updates the capability requirement via the user interface (1). The capability description has been provided by the network domain expert to support the creation of the capability requirement. After the capability requirement is input, the components residing in the computing layer start their processes automatically.

The CM is responsible for getting the capability instance from existing network nodes via the *getCapability* service (2). A capability instance is a data representation of a capability of the network node. In addition, the CM manages the capability instance for example providing an access to the instances.

The QM generates query commands to select (3) either the network nodes whose capabilities satisfy the capability requirement, or the ones having potential to meet the capability requirement. For example, a query to find a PC with free disk space more than 1GB. QM uses the transformation rule to transform the capability requirements into SPARQL (SPARQL Protocol and RDF Query Language) [10] query format. (The transformation rule is given in Appendix.) The result of the query can identify the network node by its URI (Uniform Resource Identifier).

The COC gets the URIs of the selected network nodes from the QM. Then the COC finds their *setCapability*, *createCapability* and *deleteCapability* services (4) to make their capabilities in absolute compliance with the capability requirements. The COC makes a CCPS (5.1). Lastly the COC applies the CCPS (5.2) by sending the messages residing on the CCPS to the network nodes (6).

At the network nodes layer the nodes register their services to CWSReg. According to WS-Management specification, the network nodes implement Web services allowing others to manage their resource instances, which are considered here as capability objects. However, the network nodes can also implement custom services or custom actions besides the ordinary well-defined services. (For network nodes that have been implemented other network management architectures, such as SNMP, we use a software gateway for the information exchange between a protocol used in such architectures and Web services protocol.)

V. Communication model

In this paper we apply WS-Management framework which proposes Web services as the protocol for managing network systems. WS-Management exposes the network management information model - such as CIM or custom models - as Web services. Several works [11, 12, 13] analyzed the advantages and drawbacks of the employment of Web services technology for the network management compared to the employment of classical network management architectures: SNMP or CMIP. However, Web services is a current technology trend for the networks with self-management ability as mentioned in [14].

The communication model extends WS-Management. It details custom actions and SOAP messages for the communication between the computing layer and the network nodes. The custom actions are defined in order to manipulate the capability as follows:

- *getCapability action* is to request for the capability instance.
- *getCapabilityResponse action* is a response to getCapability to return the capability instance.
- *setCapability, createCapability* and *deleteCapability action* are to respectively set, create and delete the capability objects.
- *setCapabilityResponse, createCapabilityResponse* and *deleteCapabilityResponse action* are to respectively return the result of setCapability, createCapability and deleteCapability.
- *subscribeOnCapability action* is an action to subscribe event relating to any changes that occur with the capability objects. When an event occurs, an asynchronous message notification is sent to indicate that event. An event is for example low disk space.
- *subscribeOnCapabilityResponse action* is to return the result of a subscription done by subscribeOnCapability action.
- *unsubscribeOnCapability action* is to terminate a subscription unless the subscribed capability is used further ahead.
- *unsubscribeOnCapabilityResponse action* is to return the result of unsubscribeOnCapability action.

We do not need subscribeOnCapability, unsubscribeOnCapability and their response actions for the capability allocation. But we also consider them in order to support the capability re-allocation in the future.

The request actions (getCapability, setCapability, createCapability, deleteCapability, subscribeOnCapability and unsubscribeOnCapability) and their parameters are enclosed in the message template displayed in Table 4, while the response actions' message template is shown in Table 5.

Note that XML namespaces used in the templates follow WS-Management specification.

Table 4: The request action message template.

```
<env:Header>
  <wsa:To>management service AP on node</wsa:To>
  <wsman:ResourceURI>WSDL identity and port on
node</wsman:ResourceURI>
  <wsa:ReplyTo>component URI</wsa:ReplyTo>
  <wsa:MessageID>uuid:xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx</wsa:MessageID>
  <wsa:Action>action URI</wsa:Action>
  <cap:ActionParams>
    <cap:ActionParam>param<cap:ActionParam>
    <cap:ActionValue>value<cap:ActionValue>
  </cap:ActionParams>
<env:Header>
```

Table 5: The response action message template.

```
<env:Header>
  <wsa:To>
http://schemas.xmlsoap.org/ws/2004/08/addressing/
role/anonymous
  </wsa:To>
  <wsa:Action>action URI</wsa:Action>
<env:Header>
<env:Body>
  <cap:ActionResponse>value</cap:ActionResponse>
</env:Body>
```

In the message templates we use XML elements defined in WS-Management specification and our proposed schema as follows:

- *wsa:To* (in the request action message template) indicates the transport address of management service access point (AP) on the network node,
- *wsman:ResourceURI* denotes WSDL (Web Services Description Language) identity and port on the network node,
- *wsa:ReplyTo* denotes URI of the component in the computing layer which will get the response,
- *wsa:MessageID* represents a unique message id for tracking and correlation between the request and the response.
- *wsa:Action* identifies the action URI.
- *cap:ActionParams* encloses different parameters, that corresponds to the capability description, depending on the request action. (See an example from the case study.)
- *wsa:To* (in the response action message template) denotes the requestor.
- *cap:ActionResponse* defines successful or unsuccessful status of the request action.

VI. Case Study

The case study is to configure a VLAN connection using Cisco switches. The configuration is prepared for a streaming service system.

Switches allow a network to be partitioned into logical segments through the use of VLAN. Switches which share Layer 2 VLAN communication need to be connected by a trunk. IEEE 802.1Q and Virtual Trunking Protocol (VTP) are two popular protocols for VLAN trunks. VTP has been developed on Cisco switches to centralize the creation and deletion of VLANs in a network into a VTP server switch. The server switch has VTP mode server. It can take care of creating, deleting and updating the status of existing VLANs to the other switches sharing the same VTP domain.

This example requires three switches: one server switch and two client switches, to implement a VLAN connection in the NTNU Savanne Room. Figure 3 depicts switches and their properties in the room before the configuration.

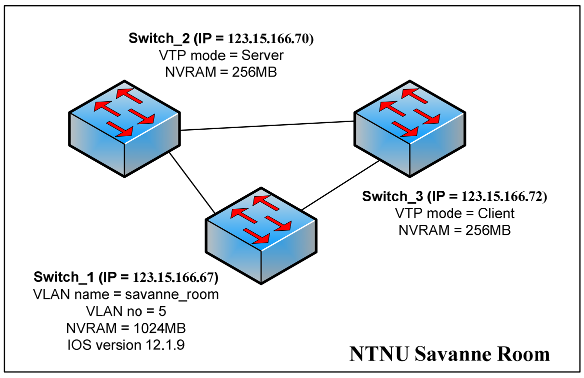

Figure 3: Existing switches in NTNU Savanne Room.

A. The capability description and the capability requirement

Figure 4 depicts the capability description of the server switch that is considered in the example. In additional to the capabilities already described in Table 1 the server switch implements VTP protocol and stores VLAN information. VTP concerns VTP domain and VTP mode. VLAN information consists of a VLAN name and a VLAN number. The VLAN information is stored in NVRAM.

The capability requirement of this example is expressed in Table 2.

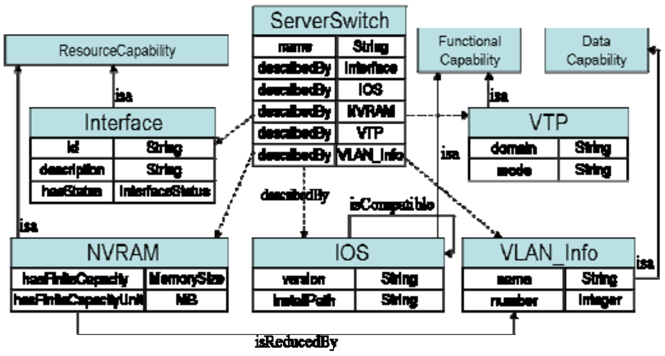

Figure 4: Server Switch Capability Description.

B. The query command

The following shows a query command which is generated to query for the server switch. The query means we need a ServerSwitch with minimum 512MB NVRAM. It must install IOS that is compatible with IOS 12.1.8a, and is reserved for VLAN in the Savanne Room. (See Appendix for details.)

We only consider the server switch as it is non-negotiable, while the client switch is negotiable. Moreover we pay attention to the VLAN information of the server switch as it will be distributed to other client switches automatically.

From the Figure 3 the query returns switch_1 as the ServerSwitch.

```
Select ?server_instance
Where
?server_instance rdf:type ServerSwitch .
?server_instance hasCapability $server_cap .

?server_cap describedBy ?nvram_instance .
?nvram_instance
  hasFiniteCapability ?memory_size .
FILTER (?memory_size >= 512) .

?server_cap describedBy ?vtp_instance .
?vtp_instance
  exclusiveTo VLAN_Savanne_Room .

?server_cap describedBy ?ios_instance .
IOS_12.1.8a compatibleWith ?ios_instance
```

C. CCPS

The following shows a CCPS of this example. It consists on two messages. The first message is to set switch_1 as server, and the second message is to set switch_2 as client.

```
<CCPS>
  <networkSystemID>NTNU_Savanne_Room</
networkSystemID>

  <!-- The first message -->
  <env:Envelop>
    <env:Header>
      <wsa:To>http://123.15.166.67/wsman</wsa:To>
      <wsman:ResourceURI>
http://schemas.tapas.item.ntnu.no/2008/04/Switch/
      </wsman:ResourceURI>
      <wsa:ReplyTo>
http://schemas.xmlsoap.org/ws/2004/08/addressing/r
ole/anonymous
      </wsa:ReplyTo>
      <wsa:MessageID>uuid:d9726315-bc91-430b-9ed8-
ce5ffb858a87</wsa:MessageID>
      <wsa:Action>
http://actions.tapas.item.ntnu.no/2008/04/Setcapab
ility
      </wsa:Action>
      <cap:ActionParams>
        <cap:ActionParam>
          VTP/mode
        </cap:ActionParam>
        <cap:ActionValue>server</cap:ActionValue>
      </cap:ActionParams>
    </env:Header>
  </env:Envelop>

  <!-- The second message -->
  <env:Envelop>
    <env:Header>
      <wsa:To>http://123.15.166.70/wsman</wsa:To>
      <cap:ActionParams>
        <cap:ActionParam>
          VTP/mode
        </cap:ActionParam>
        <cap:ActionValue>client</cap:ActionValue>
      </cap:ActionParams>
      ...
    </env:Header>
  </env:Envelop>
</CCPS>
```

VII. Conclusion

A capability self-configuration framework for networked service systems has been developed, and a case study to configure VLAN connection has been demonstrated. In the framework, there are computing components to get the capability requirement, to search for the network nodes with potential capabilities, and to compute and execute a specification CCPS automatically. The capability configuration management information needed is modeled by the capability ontology. The capability attribute is a part of the proposed ontology that supports the reasoning of the information. The configuration actions in CCPS based on

Web services consist of the action get, set, create, delete as well as subscribe on the capabilities.

The framework also provides a potential to support the modeling of service deployment and service instantiation in addition to the capability allocation that has been the focus in this paper. Extension of the capability ontology to support the modeling of service level agreements (SLA) specification based on the capability is also considered.

APPENDIX

In this section the transformation rule for the finite capacity capability and the capability attribute are provided.

A. *Rule 1 (for finite capacity capability)*

If a node is described by a finite capacity capability (expressed by <hasFiniteCapacity> element), the following query is generated to select the node's instance that has such capability with minimum defined value.

```
Select ?node_instance
Where
?node_instance rdf:type node .
?node_instance hasCapability $node_cap .
?node_cap describedBy ?capability_instance .
?capability_instance
  hasFiniteCapability ?finite_capability .
FILTER (?finite_capability >= capacity_value)
```

For example, from the Table 2 a query is generated to select a server switch that has memory size >= 512MB as follows:

```
Select ?server_instance
Where
?server_instance rdf:type ServerSwitch .
?server_instance hasCapability $server_cap .
?server_cap describedBy ?nvram_instance .
?nvram_instance
  hasFiniteCapability ?memory_size .
FILTER (?memory_size >= 512)
```

B. *Rule 2 (for arrangement attribute)*

If a node is described by an excusive capability (expressed by <arrangement rdf:resource="#Exclusive"/>), the following query is generated to select its instance, when there is a requirement that the capability is exclusive to another capability.

```
Select ?node_instance
Where
?node_instance rdf:type node .
?node_instance hasCapability $node_cap .
?node_cap describedBy ?capability_instance .
?capability_instance
  exclusiveTo another_capability
```

For example, from the Table 2 a query is generated to select a server switch that is exclusive to VLAN defined for Savanne Room as follows:

```
Select ?server_instance
Where
?server_instance rdf:type ServerSwitch .
?server_instance hasCapability $server_cap .
?server_cap describedBy ?vtp_instance .
?vtp_instance
  exclusiveTo VLAN_Savanne_Room
```

C. *Rule 3 (for compatibility attribute)*

If a node is described by a compatible capability (expressed by <compatibility rdf:resource="#Compatible"/>), the following query is generated to select its instance.

```
Select ?node_instance
Where
?node_instance rdf:type node .
?node_instance hasCapability $node_cap .
?node_cap describedBy ?cap_instance .
compatible_cap compatibleWith ?cap_instance
```

For example, from the Table 2 a query is generated to select a server switch that installs IOS that is compatible with IOS 12.1.8a as follows:

```
Select ?server_instance
Where
?server_instance rdf:type ServerSwitch .
?server_instance hasCapability $server_cap .
?server_cap describedBy ?ios_instance .
IOS_12.1.8a compatibleWith ?ios_instance
```

REFERENCES

[1] F. A. Aagesen, and P. Supadulchai, "A Capability-based Service Framework for Adaptable Service Systems," in *Proc. of 2nd Int. Conf. on Advances in Information Technology (IAIT2007)*, Thailand, Nov. 2007.

[2] *Web Ontology Language*, W3C, 2004. Available: http://www.w3.org/2004/OWL

[3] *Web Services for Management (WS-Management) Specification*, DMTF, Feb. 2008. Available: http://www.dmtf.org/standards/wsman

[4] R. Boutaba, S. Omari, and A. P. S. Virk, "SELFCON: An Architecture for Self-Configuration of Networks," in *Journal of Communications and Networks*, Vol. 3, No. 4, Dec. 2001.

[5] *Directory Enabled Network (DEN) initiative*, DMTF. Available: http://www.dmtf.org/standards/wbem/den

[6] Y. Yemini, A. V. Konstantinou, and D. Florissi, "NESTOR: An Architecture for Network Self-Management and Organization," in *IEEE Journal on Selected Areas in Communications*, Vol.18, No.5, pp.758-766, May 2000.

[7] D. Glasner, and V. C. Sreedhar, "Configuration Reasoning and Ontology For Web," in *IEEE Conf. on Service Computing (SCC 2007)*, Jul. 2007.

[8] X. Hui, and X. Debao, "A Common Ontology-Based Intelligent Configuration Management Model for IP Network Devices," in *Proc. of 1st Int. Conf. on Innovative Computing, Information and Control, 2006 (ICICIC 06)*, Aug. 2006.

[9] *Common Information Model (CIM) Standard*, DMTF. Available: http://www.dmtf.org/standards/cim

[10] *SPARQL query language for RDF*, W3C, Jan. 2008. Available: http://www.w3.org/TR/rdf-sparql-query

[11] A. Pras, T. Drevers, R. van de Meent, and D. Quartel, "Comparing the Performance of SNMP and Web Services-Based Management," in *eTransactions on Network and Service Management 1(2)*, Dec. 2004.

[12] G. Pavlou, P. Flegkas, S. Gouveris, and A. Liotta, "On Management Technologies and the Potential of Web Services," in *IEEE Communications Magazine 42(7)*, Jul. 2004.

[13] R. L. Vianna, M. J. B. Almeida, L. M. R. Tarouco, and L. Z. Granville, "Investigating Web Services Composition Applied to Network Management", in *Int. Conf. on Web Services, 2006 (ICWS'06)*, Sep. 2006.

[14] J. Schönwälder, A. Pras, and J. P. Martin-Flatin, "On the Future of Internet Management Technologies," in *IEEE Communications Magazine*, Vol. 41, Issue 10 (2003), pp.90-97, Oct. 2003.