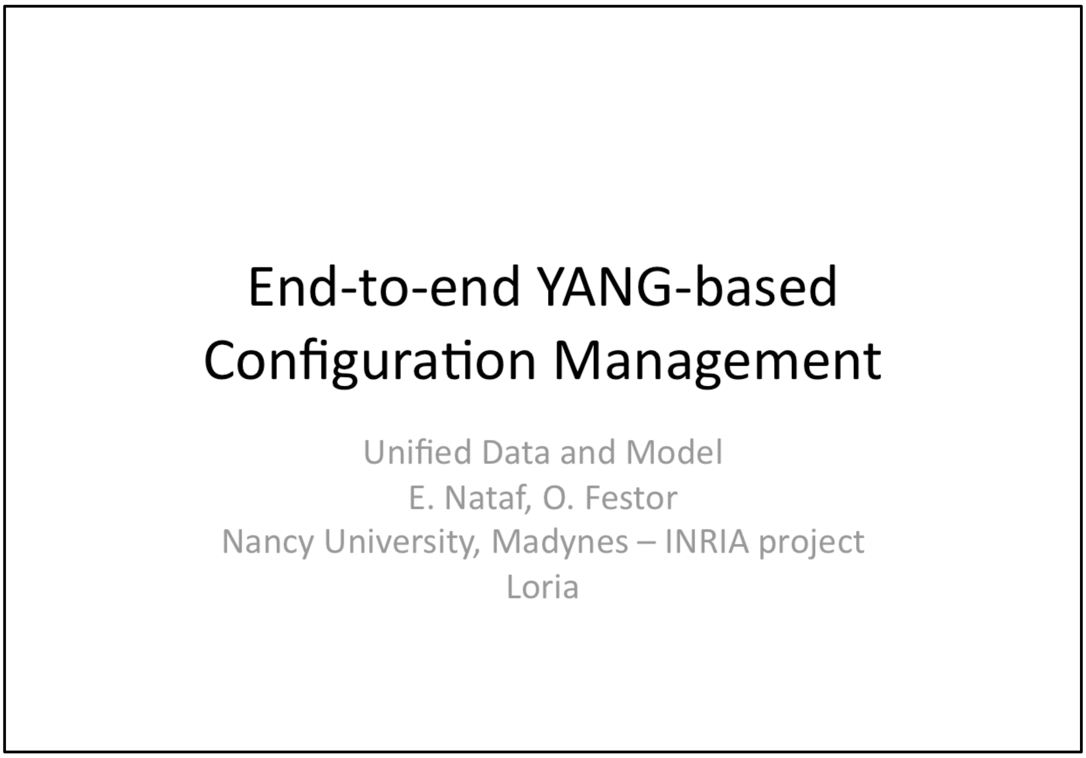
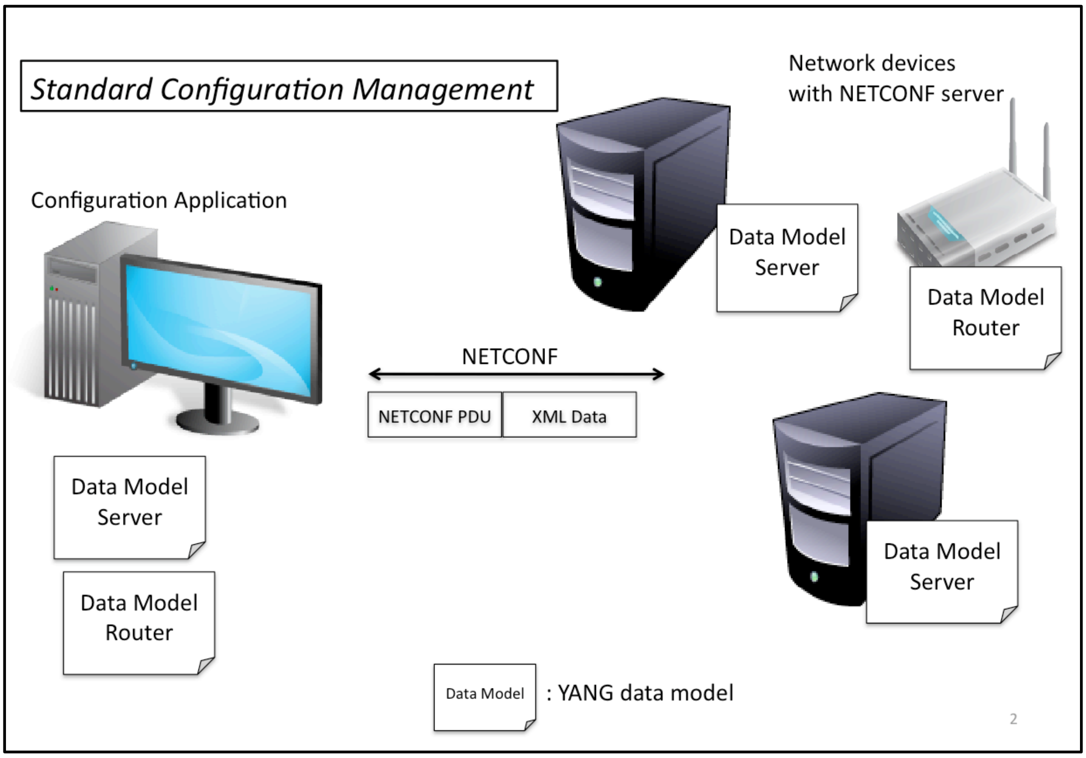


End-to-end YANG-based Configuration Management

Unified Data and Model
E. Nataf, O. Festor
Nancy University, Madynes – INRIA project
Loria

Networked resources are of increasing complexity and have to be configured to guarantee proper operation. Within the IETF, current efforts are focused on both a protocol and a data model definition language for configuration management. The NETCONF protocol describes the communication between network devices to be configured and remote configuration applications. NETCONF does not describe how configuration data is represented. This is addressed by the YANG data modeling language, the emerging proposal of the netmod standard working group.

We present in this paper the result of the integration of YANG and NETCONF in the ENSUITE open source framework. We illustrate this integration through a YANG-based configuration navigation and edition tool.



Standard Configuration Management

Network devices
with NETCONF server

Configuration Application

Data Model
Server

Data Model
Router

NETCONF

NETCONF PDU | XML Data

Data Model
Server

Data Model
Router

Data Model
Server

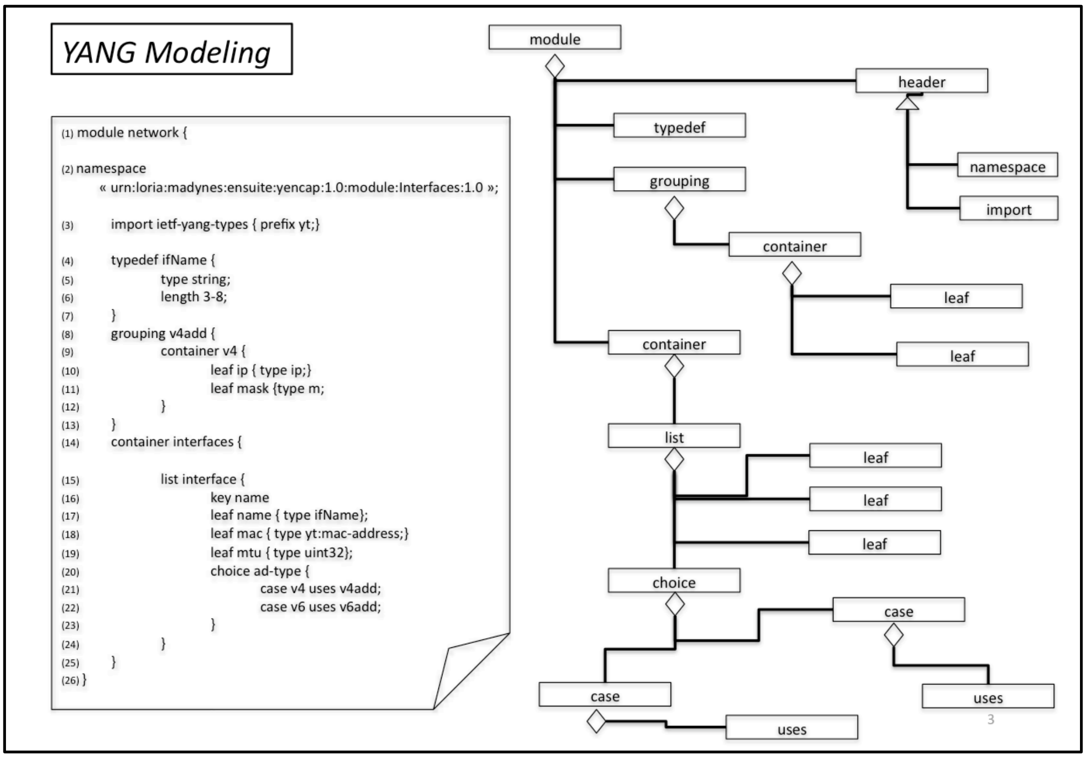
Data Model : YANG data model

2

Configuration management importance is increasing with the growing size and the complexity of network resources. In the Internet context, the netconf working group has proposed the NETCONF protocol [1] as a standard to manage configuration of network devices. This protocol is tailored to configuration operation i.e. setting and/or getting configuration data values to/from devices. Values are tansmitted in a XML document. The standardization body admits this should be improved by a data model that will give semantic to these data values and should be used as a contract between device vendors and application developers.

YANG [2] is the data modeling language proposed by the netmod working group. YANG can be compared to SMI [3] in the SNMP [4] framework because it is a data modeling language and because data values are distributed and accessible through a protocol. With YANG one can specify complex but human-readable configurations for any network device. YANG is presented as a more focused and adapted data modeling language for configuration management than XML Schema or Relax NG [5,6]. On the server side, any vendor can use such specifications to build a NETCONF server that will maintain the local configuration objects that map YANG specifications. On the client side, configuration applications need data values to operate, test or browse configurations. YANG data model specifications are a formal contract between devices' vendors and configuration applications and our goal is to provide tools helping users to ensure that the contract is respected.

The objective of this paper is to demonstrate the feasibility of an End-to-End YANG-aware management framework and to describe how it can be implemented in an open source framework. First we describe the YANG language, focusing on its major concepts. Secondly we present a parser for YANG specifications : jYang, an open source implementation we provide to the community. The third part shows how we did integrate YANG into the used NETCONF server. Finally we show a YANG browsing application and its functionalities to get and edit configuration data.



YANG Modeling

```
(1) module network {

(2) namespace
      « urn:loria:madynes:ensuite:yencap:1.0:module:Interfaces:1.0 »;

(3)       import ietf-yang-types { prefix yt;}

(4)       typedef ifName {
(5)               type string;
(6)               length 3-8;
(7)       }
(8)       grouping v4add {
(9)               container v4 {
(10)                      leaf ip { type ip;}
(11)                      leaf mask {type m;
(12)              }
(13)      }
(14)      container interfaces {

(15)              list interface {
(16)                      key name
(17)                      leaf name { type ifName};
(18)                      leaf mac { type yt:mac-address;}
(19)                      leaf mtu { type uint32};
(20)                      choice ad-type {
(21)                              case v4 uses v4add;
(22)                              case v6 uses v6add;
(23)                      }
(24)              }
(25)      }
(26) }
```

Data model configurations are grouped into YANG modules or submodules. A module is a set of data types specifications on a given subject like the configuration of network interfaces or the configuration of an application protocol parameters. Modules are the largest unit of granularity and a network device should announce which YANG modules it implements. A module defines a name space (line 2) of all its data types to ensure unique naming. Modules can reference each other (without cycle) in order to improve the reusability of YANG specifications as shown in line 3 of our network module example. The "ietf-yang-types" reference is a YANG module [7] with useful types intended to be used by other modules.

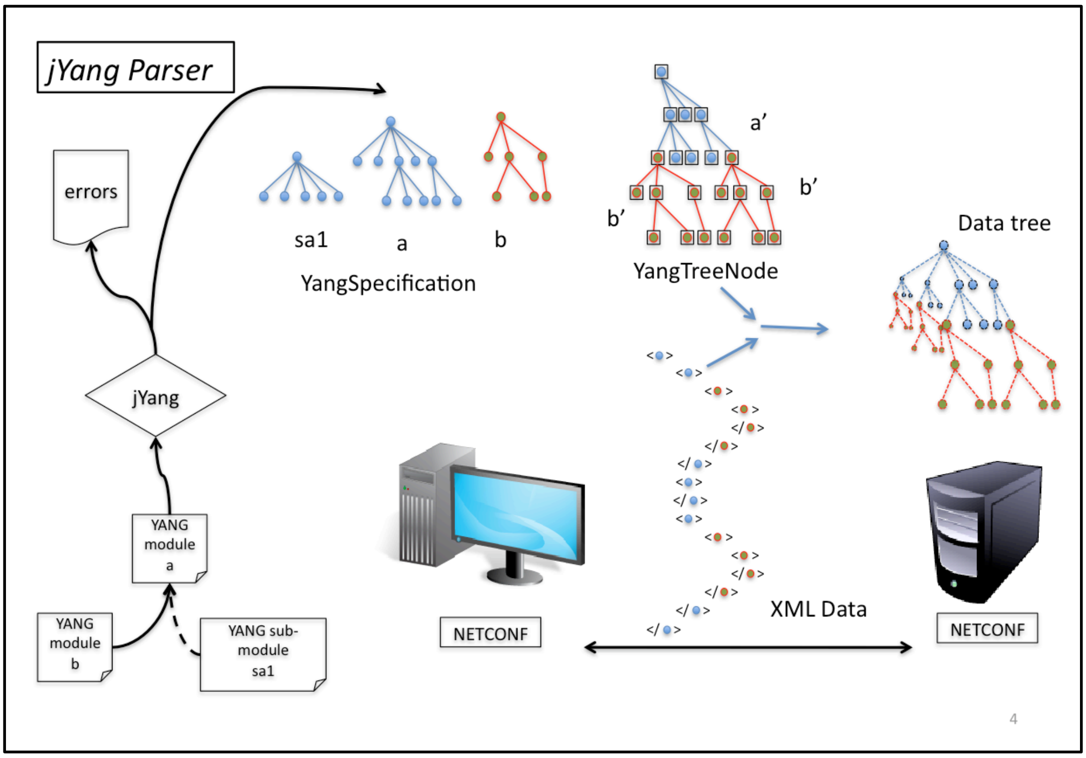
As often in data model languages there are some build-in types such as string, boolean, signed or unsigned integer on several sizes (8, 16, 32, 64 bits). These basic types can be used to create other types with a "typedef " statement (line 4) that allows more precise semantics or to add some constraints as shown at line 6 where the length of a string is limited. Another construct that improves reusability is the "grouping" statement (line 8) that allows the definition of a data model in order to use it more than once at separate places in the current module (seen line 21) or in other modules. It can be compared to a C macro definition.

Data models are mainly expressed with the following statements that are called datadef statement:

- Leaf : a value of one type.
- Container : a set of values.
- List : a set of datadefs, called an entry. An instance is a set of entry values and one value (a column) should be the key of the list (that is the value that distinguish each line).
- Leaf-list : a list of values of the same type.
- Choice ; an alternative of different cases of datadefs.

The example shows two containers (lines 9 and 14) a list (line 15) and a choice (line 20).

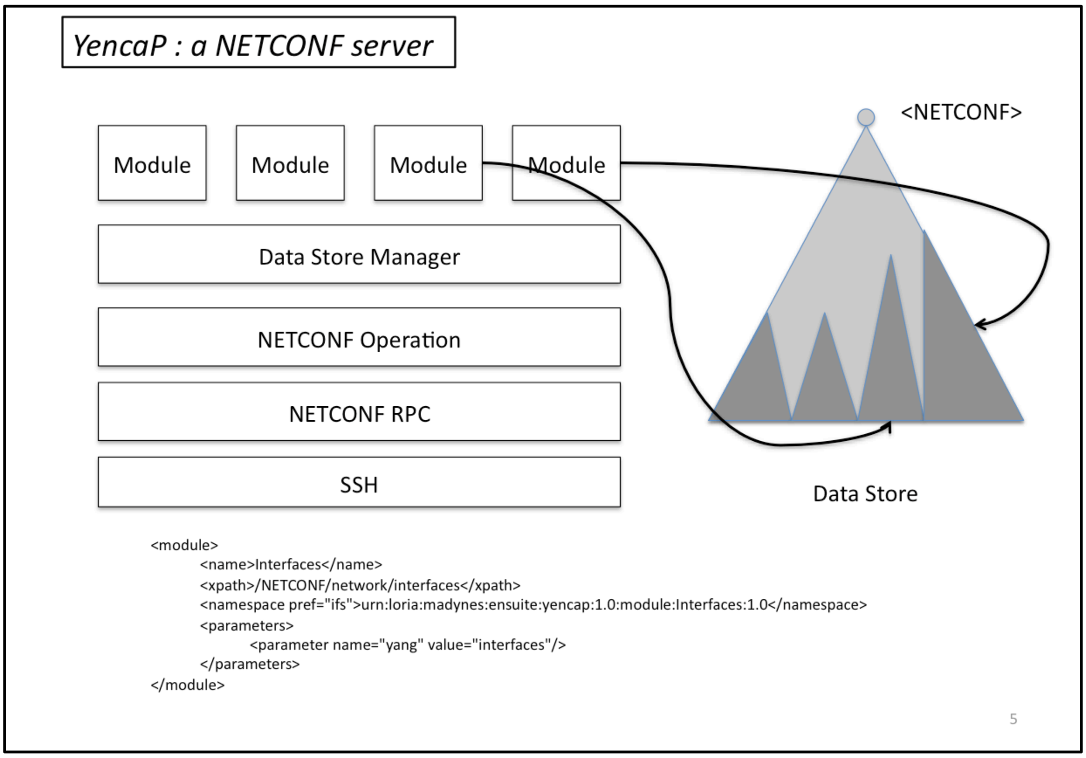
The API we propose reflects YANG statements of the data model. For each YANG statement we have a corresponding java class (see class diagram on the slide 3). A YANG module is represented as a tree of java instances (what the standard call the schema tree). The slide 3 shows as an example the java classes organization for the network module. Each java object have getters methods to follow the tree of instances. About hundred of java classes have been needed to represent any YANG specification.



jYang is an open source parser for the YANG language. It is written in java with the javaCC library(https://javacc.dev.java.net/). A jYang compilation starts with one or more YANG files references that will be loaded by the parser. All import and include statements are followed without parsing twice the same file. So for example if one just gives the *sa1* sub-module to check then the modules *a* and *b* will be too, because *sa1* belongs-to *a* and *a* import *b*. The internal representation of YANG specifications is one of the output of jYang and is composed of the java objects tree we show in slide 3. Another output can be a list of errors encountered during the parsing. Full lexical and syntax checking are done and part of semantic checks is covered. YANG allows data modelers to express some constraints like range number or string pattern. The constraints are checked when sub typing is used or when default values are set. Other constraints that can be expressed on the data value level are not checked. For example one can specify conditional presence of data depending on other data values (or hosting device capabilities).

Another useful output of jYang is the YangTreeNode that is also a java object tree where each node contains a reference to a YANG statement. This tree represents YANG specification but where typedef and grouping are copied at places where they are used. YangTreeNode is an intermediary representation of YANG specifications between static data model and data values on the wire. Its goal is to alleviate the work of any backend that focuses on NETCONF data values. As an example on the slide 4 the module *b* only contains a grouping definition that is used two times in the module *a* so the YangTreeNode from *a* that we note *a'*, contains two YangTreeNode *b'* from *b* (we do not show *sa1'* for readability of the picture).

The YangTreeNode is used to produce the YANG standard data tree from XML data of NETCONF operations. The slide 4 suggests that the data tree has more nodes than the YangTreeNode and that these nodes have a common pattern. This can be the case when a YANG list (or leaf-list) is defined because the data tree will contain each entry of the list (or each value of a leaf list). At the opposite if a specification is made with plenty of choice statements then the data tree will only show one of the cases from the NETCONF data values.



YencaP : a NETCONF server

Module

Module

Module

Module

Data Store Manager

NETCONF Operation

NETCONF RPC

SSH

<NETCONF>

Data Store

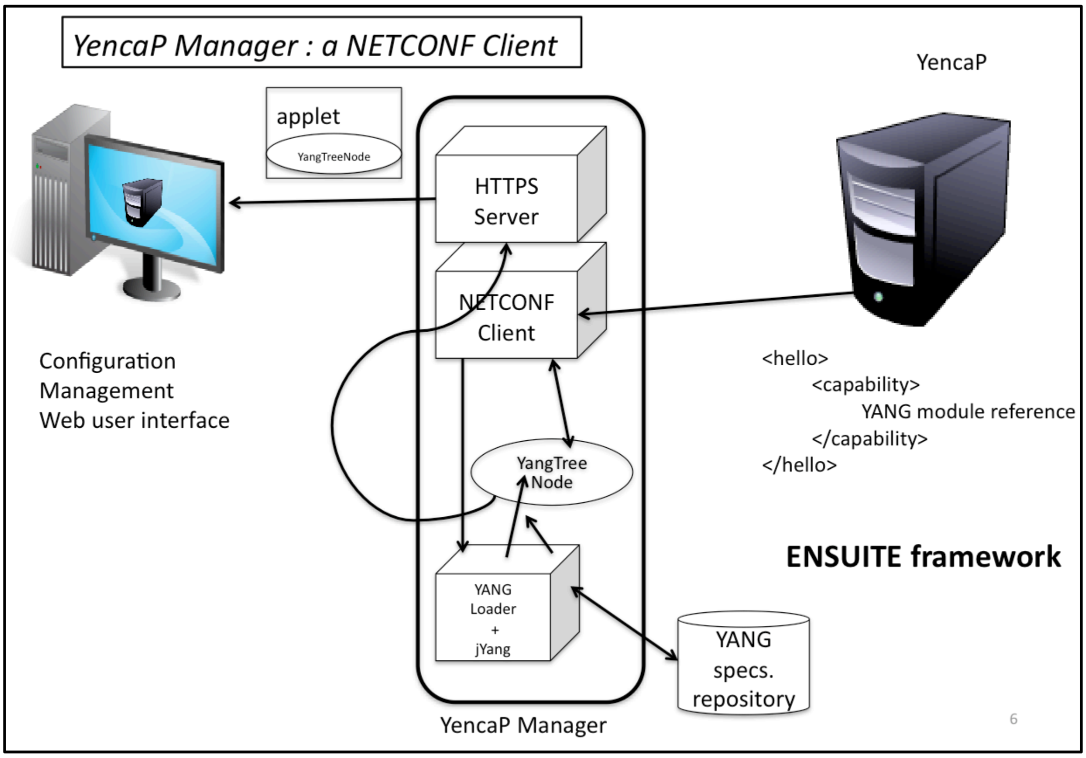
```
<module>
        <name>Interfaces</name>
        <xpath>/NETCONF/network/interfaces</xpath>
        <namespace pref="ifs">urn:loria:madynes:ensuite:yencap:1.0:module:Interfaces:1.0</namespace>
        <parameters>
                <parameter name="yang" value="interfaces"/>
        </parameters>
</module>
```

5

YencaP [8] is an implementation of the server side of NETCONF. It is an open source software initially created by our research team. It is built on top of an SSH layer to ensure security, session and connection-oriented communication as stated by the standard. The RPC layer implements the RPC mechanism with <rpc> and <rpc-reply> primitives that carries basic operations of the NETCONF operation layer as <get>, <get-config> and <edit-config> (notification are planned). The Data store Manager layer entity is responsible for maintaining a virtual database of configuration (and state) data and provides a read / write access to these data (state data being read-olny).

At the starting of a session the Data Store Manager looks for modules in a text configuration file and dynamically loads them. A module is a piece of code that accesses to specific configuration and state information with an interface compliant with NETCONF operations. For example there are modules for network interfaces, system, protocols like RIP or OLSR... When a module is loaded it must provide the location of its data by giving a path from the global root (the <netconf> node) to the root node of the module. This path is expressed with the Xpath notation. For example, the interfaces module is localized with the "/netconf/network/interfaces" expression and it maintains data under the <interfaces> node. Thus, part of the global Data Store is managed by the Data Store Manager (light grey on slide 5) and the rest is distributed among modules (dark grey sub trees). This enables modularity of the server without increasing the complexity of the Data Store Manager.

The format for the YencaP configuration file is open to any extends through a <parameters> markup that contains <parameter> items with name and value attributes. The integration of YANG into YencaP is made by adding a parameter with the "yang" name attribute and the module name as value attribute. For each module inside the YencaP server there is one and only one YANG module. One can see on the example that there is a <namespace> markup before the module parameters list. This name space is needed inside NETCONF requests to distinguish its XML naming. It must be the same as the one defined in the YANG module and can easily be extracted from the YangTreeNode.



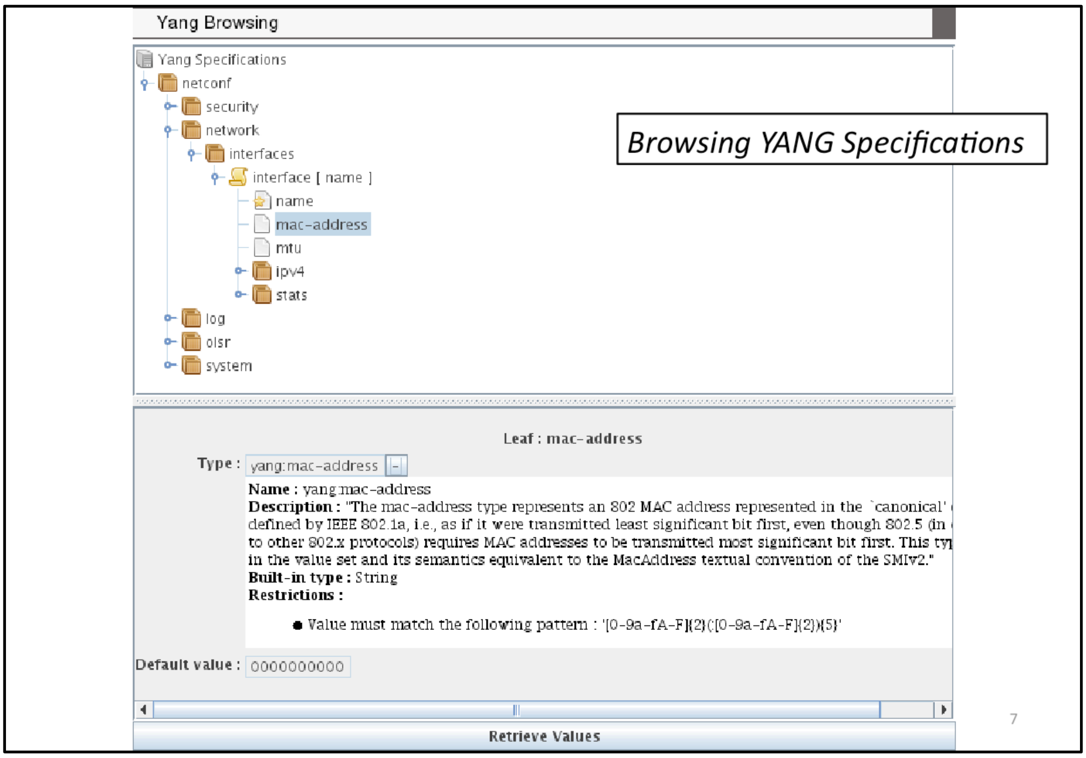
YencaP Manager : a NETCONF Client

```
<hello>
        <capability>
                YANG module reference
        </capability>
</hello>
```

YencaP Manager is an open source NETCONF application that can send queries and receive responses with any NETCONF server. The NETCONF Client can have several NETCONF sessions with different servers at one time. Each of theses sessions is initialized from the HTTPS server inside the YencaP Manager when a user opens a HTTPS session. There is a one to one mapping between HTTPS and NETCONF sessions even if two users are accessing the same NETCONF server. The couple (YencaP / YencaP Manager) forms the ENSUITE framework.

As shown in slide 5 we had to extend YencaP to announce which YANG modules it implements (together with version and revision information) as a capability in its standard hello message. On the client side, a YANG loader will be used by the YencaP Manager when such a capability is detected. We do not constraint the YencaP Manager to only work with YANG but to accept servers that are YANG enabled or not. The YANG loader gets the specifications from an external repository and builds a specific YangTreeNode for the data model maintained by the server. The YANG loader is a java program that uses jYang to dynamically parse YANG specifications. We took this choice because we suppose the YencaP Manager will discover servers without knowledge of their configuration and thus must be able to dynamically load and parse any YANG specification. There is also the creation of "glue" parts in the YangTreeNode as a virtual netconf container, that are needed to built one YangTreeNode from several YANG modules. The YANG specification repository is shown as an external element of the YencaP Manager as it should be a global repository implemented for example as a web service.

Once a HTTPS session is opened the user can ask for the configuration of a YANG enabled device. In doing so it receives a java applet that contains the YangTreeNode for this server only. The applet will be loaded by the web interface to provide the user with a graphical interface representing the configuration.



Browsing YANG Specifications

Leaf : mac-address

Type : yang:mac-address -

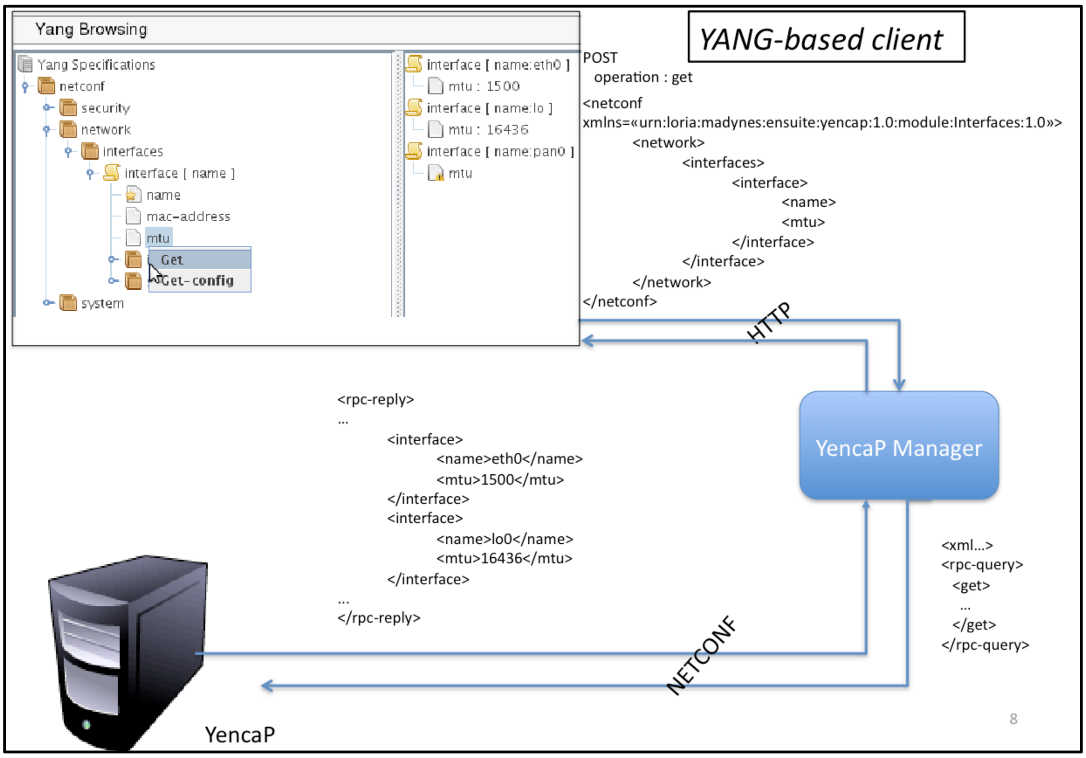
Name : yang:mac-address
Description : "The mac-address type represents an 802 MAC address represented in the `canonical'
defined by IEEE 802.1a, i.e., as if it were transmitted least significant bit first, even though 802.5 (in
to other 802.x protocols) requires MAC addresses to be transmitted most significant bit first. This ty
in the value set and its semantics equivalent to the MacAddress textual convention of the SMIv2."
Built-in type : String
Restrictions :

- Value must match the following pattern : '[0-9a-fA-F]{2}(:[0-9a-fA-F]{2}){5}'

Default value : 0000000000

Slide 7 shows the applet part of the web interface, the user gets when asking for the configuration of a device. This first view can be used as a YANG specification browser that looks like a file system browser (we use the swing Jtree interface). The tree view matches well with YANG because it defines a schema tree. Specific icons are used to distinct nodes, here NETCONF, network and interfaces are all YANG containers, interface is a YANG list and name, mac-address, mtu are YANG leaves. A YANG list can have some key inside its leaf as is the “name” leaf referenced inside brackets in the “interface” list and by a little star on its leaf icon. When selecting a leaf in this tree, the lower part of the applet shows details of the YANG specification, as the type of a leaf and constraints such default value or range intervals. A leaf type is always at least of a built-in types (as string, int8,…) and can be refined by other types with added constraints or to use an existing useful type (as a mac-address). This is the meaning of the “-” (or “+”) behind the name type. When “+” is set (by one mouse click on the “-”) then only the built-in type is displayed.

As we said in the previous page there is a “netconf” container while there is no YANG module called “netconf”. The YANG loader has created a virtual container called “netconf” and has plug it on top of the existing “network” module (as for security, log, olsr and system YANG modules).



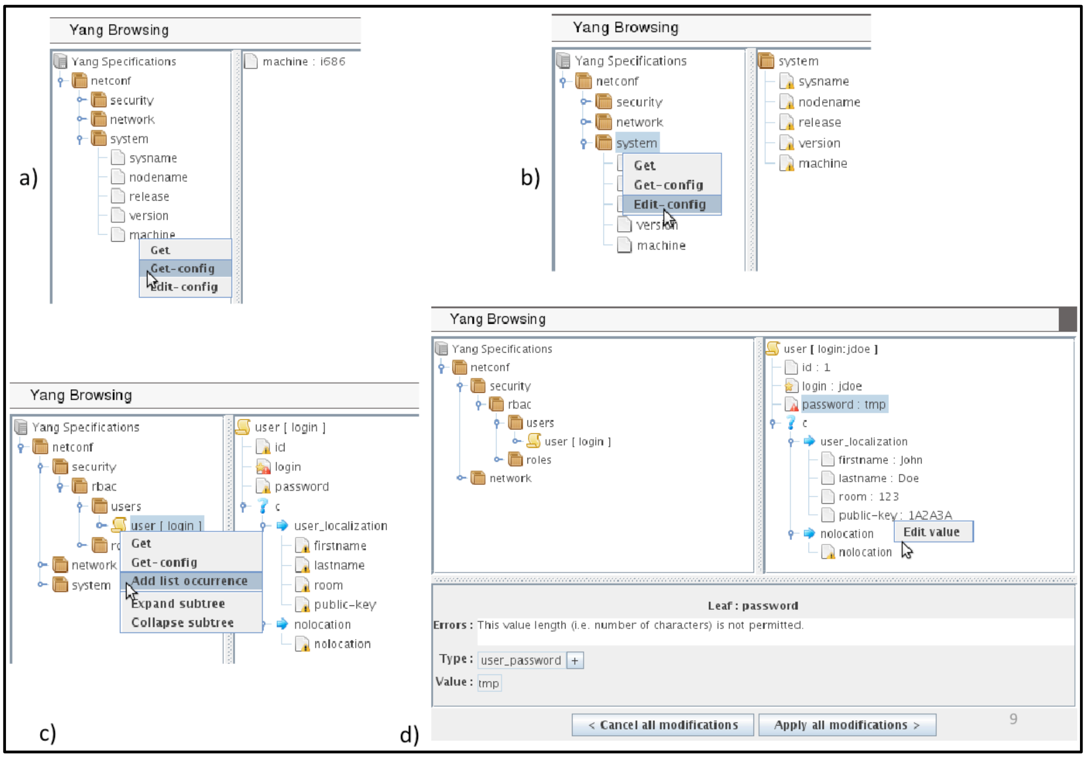
```
POST
  operation : get
<netconf
xmlns=«urn:loria:madynes:ensuite:yencap:1.0:module:Interfaces:1.0»>
        <network>
                <interfaces>
                        <interface>
                                <name>
                                <mtu>
                        </interface>
                </interface>
        </network>
</netconf>
```

```
<rpc-reply>
…
        <interface>
                <name>eth0</name>
                <mtu>1500</mtu>
        </interface>
        <interface>
                <name>lo0</name>
                <mtu>16436</mtu>
        </interface>
…
</rpc-reply>
```

```
<xml…>
<rpc-query>
  <get>
    …
  </get>
</rpc-query>
```

From the slide 7 one can request the indirectly connected NETCONF device by a mouse contextual menu that pops-up when the right button is pressed on a YANG data type. When one of the standard NETCONF operations is chosen, the request is built from the root node (here the netconf virtual container) to the tree position of the selected node. The resulting XML document is sent inside a HTTP POST request. A specific header called "operation" is used to specify which NETCONF operation must be performed on the server (get, get-config or edit-config). Note that the key of the list is added to the request while it is not explicitly asked. This is an optimization because subsequent requests on lists (and especially on list entries) will likely need the key.

When the request is received by the YencaP Manager, the latter adds the rpc and filter mechanisms to surround the XML document received and sends a valid NETCONF request. From this step we are independent of any YANG concern because we are in a full NETCONF session. So the reply is sent by the NETCONF server to the YencaP Manager and this finish the NETCONF operation. Following is simply a cleaning of NETCONF XML data until the first node of the Data Store and its forwarding to the client applet that is waiting for the response. The slide 8 shows the response on the right part of the management applet. The request is synchronous because even if one request contains several data (as can be a request on a list) all of them are returned by one response. Note we have made our protocol synchronous on top of HTTP with several asynchronous requests. We plan to allow multiple selections for the same NETCONF operation to give access to the separate part of the NETCONF Data Store in one request.

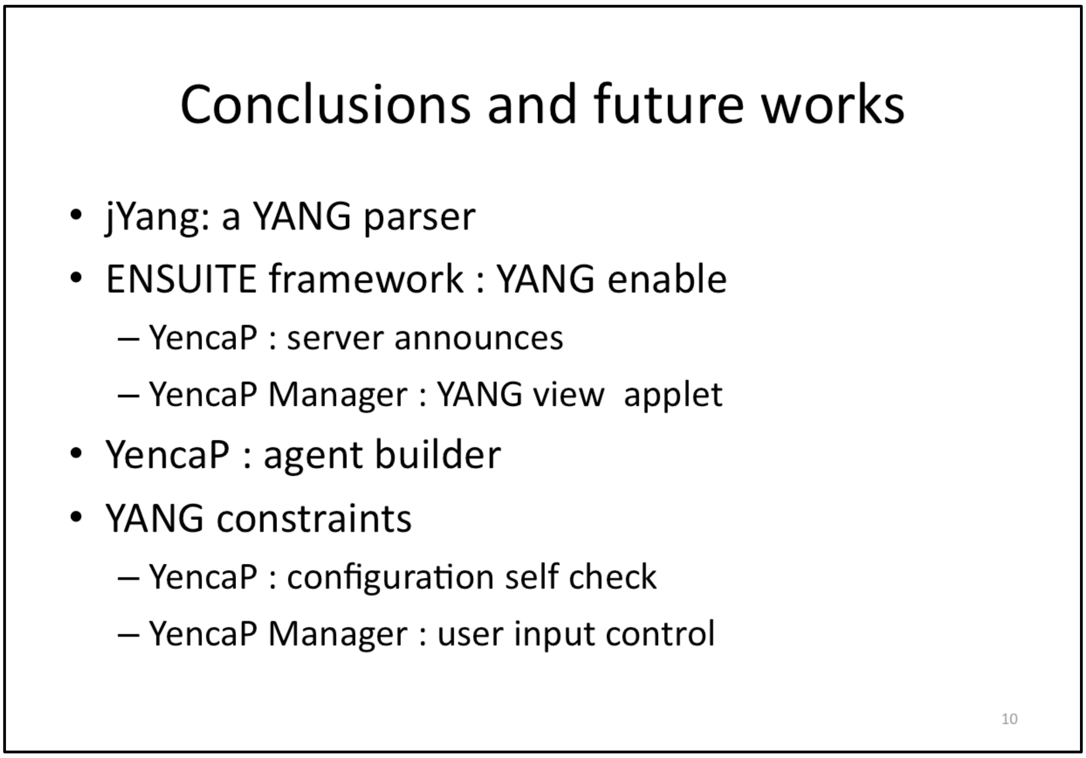


Errors : This value length (i.e. number of characters) is not permitted.

Type : user_password

Value : tmp

The slide 9 depicts some functionalities of the Yang Browsing client. Part (a) is a simple access to a leaf in a container and where we get the response of a get-config operation. Part (b) shows that when editing a container, its components are listed with a warning until a correct value is given. Part (c) is an edit-config for a list. Here a list is edited entry by entry (that we call a list occurrence) and one can see an empty list entry ready to be filled. Note that a red mark is on the “login” leaf because it is the key of the list and so its value must be set. Part (d) illustrates the choice representation and its edition. A choice case (“user-localization” or “nolocation”) can only be edited if all of its components are set. On the same part the leaf called “password” is read marked because the value is not long enough. This is an example of dynamic constraint one can check with the tool. Range values of integer or float, pattern matching of string are also checked.



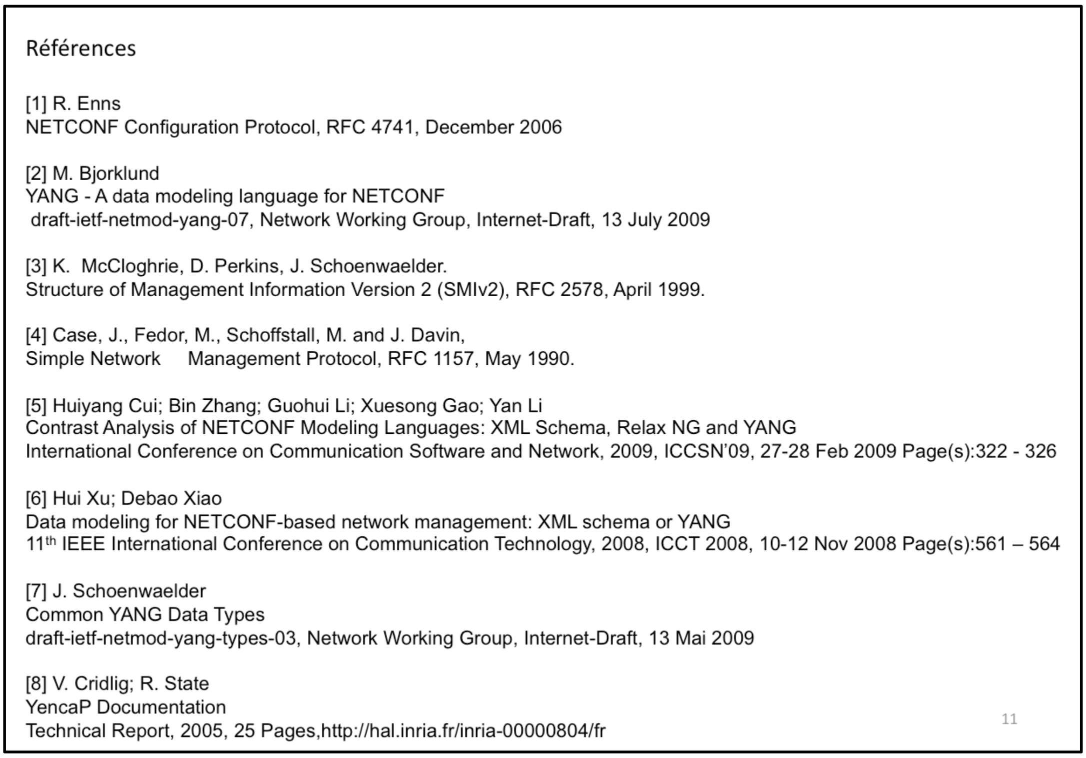
Conclusions and future works

- jYang: a YANG parser
- ENSUITE framework : YANG enable
  - YencaP : server announces
  - YencaP Manager : YANG view applet
- YencaP : agent builder
- YANG constraints
  - YencaP : configuration self check
  - YencaP Manager : user input control

We provide two contribution to the network configuration domain. The first one is a YANG parser and semantic checker close to the actual version of the draft definition of YANG. The second contribution is the support within the ENSUITE framework of YANG based models both on the server and the client side.

We plan to extend YencaP to be able to generate parts of its code from YANG specifications, or build generic parts, to ensure the server maintains a valid Data Store compliant with YANG . The server has to be able to send notifications especially those defined in YANG and must also accept user defined operations as there are YANG rpc and notification statements to do this.

We are also interested by all YANG constraints one can specify. Default value, must and presence conditions, references between values, length or pattern matching are some examples of such constraints. If one can have a NETCONF server with such knowledge this server will be enabled to autonomously checks its configuration. At the client side the constraints can ensure the manager does not make mistakes in its configuration operations and can notify users if constraints are not respected.



Références

[1] R. Enns
NETCONF Configuration Protocol, RFC 4741, December 2006

[2] M. Bjorklund
YANG - A data modeling language for NETCONF
draft-ietf-netmod-yang-07, Network Working Group, Internet-Draft, 13 July 2009

[3] K. McCloghrie, D. Perkins, J. Schoenwaelder.
Structure of Management Information Version 2 (SMIv2), RFC 2578, April 1999.

[4] Case, J., Fedor, M., Schoffstall, M. and J. Davin,
Simple Network Management Protocol, RFC 1157, May 1990.

[5] Huiyang Cui; Bin Zhang; Guohui Li; Xuesong Gao; Yan Li
Contrast Analysis of NETCONF Modeling Languages: XML Schema, Relax NG and YANG
International Conference on Communication Software and Network, 2009, ICCSN'09, 27-28 Feb 2009 Page(s):322 - 326

[6] Hui Xu; Debao Xiao
Data modeling for NETCONF-based network management: XML schema or YANG
11th IEEE International Conference on Communication Technology, 2008, ICCT 2008, 10-12 Nov 2008 Page(s):561 – 564

[7] J. Schoenwaelder
Common YANG Data Types
draft-ietf-netmod-yang-types-03, Network Working Group, Internet-Draft, 13 Mai 2009

[8] V. Cridlig; R. State
YencaP Documentation
Technical Report, 2005, 25 Pages,http://hal.inria.fr/inria-00000804/fr