Configuration Management Simplified

Executive summary

The IETF has recently standardized the NETCONF configuration management protocol and is currently in the process of standardizing a NETCONF-oriented data modeling language called YANG. These two new technologies promise to drastically simplify network configuration management. This paper shows how NETCONF and YANG can be employed to make next-generation configuration management systems considerably simpler, more understandable, and also more robust than current systems.

Why NETCONF and YANG?

The NETCONF protocol makes it possible for network management software to orchestrate transactional changes to several devices of different types. Even complex changes that affect several devices can be executed in an all-or-nothing mode. This means that a large class of error-prone recovery code in the network management system can be eliminated.

The NETCONF protocol moves the responsibility of consistency checks and error recovery to the managed devices, thus making the manager code simpler and the network management system more robust. The fact that the managed devices participate in the two-phase commit issued by the manager allows for implementing in-service configuration updates, spanning several devices in a transactional manner.

The YANG model specification language has two major implications for network management systems. First and foremost, if the managed device publishes a strict XML-based data model that it promises to adhere to, the network management system can reuse that very same model. We will see how the strict data model at the managed device is explicitly used at all layers in the network management system.

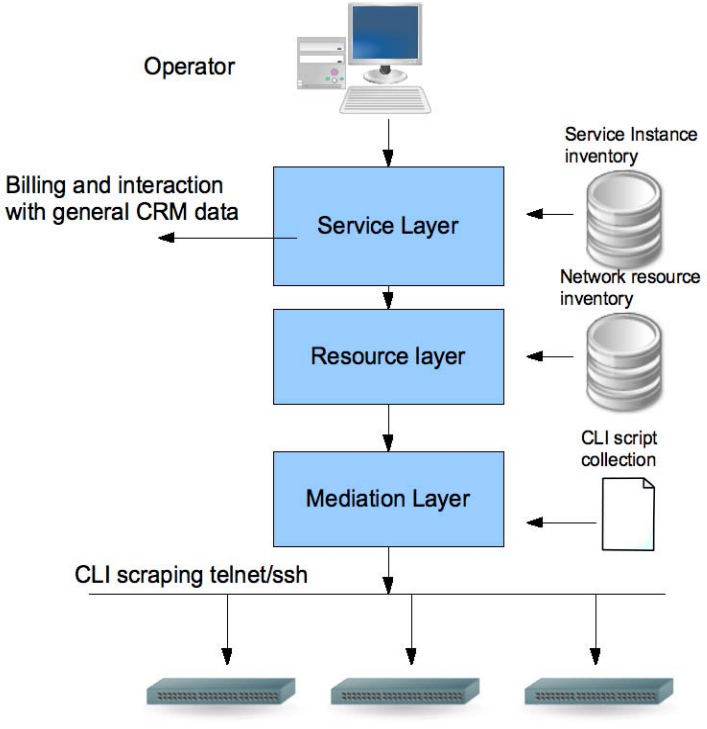
Secondly, the advent of standard YANG models for common networking tasks, such as assigning IP addresses to interfaces or changing DNS servers, means that a manager can leverage that by executing identical code towards different types of devices (assuming they all implement the same standard capability). Compare this to how standard SNMP MIBs have made it possible for network management systems to perform several tasks independently of the device model.

With YANG as a modeling language, device vendors and network management system designers now speak a common language. Prior to the introduction of NETCONF in the managed devices, vendors defined the capabilities of their devices largely through a set of CLI commands with an accompanying user guide. YANG models give the vendors a means to communicate a precise data model of the device to network management system designers. The YANG model works as formal glue between the team that designs the device and the network management design team.

YANG provides a more concise and readable notation of XML data models. There is symmetric mapping between YANG and the corresponding XML notation, allowing XML-based tools to validate, transform or filter the data model information.

Current Configuration Manager architecture

Let us define a Configuration Manager (CM) as the aspects of an OSS, NMS or EMS solution that reconfigures and provisions managed devices. Many CM solutions have a layered approach with the following three layers.



At the top resides the Service Layer. Here typical concepts are tasks like "provision a new customer" or "increase bandwidth for customer X".

The Service Layer defines concepts that relate to the day-to-day business flows for a service provider. Thus it must be easy and fast to make changes to the service models as new business scenarios emerge. The service models are typically modeled with SID, UML, or proprietary languages. The Service Layer also maintains an inventory of service instances in persistent storage.

Below the Service layer resides the Resource Layer. This is where individual devices are modeled. Usually this is done in XML Schema, UML, or proprietary languages. The task of the Resource Layer is to provide a mapping from the Service Layer to actual device manipulations. Thus the Service Layer task "provision new customer" in, for example, an ADSL provider scenario includes a series of manipulations of switches, routers, and DSLAMs. The individual devices and their configurations that are involved in the configuration change are represented in abstract form as data structures local to the CM.

At the Resource Layer, the CM maintains a database of network resource instances. Thus, this layer has

a database of all managed devices, their current configurations, their models, and also the version of the software on the devices.

If the managed devices do not have well-defined data models, as would typically be the case when the devices only exports command line interfaces, we have to model the devices at the Resource Layer. Such modeling can be automated with tools that walk through the CLI and generate an XML-based model or through using hand-written modeling languages such as UML.

Both approaches are problematic. First, the device models at the Resource Layer are bound to make assumptions that are not always right, leading to subtle errors. Second, it is very common to ignore aspects of the device that are currently not relevant at the Resource Layer, making later extensions more difficult.

Finally, below the Resource Layer resides the Mediation Layer. The task of this layer is to map changes to the CM local data structures in the Resource Layer to actual configuration change commands on the actual devices. Today, this typically includes CLI scripting towards devices.

NETCONF-centric Configuration Manager architecture

NETCONF and YANG promise to simplify all three layers in the above architecture.

First, the Mediation Layer is simplified by replacing error-prone CLI scripting with strict XML processing. The NETCONF protocol defines how to execute the configuration changes, so there is no need for CLI scripting to execute the actual configuration changes.

Second, the Resource Layer becomes simpler, since devices themselves have stringent YANG models that define the device configurations. These YANG models should be reused in the Resource Layer, since they are not only accurate, capturing fine semantic details of the managed devices, they are also considerably easier to understand. The latter fact should not be underestimated. If the CM developers have a good understanding of the devices they intend to manage it greatly enhances the chances that the management code is correct.

Third, the connection between the Service layer and the Resource Layer becomes easier. A good way to leverage the YANG models for the managed devices is to utilize technologies such as XMLBeans, Castor, Xgen or JAXB, where the YANG models can be compiled into a set of well-defined Java classes which can be used to manipulate the configuration instances. Thus, if we utilize a good Java XML binding tool, we get code at the Service Layer that in a type-safe way can manipulate all the configuration aspects of the managed devices. Since the YANG model is compiled into Java classes we can never construct structurally faulty configurations. As we will see later in this paper, the CM code can now nicely manage different software versions of the managed devices.

Another aspect of configuration management that may be improved is error management. With traditional CLI methods where there are typically no proper transactional capabilities, and no proper rollback management, the CM must be prepared to rollback configuration changes in a much more complex way than when using NETCONF.

A small example

In this section we provide a small example that makes these points from the previous section more clear. Assume we have a YANG model that looks like:

```
module acme-system {
    namespace "http://acme.example.com/system";
    prefix "acme";

    organization "ACME Inc.";
    contact "joe@acme.example.com";
    description
        "The module for entities implementing the ACME system.";

    revision 2007-11-05 {
        description "Initial revision.";
    }

    container system {
        leaf rack-slot-number {
            type string;
            description "Rack slot number for this system";
        }

        leaf-list domain-search {
            type string;
            description "List of domain names to search";
        }

      list interface {
            key "name";
            description "List of interfaces in the system";
            leaf name {
                type string;
            }
            leaf type {
                type string;
            }
            leaf mtu {
                type int32;
            }
      }
    }
}
```

The above defines precisely the data model for parts of a fictitious device. Given the appropriate Java XML binding tool, we get generated Java classes that can manipulate a *System* tree.

For example we may get an *Interface* class as:

```
public class InterFace {

    public InterFace(String nameValue);
    public void setNameValue(String nameValue);
    public String geNameValue();
    public void setTypeValue(String typeValue);
    public String getTypeValue();
    public void setMtuValue(int mtuValue);
    public int getMtuValue();
}
```

The code in the Resource Layer can use the *System* and the *Interface* classes and all other classes generated from the YANG model. The code in the Mediation Layer then uses a NETCONF client library to execute the configuration changes towards the managed devices. We can envision code similar to:

```
// Create a config tree fragment using the generated code
    System system = new System();
    Interface if = system.addInterface("eth3");
    if.setTypeValue("Ethernet");
    if.setMtuValue(1500);

    // Send it to the device using the NETCONF library
    NetconfSession sess = new NetConfSession(someHost);
    sess.editConfig(system);
```

The *editConfig()* method will marshal the system object into its XML representation

```
<system>
  <interface>
     <name>eth3</name>
     <type>Ethernet</type>
     <mtu>1500</mtu>
  </interface>
</system>
```

and pack that data into the appropriate NETCONF *edit-config* RPC, send it over SSH and collect the reply from the NETCONF agent at the managed device.

Many three-layered CMs implement the Mediation Layer as a set of independent adapters, where the adapter knows how to communicate with the specific devices. Furthermore, the CM often has a SOAP interface to the adapters. The above approach fits nicely into such an architecture. The Service and the Resource Layers can freely manipulate the high-level objects as defined above, e.g. it can manipulate *System* and *Interface* objects. Once the higher layers have decided on the appropriate configuration changes, the Mediation Layer can export a SOAP interface containing, for example, an *editConfig()* method. Hence, we do not have to abandon the three-layered architecture in the CM, we can just simplify it.

Standard YANG modules

Similar to how the SNMP community has developed a set of standard MIBs, the IETF is beginning to develop standard YANG modules for common configuration items.

Each standard YANG module is uniquely identified by an XML namespace identifier. When a NETCONF client connects to an agent, the client and the agent exchange hello messages. The hello message from the agent contains a list of Uniform Resource Identifiers (URIs) identifying the capabilities of the agent. Assume that in the future the IETF publishes a YANG module describing how a network element defines how its local syslog daemon should operate. That is a YANG version of */etc/syslog.conf* that captures all commonalities between all syslog daemons. A CM connecting over NETCONF to a managed device does not have to care in the least about what type of device it is communicating with as long as the device publishes the standard URI defining syslog operations. That is when the CM code sees the standard URI for syslog operations, the CM can reconfigure the syslog aspects of the device without knowing what type of device it is.

Managing different device software versions

The archaic technology of CLI screen-scraping lies at the heart of today's Mediation Layers. The CLI is still the most common way to reconfigure a device. This is the source of many errors and mis-configurations. If a managed device undergoes a software upgrade and spelling errors are corrected in the new CLI version, the CM code breaks. This is just bad engineering. It is the job of the Mediation Layer to perform the CLI screen-scraping, typically through "expect" style scripting.

In this approach, different versions and software loads of the managed devices are identified by explicitly letting the Mediation Layer execute various version checking commands immediately after logging in to the device.

Compare this with NETCONF which has strict methods that allow the CM to always check the versions of the managed devices. Furthermore, many software upgrades do not affect the CM - they are backwards compatible at the NETCONF layer. The CM software sees that through the NETCONF protocol and it is the responsibility of the device manufacturers to indicate version information through NETCONF.

The CLI screen scrapping code, also known as "Expect Hell", has a much more difficult task. If a managed device is upgraded, the CM developers must carefully read the change notes for the device and look for CLI changes. The CM developers cannot know whether the upgraded device will be manageable through the old CM code or not. In general this makes device upgrades in the CM network a major work item not just for the staff that runs the network, but also for the CM developers. As a result, important software upgrades for managed devices are put on hold due to CM integration issues. This also includes e.g. security updates!

How transactions affect the Configuration Manager

The NETCONF protocol has explicit transaction support. This may be the single most important change that NETCONF brings to the table. It makes it not just possible but even easy to execute multi-device, multi-vendor changes that either applies in its entirety or not all.

A NETCONF-enabled device can support a candidate data store. The candidate configuration data can be manipulated without impacting the device's current configuration. A NETCONF *Commit* operation is performed to set the actual configuration of the device to the current contents of the candidate configuration. The candidate store is utilized by the manager code to implement transactions spanning multiple devices as follows:

1. The manager modifies the candidate store on all involved devices.
2. The manager asks all devices to validate the new - not yet applied - configurations.
3. If all devices acknowledge the validation request, the manager finally orders all devices to commit the proposed changes into their running stores, effectively making the configuration change permanent.

Another key feature that has a major impact on CM design is the *Confirmed Commit* capability of NETCONF that associates a timeout with the transaction. It is the responsibility of the CM to confirm the commit within the stipulated timeout. Failing to do so, the managed devices will automatically rollback to the previous configuration. This makes it possible to try out a configuration change in live operations. If the change is bad, it can easily be backed out. If the change is so bad so that the managed device looses network connectivity with the manager, the device itself automatically backs it out.

The responsibility of error recovery is now moved from the CM to the managed devices themselves. Imagine the CM code that handles the case where, say, five devices have received a CLI configuration change and while reconfiguring the sixth device it fails. The CM has to explicitly undo the configuration changes so far performed. The probability is pretty low that the CM code correctly executes this in all error scenarios.

Most devices can validate configuration changes. However, a set of configuration changes may make sense and be correct for each individual managed device, but the net effect of the changes in combination may be bad. In the worst case scenario the CM looses network connectivity to a managed device due to the configuration change. With non-NETCONF enabled devices the only remedy here is manual physical on-site repair. In contrast a NETCONF-enabled device that looses the network connectivity to the CM, while executing a transaction, will automatically rollback.

One of the most profound changes resulting from the transaction-oriented approach is that configuration changes are made independent of their order during a transaction. In contrast, changes made through a CLI immediately affect a device's running configuration. For example, a CLI script to enable an interface must be sent to the device before a protocol can be enabled on the device. NETCONF's transaction-based approach is not procedural, but rather declarative. In this example, the NETCONF edit-configuration operation would simply specify the interface configuration and the protocol configuration without any dependencies on order of execution. The device is responsible for applying the changes in the right order. Removing device-dependent ordering constraints is a major benefit, eliminating a significant source of errors in the Mediation Layer of the CM.

Statistics gathering and notifications

In today's networks a plethora of techniques (e.g. CLI screen-scraping, Syslog, SNMP) are employed to gather statistics and runtime data from the managed devices. It is the task of a network management system to gather the individual data items from the individual devices and present network-wide, aggregated statistics.

NETCONF comes with three pieces of technology that addresses runtime data gathering and notifications.

First, the NETCONF protocol distinguishes between configuration data and non-configuration data at the protocol layer. Thus the NETCONF RPC

```
<rpc message-id="101"
          xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
       <get>
         <filter type="subtree">
           <system xmlns="http://acme.example.com/system">
               <interface/>
           </system>
         </filter>
       </get>
     </rpc>
```

will return, as an XML tree, all items under "/system/interface" that are defined as runtime data in the YANG model. The *get-config* RPC returns the configuration data only and no statistics, whereas the *get* RPC returns all data - both configuration and statistics data.

Since the output of the statistics generating functions is well defined inside the YANG models, the network management system code knows exactly the format of replies to the different statistics gathering functions. This is in stark contrast to "show stats" style CLI commands that have ad-hoc output.

Second, NETCONF provides an efficient way of transporting statistics data in bulk through its XML RPC mechanism. A single get RPC may return an arbitrarily large volume of data. This is in contrast with SNMP, where the amount of data transported by each GET command is limited by the size of a UDP packet.

Finally, the NETCONF protocol has built-in support for NETCONF notifications which allows the network management system to subscribe to well-defined streams of XML data from the managed devices. This feature does not replace SNMP traps, but complements it with the ability to send structured data. SNMP traps are unsolicited UDP messages sent from the device to the manager whereas NETCONF notifications are pulled by the manager that must first establish an SSH connection to the managed device and then subsequently subscribe to the notifications "streams" it is interested in.

The format of the NETCONF notifications is defined in the YANG models for the device and it is up to each vendor to define what they perceive as interesting notifications. We may very well see standard YANG models in the future defining well-known notification streams.

Summary

NETCONF moves the responsibility of consistency checks and error recovery to the managed devices. This enables the protocol to robustly support transactional capabilities and proper rollback management, functionality that is extremely hard to implement correctly using traditional technology.

In addition, NETCONF and its data modeling language YANG simplifies the three layers of a configuration management system as follows:

- Mediation Layer: Error-prone CLI scripting is replaced by strict XML processing.
- Resource Layer: After-the-fact modeling of devices is replaced by stringent data models shared by the network devices and the network management system.
- Service Layer: The interface to the Resource Layer is ensured to be consistent with the device data models by employing an XML binding tool to generate Java classes from data models.

Finally, NETCONF complements and enhances SNMP in the areas of statistics gathering and notifications.

References

NETCONF: RFC 4741, RFC 4742

YANG: http://datatracker.ietf.org/drafts/draft-ietf-netmod-yang/

www.tail-f.com
info@tail-f.com

Tail-f Systems Headquarter
Klara Norra Kyrkogata 31
SE-111 22, Stockholm, Sweden
Phone: +46 8 21 37 40

Tail-f Systems North America
109 S. King Street, Suite 4
Leesburg, VA 20175, USA
Phone: +1 703-777-1936