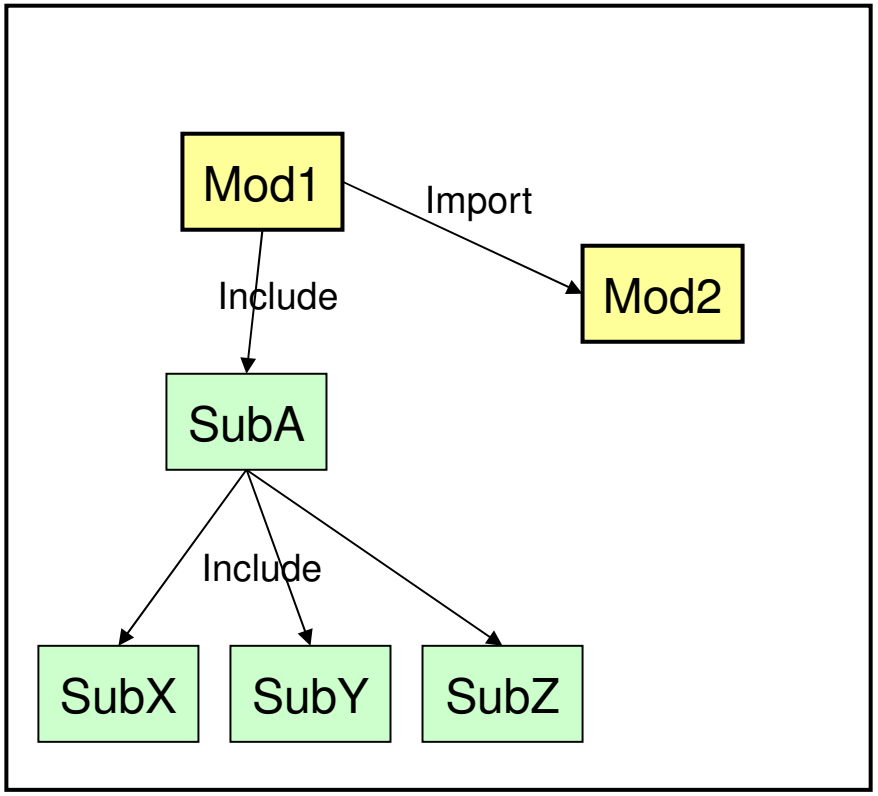
YANG High-Level Presentation
YIN
XML on the wire

IETF 72

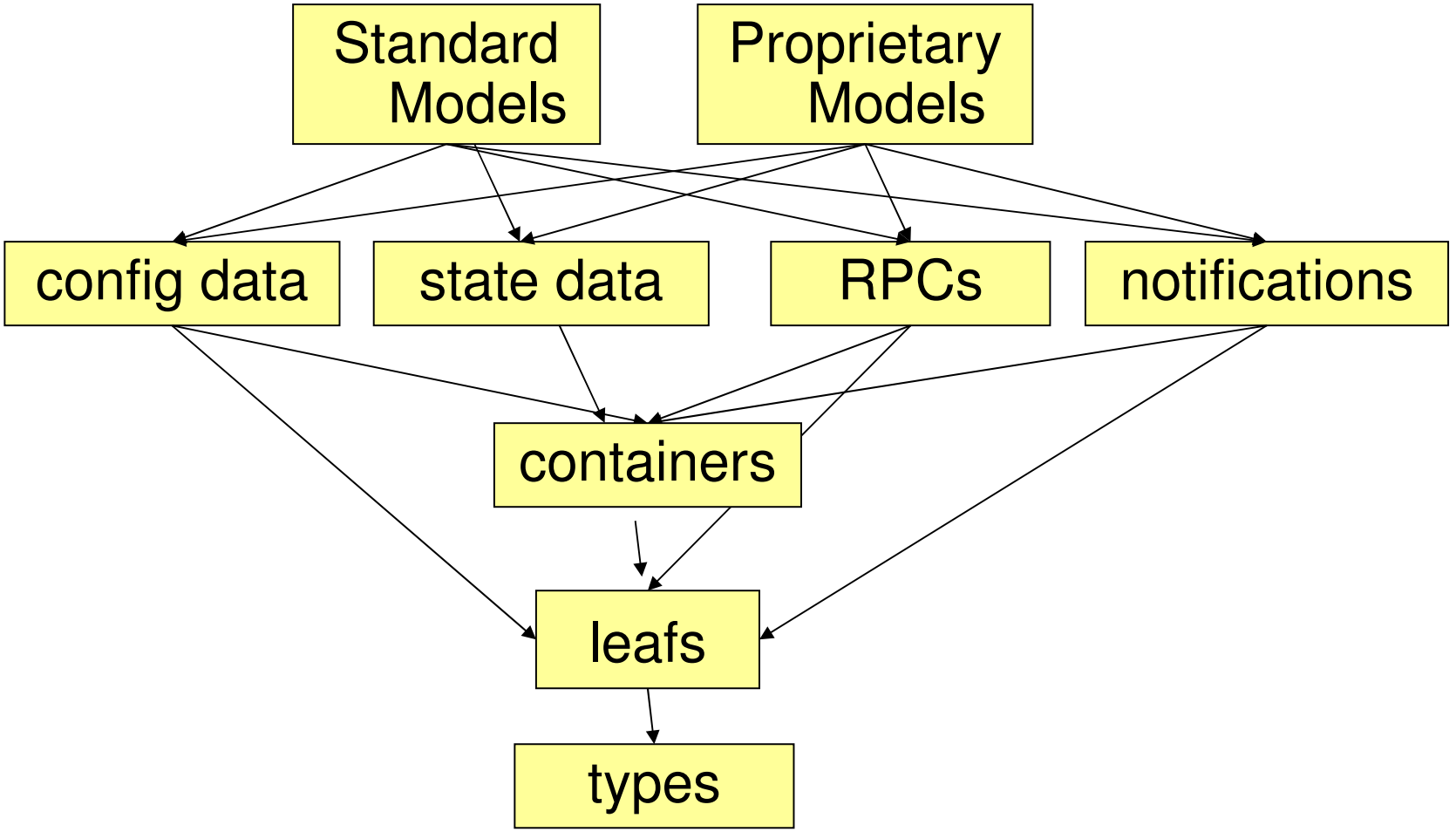
Martin Björklund
mbj@tail-f.com

YANG High-Level

- YANG data models are written in YANG modules
  - Header statements
    - namespace, prefix
  - Linkage statements
    - import, include
  - Meta statements
    - organization, contact,
    - description, reference
  - Revision history
    - revision



YANG Concepts



YANG module

- reusable simple type definitions (typedefs)
- reusable groups of nodes (groupings)
- data definitions (configuration and state data)
- operations (rpcs)
- notifications
- extensions to other modules

Data definitions

- Structure
  - container, list, leaf-list, leaf
- Syntax
  - type
- Semantics
- Integrity constraints
  - YANG allows to formally specify integrity constraints on configuration data. These constraints are enforced by the server.
- Relationships

Structure 1(3)

- container
  - Used to group related nodes in a subtree. It has no data value, only child nodes.
  - Can be purely organizational or carry some semantic meaning by its presence in the configuration data store.
- leaf
  - A leaf node has a value and no children.
  - Can have a default.
  - Can be mandatory or optional.

```
container server {
    container address {
        leaf ip { type inet:ip-address; }
        leaf port { type inet:port-number; }
    }
}
```

Structure 2(3)

- list
  - Used to define a sequence of list entries, where each list entry contains any number of child nodes of any type, including other lists.
  - A list entry is uniquely identified by the value of its key(s).
  - The order of the entries can carry semantic meaning (e.g. firewall filter rules)

```
list user {
    key login-name;
    leaf login-name { type string; }
    leaf full-name { type string; }
}
```

Structure 3(3)

- leaf-list
  - Used to define a sequence of simple values.
  - The order of the entries can carry semantic meaning (e.g. dns server list)
  - A leaf-list entry is uniquely identified by its value.

Reusable simple types

`typedef` defines a named type which can be reused in the same module or in other modules. The new type is derived from some existing type – built-in or derived.

```
typedef port-number {
    type uint16 {
        range "1..65535";
    }
    ...
}
```

Reusable groups of nodes

`grouping` defines a named group of related nodes. The grouping can be reused in the same or other modules.

```
grouping address-type {
    description "A reusable address group.";

    leaf ip {
        type inet:ip-address;
    }
    leaf port {
        type inet:port-number;
    }
}

container server {
    container address {
        uses address-type;
    }
}
```

Operations

The `rpc` statement defines a new operation

- operation name
- input parameters
- output parameters

An operation can reference data in the data model, e.g. an operation which restarts an interface can be defined.

```
rpc restart-interface {
    input {
        leaf interface-name { ... }
    }
}
```

Extensions to other modules

It is anticipated that vendors will need to add vendor-specific configuration data to standard data models. This must be done without breaking applications that only understand the standard data models.

YANG provides the `augment` statement, which is used to insert nodes into another data model. The augmentation can be conditional, so that the nodes are only inserted when a certain condition is met (e.g. some nodes are only added to interfaces of a specific type).

YIN

- YANG is designed with a simple, regular syntax, optimized for humans to read and write.

- YIN is a semantically equivalent syntax in XML
  - YANG(YIN(M)) == M
  - design goal: as readable as possible

- The XML syntax is useful for using XML tools such as XSLT for things like:
  - extract text for documentation
  - generate code
  - transform into other forms
  - display in / generate from graphical XML tools
  - ...

YIN Mapping 1(3)

YANG grammar:

```
statement = keyword [argument] (";" / "{" *statement "}")
```

YIN mapping rules:

- Each keyword is translated into an XML element.
- The optional argument is translated into an XML attribute or an XML element, depending on the keyword.
- Any substatements are translated into child elements.

YIN Mapping 2(3)

Example:

```
leaf address {
    type inet:ip-address;
}
```

```
<leaf name="address">
    <type name="inet:ip-address"/>
</leaf>
```

YIN Mapping 3(3)

When an extension is declared, the name of the extension's argument (if any) is specified, and whether the argument is mapped to an XML element or attribute in YIN.

```
module my-extensions {
    ...
    extension help {
        argument text {
            yin-element true;
        }
    }
}
```

```
module my-configs {
  ...
  import my-extensions { prefix myext; }
  container interface {
    myext:help “help msg”;
  }
```

```
<container name=”interface”>
  <myext:help>
    <text>help msg</text>
  </myext:help>
</container>
```

NETCONF XML mapping 1(4)

All nodes in a module are encoded as XML elements in the module's namespace:

```
module my-config {
    namespace “http://example.com/my-config”
    container system {
        ...
    }
}
```

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
  <config>
    <system xmlns=”http://example.com/my-config”>
      ...
    </system>
  </config>
</rpc>
```

NETCONF XML Mapping 2(4)

- container
  - XML element with no text, only XML sub elements.
- leaf
  - XML element with text data, and no XML sub elements
- leaf-list
  - Each entry in the list is encoded as one XML element like a leaf.
  - If the leaf-list is ordered-by user, then order MUST be preserved in the XML encoding.
  - If the leaf-list is ordered-by system, then the entries can be encoded in any order.

NETCONF XML Mapping 3(4)

- list
  - Each entry in the list is encoded as one XML element like a container, containing each key as a sub element, followed by the rest of the list entry content as sub elements.
  - If the list is ordered-by user, then order MUST be preserved in the XML encoding.
  - If the list is ordered-by system, then the entries can be encoded in any order.

NETCONF XML Mapping 4(4)

- Example of a list encoding:

```
list server {
    key name;
    leaf name { ... }
    leaf ip { ... }
    leaf port { ... }
}
```

```
<server>
  <name>smtp</name>
  <ip>192.0.2.1</ip>
  <port>25</port>
</server>
<server>
  <name>http</name>
  <ip>192.0.2.1</ip>
  <port>25</port>
</server>
```

Free YANG tools

- libsmi (http://www.ibr.cs.tu-bs.de/projects/libsmi)
  - generates YANG from SMIv2
  - open source
- pyang (http://code.google.com/p/pyang)
  - YANG validator
  - translates between YANG and YIN
  - generates XSD and DSDL from YANG and YIN
  - open source
- yangdump (http://www.netconfcentral.org/download)
  - YANG validator
  - generates XSD and HTML from YANG
  - free binary

Things to work on

- Feedback from IPFIX YANG users:
  - Need a mechanism to divide a module into optional-to-implement parts. A client must be able to query the server which optional parts of a module it implements.
  - Need a mechanism for a client to discover server-specific variance (for example server-specific limits on number of entries in lists).
- Rules for module revisioning
- Clarifications to the document (please review!)
- Several open issues to be discussed tomorrow