jYang : A YANG parser in java

Emmanuel Nataf, Olivier Festor

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

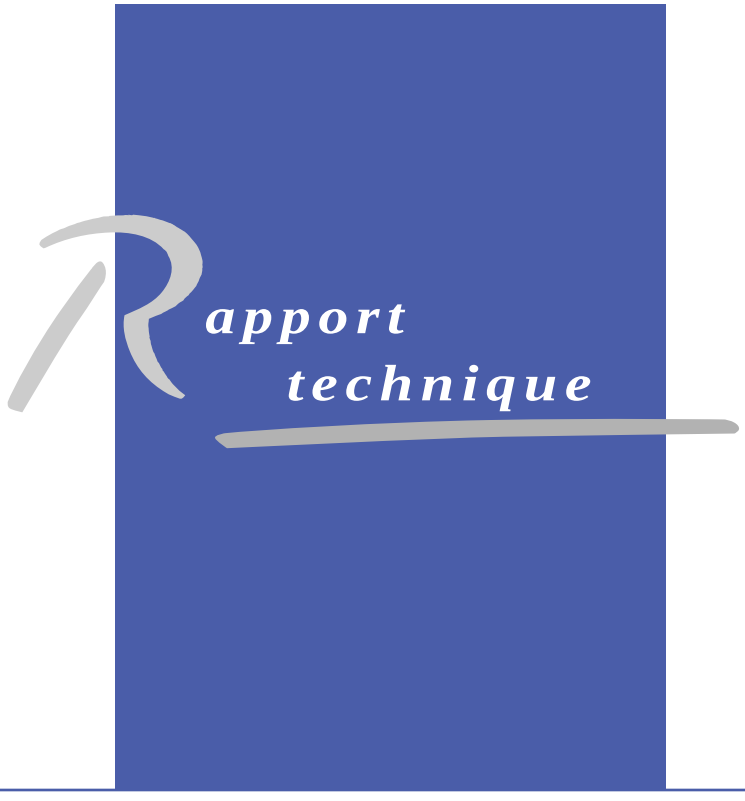
jYang : A YANG parser in java

Emmanuel Nataf — Olivier Festor

N° ????

August 2009

Thème COM



ISSN 0249-0803 ISRN INRIA/RT--????--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
NANCY - GRAND EST

jYang : A YANG parser in java

Emmanuel Nataf*†, Olivier Festor‡

Thème COM — Systèmes communicants
Équipe-Projet Madynes

Rapport technique n° ???? — August 2009 — 29 pages

Abstract: The NETCONF configuration protocol of the IETF Network Working Group provides mechanisms to manipulate the configuration of network devices. YANG is the language currently under consideration within the IETF to specify the data models to be used in NETCONF . This report describes the design and development of a syntax and semantics parser for YANG in java.

Key-words: parser, java, yang, netconf

* Maître de conférences - Nancy University
† Madynes INRIA project
‡ Directeur de Recherche INRIA

Centre de recherche INRIA Nancy – Grand Est
LORIA, Technopôle de Nancy-Brabois, Campus scientifique,
615, rue du Jardin Botanique, BP 101, 54602 Villers-Lès-Nancy
Téléphone : +33 3 83 59 30 00 — Télécopie : +33 3 83 27 83 19

jYang : un analyseur yang en java

Résumé : Dans le contexte du groupe de travail administration et opération des réseaux de l'IETF, le protocole de configuration NETCONF a été développé pour manipuler la configuration d'équipement réseau. YANG est le langage en cours de standardisation dans ce groupe. Il permet de spécifier des modèles de données utilisable dans l'approche NETCONF. Ce rapport décrit la conception et la réalisation en java d'un analyseur syntaxique et sémantique de spécifications YANG.

Mots-clés : parser, java, yang, netconf

1 Introduction

It is common in the network management world that a protocol and a data model are separated even if jointly designed, as it was already the case in the SNMP[3] protocol and its SMI[7] data modeling,COPS[4] and SPPI[6], or SMIng[8] (GDMO and CMIS or WBEM and CIM outside the IETF scope).

NETCONF [5] is the IETF standard that emerged from the netconf working group to configure network devices. The netmod[1] working group defines YANG as a candidate language to specify data models of values carried by NETCONF. This report describes a YANG parser called *jYang* that provides a syntaxic and semantic validation of YANG specifications (called modules or sub-modules).

This report first provides a short description of NETCONF where some parts are referenced by YANG. Section 3 details the YANG language concepts and section 5 details the design and implementation of the *jYang* parser.

2 NETCONF protocol

NETCONF is a client/server protocol where the server is a network device and the client a management framework that runs management applications. Protocol requests and responses focus on configuration manipulation such as getting the current configuration, update, create or delete all or some part of it. Configurations are represented in an XML document that contains two sort of data:

- configuration data that is writable and that describes configuration parameters of the NETCONF agent.
- state data that is read-only and that describes operational data such as counter or statistics.

A NETCONF agent can have several configurations each one containing several configuration data. There can be only one active configuration, called the `running` configuration, at the same time. Other configurations, called `candidate` configurations, can exist without interfering with the running one. A special commit capability (cf section 2.4) asks the agent to pass a candidate configuration as the running one.

Figure 1 extracted from [5] shows the layered protocol architecture of NETCONF. The protocol mainly defines operations and how they are carried by rpc mechanisms.

2.1 Transport protocol

NETCONF can use several connection-oriented transport protocols. It requires that a persistent connection is maintained between peers during a potentialy long term `session`. Ressources reservation can be granted for the session and any reserved ressources are released at the end of the connection.

Authentication, integrity and confidentiality must be provided by the transport protocol. A NETCONF implementation must support the SSH transport protocol mapping.

[1] http://www.ietf.org/html.charters/netmod-charter.html

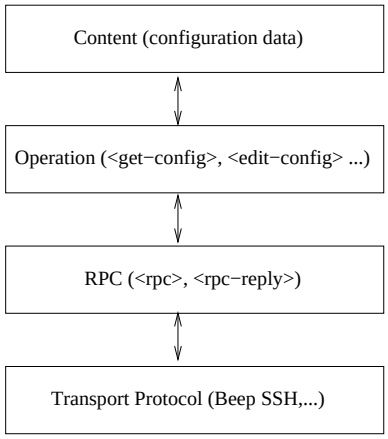


Figure 1: NETCONF protocol layers

The specification language described in this report is not bound to the transport protocol used with NETCONF.

2.2 RPC

The Remote Procedure Call on wich the NETCONF operations are built is described by two XML[2] elements : `<rpc>` for requests and `<rpc-reply>` for responses. The latter can contain a `<rpc-error>` element when an error occurs during the process of a request inside the NETCONF agent.

2.3 Operations

Basic operations are defined as XML elements :

- `<get>` : to retrieve all or part of the running configuration and state data;
- `<get-config>` : to retrieve all or part of a running or candidate configuration data;
- `<edit-config>` : to load all or part of a configuration data to a specified target running or candidate configuration;
- `<copy-config>` : to copy existing configuration data in place of a specified target running or candidate configuration;
- `<delete-config>` : to delete a candidate configuration ;
- `<lock>` : to lock the running configuration against any edit or copy config operations originated from another session or external access (like SNMP);
- `<unlock>` : to unlock a locked configuration;
- `<close-session>` : to stop the NETCONG session accepting any request but complete operation in progress;

- `<kill-session>` : to stop the NETCONF session without completing any operation in progress.

All operations are in carried `<rpc>` elements. A common element of get, edit and delete operations is a filter element (`<filter>`) that allows some filtering on data by using the hierarchical structure of XML documents.

2.4 Capabilities

Accepted operations (basic and news operations) and data are defined by capabilities. A NETCONF agent can provide more than one capabilities and an unique URI references each capabilities. Capabilities are exchanged between entities at session establishment time.

3 YANG

The YANG Internet-Draft[1] defines YANG as a data modeling language used to describe NETCONF configuration and state data. The NETCONF standard does not define such a language for its content layer (cf fig.1). The netmod working group charter[2] explains why a more hight level language than XML is needed (an old draft can be seen at : http://www.yang-central.org/twiki/pub/-Main/YangDocuments/draft-lengyel-why-yang-00.txt).

3.1 YANG specifications

A YANG specification contains formal definitions of data types that will model real data maintained by NETCONF agents. Formal definitions follow the YANG syntax. YANG provides constructs that give semantics to XML data. As an XML document is a collection of imbricated markups, YANG defines statements that can be mapped on pattern of markups. Moreover YANG allows reusability of specifications with generic statements or augmentation/extension statements.

YANG specifications are organized in modules and submodules that contain data type definitions and operation descriptions.

3.2 YANG module and submodule headers

YANG modules and submodules have some headers that are informations related to the module or submodule itself.

3.2.1 Module header

A module has mandatory headers and one optional header. The mandatory ones are the `name space` and `prefix`. For example :

```
1 module router {
2   namespace ''urn:madynes:xml:ns:yang:router'';
3   prefix router;
4 ...
```

[2] http://www.ietf.org/html.charters/netmod-charter.html

The name space at line 2 is for all data defined in the module and the prefix at line 3 can be used inside the module (when confusion is possible) to refer some data. A YANG `version` header is optional.

3.2.2 Submodule header

A submodule has one or two headers. It must have a `belongs to` statement and may have the YANG `version` statement. A submodule belongs to one and only one module. For example :

```
1 submodule routing-policies {
2   belongs-to router ;
3
4 ...
```

The submodule `routing-policies` belongs to the `router` module at line 2.

3.2.3 Yang specification meta statements

Meta statements give some general information on the module or submodule. These informations concern the organization that defines the module, the contact, the description and the reference of the YANG specification. At most four meta statements can be made. A meta statement of a specification must not be duplicated (e.g. two contact meta statement in a module).

3.2.4 Yang linkage statements

A yang specification can have `import` and `include` statements.

Import statement The syntax allows to identify another module and associate it to a prefix. For example :

```
1 module router {
2 ...
3   import yang-types {
4     prefix yang;
5   }
6 ...
```

The module `yang-types` is imported at line 3 so that any type or data defined in this module can be used in the `router` module. In order to use them without conflict, the prefix `yang` defined at line 4 must be used. For example (again in the `router` module):

```
1 ...
2 leaf network {
3   type yang:counter32;
4 }
5 ...
```

where `counter32` is defined in the `yang-types` module. The prefix used must be the same than the one defined in the prefix statement of the imported module (see section 3.2.1).

There can be several import statements but each prefix must be unique in the module. The prefix defined in a module can be used in this module. A submodule can import modules but no submodules.

Include statement The syntax allows to refer to a submodule. For example:

```
1 module router {
2 ...
3 include routing-policies;
4 ...
```

The `router` module includes the `routing-policies` submodule at line 3 so any type or data defined in the submodule can be used in that `router` module.

An included submodule must have a `belongs-to` statement with the reference of the including module (see section 3.2.2). A submodule can include other submodules but they must all belong to the same module.

3.2.5 Yang revision statement

Any yang specification should contain revision statements. There is one YANG_Revision instance for each yang revision statement and each one can contain none or one description statement.

YANG specifications describe1 data as a tree of nodes. There are two main node types; **leaf** nodes that contain data values and **construct** nodes that contain (in the hierarchical meaning) other nodes.

3.3 Leaf nodes

There are two classes of leaf nodes :

- (`leaf`) that contains one value;
- (`leaf-list`) that contains a list of values of the same type.

3.4 Construct nodes

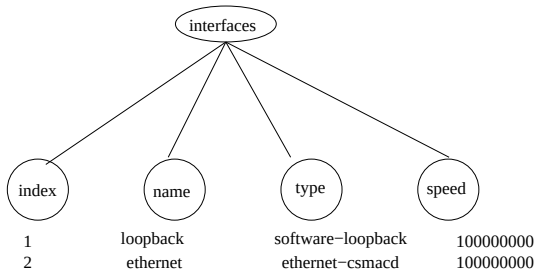
A construct node definition contains other node definitions. Value of such a node depends on the type of the construct node:

- `container` that contains other nodes and its value is composed of values of all contained nodes;
- `list` that contains other nodes and its value is composed of several values of all contained nodes. A list value can be seen as a two dimensional array and a `key` parameter of the `list` allows the reference of one instance of the list of node (an entry);
- `choice` that defines `case` constructs containing other nodes and its value is the value of contained nodes of one of the defined cases;
- `rpc` that contains other nodes and is used in the rpc mechanism of NETCONFand its value is the value of contained nodes.

- `notification` that contains other nodes and is used by NETCONF notifications and its value is the value of contained nodes.

Following is an example of a part of a YANG specification[3] that describes a table of network interfaces, a conceptual view of two entries and the XML document of this configuration:

```
list interfaces {
   key index;
   leaf index {
     type int8;
   }
   leaf name {
     type string;
   }
   leaf type {
     type string;
   }
   leaf speed {
     type int64;
   }
}
```




```
<list>
  <index>
     1
  </index>
  <name>
     loopback
  </name>
  <type>
     software-loopback
  </type>
  <speed>
     100000000
  </speed>
  <index>
     2
  </index>
  <name>
     ethernet
  </name>
  <type>
     ethernet-csmacd
  </type>
  <speed>
     100000000
  </speed>
</list>
```

3.5 Typedef

YANG defines a set of base types (integer, float, string...) and allows the definition of new types from existing ones by a `typedef` construct. For example below is the definition of a 32 bits counter from the basic unsigned integer `uint32`.

```
1 typedef counter32 {
2        type uint32;
3        description
4           "The counter32 type represents...
5        reference
6           "RFC 2578 (STD 58)";
7    }
```

New types can be used in data nodes and in other `typedef`. Depending on the base type used in a `typedef`, some restrictions can be added like a range restriction on numerical values or as a string pattern on string derived types. When defining a new type, restrictions must only restrict the value set of the base type. The new type is a sub-type of the base type.

[3] All example in this report are inspired from the draft[1]

3.6 Grouping and Uses

YANG provides a reusability concept with `grouping` and `uses` statements. A grouping is a set of definitions (leafs and construct nodes, typedef, grouping...) that can be used in other definitions with the `uses` statement. For example below is the definition of the grouping `address` with two leaf nodes at lines 2 and 5 and its usage in the `http-container` container (line 10).

```
1 grouping address {
2     leaf ip {
3       type bits (32);
4     }
5     leaf port {
6       type uint32;
7     }
8 }
```

```
9
10 container http-server {
11   leaf name {
12     type string;
13   }
14   uses address;
15 }
```

This construct is equivalent to :

```
1
2 container http-server {
3   leaf name {
4     type string;
5   }
6   leaf ip {
7     type bits (32);
8   }
9   leaf port {
10    type uint32;
11  }
12 }
```

3.7 Augmenting

The `augment` statement contains nodes and is used to add theses nodes to an existing construct node. In the specification below, a container named `login` at line 2 contains a leaf named `message` line 3 and a list `user` line 6 having several leaf nodes (just `name` is shown). The `augment` statement at line 13 refers to the list `user` under the container `login` and adds to it the leaf `uid` at line 14.

```
1
2   container login {
3     leaf message {
4       type string;
5       }
6     list user {
7       key ``name'';
8       leaf name {
9          type string;
10      }
11      ...
12    }
```

```
13 augment login/user {
14   leaf uid {
15     type uint16;
16   }
17 }
```

Note that augmenting is not the same as grouping. Grouping is used to reduce the size of a specification by using several times the same construct while augmenting allows to add nodes to an existing one. Augmenting is useful when an equimement has vendor-specific parameters added to standard ones.

3.8 Rpc

As a NETCONF agent can provide capabilities with new rpc embeded operations, YANG allows the specification of such an operation. For example the `activate-software` operation below defines data sended in a `<rpc>` message with `input` statement (line 2) and data returned in a `<rpc-reply>` with the `ouput` statement (line 7).

```
1 rpc activate-software-image {
2     input {
3         leaf image-name {
4           type string;
5         }
6     }
7     output {
8         leaf status {
9           type string;
10        }
11    }
12 }
```

3.9 Notification

A NETCONF agent can send notifications that can be specified with YANG by the `notification` statement. Nodes contained in a `specification` statement model data sent by the agent. Below is an example where the index of a failed interface (line 3) will be sent.

```
1 notification link-failure {
2   description "A link failure has been detected";
3   leaf if-index {
4       type int32 { range "1 .. max"; }
5   }
6 }
```

3.10 Extensions

YANG allows the definition of new statements when specific processes requires it. The content of an extention is to be interpreted by specific implementation. Extensions can be used anywhere in YANG specifications. In the example below, the extension `c-define` is specified and used with one name argument (line 6). Each use of an extension must be prefixed by the module prefix where the extension is defined.

```
1 extension c-define {
2          description
```

```
3             "Takes as argument a name string.
4              Makes the code generator use the given name in the
5              #define.";
6           argument "name";
7         }
```

```
1  myext:c-define "MY_INTERFACES";
```

3.11 YIN

YIN is an alternative XML-based syntax for YANG specifications. YIN specifications can be generated from YANG ones and are equivalent. The goal of YIN specifications is to enable seemless interactions with XML based tools (as XSLT). *jYang* parser allows the generation of YIN specifications from YANG.

4 *jYang*

jYang is a java parser for YANG specifications and an application programming interface offering a programmatic access in java to YANG specifications.

4.1 YANG Parser

The java parser is built with JJTree and JavaCC [4] but no external library is needed to use it.

- lexical and syntax checks are conformant to the ABNF grammar given in [1]
- semantical check covers following features :
  - name scoping and accessibility for typedef, grouping, extension, uses, leaf and leaflist, inside a module or submodule and with imported and included specifications.
  - type restriction for any type (integer, boolean, bits, float,...) and typedef
  - default value and restriction
  - augment existing node
  - Xpath for schema node in augment, leaf (of key ref type) and list (for unique statement)

4.2 Repository

jYang is an open source distribution of our toolkit under the GPL licence. The official repository is at the INRIA Gforge web site :

`http://jyang.gforge.inria.fr`

[4] https://javacc.dev.java.net

4.3 *jYang* tools

4.3.1 *jYang* parser use

jYang is distributed as a java jar file called `jyang.jar` and configured to be executable. The synoptic is :

```
java -jar jyang.jar [-h] [-f format] [-o outputfile] [-p paths] file [file]*
```

- `-h` print the synoptic
- `-f format` specifies the format for a translated output (yin format for example)
- `-o outputfile` the name of the translated output (standard output if not given) ignored if no format are given
- `-p paths` a path where to find other YANG specifications. It is needed if import or include statements are in the checked specification or if the environement variable `YANG_PATH` is not set.
- `file [file]*` specifies files containing YANG specification. It must be one specification (`module` or `submodule` for each file).

Errors Errors in YANG specifications are printed on the standard error output. *jYang* stops checking at the first lexical or syntaxical error but tring to check after a first semantical error is encountered. When such an error is detected, the current bloc statement is escaped and *jYang* passes to the next statement.

4.3.2 Programmatic access

jYang provides java classes and interfaces to parse YANG specification inside a java program. Internal representation of those specifications can be accessed throught the API defined in the section 5. Below is an example of how to parse a YANG specification.

```
1  import java.io.*;
2  import jyang.*;
3
4  public class JyangTest {
5
6      /**
7      * Simple jyang test, parses and checks one YANG specification.
8      * Imported or included modules or submodules are looked in the
9      * current directory.
10     * Error messages are on the standard output
11     *
12     * @param args YANG file name
13     */
14     public static void main(String[] args) throws Exception {
15         FileInputStream yangfile = new FileInputStream(args[0]);
16         new yang(yangfile);
```

```
17          YANG_Specification spec = yang.Start();
18          spec.check();
19      }
20  }
```

The program first gets the YANG specification file at line 15. A new jyang parser is created line 16 with this file. The lexical and syntactic check are processed at line 17 and return a YANG_specification object instance that can be semantically checked, as at line 18.

5 *jYang* API

5.1 UML class diagram

Following sections contain the UML class diagrams of the *jYang* API. UML classes (abstacts or not) are java classes and UML interfaces are java interfaces. Inheritance relations are directly mapped to the java inheritance mechanism (we have limited in the design multiple inheritance to interfaces only).

For relationships other than inheritance the API follows theses rules :

- when the cardinality is `0-1` there is a getter and a setter method with the name of the related class in the other related class. For example in figure 3 there is a method called `getArgument` in the `YANG_Extension` java class and this method returns an instance of the `YANG_Argument` java class. Such method returns `null` if there is no related instance (but some relations have no `0` lower bound and so must not return null). There is also a method called `setArgument(YANG_Argument)`.
- when the cardinality is `0-n` the getter returns a java `Vector` instance containing related instances. The getter has an extra 's', for example in the figure 2 there is a method called `getLinkages()` in the `YANG_Specification` java class. If there is no related instance, the method returns an empty java `Vector`. For the setter, as it is often used during parsing, there is a method called `add`*Class-Name* (for example `addLinkage(YANG_Linkage)`).

5.2 YANG specifications

Figure 2 shows the top level classes and interfaces hierarchy. On top is the YANG_Specification interface that can be a YANG_Module for a yang module or a YANG_SubModule for a YANG submodule.

5.3 Yang body statements

Data definitions are in body statements that can be: extension, type definition, grouping, data definition, rpc or notification. The YANG_Body interface is the common interface for all bodies in a yang specification.

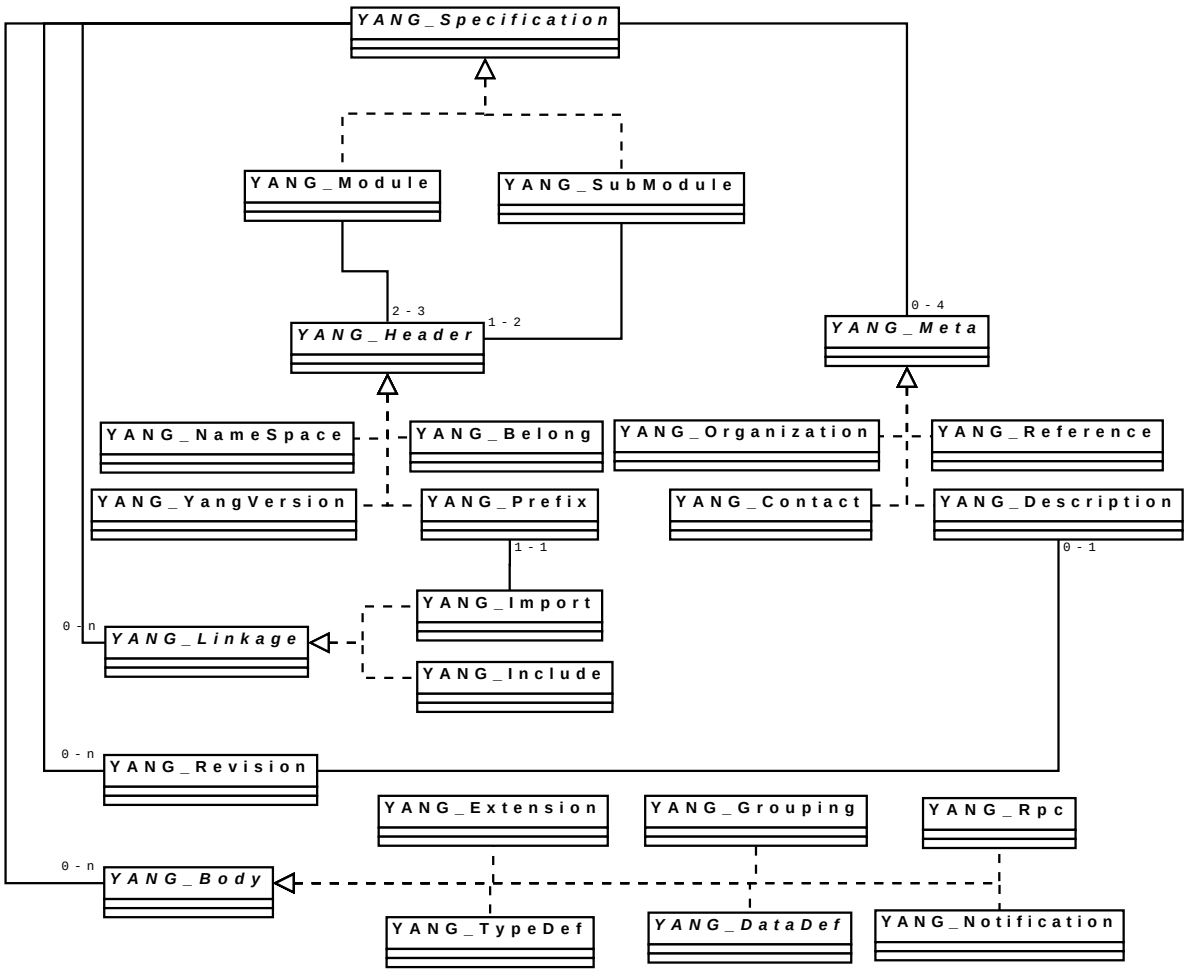


Figure 2: Module and SubModule

5.4 Bodies

5.4.1 Extension statement

An extension statement (fig. 3) can be stand alone or can contain other statements either as argument, status, description and reference. Each of these statements can occur at most once. Their description is detailed in section 5.5.

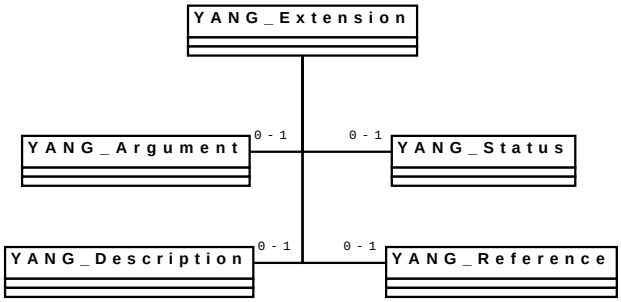


Figure 3: Extension statement classes

5.4.2 TypeDef statement

A typedef statement (fig. 4) must contain a type statement and can contain units, default, status, description and reference statements. Each of these statements can occur at most once. Their description is detailed in section 5.6.

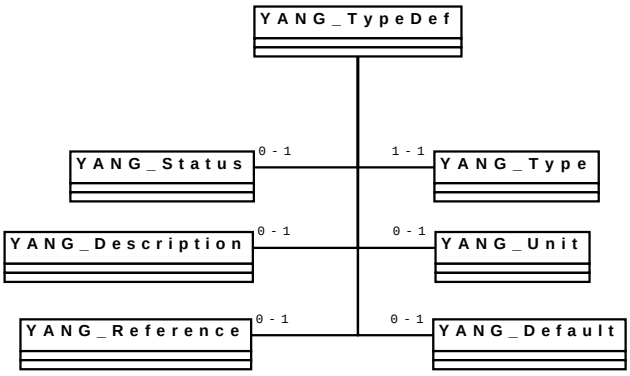


Figure 4: TypeDef statement classes

5.4.3 Grouping statement

A grouping statement (fig. 5) can be single or can contain status, description and reference statements. Each of these statements can occur at most one time. A grouping statement can also contain several other grouping, typedef and datadef statements. Their description is detailed in section 5.7.

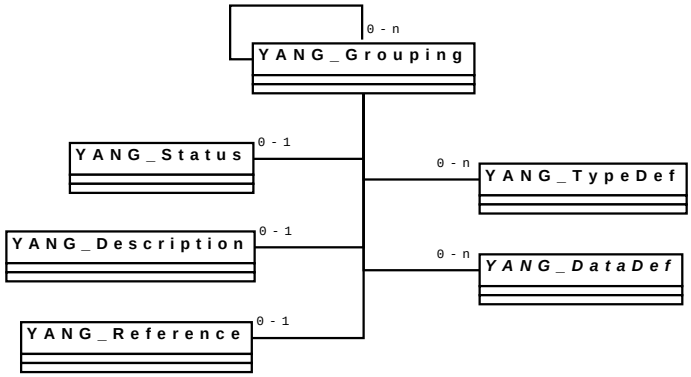


Figure 5: Grouping statement classes

5.4.4 DataDef statement

A datadef statement (fig. 6) is either a leaf, leaflist, list, choice, anyxml, uses or augment statement. Their description is detailed in section 5.8.

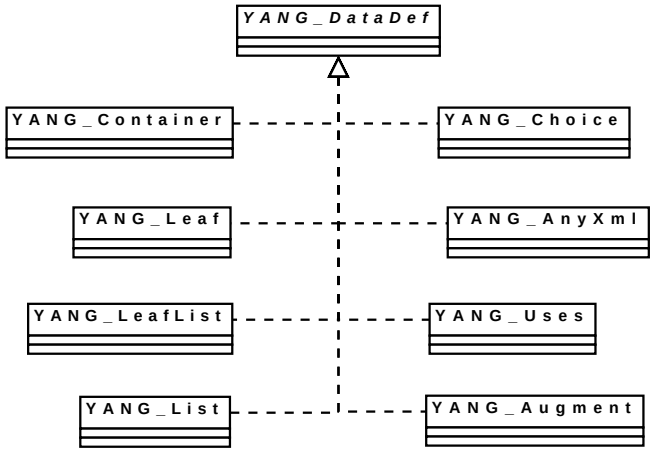


Figure 6: DataDef statement classes

5.4.5 Rpc statement

A rpc statement (fig. 7) can be alone or can contain status, description, reference, input and output statements. Each of these statements can occur at most once. A rpc statement can also contain several other grouping, typedef and datadef statements.

5.4.6 Notification statement

A notification statement (fig. 8) can be alone or can contain status, description and reference statements. Each of these statements can occur at most once.

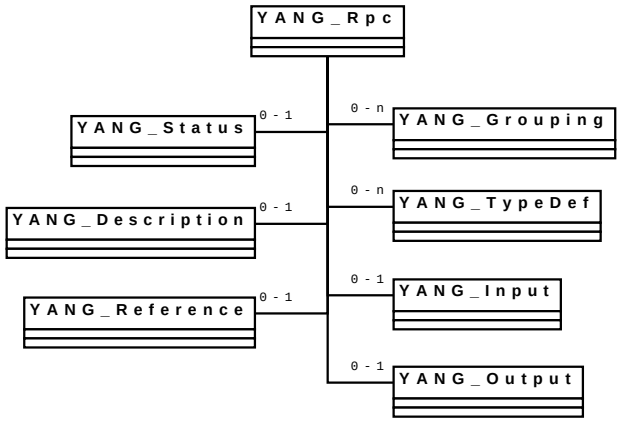


Figure 7: Rpc statement classes

A notification statement can also contain several other grouping, typedef and datadef statements.

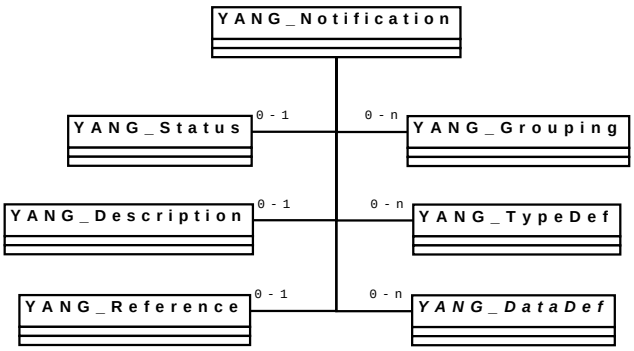


Figure 8: Notification statement classes

5.5 Extension details

This section refers to the section 5.4.1. It details all statements that can occur in an extension statement.

5.5.1 Argument statement

An argument (fig. 9) is composed of at most one yin statement. A yin statement contains either the “true” or the “false” string.

There is no more syntax checking needed by other extension substatements (description, status and reference).

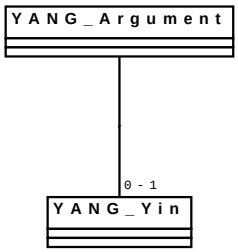


Figure 9: Argument statement classes

5.6 Typedef detail

This section refers to the section 5.4.2. It details all statements that can occur in a typedef statement.

5.6.1 Type statement

A type (fig. 10) is composed of either one or more enum statement or only one of the specification or restriction statement.

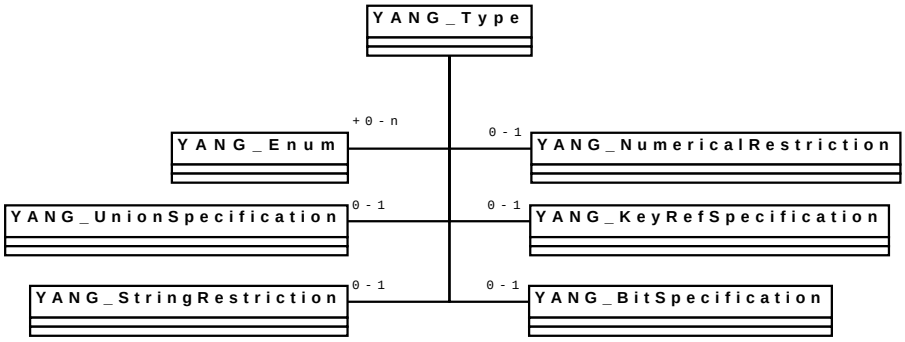


Figure 10: Type statement classes

There is no more syntax checking needed by other typedef substatements (description, status, default and units). Default and units statements are subject to semantical checking.

5.7 Grouping detail

This section refers to the section 5.4.3. It does not detail any statement like status, description and reference. Typedef is detailed in the section 5.6. The `data-def` statements are detailed in the section 5.8.

5.8 Data def details

This section refers to the section 5.4.4. It details those statements that can be a `data-def` statement.

5.8.1 Container statement

A `container` statement (fig. 11) can contain several must, typedef, grouping and data-def statements. Presence, config, status, description and reference statements are optional.

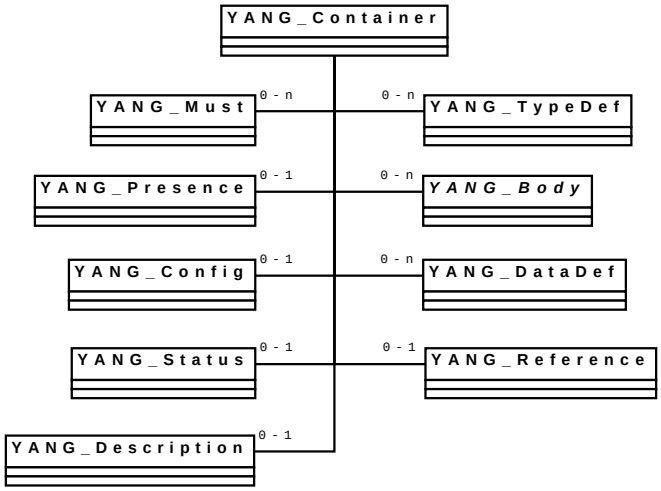


Figure 11: Container statement classes

5.8.2 Leaf statement

A `leaf` statement (fig. 12) must contain one type statement (see section 5.6.1) and several must statements. Units, default, config, mandatory, status, reference and description are optional.

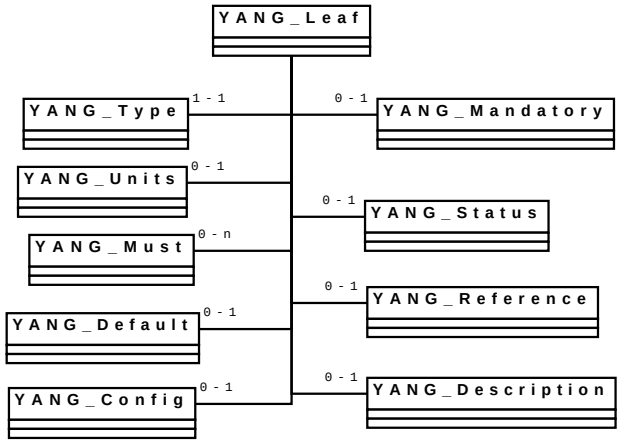


Figure 12: Leaf statement classes

5.8.3 Leaf List statement

A `leaf-list` statement (fig. 13) must contain one type statement (see section 5.6.1), several must statements. Units, default, config, min element, max element, mandatory, status, reference and description are optional.

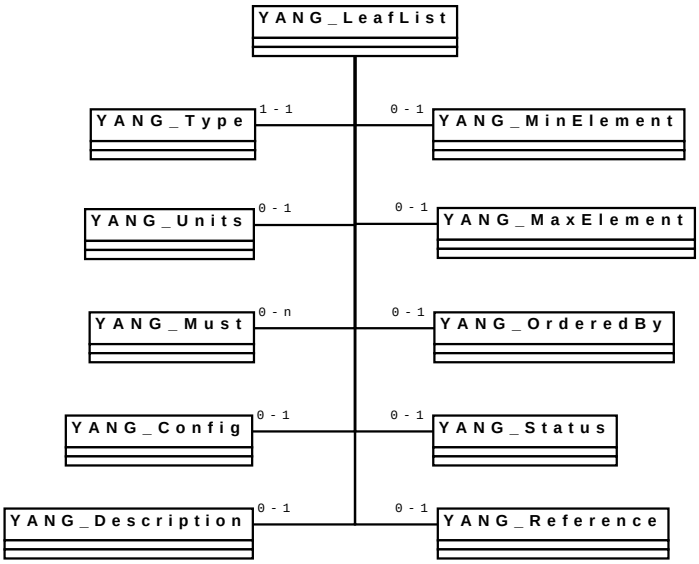


Figure 13: Leaf list statement classes

5.8.4 List statement

A `list` statement (fig. 14) can contain several must, unique, typedef and grouping statements and must contain at least one data-def statement. Key, min element, max element, ordered-by, status, description and reference are optional.

5.8.5 Choice statement

A `choice` statement (fig. 15) can contain several short-case or case statements that are detailed in section 5.9. Default, mandatory, status, description and reference are optional.

5.8.6 Any-xml statement

An `any-xml` statement (fig. 16) can contain a config, mandatory, status, description and reference statements.

5.8.7 Uses statement

An `uses` statement (fig. 17) can contain a status, description, reference and refinement statements. The refinement is detailed in section 5.10

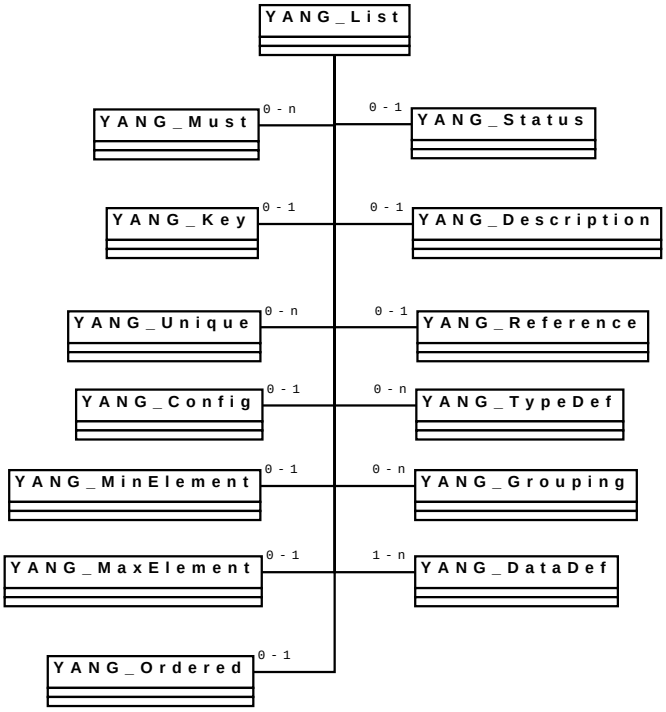


Figure 14: List statement classes

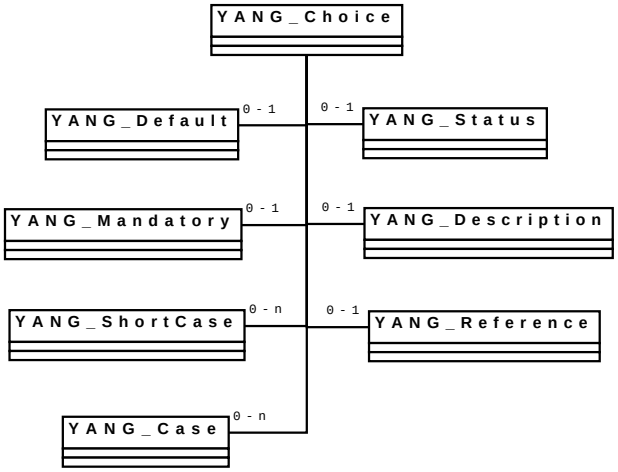


Figure 15: Choice statement classes

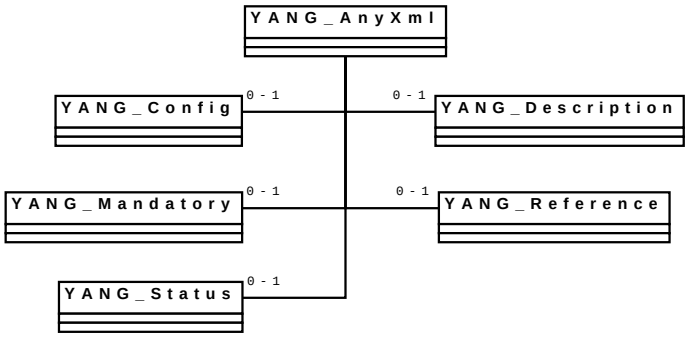


Figure 16: Any-xml statement classes

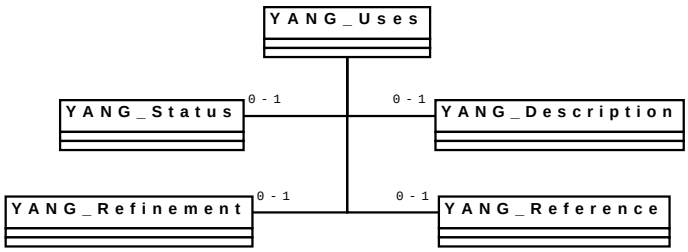


Figure 17: Uses statement classes

5.8.8 Augment statement

An `augment` statement (fig. 18) can contain at least one datadef or case statements or one input or output statements. It depends on the augmented node. When, status, description and reference statements are optional.

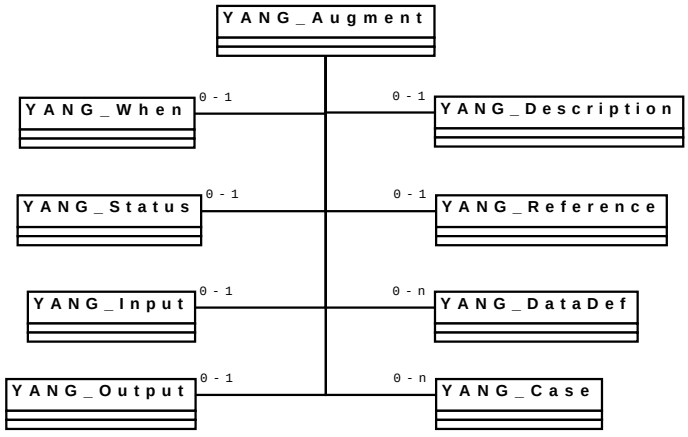


Figure 18: Augment statement classes

5.9 Case and Short Case statements

Case and short case use are described in section 5.8.5.

5.9.1 Case statement

A `case` statement (fig. 19) can contain several case-data-def statements. Status, description and reference are optional. Case-data-def is detailed in section 5.9.3.

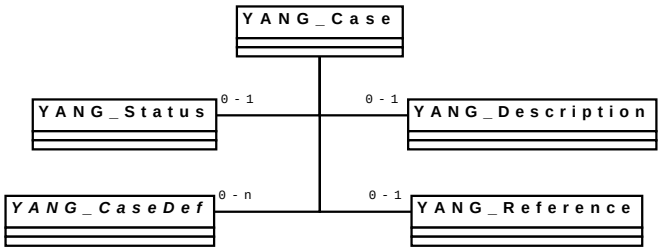


Figure 19: Case statement classes

5.9.2 Short Case statement

A short `case` statement (fig. 20) can be either a container, leaf, leaf-list, list or any-xml statements.

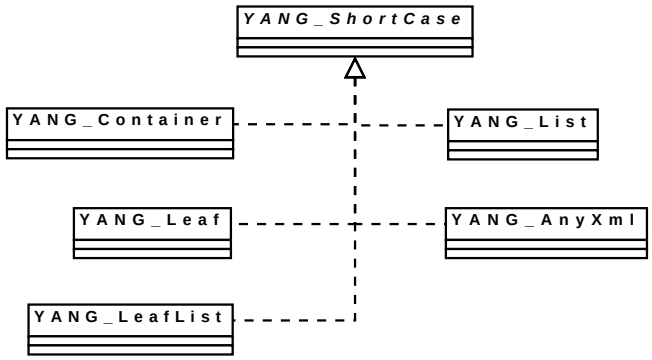


Figure 20: Short Case statement classes

5.9.3 Case Data Def statement

A case data def statement (fig. 21) can be either a container, a leaf, a leaf-list, a list, an any-xml, an uses or an augment statements. Case data def use is described in section 5.9.1.

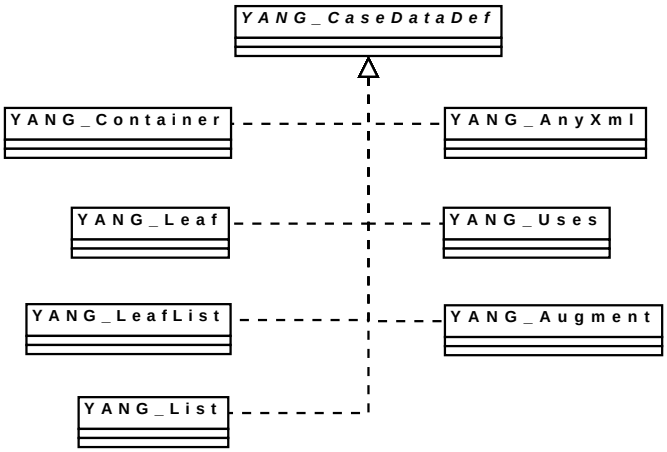


Figure 21: Case Data Def statement classes

5.10 Refinement statement

The refinement statement (fig. 22) can be a refinement of a container, leaf, leaf-list, choice or any-xml statement. Refinement use is described in section 5.8.7.

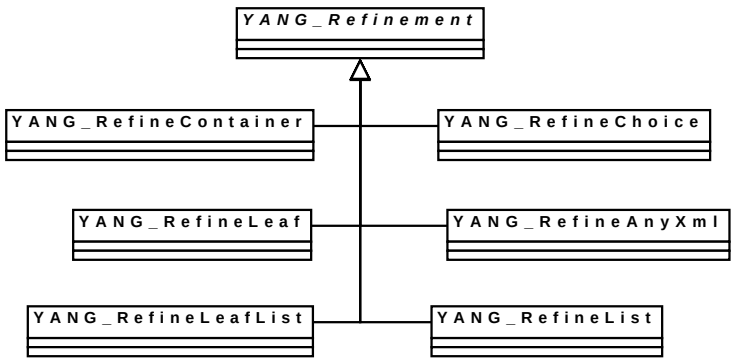


Figure 22: Refinement statement classes

5.10.1 Refined Container statement

A refined `container` statement (fig. 23) can contain several must and refinement statements. Presence, config, description and reference are optional.

5.10.2 Refined Leaf statement

A refined `leaf` statement (fig. 24) can contain several must statements. Default, config, description and reference are optional.

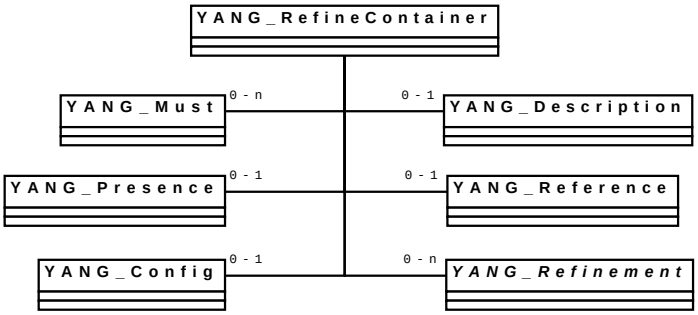


Figure 23: Refine Container statement classes

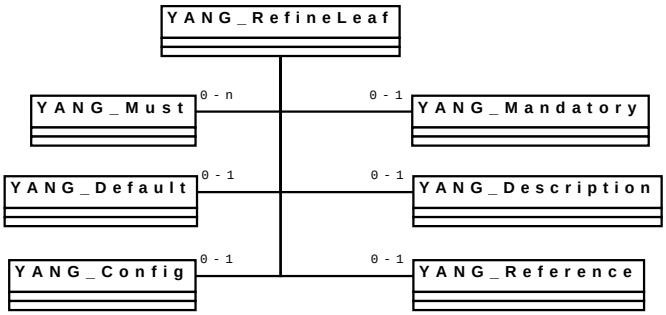


Figure 24: Refine Leaf statement classes

5.10.3 Refined Leaf List statement

A refined `leaf-list` statement (fig. 25) can contain several must statements. Config, min-element, max-element, description and reference are optional.

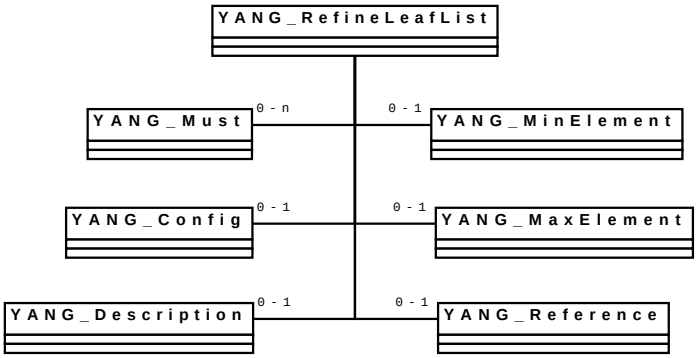


Figure 25: Refine Leaf List statement classes

5.10.4 Refined List statement

A refined `list` statement (fig. 26)) can contain several must and refinement statements. Config, min-element, max-element, description and reference are optional.

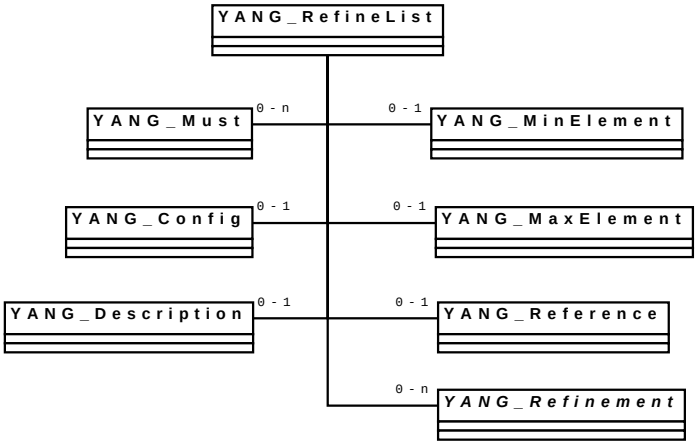


Figure 26: Refine List statement classes

5.10.5 Refined Choice statement

A refined `case` statement (fig. 27) can contain several refine case statements. Default, mandatory, description and reference are optional.

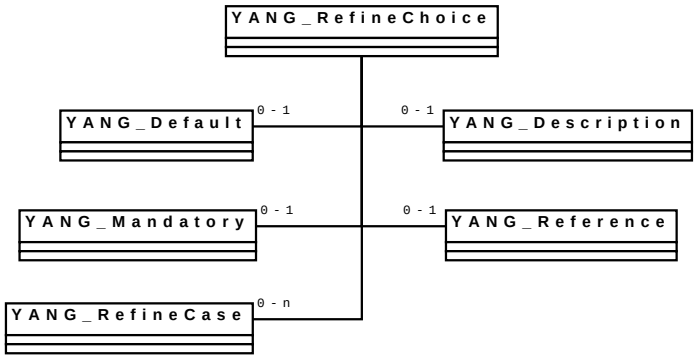


Figure 27: Refine Choice statement classes

5.10.6 Refined Any-xml statement

A refined `any-xml` statement (fig. 28) optionaly contains a config, mandatory, description and reference statements.

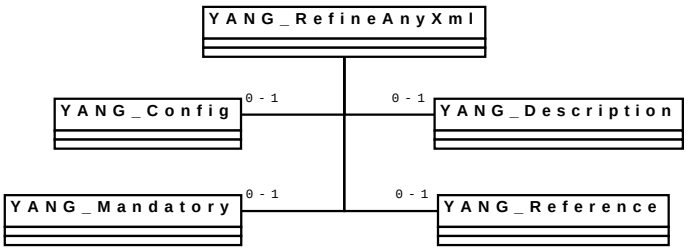


Figure 28: Refine Any-xml statement classes

5.11 Global view

The figure 29 shows all classes and their inheritance relationships.

6 Conclusions and future work

This report describes the *jYang*parser and its API. The work is based on an early release of the draft[1]. futhur revisions will follow the YANG evolution.

jYang allows a static parsing of YANG specifications but there are several other checks that need to be done at the execution time. We plan to define some mechanisms to ensure that a NETCONF agent realizes such checks. The list below may be not exhaustive but draws our main goals :

- YANG specifications can use an `object-instance` data type that refers to an existing element in a configuration. A NETCONF agent must verify that the refered element effectively exists, or has a default value.

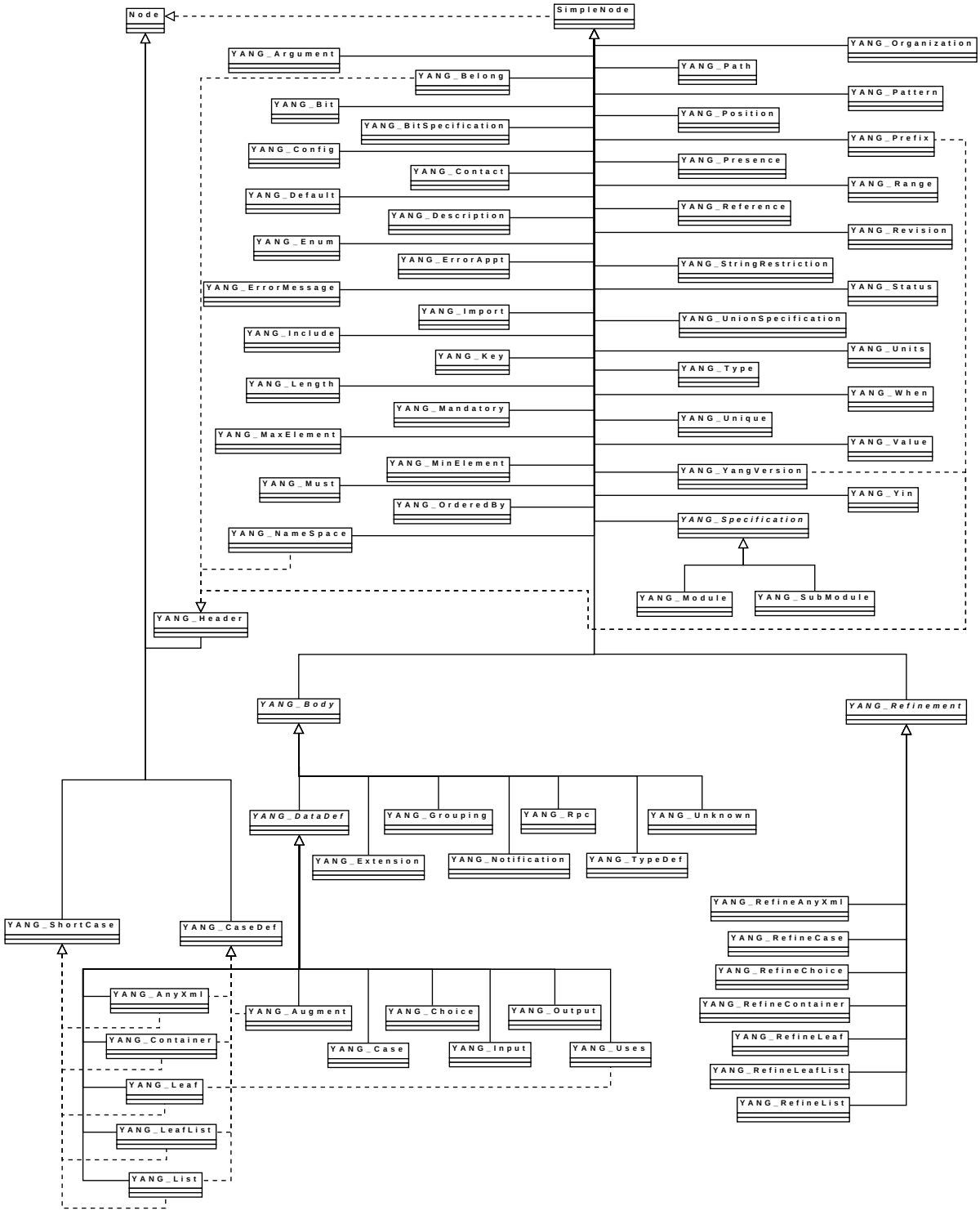


Figure 29: YANG Classes and Interfaces

- YANG specifications can define new operations and notifications. A NETCONF agent must provide them on top of the RPC mechanism.

These evolutions will be bound to a particular NETCONF implementation.

References

[1] M. Bjorklund. YANG - A data modeling language for NETCONF, August 2008. http://www.ietf.org/internet-drafts/draft-ietf-netmod-yang-01.txt.

[2] Tim Bray, Eve Maler, C. M. Sperberg-McQueen, and Jean Paoli. Extensible markup language (XML) 1.0 (second edition). first edition of a recommendation, W3C, October 2000. http://www.w3.org/TR/2000/REC-xml-20001006.

[3] J.D. Case, M. Fedor, M.L. Schoffstall, and J. Davin. Simple Network Management Protocol (SNMP). RFC 1157 (Historic), May 1990.

[4] D. Durham, J. Boyle, R. Cohen, S. Herzog, R. Rajan, and A. Sastry. The COPS (Common Open Policy Service) Protocol. RFC 2748 (Proposed Standard), January 2000. Updated by RFC 4261.

[5] R. Enns. NETCONF Configuration Protocol. RFC 4741 (Proposed Standard), December 2006.

[6] K. McCloghrie, M. Fine, J. Seligson, K. Chan, S. Hahn, R. Sahita, A. Smith, and F. Reichmeyer. Structure of Policy Provisioning Information (SPPI). RFC 3159 (Proposed Standard), August 2001.

[7] M.T. Rose and K. McCloghrie. Structure and identification of management information for TCP/IP-based internets. RFC 1155 (Standard), May 1990.

[8] F. Strauss and J. Schoenwaelder. SMIng - Next Generation Structure of Management Information. RFC 3780 (Experimental), May 2004.

Centre de recherche INRIA Nancy – Grand Est
LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-0803