jYang : yang java parser

Emmanuel Nataf — Olivier Festor

N° ????

Avril 2008

Thème COM

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

jYang : yang java parser

Emmanuel Nataf[*][†], Olivier Festor[‡]

Thème COM — Systèmes communicants
Équipe-Projet Madynes

Rapport technique n° ???? — Avril 2008 — 27 pages

Abstract: The NETCONF configuration protocol of the IETF Network Working Group provides mechanisms to maniupulate the configuration of network devices. YANG is the language on the standard track for specify such configuration. This report describes the use of a yang syntactic and semantic parser.

Key-words: parser, java, yang, netconf

[*] Maître de conférence - Université Nancy2
[†] Madynes INRIA project
[‡] Directeur de Recherche INRIA

jYang : un analyseur yang en java

Résumé : Dans le contexte du groupe de travail réseaux de l'IETF, le protocole de configuration NETCONF permet de manipuler la configuration de matériel réseau. YANG est le langage en cours de standardisation permettant de spécifier des configurations. Ce rapport décrit un analyseur syntaxique et sémantique de spécifications yang

Mots-clés : parser, java, yang, netconf

1 Introduction

It is frequent in the network management world that a protocol and a data model are separatly but conjointly designed, as SNMP[3] protocol and its SMI[7],CO-PS[4] and SPPI[6], or SMInG[8] (GDMO and CMIS or WBEM and CIM outside the IETF scope).

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

This report gives first a short description of NETCONF where some parts are referenced by YANG. The part 3 details language concepts and the last part shows the implementation of *jYang*.

2 NETCONF protocol

The 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 focuse on configuration manipulation as getting the current configuration, update, create or delete it or some part of it. Configuration are modeled by an XML document that contains two sort of data :

- configuration data that could be writable and that described configuration parameter of the NETCONF agent.
- state data that are read-only and that described operational data such as counter or statistics.

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

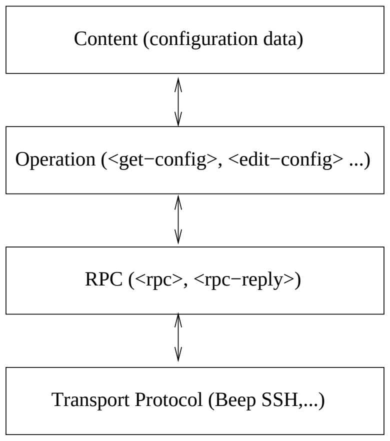

Content (configuration data)
Operation (<get−config>, <edit−config> ...)
RPC (<rpc>, <rpc−reply>)
Transport Protocol (Beep SSH,...)

Figure 1: NETCONF protocol layers

2.1 Transport protocol

The specification language described in this report has no relationship with the transport protocol used with NETCONF.

2.2 RPC

The Remote Procedure Call is described by two XML[2] elements : `<rpc>` for request and `<rpc-reply>` for response. This latter could 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 data (configuration and state);
- `<get-config>` : to retrieve all or part of configuration data;
- `<edit-config>` : to update configuration data;
- `<copy-config>` : to copy existing configuration data in place of current configuration;
- `<delete-config>` : to delete a configuration (but not the current one);
- `<lock>` : to lock any configuration operations;
- `<unlock>` : to unlock a locked configuration;
- `<close-session>` : to stop the NETCONG agent accepting any request but achieves operation in progress;
- `<kill-session>` : to stop the NETCONF agent without achieving any operations in progress.

All these operations are in `<rpc>` elements. A common element of getting, edit or 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 (basics and news operations) and data are defined by capabilities. A NETCONF agent could provides more than one capabilities and an unique URI reference each capabilities.

3 YANG

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

3.1 YANG specifications

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

3.2 YANG module and submodule headers

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

3.2.1 Module header

A module could have two or three headers. It must have a `name space` and a `prefix`. For example :

```
module router {
  namespace ‘‘urn:madynes:xml:ns:yang:router’’;
  prefix router;
...
```

the name space is for all data defined in the module and the prefix could be used inside the module (when confusion is possible) to refer somes data. A YANG `version` header is optional.

3.2.2 Submodule header

A submodule could have 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 :

```
submodule routing-policies {
  belongs-to router;
...
```

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. There could be at most four meta statement. A meta statement of a specification must not be dupplicated (i.e. two contact meta statement in a module).

3.2.4 Yang linkage statements

A yang specification could have import and include statements.

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

Import statement The syntax allows to refer module and associated prefix. For example :

```
module router {
...
import yang-types { prefix yang;}
...
```

The module `yang-types` is imported so 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` must be used. For example (again in the `router` module):

```
...
leaf network {
  type yang:counter32;
}
...
```

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

There could be several import statements but each prefix must be unique in the module. The prefix defined in a module could be used in this module. A submodule could import modules but no submodule could be imported (just module could be).

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

```
module router {
...
include routing-policies;
...
```

The module `router` include the submodule `routing-policies` so any type or data defined in the submodule could be used in the module `router`.

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

3.2.5 Yang revision statement

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

YANG specification describes data as a tree of nodes.

There are two main node types; leaf nodes that contains data values and construct nodes that contains (in the hierarchical meaning) other nodes.

3.3 Leaf nodes

There is two classes of leaf nodes :

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

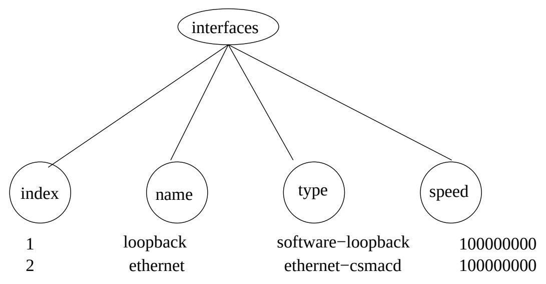
3.4 Construct nodes

Three construct nodes are defined :

- `container` that contains other nodes
- `list` that contains a list of other nodes and where there could be several instances of the list of nodes. A list could be seen as a two dimensions array and a `key` parameter of the `list` allows the reference of one instance of the list of node (an entry);
- `choice` that contains `case` constructs containing other nodes;
- `rpc` that contains other nodes and is used in rpc mechanism of NETCONF.
- `notification` that contains other nodes and is used by NETCONF notifications.

Following is an example of a part of a YANG specification[2] 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>
```

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

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`.

```
 typedef counter32 {
        type uint32;
        description
           "The counter32 type represents...
        reference
           "RFC 2578 (STD 58)";
    }
```

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

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 and its using in the `http-container` container.

```
 grouping address {
     leaf ip {
       type bits (32);
     }
     leaf port {
       type uint32;
     }
 }
```

```
container http-server {
  leaf name {
    type string;
  }
  uses address;
}
```

This construct is equivalent to :

```
container http-server {
  leaf name {
    type string;
  }
  leaf ip {
    type bits (32);
  }
  leaf port {
    type uint32;
  }
}
```

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 `login` contains a leaf `message` and a list `user` having several leaf nodes (just `name` is showed). The `augment` statement refers to the list `user` and add to it the leaf `uid`.

```
container login {
  leaf message {
    type string;
    }
  list user {
    key ‘‘name’’;
    leaf name {
       type string;
    }
    ...
  }
```

```
augment login/user {
  leaf uid {
    type uint16;
  }
}
```

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 operation. For example the `activate-software` operation below defines data sended in a `<rpc>` message with `input` statement and data returned in a `<rpc-reply>` with the `ouput` statement.

```
rpc activate-software-image {
    input {
       leaf image-name {
         type string;
       }
    }
    output {
       leaf status {
         type string;
       }
    }
}
```

3.9 Notification

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

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

3.10 Extensions

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

```
extension c-define {
         description
           "Takes as argument a name string.
           Makes the code generator use the given name in the
           #define.";
         argument "name";
       }

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

4 *jYang*

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

4.1 YANG Parser

The java parser is build with JJTree and JavaCC [3] but no external library are needed to use it.

- lexical and syntax check are comformant 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)

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

4.2 *jYang* tool

4.2.1 *jYang* 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.

4.2.2 Errors

Errors in YANG specification are printed on the standard error output. *jYang* stop checking at the first lexical or syntaxical error but try to check after a first semantical error is encountered. When such error is detected the current bloc statement is escaped and *jYang* pass to the next statement.

5 *jYang* API

5.1 UML class diagram

Following sections contain 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 java inheritance mechanism (we have limited 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 return 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 return 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 return 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

The figure 2 shows the top level of java classes and interface hierarchy. On top is the YANG_Specification interface that could be a YANG_Module for a yang module or a YANG_SubModule for yang submodule.

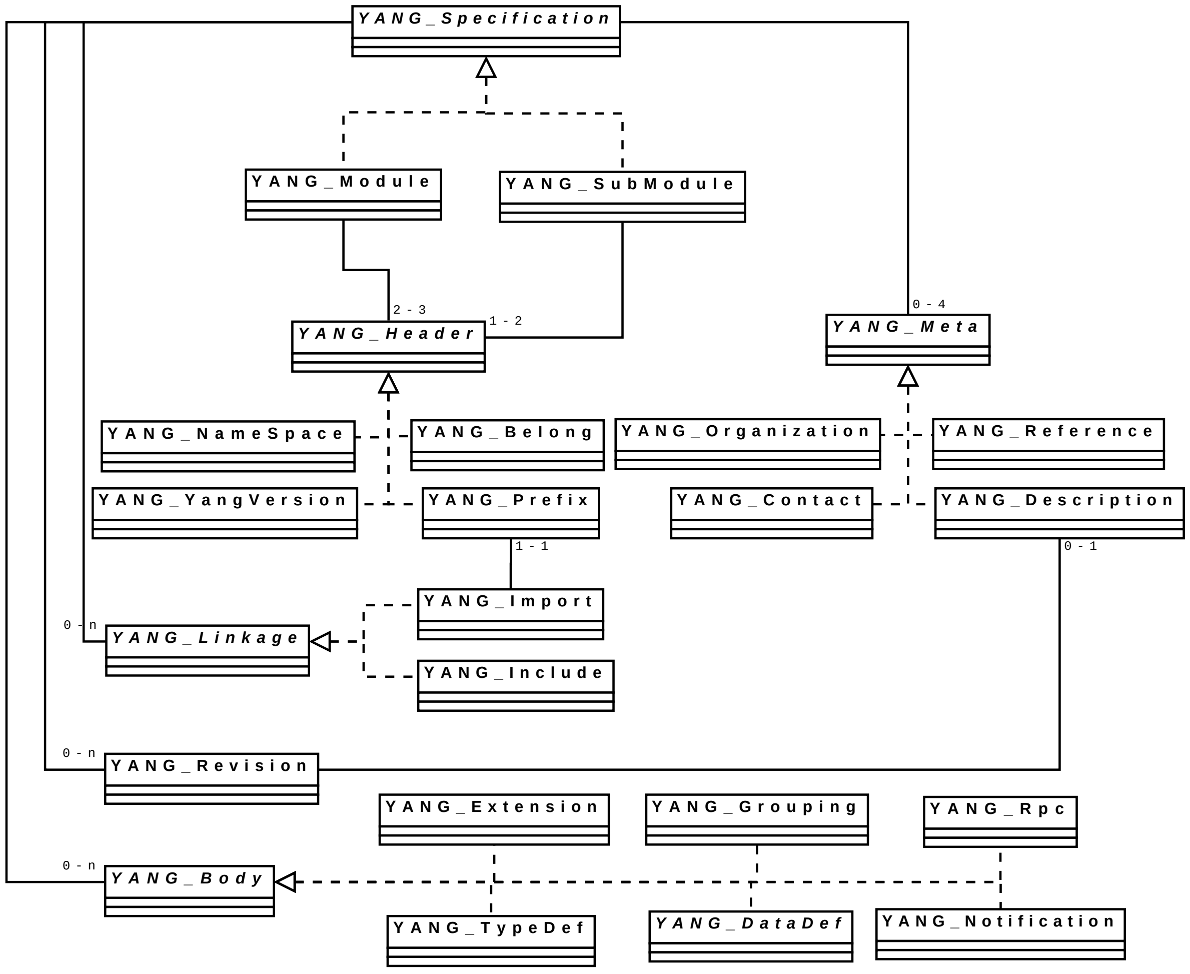


Figure 2: Module and SubModule

5.3 Yang bodies statements

Data definition are in bodies statements that could 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.

5.4 Bodies

5.4.1 Extension statement

An extension statement (fig. 3) could be stand alone or could contains other statement as argument, status, description and reference. Each of these statement could occurs at most one time. Their description are detailed in section 5.5.

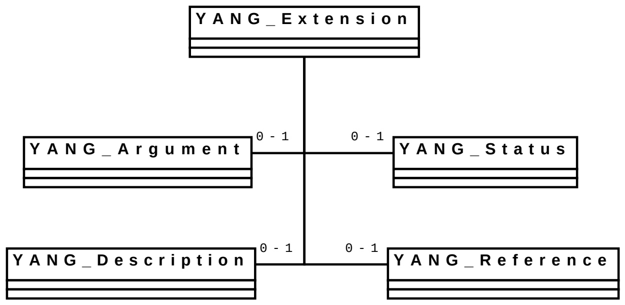


Figure 3: Extension statement classes

5.4.2 TypeDef statement

A typedef statement (fig. 4) must contains a type statement and could contains units, default, status, description and reference statement. Each of these statement could occurs at most one time.Their description are detailed in section 5.6.

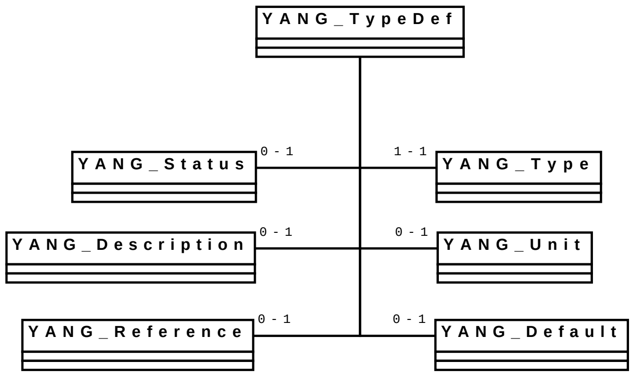


Figure 4: TypeDef statement classes

5.4.3 Grouping statement

A grouping statement (fig. 5) could be stand alone or could contains status, description and reference statement. Each of these statement could occurs at most one time. A grouping statement could also contains several other grouping, typedef and datadef statements. Their description are detailed in section 5.7.

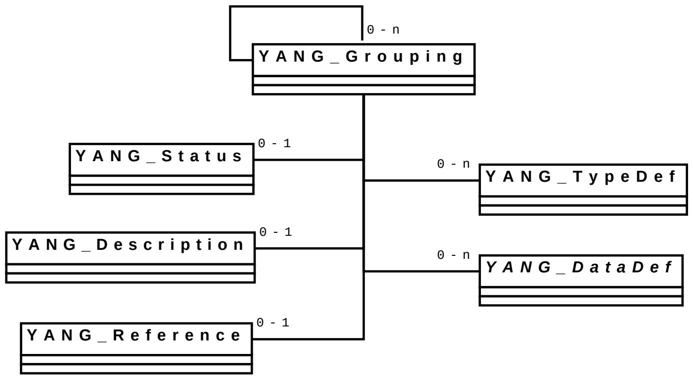


Figure 5: Grouping statement classes

5.4.4 DataDef statement

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

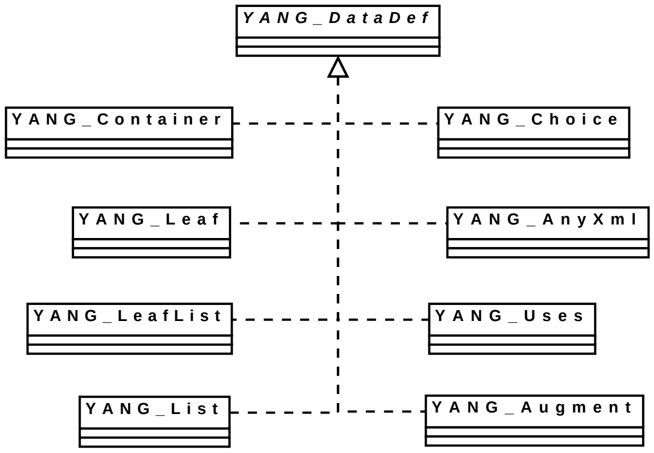


Figure 6: DataDef statement classes

5.4.5 Rpc statement

A rpc statement (fig. 7) could be stand alone or could contains status, description, reference, input and output statement. Each of these statement could occurs at most one time. A rpc statement could also contains several other grouping, typedef and datadef statements.

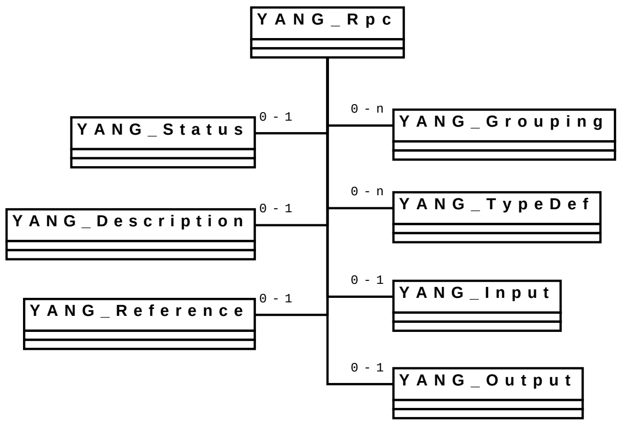


Figure 7: Rpc statement classes

5.4.6 Notification statement

A notification statement (fig. 8) could be stand alone or could contains status, description and reference statement. Each of these statement could occurs at most one time. A notification statement could also contains several other grouping, typedef and datadef statements.

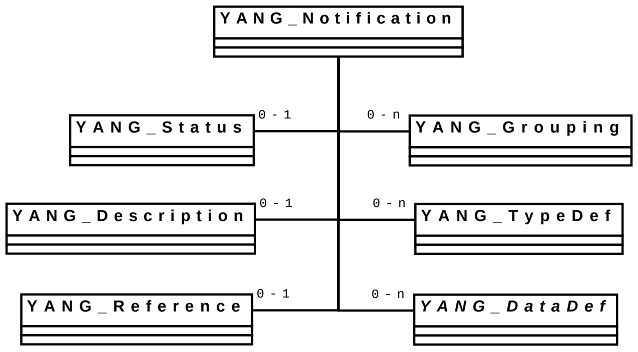


Figure 8: Notification statement classes

5.5 Extension detail

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

5.5.1 Argument statement

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

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

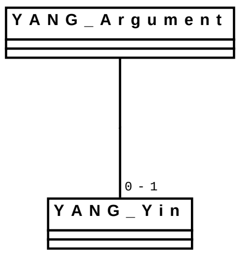


Figure 9: Argument statement classes

5.6 Typedef detail

This section refers to the section 5.4.2. It details all statement that could occurs 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.

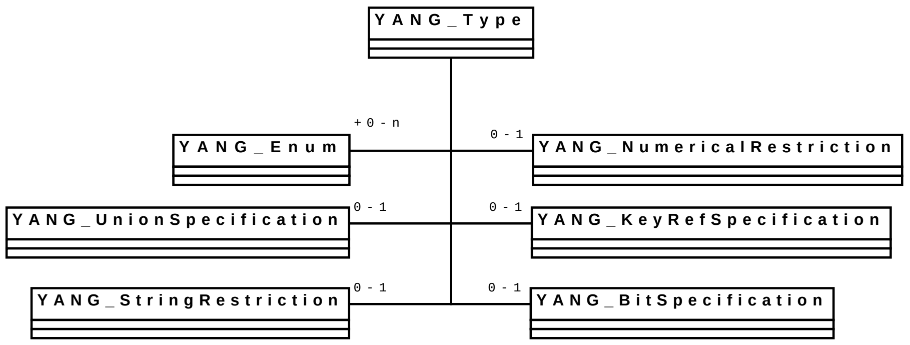


Figure 10: Type statement classes

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 details any statement as status, description and reference does not need more syntax checking and typedef is detailed in the section 5.6. The data-def statement are detailed on the section 5.8.

5.8 Data def detail

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

5.8.1 Container statement

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

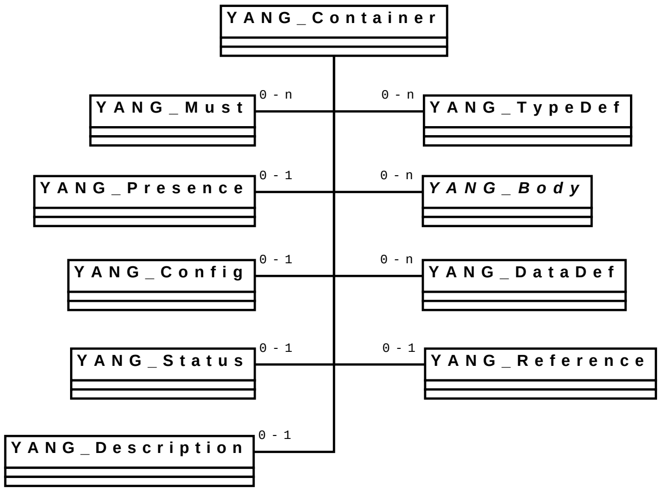


Figure 11: Container statement classes

5.8.2 Leaf statement

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

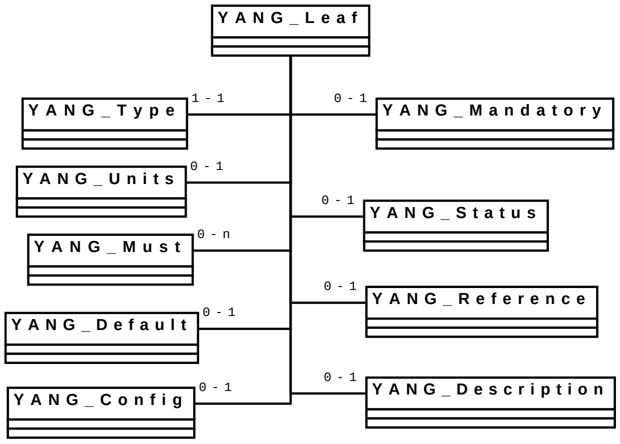


Figure 12: Leaf statement classes

5.8.3 Leaf List statement

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

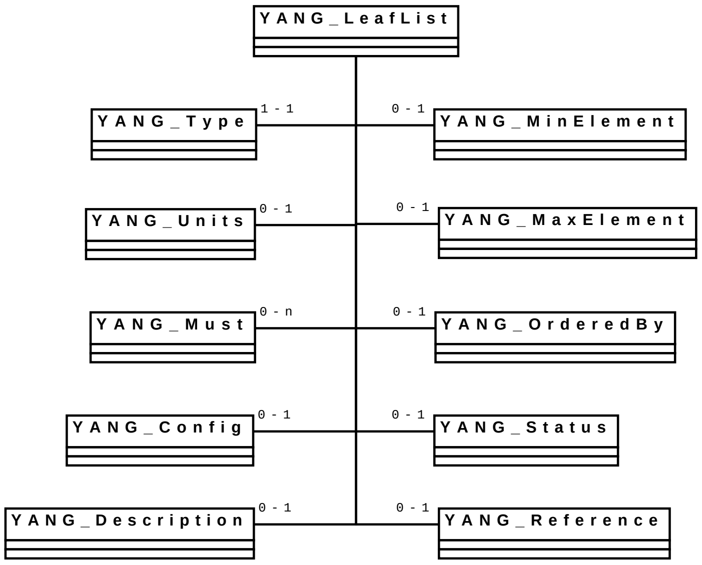


Figure 13: Leaf list statement classes

5.8.4 List statement

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

5.8.5 Choice statement

A choice (fig. 15) could contains several short-case or case statements that are detailed in section 5.9. Default, mandatory, status, description and reference are optionals.

5.8.6 Any-xml statement

An any-xml (fig. 16) could contains a config, mandatory, status, descrition and reference statement.

5.8.7 Uses statement

An uses (fig. 17) could contains a status, description, reference and refinement statement. The refinement is detailed in section 5.10

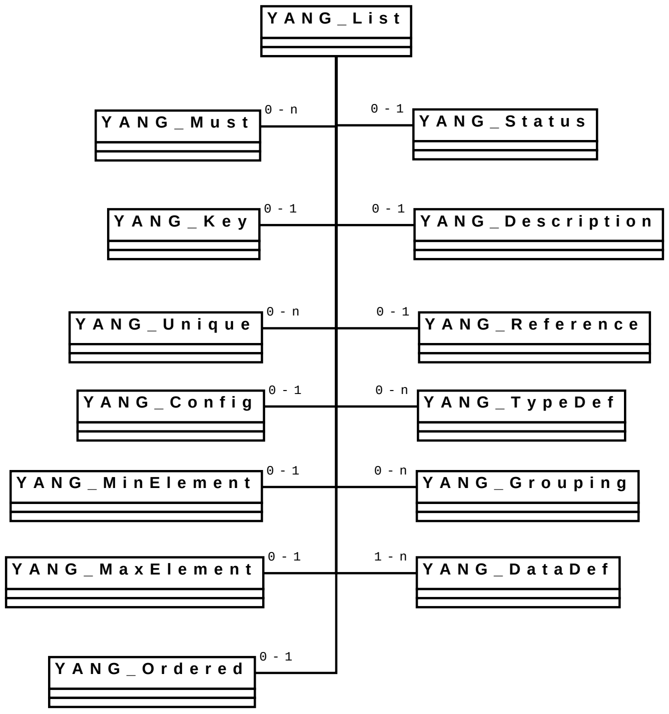


Figure 14: List statement classes

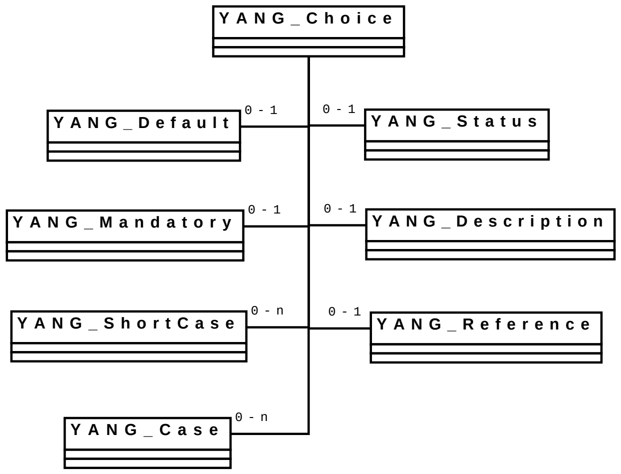


Figure 15: Choice statement classes

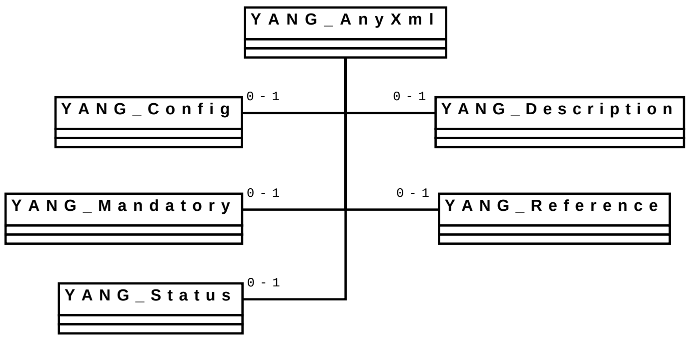


Figure 16: Any-xml statement classes

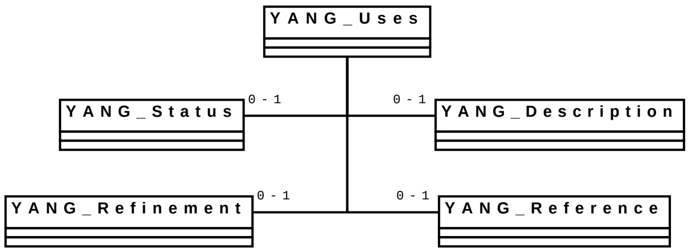


Figure 17: Uses statement classes

5.8.8 Augment statement

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

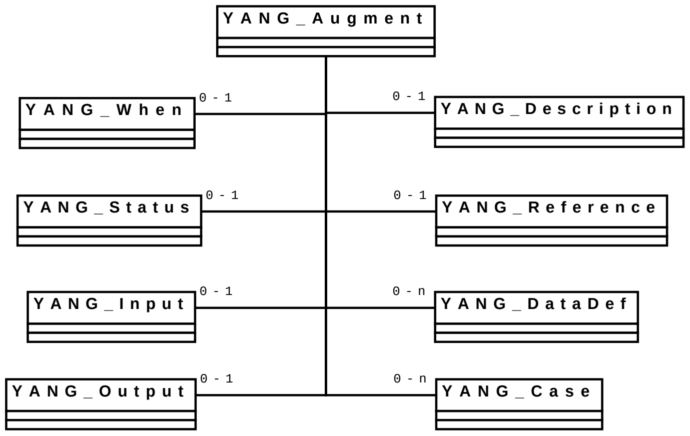


Figure 18: Augment statement classes

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 (fig. 19) could contains several case-data-def statements. Status, description and reference are optionals. Case-data-def is detailed in section 5.9.3.

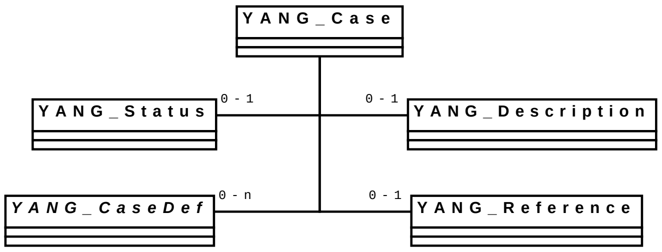


Figure 19: Case statement classes

5.9.2 Short Case statement

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

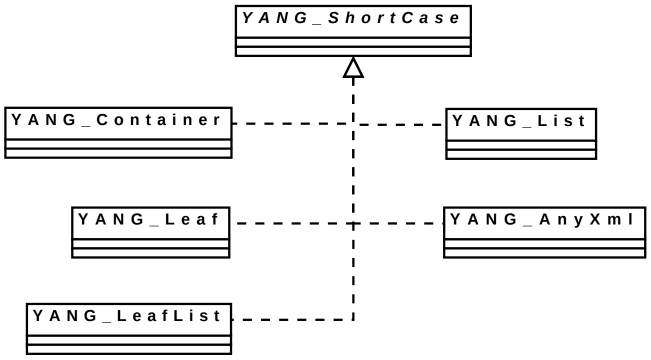


Figure 20: Short Case statement classes

5.9.3 Case Data Def statement

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

5.10 Refinement statement

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

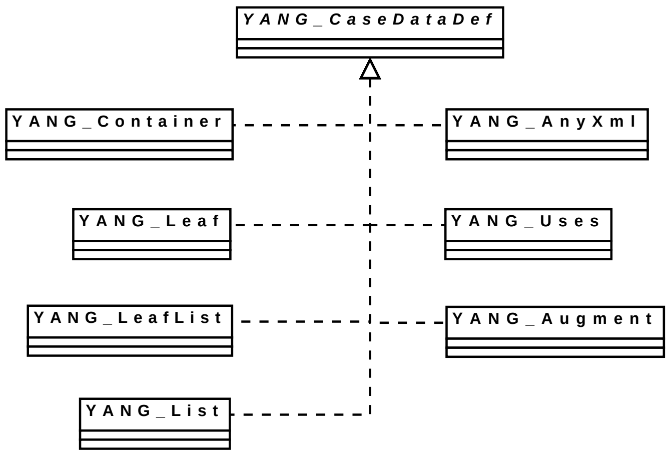


Figure 21: Case Data Def statement classes

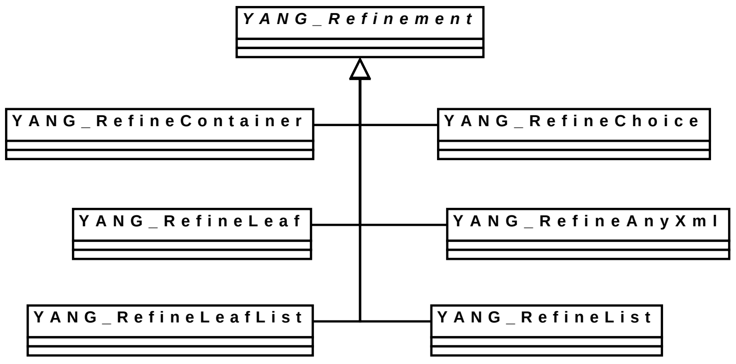


Figure 22: Refinement statement classes

5.10.1 Refine Container statement

A refine container (fig. 23) could contains several must and refinement statements. Presence, config, description and reference are optionals.

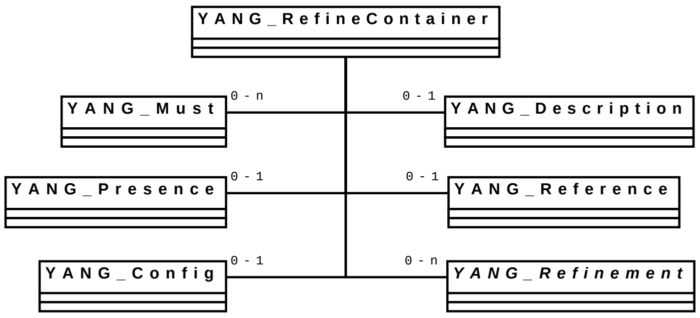


Figure 23: Refine Container statement classes

5.10.2 Refine Leaf statement

A refine leaf (fig. 24) could contains several must statements. Default, config, description and reference are optionals.

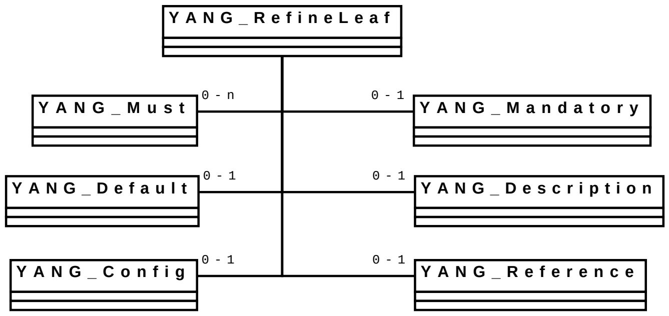


Figure 24: Refine Leaf statement classes

5.10.3 Refine Leaf List statement

A refine leaf list (fig. 25) could contains several must statements. Config, min-element, max-element, description and reference are optionals.

5.10.4 Refine List statement

A refine list (fig. 26)) could contains several must and refinement statements. Config, min-element, max-element, description and reference are optionals.

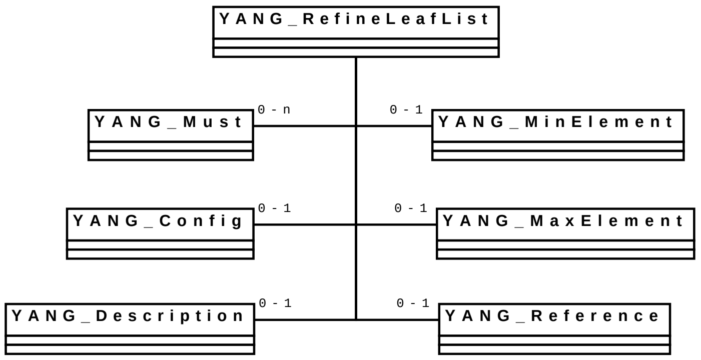


Figure 25: Refine Leaf List statement classes

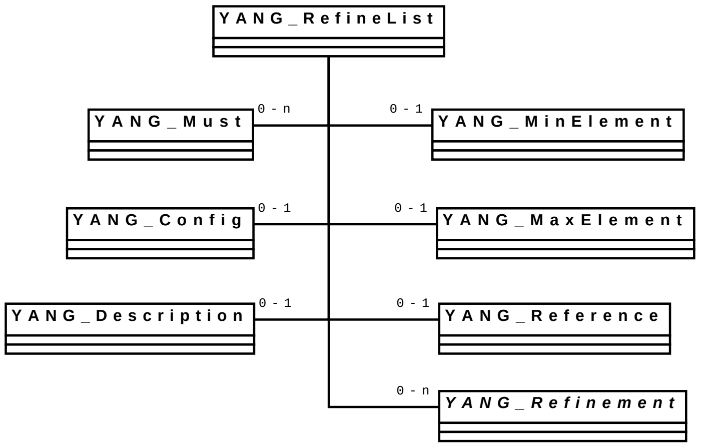


Figure 26: Refine List statement classes

5.10.5 Refine Choice statement

A refine case (fig. 27) could contains several refine case statements. Default, mandatory, description and reference are optionals.

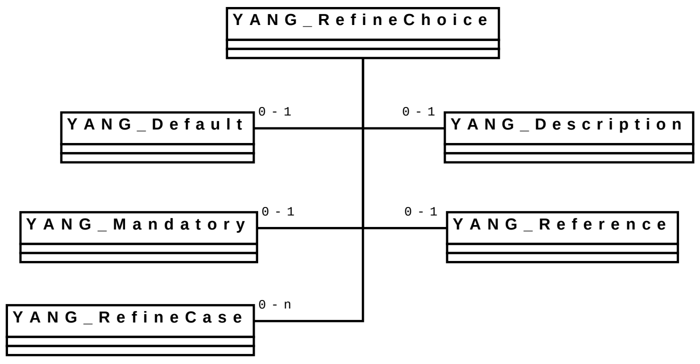


Figure 27: Refine Choice statement classes

5.10.6 Refine Any-xml statement

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

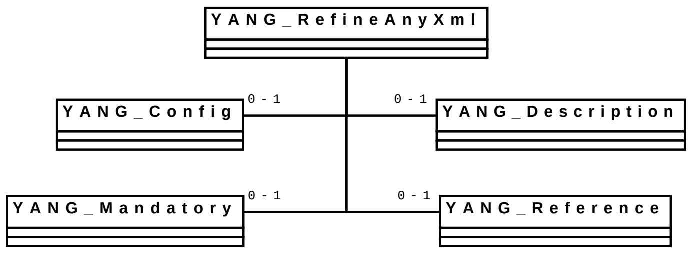


Figure 28: Refine Any-xml statement classes

5.11 Global view

The figure 29 shows all classes and their inheritance relationships.

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.

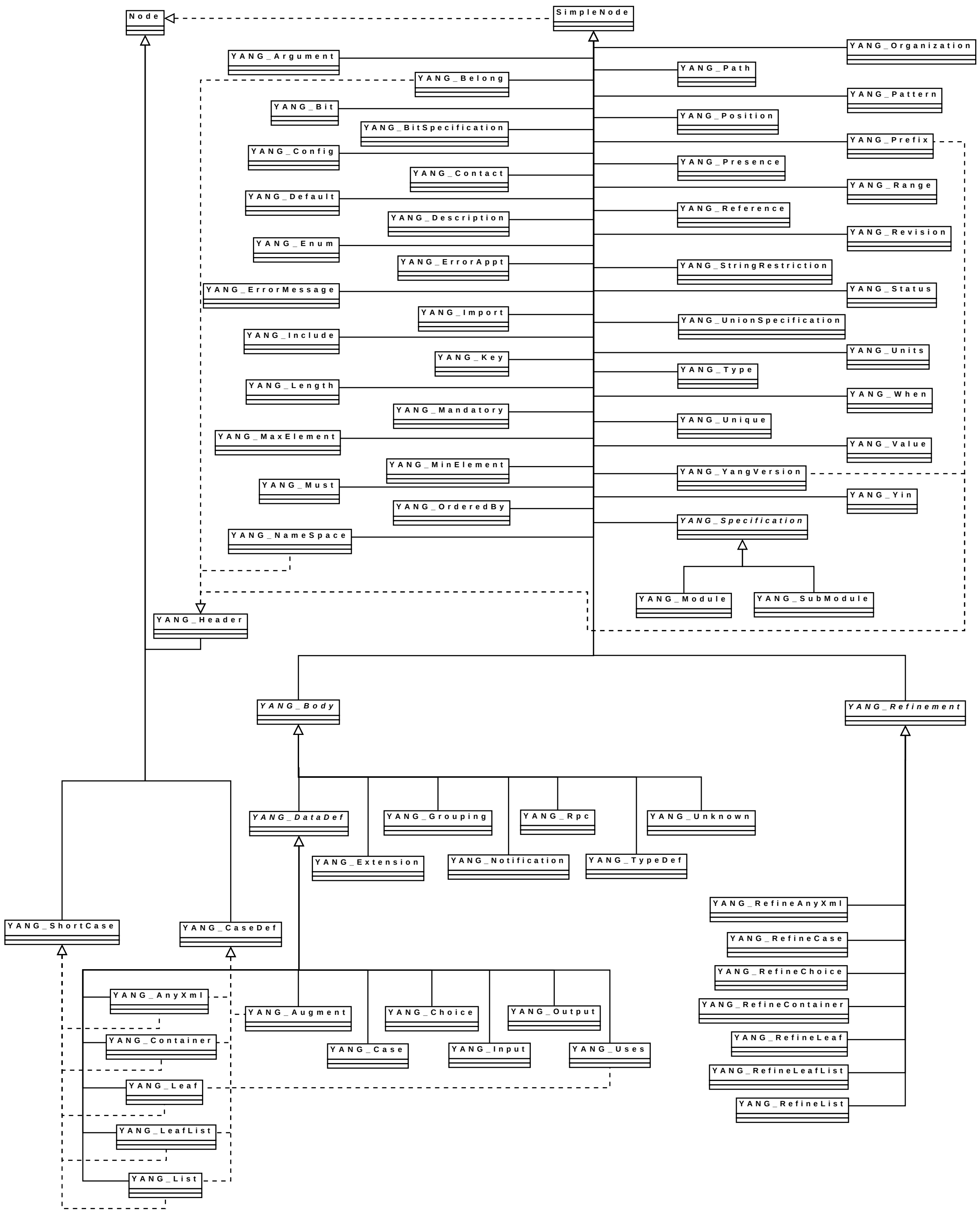


Figure 29: YANG Classes and Interfaces

[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.

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