CLICON data model

Olof Hagsand and Benny Holmgren

July, 2014

Table of contents

1 Introduction

Data modeling languages are used in software engineering to model data types in order to generate code, database entries and to enable transfer of information in communication protocols. There are several data modeling languages including UML, XML Schema and YANG [4].

CLICON uses a limted YANG as data model.

The features of the CLICON data model can be summarized as follows:

- It defines keys and variables of embedded databases in the CLICON system.
- It is mapped to CLI configuration statements using CLIgen.
- In CLICON backend plugins, semantics is added to the data model.
- The data is automatically mapped to XML, CLI and NETCONF [3] statements, making it easy to import and export.

2 Generating XML

A CLICON data model specifies valid XML statements. CLICON routinely maps from its internal (key-based) representation to XML for saving and loading files, or transfering data via Netconf, for example.

This is illustrated below by example for the four basic language constructions.

2.1 Leaf

```
leaf a{
   type int;   --> <a>42</a>
}
```

2.2 Leaf-list

```
leaf-list a{       <a>12</a>
   type int;   --> <a>42</a>
}                  <a>33</a>
```

Note, each leaf-list must be within a single container statement.

2.3 Container

```
container a{      <a>
   leaf x;         <x>42</x>
   leaf y;   -->   <y>22</y>
}                 </a>
```

Note, type statement of leaf has been omitted.

2.4 List

```
list a{          <a>
   key x;          <b>he</b>
   leaf x; -->     <x>12</x>
   leaf b;        </a>
}                 <a>
                    <x>13</x>
                    <b/>
                  </a>
```

3 Generating CLI configuration statements

The YANG data model can be used to map to CLI configuration statements including help-texts, such as set/delete/modify operations. The mapping is straightforward.

3.1 Generating syntax

CLICON generateas a CLIgen tree of commands if the `CLICON_CLI_GENMODEL` option is set. The generated syntax can be used by including a sub-tree syntax in CLIgen, thus producing CLIgen commands automatically.

Let the configuration file `clicon.conf` contain the following settings:

```
CLICON_CLI_GENMODEL            1
CLICON_CLI_GENMODEL_TYPE       VARS
CLICON_CLI_GENMODEL_COMPLETION 0
```

Also define an 'example' datamodel as follows:

```
module example{
  list a{
    key x;
    leaf x{
      type int32;
    }
    container b{
      leaf-list y{
        type string;
      }
    }
    leaf z{
       type date-and-time;
```

```
        mandatory true;
      }
    }
  }
```

In the CLICON CLI specification (this is usually a file in the frontend/ dir of the installation), insert the reference statement: "@datamodel:example":

```
  set @datamodel:example, cli_merge();
```

This will generate a CLIgen syntax as follows:

```
  set a <x:int>; {
    b; {
      y <y:string>;
    }
    z <z:time>;
  }
```

This syntax will allow the following example CLI statements:

```
  > set a 42
  > set a 42 b y bar
  > set a 42 b y foo
  > set a 42 z 2008-09-21T18:00:00
  > set a 43 b
```

The configuration statement above results in the following XML:

```
  <a>
    <x>42</x>
    <b>
      <y>bar</y>
      <y>foo</y>
    </b>
    <z>2008-09-21T18:00:00</z>
  </a>
  <a>
    <x>43</x>
    <b/>
  </a>
```

And the following database settings:

```
a[0]   $!x=42 $z="2008-09-21T18:00:00"
a[0].b $!x=42 $y="bar" $y="foo"
a[1]   $!x=43
```

Note that the last entry (a[1]) does not satisfy the 'mandatory' statement of 'z'. It will therefore fail in the validate phase.

3.2 Callbacks

The example above did not include callback information. That is, which function to invoke on a completed command. The following CLIgen syntax shows the full callback functions with arguments.

```
set a <x:int>,    cli_merge("a[] $!x"); {
  b,              cli_merge("a[].b $!x"); {
    y <y:string>, cli_merge("a[].b $!x $y");
  }
  z <z:time>,     cli_merge("a[] $!x $z");
}
```

Note that the callback (`cli_merge()`) given in the specification is repeated for every instance in the tree. Further, a generated argument that fits the `cli_set/cli_merge/cli_delete` function family is automatically generated. For an explanation of these key arguments, please see Section 4.

This means that the same generation may also be used for deleting syntax constructs:

```
delete @datamodel:example, cli_delete();
```

3.3 Leafs

Mapping of YANG leafs to CLIgen configuration syntax has some variants depending on the setting of the `CLICON_CLI_GENMODEL` option. This option can be set to `NONE`, `VARS` or `ALL`.

If `CLICON_CLI_GENMODEL` is set to `NONE` or set to `VARS` *and* the leaf is a key variable in a yang list, the mapping is as follows:

```
leaf a;         --> <a>
```

where the value of `a` will overwrite any existing entry in a database.

If `CLICON_CLI_GENMODEL` is set to `ALL` or set to `VARS` and the leaf is not a key variable in a yang list, the mapping is as follows:

```
leaf a;         -->  a <a>;
```

The difference is the occurence of the `a` keyword.

3.4 Leaf-list

A YANG leaf-list is mapped as follows:

```
leaf-list a;    -->  a <a>;
```

The difference with ordinary variables is not visible in the CLIgen syntax, only in how keys are added to a database. A new value is added to the database, it does not overwrite an existing unless the index variable matches an existing entry.

Again `CLICON_CLI_GENMODEL` controls the generation of keyword.

3.5 Container

```
container a{      a; {
   leaf x;          x <x:int>;
   leaf y;    -->   y <y:int>;
  }               }
```

The mapping is straightforward. Note again that `type` is a mandatory sub-statement in the yang sspecification, but omitted here for brevity.

3.6 List

```
list a{           a <x:int>; {
   key x; -->       y <y:int>;
   leaf x;        }
   leaf y;
}
```

or if `CLICON_CLI_GENMODEL` is `ALL`:

```
list a{           a x <x:int>; {
   key x; -->       y <y:int>;
   leaf x;        }
   leaf y;
}
```

3.7 Completion

If the `CLICON_CLI_GENMODEL_EXPAND` option is set to `1`, automatic completion entries will be added to the generated syntax. This is convenient if existing values should be used.

```
list a{
  key x;   -->     a <x:int> | <x:int expand_dbvar("candidate a[] $!x")>;
  leaf x;
}
```

The `expand_dbvar()` function completes the current database values. For example, if there are two existing entries, the user will be prompted those entries as potential values for `x` when making a $< TAB >$.

3.8 Help text

Help-texts added to the specification are passed to CLIgen. Note that when dual symbols are generated (`b` in the example below), the original help text will be used in both places.

```
leaf a{
  type int32;                 --> <b:int>("A helptext");
  description "A helptext";
}
```

4 Mapping to database keys

4.1 Key usage in the embedded database

The underlying CLICON database uses simple embedded databases for storage of *variables* referenced by *keys*. A key is a sequence of elements delimited by periods.

Example of a simple *key specification*:

```
a.b             $name:string
```

Each key (`a.b` in the example) defines a set of typed values, stored as binary data. In the example, the key has the single string variable `name`. The type of the value is 'string'. If the type is omitted, it defaults to string.

In an example database, the variable is assigned a value, such as 'osiris':

```
a.b             $name=osiris
```

The `clicon_dbctrl` may be used to list the key values in a running system. Example:

```
> clicon_dbctrl -d /usr/local/share/clicon/db/candidate_db -p
a.b
--------------------
     type: string len:   5 data: "name"
     type: string len:   7 data: "osiris"
```

Note that the output from `clicon_dbctrl` is somewhat more verbose than the notation used in this document.

4.2 Mapping from YANG datamodel to keys

This section shows how the YANG-based CLICON data model is mapped to key specifications. Example key instantiation is also shown (ie example database content) in the following sections.

4.3 Leaf

```
leaf a{
  type int32;   -->    $a  -->  $a=42
}
```

4.4 Leaf-list

```
leaf-list a{
  type int32; -->    $a  --> $a=42 $a=55 $a=88
}
```

4.5 Container

```
container a{
   leaf x;              a $x $y         a $x=42 $y=33
   leaf y;        -->   a.b        -->  a.b
   container b;
}
```

4.6 List

```
list a{             a[]   $!x $y     a.1   $!x=44  $y=99
   key x;     -->   a[].b $!x    --> a.2   $!x=42
   leaf x;                           a.2.b $!x=42
   container b;                      a.n=2
}
```

Note that the index variables are prepended with an exclamation mark.

Further, the list key specification encodes keys in the database using a special index key (`a.n` in the above example). This index key is used to keep track of list entries.

4.7 Hierarchy

Keys may be structured in hierarchies, thus the following shows an hierarchy of lists:

```
a.b[]          $!name $x:int
a.b[].c[]      $!name $x:int $!y
```

An example of values of such a key specification is:

```
a.b.0          $name=T0 $x=12
a.b.0.y.0      $name=T0 $x=22 $y=F
a.b.1.y.0      $name=T3 $x=14 $y=C
a.b.3.y.0      $name=T9 $x=42 $y=K
```

5 Summary

The CLICON datamodel is a limited YANG specification. The YANG specification has three prposes in CLICON: (1) validation and generation of database keys; (2) validation of NETCONF XML; (3) generating of CLI configutration statements.

In this document the nature of the YANG specification and the mappings from YANG statements to database keys and CLI commands have bee given in some detail.

CLICON code, including several of these examples is GPLv3 and is available via github `https://github.com/clicon/clicon`. Commercial licenses are available.

References

[1] Olof Hagsand, "CLIgen Tutorial", Technical documentation, April, 2011

[2] Olof Hagsand and Benny Holmgren, "CLICON Tutorial", Technical documentation, 2013.

[3] R. Enns, Ed, "NETCONF Configuration protocol", RFC 4741, IETF, Dec, 2006

[4] M. Björklund, Ed, "YANG - A data modeling language for the Network Configuration Protocol (NETCONF)", RFC 6020, IETF, Oct, 2010

Appendix A: Data model example

This appendix show several of the features in this document by a pre-defined example. The example is a part of the CLICON release.

The datamodel in this example is:

```
module datamodel{
    list a{
key x;
leaf x{
    type int32;
}
container b{
    leaf-list y{
type string;
    }
}
leaf z{
    type string;
    mandatory true;
}
    }
}
```

The first step is to install CLICON and the example (see also the CLICON tutorial [2]).

```
> configure         # Configure clicon to platform
> make              # Compile
> sudo make install # Install libs, binaries, and config-files
> sudo make install-include # Install include files (for compiling)
> cd appdir/datamodel
> make
> sudo make install
```

Second step is to start the `clicon_backend` daemon and the `clicon_cli` frontend, and try out the CLI. For example:

```
> clicon_cli
# set a
  <x>
# set a 42
# validate
Config error: key a.0: Missing mandatory attribute: z
CLI command error
# set a
  42
  <x>
# set a 42 z
  <z>
# set a 42 z 99
# validate
# set a 42
  <cr>
  b
  z
# set a 42 b
  <cr>
  y
# set a 42 b y 23
# show configuration xml
<a>
  <x>42</x>
  <b>
    <y>23</y>
  </b>
  <z>99</z>
</a>
#
```