CLICON data model

Olof Hagsand and Benny Holmgren

September, 2013

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].

The CLICON data model is a simple and compact specification language meant to be used in conjunction with CLIgen [1] and embedded databases. The contructs in the CLICON data model can be mapped to more powerful data modeling languages, but is not as expressive as many of those languages, but is "good enough" for most modeling.

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 can be 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 Data model

The data modeling language is inspired by C-programming and the CLIgen specification language, with some small differences. The following shows a simple example:

```
a.b[x]{
  <x:int>;
  c {
    <y[]:string>;
  }
  <z:time mandatory>;
}
```

The example illustrates the basic layout of a CLICON data model. First, it contains *keywords* (`a`, `b` and `c`) and *variables* (`x` and `y`).

Second, keys and variables are structured using curly braces{}. By default, every level of hierarchy is delimited as follows:

```
a{
  b;
  c;
}
```

However, as an optimization, the symbols may be given linearly when there is only one option. For example, the following two specifications are equal:

```
                a{
a b;  <-->        b;
                }
```

2.1 Generating data

The example data model above can be used to generate data in different formats. An example XML data is shown below. In Section 3 and beyond, mapping is shown to other data representations.

```
<a>
  <b>
    <x>23</23>
    <c>
      <y>kalle</y>
    </c>
    <z>2013-09-03T11</z>
  </b>
  <b>
    <x>23</23>
    <c>
      <y>nils</y>
      <y>anders</y>
    </c>
    <z>2013-09-03T12</z>
  </b>
</a>
```

2.2 Variables and types

Variables are given as:

```
<name:type options*>
```

The variable types are the same simple types defined by CLIgen [1]:

Type	Example	C-Type
number	42	`int32`
long	78432798732	`int64`
string	"a string"	`char*`
rest	"rest of line"	`char*`
ipv4addr	1.2.3.4	`struct in_addr`
ipv4prefix	1.2.3.0/24	`struct in_addr` `uint8`
ipv6addr	2001:45ce::1	`struct in6_addr`
ipv6prefix	2001:45ce::/48	`struct in6_addr` `uint8`
macaddr	f0:de:f1:1b:10:47	`char[6]`
uuid	f81d4fae-7dec-11d0-a765-00a0c91e6bf6	`char[16]`
time	2008-09-21T18:57:21.003456	`timeval`
url	http://www.acme.com	`char*` `char*` `char*`

2.3 Variable options

A variable may have options. Many of these are inherited from CLIgen [1]:

Option	Syntax	Descr
default	default:(*type*)*value*	Assign a default value if not given
range	range[*lower*:*upper*]	Allowed value range for numbers
mandatory		This variable is mandatory in list
choice	choice:*s* |... |*s*	Choice of values

Examples:

```
<a:int default:(int)42>
<b:string default:"42">
<c:int range[-3:5]>
<d:string mandatory>
<e:string choice:x|y|z>
```

The `default` and `mandatory` options apply when a variable is within the context of a set or list, and controls the behaviour when the variable is *not* specified:

- mandatory. If the variable is not given, validation will fail. Index variables are always mandatory.
- default. If the variable is not given, it will be assigned the default value.

2.4 Variable vectors

A variable vector is a list of variables with the same specification. For example, the following specification fulfils the XML statement on the right:

```
                    <a>
                      <x>9</x>
a <x[]:int> -->       <x>-8909</x>
                      <x>83</x>
                    </a>
```

2.5 Sets

A set specification statement is a simple grouping of sub-elements. Each sub-element can occur at most once in the set, unless it is declared as `mandatory` where it must exist exactly once.

Example, the following set statement properly specifies the XML:

```
a{                              <a>
  <x:int>;                        <x>9</x>
  <y:int mandatory>; -->          <y>33</y>
  <z:int>;                        <b/>
  b;                            </a>
  c <k:int>;
}
```

2.6 List

A list specifies a set of data with an *index* variable which is unique within each list element. An index variable is by necessity always mandatory.

In the following example, `x` is an index variable:

```
                      <a>
a[x]{                   <x>9</x>
  <x:int>;              <b><z>12</z></b>
  <y:int>;    -->     </a>
  b <z:int>;          <a>
}                       <x>10</x>
                        <y>8</y>
                      </a>
```

3 Mapping to YANG

This section describes how the CLICON data model can be mapped to YANG specifications. Note that this mapping is only an example, there are several details in the mapping that are not shown. In general though, since

CLICON is a less expressive language, it is easier to map to YANG than the other way around.

3.1 Variables

A CLICON variable corresponds to YANG *leaf*.

```
                                  leaf a{
<a:string default:"xx"> -->         type string;
                                    default "xx";
                                  }
```

Further, the keyword "mandatory" is mapped to the YANG "mandatory" statement. Simple type-mappings are straightforward. There are no derived types in CLICON.

3.2 Vector

A CLICON vector corresponds to YANG *leaf-list*.

```
                 leaf-list a{
<a[]:int> -->      type int;
                 }
```

3.3 Set

A CLICON set corresponds to YANG *container*.

```
a{                            container a{
   <x:int>;                     leaf x{ type int; }
   <y:int mandatory>  -->       leaf y{ type int; mandatory true; }
   b;                           container b;
  }                           }
```

3.4 List

A CLICON list translates to YANG *list*.

```
a[x]{                         list a{
   <x:int>;                     key x;
   b;   -->                     leaf x { type int;}
}                               container b;
                              }
```

4 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.

4.1 Variables

```
<a:int>   -->    <a>42</a>
```

4.2 Vector

```
                 <a>12</a>
<a[]:int> -->    <a>42</a>
                 <a>33</a>
```

4.3 Set

```
a{                <a>
   <x:int>;         <x>42</x>
   <y:int>; -->     <y>22</y>
   b;               <b></b>
  }               </a>
```

4.4 List

```
a[x]{             <a>
   <x:int>; -->     <x>12</x>
   b;             </a>
}                 <a>
                    <x>13</x>
                    <b/>
                  </a>
```

5 Generating CLI configuration statements

The CLICON data model can be used to map to CLI configuration statements including help-texts, such as set/delete/modify operations. The mapping is straightforward since the CLICON data modeling language is nearly equal to CLIgen specification statements. However, there are differences.

5.1 Generating syntax

CLICON generateas a CLIgen tree of commands if the `CLICON_CLI_GENMODEL` option is set to anything else than "OFF". Setting it to something else will generate a syntax tree with that name. This tree can be used by the "reference" syntax in CLIgen, thus producing CLIgen commands automatically.

Assume for example that the the configuration file `clicon.conf` contains the following settings:

```
CLICON_CLI_GENMODEL         MODEL
CLICON_CLI_GENMODEL_TYPE    VARS
CLICON_CLI_GENMODEL_EXPAND 0
```

Also assume that the CLICON data model is:

```
a[x]{
  <x:int>;
  b {
    <y[]:string>;
  }
  <z:time mandatory>;
}
```

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

```
set @MODEL, 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
> set a 42 b y foo
> set a 42 z 2008-09-21T18:00:00
> set a 42
```

5.2 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 6.

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

```
delete @MODEL, cli_delete();
```

5.3 Variables

Mapping of CLICON data model variables to CLIgen configuration syntax is as follows:

```
<a>        -->  a <a>
```

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

Note that the `CLICON_CLI_GENMODEL` option controls the generation of variables. If set to `ALL` it applies to all variables. If set to `VARS` it will not apply to index variables.

By default, the setting is `VARS`. The difference is shown below where the `ALL` setting generates a keyword also for the index variable `x`:

```
a[x]{          ALL    a x <x:int>; {
  <x:int>;     -->      y <y:time>;
  <y:time>;             }
}              VARS   a <x:int>; {
               -->      y <y:time>;
                      }
```

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

5.4 Vector

A CLICON vector is mapped as follows:

```
<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 match an existing entry.

5.5 Set

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

5.6 List

```
a[x]{               a <x:int>; {
   <x:int>; -->       y <y:int>;
   <y:int>;           b;
   b;               }
}
```

5.7 Choice

```
<a:string choice:x|y|z> -->  a (x|y|z);
```

5.8 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.

```
a[x]{               a {
  <x:int>;   -->      <x:int>;
                      <x:int expand_dbvar("candidate a[] $!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 $< TAB >$.

5.9 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.

```
a("A helptext") <b>("B helptext");
   -->
a("A helptext") b("B helptext") <b>("B helptext");
```

6 Mapping to database keys

6.1 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.

6.2 Mapping from CLICON data model to keys

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

6.3 Variables

```
<a:int>   -->    $a  -->  $a=42
```

6.4 Vector

```
<a[]:int> -->    $a  --> $a=42 $a=55 $a=88
```

Note that there is no key specification for vectors.

6.5 Set

```
a{
   <x:int>;       a $x $y          a $x=42 $y=33
   <y:int>; -->   a.b        -->   a.b
   b;
  }
```

6.6 List

```
a[x]{             a[]   $!x $y     a.1   $!x=44  $y=99
   <x:int>; -->   a[].b $!x    --> a.2   $!x=42
   <y:int>;                       a.2.b $!x=42
   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.

6.7 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
```

References

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

[2] "CLICON Tutorial", Technical documentation, 2011.

[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:

```
a[x]{
  <x:int>;
  b {
    <y[]:string>;
  }
  <z:string mandatory>;
}
```

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:

```
# 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>
#
```