sncl Documentation

Release 0.3

Lucas Terças

Nov 26, 2021

Contents

sNCL is a DSL (Domain Specific Language) that aims to simplify the authoring of multimedia applications. It is based on NCL (Nested Context Language), the standard language of the Brazilian Digital TV System.

sNCL can be installed following the instructions below:

CHAPTER 1

Getting started

1.1 Installing sNCL

sNCL relies on Lua and LuaRocks, which can be installed from the standard repositories of most distros. LuaRocks is a plugin manager for Lua.

For example, on **Ubuntu Linux** and **Arch Linux**:

```
sudo apt-get install lua luarocks
sudo pacman -S lua luarocks
```

After LuaRocks and Lua are installed, sNCL can be installed using LuaRocks. This command will install sncl and all the Lua plugins it requires.

```
sudo luarocks install sncl
```

Todo: How to install cloning the github repo

```
git clone https://github.com/TeleMidia-MA/sncl
cd sncl
sudo luarocks make
```

Todo: How to install on Windows and MacOS?

1.2 Running an sNCL program

Todo: Add some instructions on how to run an sncl program.

```
cd sncl/spec
sncl example.sncl
```

It will generate a file called example.ncl, to specify a different file, you can use

```
sncl example.sncl -o other-file.ncl
```

And can be used according to the examples and tutorials:

CHAPTER 2

The NCM Model

sNCL (simpler Nested Context Language) is a language made to ease the development of hypermedia applications for the Ginga-NCL middleware, which has as the standard language NCL (Nested Context Language), a XML application. The sNCL compiler translates sNCL files into NCL files, and then these NCL files can be played by the Ginga middleware.

Since both are based on the same model, the NCM (Nested Context Model), many of the concepts of NCL are used in sNCL, however, sNCL introduces new elements, such as macros and templates, that are not comtemplated in the model.

One of the basic entities of the NCM is a node, that can be either a Media node or a Composition node. Medias in sNCL be anything, from a image, video or audio to a Lua script, HTML document or Java code, the type of Media node is also called its subclass, which are used to better define the interpretation of the content.

A Media can have interfaces, which can be separated into properties, anchors, ports and switch ports. A anchor is a subset of the informations of the node, for example, a snippet of a video, or a part of a text file, and properties are informations like the color of the background of a media or its position on the screen. The switch and switch port interfaces are not yet implemented in sNCL, a table with the information of what elements are implemented, will be implemented or won't be implemented can be found at ().

Another important element of the model is the Link element, which sets up temporal and spacial relationships between nodes.

2.1 Medias

2.2 Contexts

Todo: Explain context, and access to elements inside of the context

In NCL,

2.3 The compile process:

The compiler first turns the sNCl file in a Lua table (called symbol table), that is indexed by the Ids of the elements. This table is then used to generate the final NCL document.

For example, this sNCL file:

Generates the following Lua table:

```
media1 = {
   _type = "media",
   hasEnd = true,
   id = "media1",
   line = 9,
   properties = {
      left: '"50%"'
   },
   sons = {},
   type = '"text/html"'
}
```

This example is pretty straightforward. The table creates has the properties of the sNCL element, plus some meta information, like the line number it was created, the elements that are nested inside of it (its sons).

The next element shows the use of the Region element, which serves to reuse the properties of the Media element:

```
region region1
   top: 10%
   left: 50%
end
media media1
   type: "text/html"
   rg: region1
end
media media2
   type: "text/html"
   rg: region1
end
```

The element '__descregion1' is created by the compiler, and it is necessary because in NCL, a Media can not refer directly to a Region. It has to refer to a Descriptor, and then the Descriptor has to refer to said Region. Both Medias now have the 'left' and 'top' properties, but you do not have to declare it twice for each Media

```
head = {
   __descregion1 = {
      _type = "descriptor",
      id = "__descregion1",
      region = "region1"
   },
   region1 = {
      _type = "region",
      hasEnd = true,
      id = "region1",
      line = 3,
      properties = {
         left = '"50%"',
         top = '"10%"'
      },
```

(continues on next page)

(continued from previous page)

```
        sons = {}
    }
}
body = {
    media1 = {
        _type = "media",
        descriptor = "__descregion1",
        hasEnd = true,
        id = "media1",
        line = 7,
        properties = {},
        region = "region1",
        sons = {},
        type = '"text/html"'
    }
    media2 = {
        _type = "media",
        descriptor = "__descregion1",
        hasEnd = true,
        id = "media2",
        line = 11,
        properties = {},
        region = "region1",
        sons = {},
        type = '"text/html"'
    }
}
```

As can be seen, all the tables up to look alike. This is because they are all presentation elements, so they are created the same way. All have id, _type, sons, properties and others informations that are exclusive to each, like the descriptor and region in the case of the Medias that have a Region.

The next example shows the state of the symbol table with a Link element:

```
media media1
    type: "text/html"
end
media media2
    type: "text/html"
end
onBegin media1 do
    start media2
        delay: 20s
    end
end
```

```
head = {
    OnBeginStart = {
        _type = "xconnector",
        action = {
          start = 1
        },
        condition = {
          onBegin = 1
        },
        id = "OnBeginStart",
```

(continues on next page)

(continued from previous page)

```
      properties = { "delay" }
   }
}
body = {
   [1] = {
      _type = "link",
      actions = {
         [1] = {
            _type = "action",
            component = "media2",
            father = <table 1>,
            hasEnd = true,
            line = 10,
            properties = {
               delay = '"20s"'
            },
            role = "start"
         }
      },
      conditions = {
         [1] = {
            _type = "condition",
            component = "media1",
            father = <table 1>,
            hasEnd = false,
            line = 6,
            properties = {},
            role = "onBegin"
         }
      },
      hasEnd = true,
      line = 11,
      properties = {},
      xconnector = "OnBeginStart"
   },
   media1 = {
      _type = "media",
      hasEnd = true,
      id = "media1",
      line = 2,
      properties = {},
      sons = {},
      type = '"text/html"'
   },
   media2 = {
      _type = "media",
      hasEnd = true,
      id = "media2",
      line = 5,
      properties = {},
      sons = {},
      type = '"text/html"'
   }
}
```

2.4 Macros

sNCL also has a new element, the **macro** element, that is neither a Representation Element or a Relantionship Element.This new element behaves exactly like a macro is supposed to.

```
macro macro1 (mName, mType)
   media mName
      type: mType
   end
end
```

2.5 Templates

CHAPTER 3

Default Properties

These are the properties of the Ginga-NCL Player:

Properties	Default Values
top, left, bottom, right, width, height	0
location	0
size	0
bound	0
plan	“video”
baseDeviceRegion	no default
deviceClass	
explicitDur	
background	transparent
visible	true
transparency	0%
rgbChromaKey	nil
fit	nil
scroll	none
style	nil
soundLevel, trebleLevel, bassLevel	1
balanceLevel	0
zIndex	0
fontColor	white
fontAlign	left
fontFamily	
fontStyle	normal
fontSize	no default
fontVariant	normal
fontWeight	normal
player	no default
reusePlayer	false

Continued on next page

Table 1 – continued from previous page

Properties	Default Values
playerLife	close
moveLeft, moveRight, moveUp, moveDown	nil
focusIndex	nil
focusBorderColor	
selBorderColor	
focusBorderWidth	
focusBorderTransparency	
focusSrc, focusSelSrc	nil
freeze	false
transIn, transOut	empty string

CHAPTER 4

sNCL Elements

4.1 Media Element

The media element defines an media object, that can be an image, video, text and even HTML documents or Lua scripts.

Its syntax is defined as:

```
Media = "media" * Id *(Comentario + MacroCall + Area + Propriedade)^0 * end
Area = "area" * Id * (Comentario + Propriedade)^0 * "end"
```

It is identified univocally by the **id** field, for example, the code below declares a media object that is a HTML document and has the id “media1”. In this case, no other element in the entire application may have the id “media1”.

```
media media1
   type: "text/html"
end
```

The media element must have either a **type**, a **source** or **refer** to another element, so the player knows what is the type of the media object.

```
media media1
   type: "text/html" -- a type
end
media media2
   src: "docs/index.html" -- a source
end
media media3
   refer: media2 -- media3 refers to media2
end
```

In addition to specifying the type of the media object, or what the object is, it can also be specified where the object will appear in the screen, the location of it, the list of these other possible properties is in *Default Properties*

```
media media4
   -- a media with margin of 15 pixels on both sides
   src: "medias/image.jpg"
   left: 15px
   right: 15px
end
```

4.1.1 Area Element

The area element defines an anchor (a part of the information of the media element) that may be used in relationships with other objects.

```
Area = "area" * Id * (Comentario + Propriedade)^0 * "end"
```

Anchors can represent:

- Spatial portions of images (begin, end, first, last)
- Temporal portions of continuous media content (begin, end, coords, first, last)
- Textual segments

For example, a temporal portion of a video can used like the example below. When the *media1* gets in 20s, *media2* will start.

```
port pBody media1

media media1
   src: "medias/video1.jpg"
   area area1
      begin: 20s
   end
end

media media2
   src: "medias/image2.jpg"
end

onBegin media1.area1 do
   start media2 end
end
```

4.2 Link Element

The syntax of the link element is:

```
Link = Condition^1 * (Comentario + Propriedade + Action)^0 * end

Condition = AlphaNumeric * Id * TermCond
TermCond = ("and" * Condition) + ("do")

Action = AlphaNumeric * Id * (Comentario + Propriedade) * "end"
```

```
onBegin media1 do
   start media2 end
end
```

The link element must have at least 1 condition and 1 action, in the case above, the condition is "*onBegin media1*" and the action is "*start media2*", meaning that, when the media1 begin, the media2 will start.

The condition and the action can also have properties, like a delay:

```
onBegin media1 do
   start media2 end
   delay: 10s
end

onBegin media1 do
   start media2
      delay: 10s
   end
end
```

As seen in the syntax of the element, it can have multiple conditions and actions. To declare more than 1 action, you simply add it, like a son element:

```
onBegin media1 do
   start media2 end
   start media3 end
end
```

And for multiple conditions, you can concatenate then with the "*and*" keyword:

```
onBegin media1 and onEnd media2 do
   start media3 end
end
```

In this stage of development, the compiler only accepts the *and* value, so, the link will only activate when media1 begin and media2 end. Adding the *or* value will come in later stages.

Below is a list of the accepted conditions and actions:

Conditions	Event Type
onBegin	
onEnd	
onAbort	
onPause	
onResume	
onSelection	
onBeginSelection	
onEndSelection	
onAbortSelection	
onPauseSelection	
onResumeSelection	
onBeginAttribution	
onEndAttribution	
onPauseAttribution	
onResumeAttribution	
onAbortAttribution	

Actions	Event Type
start	
stop	
abort	
pause	
resume	
set	

4.3 Context Element

The context element defines

Its syntax is defines as:

```
Context = "context" * Id * (Comentario + Port + Propriedade + Media + Context + Link
↪+ MacroCall)^0 * "end"
```

As can be seen in the grammar especification, a context element can nest other elements, like *Media Element*, *Macro Element*, *Link Element* and other contexts.

Elements that are inside of a context are only visible to the elements of the same context, meaning that, in the example below, the action of the link in the line 10 can not see the media **m1**, and the action of the line 18 neither.

```
1  context c1
2     media m1
3        src: "medias/image1.jpg"
4     end
5  end
6
7  context c2
8     media m2
9        src: "medias/image2.jpg"
10    end
11    onBegin m2 do
12       start m1 end
13    end
14 end
15
16 media m3
17    src: "medias/image3.jpg"
18 end
19
20 onBegin m3 do
21    start m1 end
22 end
```

4.4 Region Element

The region element defines the initial values of the region of the screen where the media element will appear.

```
Region = "region" * Id * (Comentario + Region + Propriedade + MacroCall)^0 * "end"
```

On the example below,

```
port pBody1 media1
port pBody2 media2

region rgFullScreen
   width: 100%
   height: 100%
   region rgMidScreen
      width: 50%
      height: 50%
      bottom: 25%
      right: 25%
   end
end

media media1
   rg: rgFullScreen
   src: "medias/image1.jpg"
end

media media2
   rg: rgMidScreen
   src: "medias/image2.jpg"
end
```

4.5 Switch Element

4.6 Macro Element

The macro element is

```
Macro = "macro" Id * (Comentario * MacroCall * Propriedade + Media + Area + Context +␣
↪Link + Port + Region)^0 * "end"
```

It behaves like the standard definition of macro, it replaces the words of what it receives as an argument:

```
macro macro1 (mName, mSource)
   media mName
      src: mSource
   end
end

*macro1("media1", "medias/image1.png")
```

The example above creates the one shown below. Note that, even if the argument is passed as a string *"media1"*, when the macro is resolved, it don't become a string, since it is an Id.

```
media media1
   src: "medias/image1.png"
end
```

Macro can contain other macros, and call other macros inside of them, however, recursion is not allowed (it can not call itself, its parent macros or macros that are declared after itself).

```
macro macro1()
   *macro3() -- NOT ALLOWED, macro3 is declared after
   macro macro2()
      *macro1() -- NOT ALLOWED, macro1 is the parent of macro2
      macro macro3()
         *macro1() -- NOT ALLOWED EITHER
      end
   end
   *macro1() -- NOT ALLOWED, macro1 can not call itself
end

macro macro4()
end

macro macro5()
   *macro4() -- ALLOWED
end
```

CHAPTER 5

sNCL by Example

The applications used in the tutorial and the media content can be found in sNCL Tutorials

5.1 Example 1: Hello world

This first example shows a simple multimedia application, that only shows one image. It consists of a *port* element and a *media* element.

5.2 Example 2: Playing a video and an image

TODO.

5.3 Example 3: Playing a video in loop

In this example,

5.4 Exemplo 4: Slideshow (macros)

5.5 Example 5: Slideshow with buttons (macros)

TODO

5.6 Exemplo 6: Allen's operators (macros)

Todo: Seria interessante colocar uma introdução sobre o que são os operadores de Allen.

5.6.1 Precedes e Preceded By

A media1 acontece antes da media2, ou a media2 é precedida pela media1.

```
macro precedes (A, B, delay)
  onBegin A do
    start B
      delay: delay
    end
  end
end

media media1
   src: "media2.mp4"
end

media media2
  src: "media2.mp4"
end

precedes(media1, media2)
```

5.6.2 Meets e Met By

A media1 encontra a media2

```
macro meets (A, B)
  onEnd A do
    start B end
  end
end

media media1
  src: "media2.mp4"
end

media media2
  src: "media2.mp4"
end

meets (media1, media2)
```

5.6.3 Overlaps e Overlapped By

A media1 sobrepõe a media2

```
TODO.
```

5.6.4 Starts e Started By

A media1 começa a media2, ou a media2 é começada pela media1.

```
macro starts (A, B)
  onBegin A do
    start B end
  end
end

media media1
  src: "media1.mp4"
end

media media2
  src: "media2.mp4"
end

starts (media1, media2)
```

5.6.5 During e Contains

A media1 acontece durante a media2, ou a media2 contém a media1.

```
TODO.
```

5.6.6 Finishes e Finished By

A media1 acaba a media 2, ou a media2 é acabada pela media1.

```
macro finishes (A, B)
  onEnd A do
    stop B end
  end
end

media media1
  src: "media1.mp4"
end

media media2
  src: "media2.mp4"
end

finishes (media1, media2)
```

5.6.7 Equals

A duração de ambas as mídias são iguais.

```
macro equals (A, B)
  onBegin A do
    start B end
  end
  onEnd A do
    stop B end
  end
end

media media1
  src: "media1.mp4"
end

media media2
  src: "media2.mp4"
end

equals (media1, media2)
```

CHAPTER 6

sNCL full grammar specification

This page presents the grammar of the language. It follows the specification used in LPeg, the tool used in the compiler for grammar especification.

An **"+"** between elements means an *or*, an **"*"** means an *and*.

"(" and **")"** group elements together, and the repetition of the group, or of a single element, is represented using the **"^"** operator, **"^1"** means *one or more*, **"^0"** means *0 or more*, and **"^-1"** means *one or none*.

Elements between "" are literals, the others are non-terminal.

```
Start = (Comentario + Context + Media + Area + Port + Region + Link + Macro)^0

Comentario = "--" * (AlphaNumeric + Punctuation)^0
Propriedade = AlphaNumeric * ":" * (String + AlphaNumeric)

Context = "context" * Id * (Comentario + Port + Propriedade + Media + Context + Link
↪+ MacroCall)^0 * "end"

Media = "media" * Id *(Comentario + MacroCall + Area + Propriedade)^0 * end

Area = "area" * Id * (Comentario + Propriedade)^0 * "end"

Port = "port" * Id * AlphaNumeric

Region = "region" * Id * (Comentario + Region + Propriedade + MacroCall)^0 * "end"

Link = Condition^1 * (Comentario + Propriedade + Action)^0 * end

Condition = AlphaNumeric * Id * TermCond
TermCond = ("and" * Condition) + ("do")

Action = AlphaNumeric * Id * (Comentario + Propriedade) * "end"

Macro = "macro" Id * (Comentario * MacroCall * Propriedade + Media + Area + Context +
↪Link + Port + Region)^0 * "end"
```

(continues on next page)

(continued from previous page)

```
MacroCall = "*" * AlphaNumeric * "(" * Params^-1 * ")"

Params = AlphaNumeric * ("," * AlphaNumeric)^0
```

CHAPTER 7

sNCL vs. NCL

Todo: Copiar a tabela de completude para cá.

CHAPTER 8

TO-DO

8.1 To-Do (sNCL)

Implemented	To Be Implemented	Won't Be Implemented
Media	Switch	Descriptor
Context	Switch Port	Connector
Area		Descriptor Switch
Region		Any of the bases:
Macro		• Descriptor Base
Template		• Region Base
Port		• etc…
Anchor		
Property		
Link		

8.2 To-Do (Documentation)

Todo: Explain context, and access to elements inside of the context

original entry

Todo: Seria interessante colocar uma introdução sobre o que são os operadores de Allen.

original entry

Todo: How to install cloning the github repo

original entry

Todo: How to install on Windows and MacOS?

original entry

Todo: Add some instructions on how to run an sncl program.

original entry

Todo: Adicionar os exemplos do Garrincha

original entry

Todo: Copiar a tabela de completude para cá.

original entry

Todo: Adicionar os exemplos do Garrincha

CHAPTER 9

Indices and tables

- genindex
- modindex
- search