NuGinga Playcode: A web NCL/NCLua authoring tool for Ginga-NCL digital TV applications

Dina Nogueira, Lois Nascimento, Michael Mello, Rodrigo Braga
SIDIA: Instituto de Ciência e Tecnologia
Manaus, Amazonas
{dina.nogueira,lois.nascimento,michael.bittencourt,rodrigo.braga}@sidia.com

ABSTRACT

Entering the world of Interactive Digital Television (iDTV) applications can lead to an exhaustive process that involves reading extensive standards and may need a robust middleware solution to run Ginga-NCL applications, which is a subset of Ginga and the standard for interactivity of digital television adopted in Latin America countries. Even though there are some open and commercial solutions for development of Ginga applications available on the market, most of these solutions present downsides such as deep dependency of Ginga engine implementation, need of complex environment setup and they are not very intuitive NCL/NCLua coding platforms.

To solve these issues, we created an IDE developed in web technology that makes possible for students and professionals to learn how to develop Ginga-NCL applications. With this tool, developers can write, validate and experiment their applications entirely on the web environment, with no software installation required.

KEYWORDS

Ginga, coding platform, NCL coding tool, interactivity, digital television, NCL, Lua, IDE

1 INTRODUCTION

With the advance of Digital Television, broadcasting television (TV) companies have the possibility to send interactive applications within the channel so the user can play a game or answer a quiz while watching a journal or get the details of a soccer match, all on TV. And, Ginga is the middleware specification that made this interaction possible.

In order to write a software, such as a Ginga application, a developer could use a simple code editor or even an integrated development environment (IDE) that facilitates the writing by having features like syntax highlighting and auto complete. Generally, a IDE has a source code editor, build automation tools and a debugger.

Ginga-NCL is a subset of Ginga [16] and as a particular standard for Latin digital television, learning how to implement Ginga applications may be a challenging and laborious process since many requirements could be necessary to run applications, and there are only few IDEs and open tests suites available for the Ginga community.

In: XIX Workshop de Ferramentas e Aplicações (WFA 2020), São Luís, Brasil. Anais Estendidos do Simpósio Brasileiro de Sistemas Multimídia e Web (WebMedia). Porto Alegre: Sociedade Brasileira de Computação, 2020.
© 2020 SBC – Sociedade Brasileira de Computação.
ISSN 2596-1683

There are some open and closed source solutions available for development of Ginga applications on the market, however most of these solutions have a deep dependency of middleware implementation, require a complex environment setup and are not very intuitive NCL/NCLua coding platforms [20].

In some of the available solutions, the common process to create and test Ginga applications could have up to five steps. First, users would have to setup their environment and then install a Ginga middleware. These two steps by themselves might present issues and require some level of understanding of the Ginga middleware involved. After that, users would create an application and to execute it, may need to embed the application in the required system or device that contains the middleware, so they could finally run it.

Due to these reasons, a web-based IDE was developed aiming a simple coding interface where any developer could easily experiment some preloaded Ginga applications, adapting these examples or even coding a new application. The platform runs entirely on the browser, compiling the application on the client side, therefore it can run regardless of internet connection.

2 RELATED WORKS

There are numerous tools designed to introduce developers to new technologies, making it easy to get started quickly.

An online tool for Web development is JSFiddle [6] that allow developers to create and get started with their own application, allowing to code in HTML, JavaScript and CSS. Besides that, users can develop more complex projects, being able to add 3rd party libraries, such as, ReactJs [7], Vue.js [10], Lodash [1], etc. It can be used as a complete IDE to Web development and make it easy to prototype and test snippets of code, entirely on the web browser.

Other interesting tool is Rust Playground [8] that is simpler than JSFiddle and provides an environment to code and test using Rust language. Users can write, compile and execute Rust code in a web browser. It is similar to C++ Shell [3], which is focused on C++ language.

Regarding Ginga-NCL development, the NCL Eclipse [12] is a plugin to Eclipse IDE [4] that validates if NCL code is in compliance with the ABNT standard and provides a Ginga-NCL Emulator to test code. The main feature gap of NCL Eclipse plugin is that it does not have support to NCLua code, and it only works with NCL code. Another proposal is Gingaway [13] plugin to Eclipse, that combines the NCL Eclipse and Lua Eclipse (plugin that configure Eclipse to Lua development), providing a way to create NCL/NCLua projects, with syntax error feedback. Users can set other middleware runners in order to execute their application, for example Ginga-NCL Virtual set-top box.

The NCL composer [18] is a expansible IDE to create Ginga-NCL applications that allows create graphic interfaces using drag and drop, but this feature is just available to define regions. This IDE enables extensions through plugins to add features to it as media creation by dragging and dropping. This IDE also allows to create NCL applications that define the components behaviour, the structural and temporal organization.

The Dr. Nau [15] is another approach that allows high-level abstractions, with it, users can create diagrams to design relationship between NCL components and tags allowing structural design where it is possible create interactivity applications or prototypes without write any code and then export the result the XML on NCL format.

3 NUGINGA PLAYCODE

This section presents the NuGinga Playcode [1] tool in further details. On section 3.1 an overview of the tool is presented, pointing out its main advantages, on section 3.2 the key features and finally, on section 3.3 details of the architecture are presented along with the main components and their interactions

3.1 Overview

NuGinga Playcode is a web-based IDE which is designed to allow users to edit and run NCL and Lua code on the web page. It allows developers to prototype and test their ideas with quick preview. Similar to other IDEs focused on Web code, such as JSFiddle.

The code written by the user is executed through NuGingaJS [14] engine, which is a full portable Ginga middleware that provides support to Ginga NCL and can run Lua code in the browser environment using Emscripten [5], that allows to compile C/C++ code to asm.js [2] or WebAssembly [11]. This way users can code from anywhere, just needing a web browser.

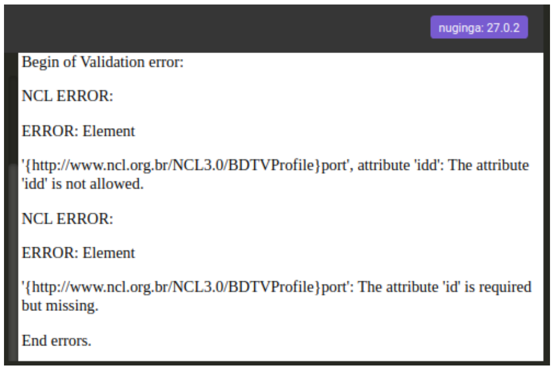
In addition, since it is a non-dependent platform, the NuGinga Playcode enables Ginga developers to execute applications without having to understand about the middleware involved, as it is embedded in the web platform and transparent to the final user. Therefore, developers can easily run NCL and NCLua code with no middleware background knowledge, no requirement of environment setup, middleware installation or need to embed the application on a system that runs the Ginga middleware.

3.2 Key Features

The NuGinga Playcode interface was thought to be minimalist and easy-to-use. It is splitted in three main components highlighted on Figure 1:

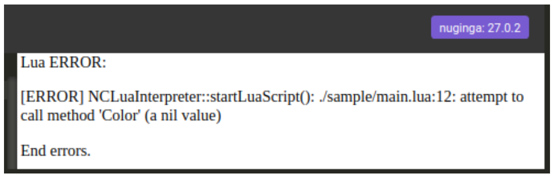
- *NCL Code Editor (label A):* provides a XML-based editor to user, where it can identifies any XML syntax error.
- *Lua Code Editor (label B):* provides an colored editor to write Lua code, it can be accessed on NCL editor as *main.lua* file.
- *Interactive Application Frame (label C):* shows the interactive application resulting from the combination of NCL/Lua codes executed in NuGingaJS engine.

[1]A demonstration video is available on https://bit.ly/3nyHizH.



```
Begin of Validation error:

NCL ERROR:

ERROR: Element

'{http://www.ncl.org.br/NCL3.0/BDTVProfile}port', attribute 'idd': The attribute
'idd' is not allowed.

NCL ERROR:

ERROR: Element

'{http://www.ncl.org.br/NCL3.0/BDTVProfile}port': The attribute 'id' is required
but missing.

End errors.
```

Figure 2: NCL validation error



```
Lua ERROR:

[ERROR] NCLuaInterpreter::startLuaScript(): ./sample/main.lua:12: attempt to
call method 'Color' (a nil value)

End errors.
```

Figure 3: Lua validation error

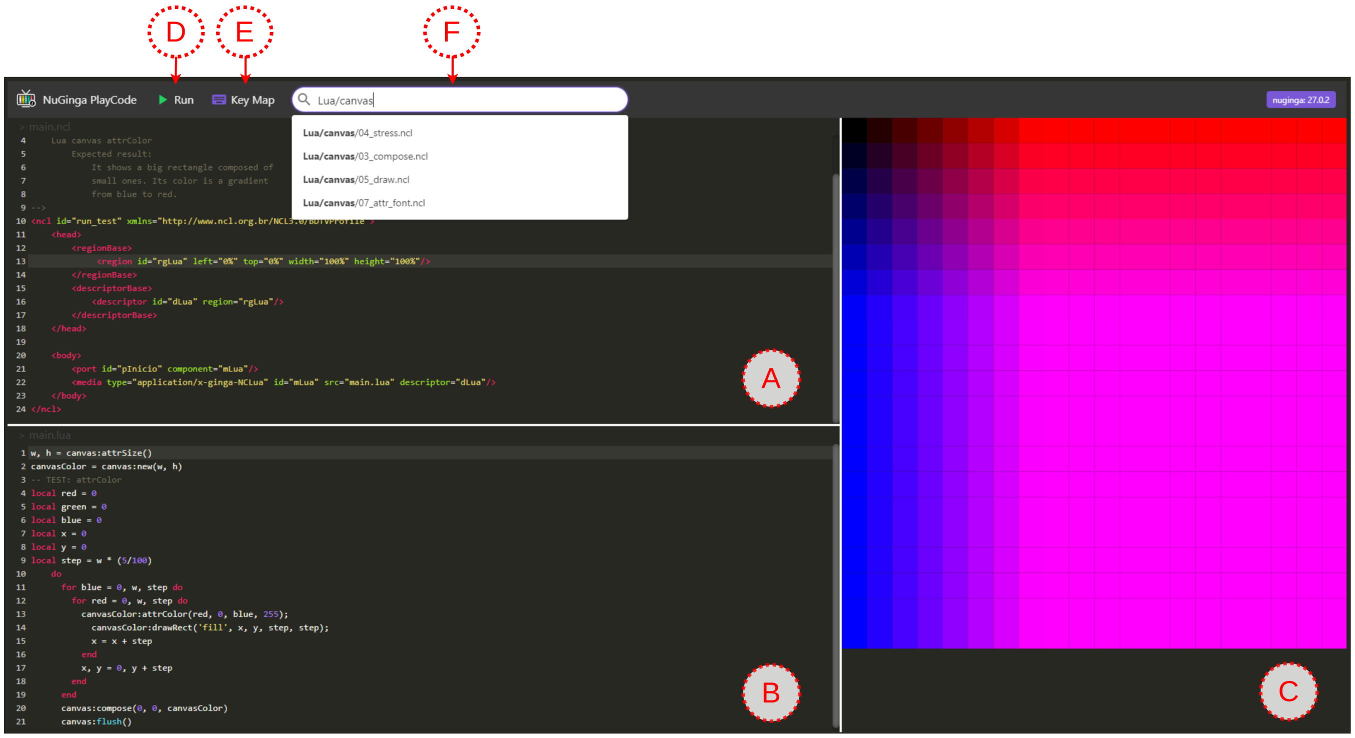
During the coding process, an important feature available on some coding editors and IDEs is the code validation. This feature consists of checking if the code is in compliance with the standards and recommendations. The [17] proposes a model-driven approach that verifies the structural and behavioral properties of NCL document in order to guarantee its conformance with Ginga-NCL standard. In the NuGinga PlayCode, we have XSD schema validation in order to check if NCL code is according to Ginga-NCL standard, the user receives this feedback every time the code is executed through the run button (Figure 1 label D) as shown in Figure 2. Besides that, we show the interpreter outputs information regarding errors of the Lua script execution as demonstrated in Figure 3.

The interaction between viewers and an Interactive Digital TV (iDTV) application is made via remote control [19]. In this context, the PlayCode handles the user input by mapping the remote control events into keyboard events (Figure 1 label E), therefore, the user can test Ginga applications with these events successfully on the web application. Finally, on label F is located a search input where the developer can select, modify and test Ginga applications examples that are preloaded on the platform.

3.3 The Architecture

This platform was developed as a full web application owing to the use of the NuGingaJS middleware, which is the core engine that enables to run Ginga applications entirely on the browser. The Figure 4 shows the architecture used on the NuGinga Playcode development. The project uses *Vue Framework* and has the following components: navigation bar, keyboard Map, NCL Editor, Lua Editor and the main app component.

Vue is a progressive framework for building user interfaces, which is focused on the view layer only and easy to integrate with other libraries or existing projects. Besides that, Vue is also capable of powering Single-Page Applications combined with supporting



```
    Lua canvas attrColor
        Expected result:
            It shows a big rectangle composed of
            small ones. Its color is a gradient
            from blue to red.
-->
<ncl id="run_test" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
    <head>
        <regionBase>
             <region id="rgLua" left="0%" top="0%" width="100%" height="100%"/>
        </regionBase>
        <descriptorBase>
            <descriptor id="dLua" region="rgLua"/>
        </descriptorBase>
    </head>

    <body>
        <port id="pInicio" component="mLua"/>
        <media type="application/x-ginga-NCLua" id="mLua" src="main.lua" descriptor="dLua"/>
    </body>
</ncl>
```

```
w, h = canvas:attrSize()
canvasColor = canvas:new(w, h)
-- TEST: attrColor
local red = 0
local green = 0
local blue = 0
local x = 0
local y = 0
local step = w * (5/100)
    do
      for blue = 0, w, step do
        for red = 0, w, step do
          canvasColor:attrColor(red, 0, blue, 255);
            canvasColor:drawRect('fill', x, y, step, step);
            x = x + step
          end
          x, y = 0, y + step
        end
      end
      canvas:compose(0, 0, canvasColor)
      canvas:flush()
```

Figure 1: NuGinga coding platform

tools and libraries. Furthermore, Vue uses the concept of a component system, allowing to build large-scale applications composed of small, self-contained, and reusable components. [10].

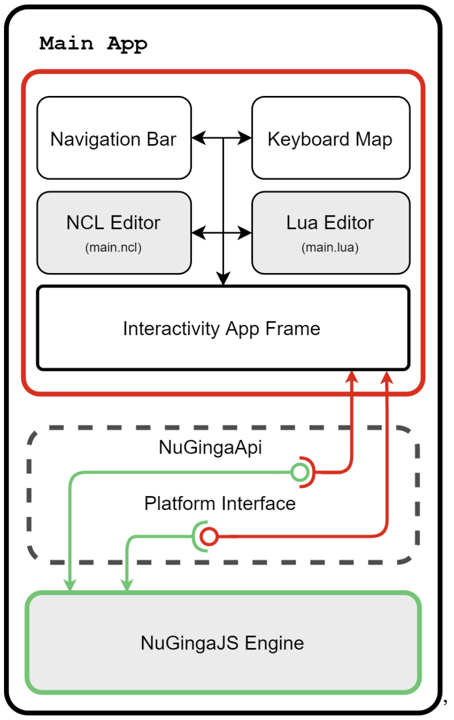


Figure 4: NuGinga Playcode architecture

The main app defines the NuGinga Playcode template, that splits the components into the areas highlighted on Figure 1. The Navigation bar template defines the run button, keyboard map and search bar items. The keyboard map has its own component and is imported to the Navigation bar.

The search bar uses a set of Ginga-NCL samples, which titles are available to the user. When the user selects a title, an event is emitted to the code editors via Event Bus, a communication bus that connects all Vue components, represented in black arrows on Figure 4. NCL and Lua editors wait for this event and then load the code on their edition area.

The code editors use a Vue component called Code Mirror, a versatile open-source Javascript editor for the browser. It is specialized for editing code, and comes with a number of language modes and addons that implement more advanced editing functionality [9]. Other assignments of editors components are to resize the edition areas and save code changes.

The Main App incorporates all Vue components and contains the Interactivity App Frame. On this frame, the Ginga application is rendered using the NuGingaJS engine in the background through the integration layer represented as the dashed area in the architecture Figure 4.

The engine requires the implementation of a Platform Interface to configure some Ginga modules that are restricted to the platform. The interface handles Short Message Service (SMS) and Transmission Control Protocol (TCP) communications, file system access and Ginga platform Settings. Furthermore, to run applications, the Playcode platform uses the following NuGingaApi methods:

- *setConfiguration:* This function is called to apply required settings from the platform, such as language, screen size, channels, among others.
- *createFileInSandbox:* The platform uses this function to push NCL and Lua code files into the browser sandbox.
- *loadAppByRawNCL:* This function is called to load and execute the Ginga-NCL application.

Another duty handled by the integration layer is the error validation display. The Figure 5 shows a sequence diagram of the code validation error process.

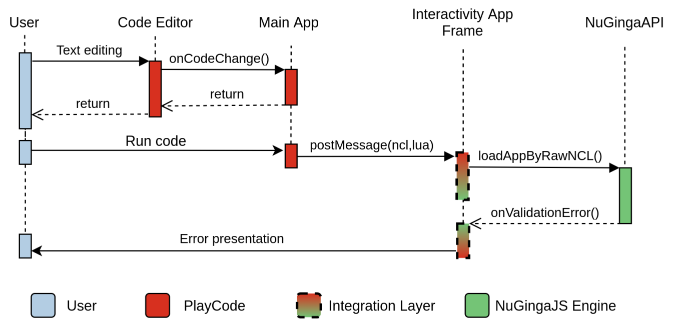


Figure 5: Sequence diagram of the code validation

First, when the user modify the code in the edition area, the Code Editor components notify the Main App, that saves the new content. Then, when the user clicks the run button, the Main App sends the code strings to the Interactivity App Frame via postMessage(ncl,lua). The frame then calls NuGingaApi createFileInSandbox() method to push the files into the sandbox and calls loadAppByRawNCL(), that loads, processes and runs the Ginga application to the user. If there is an error in code validation, the Platform Interface receives it from the engine output as JavaScript object notation (JSON), parses and display it on the frame to the user, as shown on Figure 2 and 3.

4 FUTURE WORKS

The PlayCode aims to facilitate the learning, developing and testing processes of Ginga applications and also contribute to DTV and Ginga community. We intend as future work to expand the features currently available in our tool, introducing file upload and folder trees, improve code validation, among other features. In addition, we plan to collect usability evaluation from Ginga developers in order to improve the user experience for this tool and also adding the evaluating for this interface through tests to measure user acceptance, experience, and usability.

5 CONCLUSION

The NuGinga Playcode is web-based tool created for rapidly development of Ginga-NCL applications, concentrating NCL and Lua editors, application rendering, and compilation debugger. It aims to facilitate the ramp up of new iDT application developers by providing a built-in Ginga engine to run NCL applications entirely on the browser, dispensing environment setups and internet connection. With this tool developers can create new tests and experiment sample applications available on the web-based IDE.

6 ACKNOWLEDGMENT

This work was partially supported by Samsung Eletronica da Amazônia Ltda, under the informatics law no 8.387/91.

REFERENCES

[1] 2013. Lodash. lodash.com. [Online; accessed 01-October-2020].
[2] 2020. asm.js. http://asmjs.org/. [Online; accessed 10-September-2020].
[3] 2020. C++ Shell. http://cppshell.com/. [Online; accessed 10-September-2020].
[4] 2020. Eclipse Foundation. https://www.eclipse.org/. [Online; accessed 10-September-2020].
[5] 2020. Emscripten. https://developer.mozilla.org/pt-BR/docs/Mozilla/Projects/Emscripten. [Online; accessed 10-September-2020].
[6] 2020. JSFiddle - Code Playground. https://jsfiddle.net/. [Online; accessed 10-September-2020].
[7] 2020. react.js. https://pt-br.reactjs.org/docs/getting-started.html. [Online; accessed 10-September-2020].
[8] 2020. Rust Playground. https://play.rust-lang.org/. [Online; accessed 10-September-2020].
[9] 2020. This is CodeMirror. https://codemirror.net/index.html. [Online; accessed 10-September-2020].
[10] 2020. vue.js. https://vuejs.org/v2/guide/. [Online; accessed 10-September-2020].
[11] 2020. WebAssembly. https://webassembly.org/. [Online; accessed 10-September-2020].
[12] Roberto Gerson Azevedo, Carlos Soares Neto, Mario Teixeira, Rodrigo Santos, and Thiago Gomes. 2011. Textual authoring of interactive digital TV applications. *EuroITV'11 - Proceedings of the 9th European Interactive TV Conference.* https://doi.org/10.1145/2000119.2000169
[13] M. F. DE H BELTRÃO FILHO. 2008. GingaWay-Uma Ferramenta para Criação de Aplicações Ginga NCL. , 62 pages.
[14] Rodrigo Braga et al. [n.d.]. NuGingaJS: a full portable ITU-T H. 761 Ginga middleware for DTV and IPTV. https://doi.org/10.1145/3323503.3360301
[15] Sandra Casas, Franco Herrera, Fernanda Oyarzo, and Franco Trinidad. 2019. *Dr. Nau, a Web Generator of Interactive Applications for Digital TV.* 71–86. https://doi.org/10.1007/978-3-030-23862-9_6
[16] Associação Brasileira de Normas Tecnicas. 2018. Televisão digital terrestre - Codificação de dados e especificações de transmissão para radiofusão digital. Parte 2: Ginga-NCL para receptores fixos e móveis - Linguagem de aplicação XML para codificação de aplicações.
[17] Joel dos Santos, Christiano Braga, and Débora Muchaluat-Saade. 2013. Automating the analysis of NCL documents with a model-driven approach. *WebMedia 2013 - Proceedings of the 19th Brazilian Symposium on Multimedia and the Web*, 193–200. https://doi.org/10.1145/2526188.2526214
[18] Gomes Soares L.F Laiola Guimarães R., Monteiro de Resende Costa R. [n.d.]. Composer: Authoring Tool for iTV Programs. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-69478-6_7
[19] R Rodrigues and R Soares. 2006. Produción de Contenido Declarativo para TV Digital. *XXXIII SemiSH, Brasil* (2006).
[20] Franco Trinidad. 2019. Dr. Nau, a Web Generator of Interactive Applications for Digital TV. In *Applications and Usability of Interactive TV: 7th Iberoamerican Conference, jAUTI 2018, Bernal, Argentina, October 16–18, 2018, Revised Selected Papers*, Vol. 1004. Springer, 71.