UNIVERSIDAD DE GRANADA

TRABAJO FIN DE GRADO

INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

Desarrollo de un videojuego para la configuración y análisis de redes de computadores

GNS3sharp

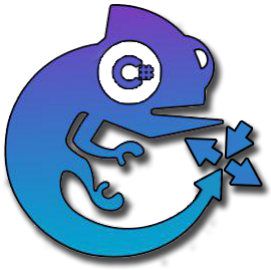
Autor
Ángel Oreste Rodríguez Romero

Directores
Juan José Ramos Muñoz
Jonathan Prados Garzón

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, agosto de 2018



Desarrollo de un videojuego para la configuración y análisis de redes de computadores

GNS3sharp

Autor
Ángel Oreste Rodríguez Romero

Directores
Juan José Ramos Muñoz
Jonathan Prados Garzón

Desarrollo de un videojuego para la configuración y análisis de redes de computadores: GNS3sharp

Ángel Oreste Rodríguez Romero

Palabras clave: palabra_clave1, palabra_clave2, palabra_clave3, ...

Resumen

El jugar ha sido desde siempre un gran amigo de la educación. Aprender jugando es un lema que cada vez se repite más. Los videojuegos, concretamente, toman en cierta forma el relevo de los juegos tal y como tradicionalmente estos se han entendido y amplían sus posibilidades.

La era digital afecta a casi todos los ámbitos de nuestro entorno. Las redes no iban a quedar excluidas de ese avance. Así, se pueden encontrar decenas de implementaciones virtuales de redes de telecomunicaciones, permitiéndonos visualizar su funcionamiento evitando el desembolso que equivale una real.

Digitalizados ambos ámbitos, ¿por qué no unirlos? ¿Y por qué no unirlos con un propósito educacional?

El presente documento pretende realizar un acercamiento a tal propósito. Se listará una serie de tecnologías que permiten llevar esto a cabo así como el desarrollo de mi aproximación.

Development of a videogame for the configuration and analysis of computer networks: GNS3sharp

Ángel Oreste, Rodríguez Romero

Keywords: Keyword1, Keyword2, Keyword3, ...

Abstract

Playing has always been a great friend of education. Learning by playing is a motto that is repeated more and more. Videogames, in particular, take over from games as they have traditionally been understood and expand their possibilities.

The digital age affects almost every area of our environment. Networks would not be excluded from this development. Thus, dozens of virtual implementations of telecommunication networks can be found, allowing us to visualize them working, avoiding the disbursement that is equivalent to a real one.

Digitized both areas, why not unite them, and why not unite them for an educational purpose?

This document is intended to bring this about. A number of technologies will be listed that allow this to be done as well as the development of my approach.

Yo, **Ángel Oreste Rodríguez Romero**, alumno de la titulación Ingeniería de Tecnologías de Telecomunicación de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 25351379C, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Ángel Oreste Rodríguez Romero

Granada a 1 de septiembre de 2018.

D. **Juan José Ramos Muñoz**, Profesor del Área de Telemática del Departamento TSTC de la Universidad de Granada.

D. **Jonathan Prados Garzón**, Profesor del Área de Telemática del Departamento TSTC de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado ***Desarrollo de un videojuego para la configuración y análisis de redes de computadores: GNS3sharp***, ha sido realizado bajo su supervisión por **Ángel Oreste Rodríguez Romero**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 1 de septiembre de 2018.

Los directores:

Juan José Ramos Muñoz **Jonathan Prados Garzón**

Agradecimientos

A Juanjo, por su increíble paciencia e inestimable ayuda, tanto en nuestras tutorías presenciales como aquellas improvisadas por Telegram. A todos aquellos profesores que confiaron en mí y en mis capacidades más que yo mismo en tantos momentos. A todos aquellos compañeros como Alfonso que no solo me facilitaron la vida académica con su conocimiento, sino también con su compañía. A Alberto, que aunque algo reticente de primeras, está dispuesto a echarme una mano de pedírselo. A mis compañeros de Francia, que me impulsaron a ampliar mis conocimientos. A Antonio por el logo tan genial que ha hecho. A mi grupo, porque sin él no habría tenido el ánimo suficiente durante este año para afrontar el proyecto. A todos aquellos amigos que me oyeron quejarme de mi proyecto con estoicismo. A la comunidad de StackOverflow que de tantos apuros me ha sacado.

Pero ante todo, a mis padres, pilar fundamental y soporte absoluto de toda mi vida, universitaria o no. Por la educación que me han dado, por todos aquellos caprichos que me permitieron y por velar siempre por mi salud.

Índice general

Capítulo 1

Introducción

fgxdh

1.1. Introducción

huigk

1.2. Los videojuegos docentes

Mucha bibliografía

1.3. Nuestro problema

Aquí puedes poner tasas de abandono escolar

1.4. Nuestra solución

Explicar qué has hecho

1.5. Motivación personal

```
foreach(Node n in handler.Nodes){
        Console.Write("host: {0}, port: {1}, name: {2}, component: {3}
            ",
                n.ConsoleHost, n.Port, n.Name, n.GetType().ToString())
                    ;
```

```
        foreach(Link link in n.LinksAttached){
                Console.Write($", link: {link.ID}");
        }
        foreach(Dictionary<string,dynamic> port in n.Ports){
                Console.Write($",\n\tport, adapter number: {port["
                    adapterNumber"]}");
                Console.Write($",\n\tport, port number: {port["
                    portNumber"]}");
                Console.Write($",\n\tport, link: {port["link"]}");
        }
        Console.WriteLine();
}
```

Capítulo 2

Introducción

fgxdh

2.1. Descripción del problema

huigk

2.2. Estructura del trabajo

```
foreach(Node n in handler.Nodes){
        Console.Write("host: {0}, port: {1}, name: {2}, component: {3}
            ",
                n.ConsoleHost, n.Port, n.Name, n.GetType().ToString())
                    ;
        foreach(Link link in n.LinksAttached){
                Console.Write($", link: {link.ID}");
        }
        foreach(Dictionary<string,dynamic> port in n.Ports){
                Console.Write($",\n\tport, adapter number: {port["
                    adapterNumber"]}");
                Console.Write($",\n\tport, port number: {port["
                    portNumber"]}");
                Console.Write($",\n\tport, link: {port["link"]}");
        }
        Console.WriteLine();
}
```

Capítulo 3

Estado del arte

El estado del arte se define como el nivel de desarrollo de un ámbito concreto, generalmente relacionado con el mundo técnico-científico.

3.1. Motores de videojuegos

3.1.1. Motores más famosos

3.1.2. Nuestra elección: Unity

huigk

3.2. Simuladores de redes

3.2.1. Simuladores más famosos

3.2.2. Nuestra elección: GNS3

Nodos

Enlaces

La API de GNS3

3.3. Juegos docentes

Con todo lo anterior nuestro objetivo es tal. Ya hay ejemplos

Capítulo 4

Diseño de la solución

Expuestas ya las distintas tecnologías que serán implementadas en nuestro trabajo, queda definir de qué modo estas serán utilizadas y cuál será su papel en la creación del proyecto.

4.1. Diseño de la API

La programación se rige por **capas de abstracción**. Partiendo de conceptos concretos se desarrolla una capa de abstracción haciendo uso de ellos que permite elevar el trato de elementos concretos un nivel por encima[1]. De esta forma ganamos en eficiencia y agilidad de escritura sin perder flexibilidad de desarrollo.

Una **API**, a grandes rasgos, no es más que una capa de abstracción sobre un lenguaje de programación o, más aún, sobre un framework del mismo. Comprende una serie de funciones que facilitan en mayor o menor medida trabajar sobre un cierto motivo. Como ejemplo de API famosa tenemos la de Google Maps, que contiene un compendio de métodos para JavaScript que permiten interactuar directamente con la plataforma de Google y crear nuestros programas jugando con ella[2]. Podemos verlo como una biblioteca de funciones.

Como se ha citado previamente, GNS3 hace uso de una **API REST** (*REpresentational State Transfer*). Esto es, mediante una serie de métodos (los conocidos GET, POST...) asociados a una URI concreta podemos interactuar **vía web** con una aplicación. La diferencia fundamental con otra clase de servicios web es que REST está orientado a recursos y no a métodos. Esto permite a la web utilizar comunicaciones sin estado, facilitando de este modo su escalabilidad[3].

Aunque de increíble utilidad (ya veremos qué papel concreto juega en

nuestro trabajo), necesitamos algo más para poder propiciar la interacción entre el simulador de redes y el videojuego. Necesitamos crear nuestra propia API que haga uso de la API REST de GNS3 y que defina métodos que permita a Unity interactuar con el simulador de redes de forma automática.

4.1.1. Elección del lenguaje

En pleno 2018 la cantidad de lenguajes de programación existentes roza el absurdo. Desde el tradicional C, pasando por el multifuncional Java, el sencillo Python o incluso los llamados lenguajes esotéricos como LOLCODE[4]. De entre todos ellos nosotros elegiremos uno sobre el que trabajar. Esta decisión está condicionada, como es natural, por el motor de videojuegos a utilizar.

Ya que nuestra intención es que el motor sea capaz de establecer interacción con el simulador, es necesario que la API a desarrollar esté escrita en un lenguaje con el que el propio motor sea capaz de trabajar. **C#** parecía la opción más sensata. ¿Por qué? Las razones se exponen a continuación:

- **Porque es el lenguaje más usado en Unity**. Unity admite varios lenguajes de programación con los que desarrollar los scripts asociados a los juegos. C++, usado en otros muchísimos otros motores de videojuegos como Unreal Engine, es uno de ellos. Aunque se trate de un lenguaje increíblemente potente y eficiente, su complejidad de uso es mucho mayor, ya que se encuentra a más bajo nivel. JavaScript es otro de ellos, pero no es tan recomendable como C#, pues entre otras razones, a diferencia de JS, C# es fuertemente tipado[5]. En general, C# es el lenguaje usado por defecto en Unity y el recomendado por todos, documentación incluida.
- **Porque son más motores quienes lo utilizan**. Unity está en el podio de entre los motores de videojuegos más usados en el mundo. Como tal se convierte en un referente. El resto de motores miran hacia él y, si quieren atraer a nuevos programadores, tendrán que hacer de su incursión en el nuevo motor algo sencillo. Este es el caso de Godot Engine, que unos meses atrás decidió incluir tal lenguaje entre los soportados[6]. Esto quiere decir que la API no solo podrá ser usada en juegos creados en Unity, si no que su terreno de juego se verá ampliado. CryEngine es otro de los motores que permiten el uso de C# como lenguaje de scripting.
- **Porque, ante todo, es un gran lenguaje**. C#, similar en cuanto a sintaxis a Java, nació como respuesta a este de la mano de Microsoft. Se trata de un lenguaje de propósito general, aunque es usado primordialmente para la construcción de aplicaciones para infraestructuras

Windows. Es uno de los lenguajes que componen la plataforma .NET de Microsoft. Tal es su importancia que a día de hoy se posiciona como el cuarto lenguaje de programación más usado a nivel mundial[7]. Alguna de las características que lo hacen especialmente atractivo:

- Es un **lenguaje de programación orientado a objetos**, con lo que posee todas las características propias de estos (encapsulado, herencia y poliformismo). Sin embargo, no admite multiherencia. Su componente fundamental es una unidad de encapsulamiento de datos y funciones llamada `type` o "tipo". C# tiene un sistema de tipos unificado, donde todos los tipos en última instancia comparten un tipo de base común.
- Aunque es principalmente un lenguaje orientado a objetos, también **toma prestado del paradigma de programación funcional**. Las funciones pueden ser tratadas como valores mediante el uso de delegados, permite el uso de expresiones lambda, acercándose a los patrones declarativos del paradigma funcional...
- Admite **tipado estatico**, lo cual se traduce en que el lenguaje obliga a que haya coherencia entre los tipos durante el tiempo de compilado. El tipado estático elimina un gran número de errores antes de que se ejecute un programa. Desplaza la carga del momento de la ejecución hacia el compilador para verificar que todos los tipos en un programa encajan correctamente. Esto hace que las aplicaciones grandes sean mucho más fáciles de administrar, más predecibles y más robustas. Además, la escritura estática permite que herramientas como IntelliSense en Visual Studio ayuden a desarrollar, pues conoce el tipo de una variable determinada y, por lo tanto, qué métodos puede utilizar está habilitada a usar. C# incluye además el tipo `dynamic` que permite sortear el tipado estático y dejar que el tipo de variable se averigüe durante el momento de la ejecución[8].

Aclarada las razones, pasamos a analizar el uso que le daremos a este lenguaje.

4.1.2. La API

Antes de desarrollar la API, que llamaremos **GNS3sharp** por un ingenioso juego de palabras entre GNS3 y C#, es necesario reflexionar sobre la forma que tendrá. Para dar soluciones, tenemos que plantearnos antes las preguntas adecuadas:

¿Cómo debe interactuar con el simulador?

Como ya se dijo con anterioridad, GNS3 crea un servidor en el equipo desde el que se ejecuta. Este servidor acepta peticiones REST, creando así una suerte de interacción con el programa sin necesidad de poner las manos en él de forma directa. Se abre de esta forma al mundo del scripting y así a nosotros para crear un conjunto de métodos que faciliten su acceso y gestión.

Tratándose de una API REST, lo único que nos es necesario para la conexión es un **cliente web**. Será pues sobre esto sobre lo que se base primordialmente la relación entre nuestra librería y el simulador de redes como tal. A continuación veremos que habremos de emplear alguna tecnología más.

¿Cómo queremos que interactúe con el exterior?

Es importante tener esto claro, pues el uso de la librería y de sus métodos por parte de una aplicación externa deben ser lo más cómodos posibles. Aquel que vaya a hacer uso de ellas debe tener un esquema claro del modo en que puede gestionar la interacción, haciéndola cercana pues al usuario (más bien desarrollador) último.

GNS3 funciona a través de proyectos. En un proyecto se puede incluir tantos nodos como se desee e interconectarlos a placer. Las posibilidades son infinitas. La singularidad del proyecto apenas pasa por permitir arrancar o apagar todos los nodos contenidos en él a la vez.

Siendo así, se ha optado por concentrar lo crucial de la interacción entre el simulador y el desarrollador en **una sola clase-objeto**. Esta clase, que se explicará con detalle más adelante, permitirá hacer de puente entre el proyecto y los distintos elementos que lo componen y aquel que haga uso de ella.

Más importante aún, ¿qué queremos que haga?

Todo lo explicado hasta ahora es correcto: definimos una forma con la que el código pueda acceder al simulador y reflexionamos en la manera en la que el desarrollador que la use pueda sacarle partido. Sin embargo, ¿qué significa entonces *sacarle partido*?

Para responder no hay más que irse al propio GNS3 y ver todas las opciones que nos ofrece: desplegar nodos, enlazarlos entre sí, cortar esos enlaces, arrancar todos los nodos juntos, parar algunos de ellos a voluntad...

Y por supuesto y aún más importante, **gestionar los aparatos desplegados desde dentro**. Es aquí donde está su verdadero potencial. Manejar

un switch conectado a varios PCs, modificar sus VLANs dinámicamente; un router conectado a varias redes distintas, desactivando y activando algunas de ellas en función de un booleano con el que estemos trabajando... Y todo esto sin necesidad de acceder a GNS3 directamente; todo mediante scripting. ¡Se abre un mundo de posibilidades!

4.1.3. Estructura de clases

Nuestra API hará uso de todas las características propias de los lenguajes orientados a objetos, ya expuestas. A nivel de estructura, la que nos interesa citar ahora es la **herencia**.

La herencia es un mecanismo por el que una clase hija (llamémosla `B`) va a heredar los métodos y las propiedades de una clase padre (llamémosla `A`). La cantidad de elementos heredados entre padre e hijo puede ser determinado en C# mediante el uso de modificadores de acceso. Como dato adicional, este lenguaje no admite herencia de constructores, así que nuestra clase `B` tendrá que definir su propio constructor (o bien explicitar su herencia respecto al de `A`).

Aclarado el concepto de herencia, mostramos la estructura básica sobre la que nuestra librería será construida:

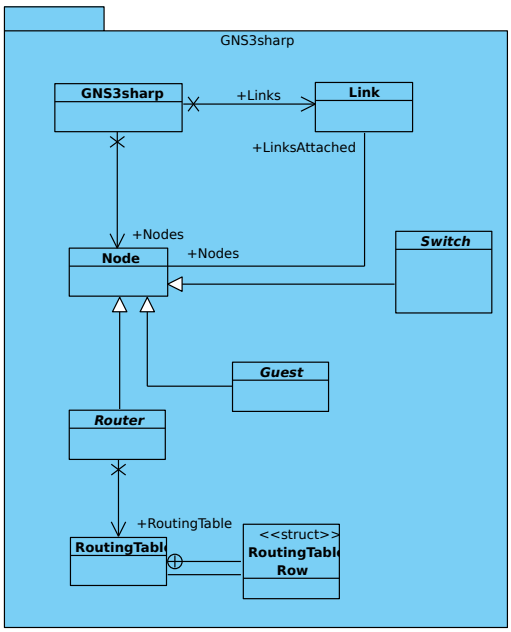


Figura 4.1: Boceto de diagrama UML de la API

A continuación pasamos a desarrollar cada una de las clases conceptualmente:

GNS3sharp, la clase principal

Tal y como expusimos en la sección 4.1.2, crearemos una clase principal que gestionará toda la interacción con el simulador de redes. Hemos llamado a esta clase del mismo modo que a la API, `GNS3sharp`. Recalcamos entonces: los objetos que se encarguen de gestionar los proyectos serán del tipo `GNS3sharp`.

¿Y en qué consistirá está gestión concretamente? Básicamente, esta clase debe encargarse de:

1. Establecer la conexión con el servidor de GNS3 y recopilar toda la información acerca del proyecto que se pretende controlar.

2. Convertir toda es información en objetos útiles que puedan ser utilizados.

3. Crear una gestión eficiente de esos recursos de manera que sean fácilmente accesibles y manipulables.

Estos recursos de los que hablamos serán en gran medida los representados por las clases que se desarrollarán justo debajo.

Sin embargo, la creación de proyectos y despliegue de nodos en los mismos quedarán excluidos, ya sea por la complejidad añadida que conlleva o porque no son funciones que nos sean vitales para la construcción de juegos. Estas funcionalidades deberán ser llevadas a cabo manualmente. Más adelante podrán ser controladas desde la librería sin problema.

Node

////////////////////////// Esto mejor para el capítulo 3 //////// Cada elemento de una red está representado en GNS3 por un elemento llamado **nodo**. Estos nodos, que pueden ser desde un router a un switch, no son más que virtualizaciones de aparatos reales. De normal, estas virtualizaciones se realizan en base a imágenes de los sistemas operativos que se integran en los aparatos. Así, podemos tener varios routers distintos de Cisco montados sobre la misma estructura, permitiéndonos jugar con ellos con verdadera facilidad. //

Sin lugar a dudas es aquí donde se encuentra la característica más interesante de GNS3 y es hacia donde nuestros esfuerzos deben dirigirse. Dado que cada nodo representa un aparato de la red, lo ideal es que ese aparato pueda ser convertido a un objeto en nuestra biblioteca desde el que se nos permita su control.

Esta es la finalidad de la clase `Nodo`. Su deber será el de contener todos los parámetros necesarios para habilitar la conexión con el nodo y así abrir un canal de comunicación con él; un canal que habilite tanto el envío como la recepción de mensajes.

Para la creación de las instancias de la clase se tendrá que recurrir a `GNS3sharp`, que mediante los datos que recoja del servidor de GNS3 será capaz de crear asimismo el objeto.

Como GNS3 admite todo tipo de aparatos de red, cada uno con sus peculiaridades, lo más sensato es crear una clase para cada uno de esos equipos. Esta individualización permite definir métodos propios para cada elemento y, de este modo, facilitar su uso final. Tal y como se puede ver en el diagrama de la figura 4.1, definiremos tres clases principales (`Guest`, `Router` y `Switch`) que heredarán de `Node`. De estas clases nacerán asimismo otra serie de clases referidas a aparatos concretos y a no tipos genéricos.

Link

Otras clases

4.1.4. GitHub y la comunidad

4.2. Diseño del videojuego

4.2.1. Interacción con el simulador

4.2.2. Propuestas de modelo de videojuego

Bibliografía

[1] Pavol Návrat. Hierarchies of programming concepts: Abstraction, generality, and beyond. *Sigse Bulletink*, 26(3):17–28, 1994. Available at `https://dl.acm.org/citation.cfm?id=187397`.

[2] Google maps platform. Available at `https://cloud.google.com/maps-platform/?hl=es`.

[3] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, 2000. Available at `https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm`.

[4] Esolang, the esoteric programming languages wiki. Available at `https://esolangs.org/wiki/Hello_world_program_in_esoteric_languages`.

[5] Joseph Hocking. *Unity in action*. Manning, 2000.

[6] Ignacio Roldán Etcheverry. Introducing c# in godot. Available at `https://godotengine.org/article/introducing-csharp-godot`.

[7] Armina Mkhitaryan. Why is c# among the most popular programming languages in the world? Available at `https://medium.com/sololearn/why-is-c-among-the-most-popular-programming-languages-in-the-world-ccf26824ffcb`.

[8] Joseph Albahari and Ben Albahari. *C# 7.0 in a Nutshell*. O'Reilly Media, 2018.