UNIVERSIDAD DE COSTA RICA

SISTEMA DE ESTUDIOS DE POSGRADO

MAESTRÍA EN COMPUTACIÓN

TRABAJO FINAL DE INVESTIGACIÓN APLICADA

NodeTortoise: aplicación web para simulación participativa

Elaborado por: Oscar Martínez Mora

oscar.martinez.mora@gmail.com

Teléfono: 8730-0626

Director: M.Sc. Alan Calderón

NOVIEMBRE DEL 2015

Índice de contenido

Antecedentes

NetLogo es un ambiente de simulación basada en agentes, utilizado por decenas de miles de estudiantes, profesores e investigadores de todo el mundo. NetLogo puede ser útil en la enseñanza de las ciencias y sirve como un primer acercamiento de niños y jóvenes a la programación. Viene con una gran biblioteca de simulaciones existentes, tanto participativos y tradicionales, que se pueden utilizar y modificar.

En las simulaciones NetLogo tradicionales, la simulación se ejecuta de acuerdo con las reglas que el autor de simulación especifica. No obstante, NetLogo cuenta con la plataforma HubNet, la cual añade una nueva dimensión a NetLogo permitiendo que las simulaciones se ejecuten sólo basadas en las reglas prestablecidas durante la programación, sino que, estas permitan la participación humana directa durante la ejecución.

Además de lo anterior, HubNet permite para la interacción participativa de distintos usuarios por medio de una red de computadoras, permitiendo utilizar NetLogo para ejecutar simulaciones participativas entre distintos usuarios. En una simulación participativa, una clase entera puede colaborar en la definición de la dinámica de un sistema, ya que cada alumno controla una parte del sistema mediante el uso de un dispositivo individual, tal como un ordenador en red (The Center for Connected Learning, s.f.).

NetLogo puede llegar a ser de gran utilidad para niños y jóvenes en las escuelas, así como equipos de investigación y en general personas interesadas en una simulación participativa. No obstante, dadas las características de la arquitectura de NetLogo y HubNet, la interacción participativa se ve circunscrita a una red de área local, limitando así la participación de distintos actores fuera de dicha red y por ende las posibilidades de compartir el conocimiento.

Precisamente por los puntos expuestos anteriormente, se dio el auge de las aplicaciones web en los últimos tiempos. “Las aplicaciones web son populares debido a lo práctico del navegador web como cliente ligero, a la independencia del sistema operativo, así como a la facilidad para actualizar y mantener aplicaciones web sin distribuir e instalar software a miles de usuarios potenciales” (Wikipedia, s.f.).

Las aplicaciones web se basan en la arquitectura cliente-servidor. En esta un servidor, por medio de un formato de comunicación estándar (http) le brinda información a un cliente. En este caso, el único requisito es que el cliente sepa cómo comunicarse con el servidor y cómo interpretar la respuesta recibida. Es gracias a esta ventaja que se popularizaron las aplicaciones web. Gracias a la plataforma web fue posible ejecutar múltiples aplicaciones en distintos sistemas operativos y desde un mismo cliente, todo esto con el único requisito de contar con un navegador web, el cual en la mayoría de casos viene incluido en muchos sistemas operativos.

Las aplicaciones web representan facilidad para el usuario final, pero también lo es para el encargado de la aplicación, ya que le permite darle mantenimiento de manera sencilla, sin necesidad de ir a actualizar todos los clientes, uno por uno. Con una única actualización en el servidor, todos los usuarios se verán automáticamente beneficiados.

Tal como se mencionó antes, HubNet permite ejecutar las simulaciones de NetLogo en una red. No obstante, pese a que utiliza el modelo cliente-servidor, para que esto sea posible todos los usuarios deben descargar el software, cumplir con los requisitos y llevar a cabo las configuraciones necesarias. Sería más sencillo que los usuarios simplemente accedan una dirección web y sin necesidad de instalar nada más, puedan interactuar y colaborar en la simulación.

Para HubNet, existe una extensión que permite la conexión con clientes web, por medio de Web Sockets. No obstante, de esta solo existe el código fuente, el cual debe ser compilado y posteriormente realizar la configuración necesaria para permitir la conexión con un cliente web. Existen dos versiones de esta extensión, la primera fue

creada por Josh Cough[1] en el 2011 y de la cual se describe que es una extensión de NetLogo que permite a los clientes de JavaScript conectarse a NetLogo, no obstante indica que es experimental y no se mantiene activamente (Cough, 2011). La segunda extensión fue creada por Charlie Turner (University of California) en el 2013[2] (versión UC Davis), el cual es un repositorio clonado originalmente de HubNet-Web-Extensión (Turner, 2013).

Dentro del mismo repositorio de ambas extensiones, está presente un cliente JavaScript que permite conectarse a un servidor HubNet desde un navegador web. La funcionalidad de este cliente JavaScript es más limitada en comparación con la del cliente HubNet regular, ya que permite únicamente el despliegue de la simulación y controlar el movimiento del agente por medio de cuatro controles básicos: arriba, abajo, derecha e izquierda. A continuación se muestran dos capturas de pantalla donde es posible observar las diferencias entre ambos clientes[3].

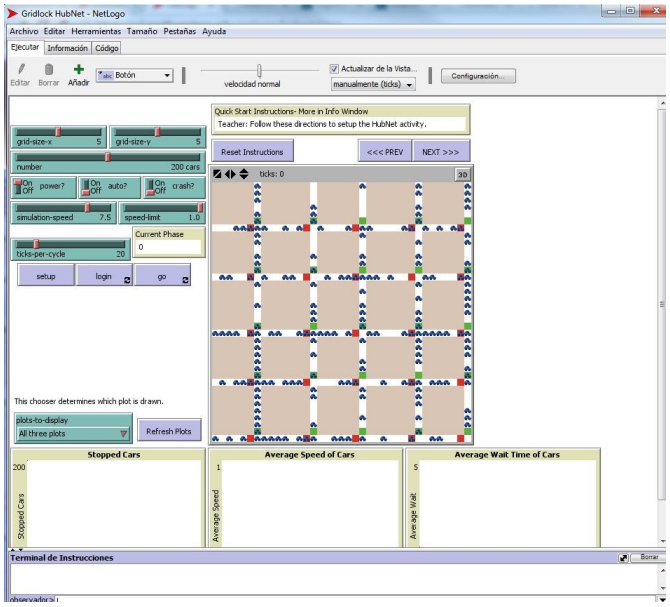

Imagen 1: captura de pantalla del cliente HubNet regular

[1] https://github.com/NetLogo/HubNet-Web-Extension
[2] https://github.com/cjt8109/hubnet-web-extension
[3] La captura de pantalla del cliente JavaScript fue tomado de un video en YouTube, ya que durante el proceso de investigación sobre el tema, no fue posible lograr la conexión entre este cliente y el servidor HubNet. Se revisó el código del cliente JavaScript y se constató que solo existe código para las funcionalidades mostradas en el video obtenido. Puede consultar este video en la dirección: http://www.youtube.com/watch?v=AELAYxwUvAE

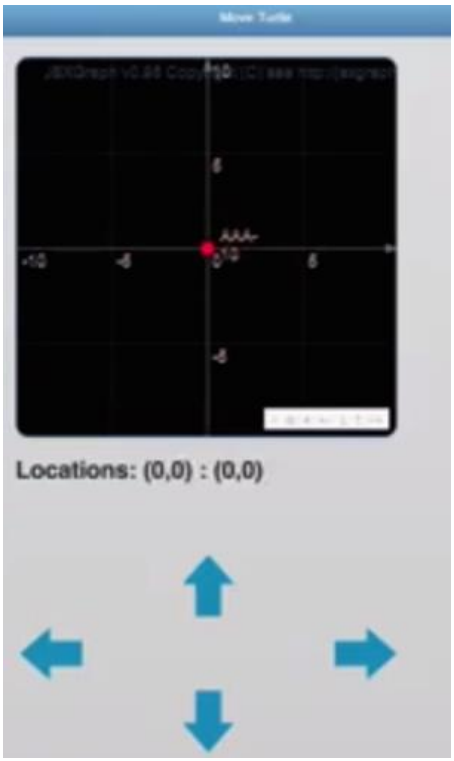

Imagen 2: captura de pantalla del cliente JavaScript incluido con HubNet-Web-Extension

Las capturas de pantallas anteriores, muestran las diferencias entre el cliente NetLogo regular y el cliente JavaScript implementado con la HubNet-Web-Extension. Ambas interfaces corresponden a una simulación específica.

En el caso de la versión UC Davis, el cliente JavaScript fue modificado para adaptarse a la plataforma Apache Cordova y jQuery Mobile y con esto poder ser ejecutado en dispositivos iOS, según se identificó en la revisión del código realizada. El código fuente de estas extensiones se encuentra disponible en GitHub.

Diagrama 1: funcionamiento de HubNet utilizando el cliente JavaScript

Basado en lo expuesto en los párrafos anteriores, podemos indicar que:

1. No es posible ejecutar NetLogo/HubNet desde una plataforma web. Actualmente funciona por medio de sockets de Java y su conexión cliente-servidor se limita a una red de área local.
2. Existe una extensión de HubNet que permite que distintos clientes que soporten esta tecnología, puedan desplegar simulaciones entregadas desde un servidor HubNet. De esta extensión existe el código fuente, el cual debe ser compilado antes de ser utilizado.
3. Se desconoce qué tan limitado es el cliente JavaScript incluido con la extensión web para HubNet. Basados en la captura de pantalla encontrada, se puede visualizar que permite el despliegue de la simulación y controlar el movimiento de cuatro controles básicos: arriba, abajo, derecha e izquierda.
4. Se desconoce si el cliente JavaScript permite la conexión por medio de Internet o se limita a una red local.
5. NetLogo/HubNet actualmente no existe en la forma de aplicación web. Para ejecutar NetLogo, es necesario no sólo descargar y configurar el servidor HubNet, si no que todos los usuarios deben descargar y configurar el software necesario.

Se intentó compilar y hacer funcionar la extensión web para HubNet. El proceso fue infructuoso, dada la escasa documentación al respecto, a pesar de que se obtuvo ayuda en el foro oficial de NetLogo Developers y se intentó entablar comunicación con el creador de la extensión, del cual no se obtuvo respuesta.

Durante el proceso de investigación, se encontró que actualmente existe un proyecto para llegar a migrar NetLogo a una plataforma 100% web. Este proyecto es NetLogo Web, el cual es la reimplementación de NetLogo en el navegador, utilizando JavaScript y HTML, y que busca reemplazar la tecnología basada en applets, ya que estos no funcionan en los dispositivos móviles y son una tecnología obsoleta (Bertsche, 2015). NetLogo Web fue conformado como proyecto en el 2015 (la primera revisión en GitHub

data del 17 de junio del 2015), previo a estas fechas se manejaban dos proyectos independientes Galapagos y Tortoise, los cuales forman parte del proyecto NetLogo y cuentan con el apoyo de Northwestern University (Bertsche, 2012).

La función de Tortoise es convertir el código de NetLogo a código JavaScript y es quien provee el motor de simulación, el cual está basado en CoffeeScript/Javascript (Bertsche, 2015). Por su parte Galapagos proporciona el servidor web que aloja Tortoise (Bertsche, 2015). Actualmente estos proyectos se integraron dentro del proyecto NetLogo Web.

Pese a que Tortoise y Galapagos cuentan con cerca de tres años de desarrollo, no obstante, la documentación es escasa y la percepción personal es que el conocimiento está concentrado en las personas que están dentro del proyecto, ya que en el foro de NetLogo, ellos son los únicos que llegaron a responder las dudas planteadas.

Descripción del problema

Es en este contexto donde se enmarca esta propuesta de Trabajo Final de Investigación Aplicada. Se creará una aplicación web que permita la ejecución de simulaciones de NetLogo/HubNet.

Se pretende facilitar el acceso de los usuarios de NetLogo/HubNet a la herramienta. Actualmente ejecutar NetLogo conlleva una serie de procedimientos que quizás puedan resultar sencillos para una persona inmersa en el mundo de la computación. Pero no debemos perder de vista que NetLogo está orientado tanto a adultos, como niños y jóvenes, así como personas cuyo trabajo implica el uso de simulaciones de NetLogo, pero cuya labor pertenece a otras áreas del saber. De igual forma, debemos pensar en que muchas ocasiones NetLogo/HubNet serán necesarios en laboratorios de computación, por lo que el hecho de no ser necesaria ninguna instalación ni configuración, facilitaría la administración.

Otro de los inconvenientes que presenta el modelo actual de HubNet/NetLogo, es que está principalmente orientado a trabajar en un área de red local (LAN), por lo que una aplicación web instalada en un servidor con IP pública, permitiría brindar acceso a la herramienta desde cualquier parte del mundo, por medio de una conexión a Internet.

Esta aplicación web trabajaría bajo el modelo “Software as a Service” (SaaS), donde la aplicación estará instalada en un servidor web y los usuarios podrán acceder a ella por medio de un portal web, desde el cual será posible controlar la autorización y la autenticación, además de contar con otro tipo de funcionalidades para facilitar su administración. La propuesta actual consiste en crear una plataforma fácil de instalar y configurar, que no necesite de grandes requerimientos de hardware, que permita agregar funcionalidades de forma sencilla, en un único lenguaje de programación y de uso extendido (JavaScript)[4] y con documentación técnica disponible.

[4] Según Gartner, en el 2014 JavaScript ocupó el segundo lugar en el ranking de lenguajes de programación más utilizados.

Las principales diferencias de esta propuesta con el cliente JavaScript existentes son:

- Simulación colaborativa mediante la interacción de varios usuarios sobre los controles de una misma simulación vía web.
- Podrá ser instalada en un servidor web y los usuarios finales solo necesitarán acceder a dicho sitio, sin necesidad de descargar e instalar el software y tener que preocuparse por sistemas de "firewall" que bloqueen las conexiones o configuraciones de red.
- Posibilidad de acceso desde cualquier parte, por medio de una conexión a Internet, con la posibilidad de utilizar el puerto HTTP estándar, utilizado en la navegación web convencional.

En lo que se refiere a Galapagos, es un proyecto que luce similar a este Trabajo Final de Investigación Aplicada planteado, no obstante, su naturaleza es distinta. Durante el proceso de investigación, se pudo comprobar algunos de los inconvenientes de Galapagos:

- El proceso de instalación y configuración es complicada, necesita de mantener una versión de la máquina virtual Java compatible, así como la instalación y configuración del Framework Play y finalmente el proceso de configuración del mismo Galapagos.
- Necesita de una cantidad considerable de recursos, por lo que la idea de Galapagos pasa más por el lado de la centralización, dificultando la oportunidad de que una persona pueda descargar e instalar de forma sencilla, su propia instancia.
- Tiene una arquitectura muy compleja y está escrito en CoffeeScript y principalmente en Scala[5] y es necesario mantener un conocimiento bastante amplio en el Framework Play para poder llegar a realizar alguna modificación (Bertsche, 2015), lo cual, aunado a la escasa documentación existente, produce que sea muy complicado el llevar a cabo modificaciones en la plataforma, para poder agregar funcionalidades.

[5] Según Gartner, en el 2014 Scala ocupó el onceavo lugar en el ranking de lenguajes de programación más utilizados.

- Actualmente no cuenta con soporte multiusuario, por lo que si una instancia de un modelo es accedida por varios usuarios, para cada uno de ellos representa una instancia independiente de las otras.

Justificación

NetLogo es utilizado por decenas de miles de estudiantes, profesores e investigadores de todo el mundo (The Center for Connected Learning, s.f.). Desde niños, hasta adultos pueden llegar a ver los beneficios de esta herramienta en sus actividades académicas y de investigación. Pero esto nos indica, entonces, que no todos los usuarios potenciales de NetLogo y HubNet son profesionales de la computación, con el conocimiento necesario para compilar y realizar las configuraciones necesarias para poder hacer trabajar una extensión, como lo sería la extensión web.

¿Por qué NetLogo/HubNet?

NetLogo permite a los estudiantes abrir simulaciones y "jugar" con ellas explorando su comportamiento bajo diferentes condiciones. También es una "herramienta de autoría", que permite a los estudiantes, a los profesores y a los desarrolladores de planes de estudio, crear sus propios modelos. NetLogo es suficientemente simple para permitir que estudiantes y maestros ejecuten fácilmente simulaciones o que incluso creen su propia simulación. Y, es lo suficientemente avanzado como para servir como una poderosa herramienta para los investigadores en muchos campos (San Francisco State University, s.f.).

Es fácil encontrar artículos sobre la creación de modelos científicos en NetLogo. Esto nos demuestra la utilidad de esta herramienta en distintos campos del quehacer humano. Es por esta razón que debe mantenerse como una aplicación fácil de obtener y ejecutar.

Imaginemos que el acceso a las simulaciones de HubNet fuera tan sencillo como ingresar a una dirección web, sin necesidad de descargar e instalar un software, sin tener que lidiar con problemas de “firewall” o configuraciones de red, accesible desde cualquier equipo en cualquier parte del planeta únicamente por medio de una conexión a Internet.

Para comprender las ventajas de contar con una aplicación web para NetLogo/HubNet, se puede usar como analogía los clientes de correo electrónico. Existen clientes de correo electrónico, como Outlook o Thunderbird, donde el usuario debe instalar y configurar la aplicación, ingresando información de la dirección IP del servidor de correos, el puerto utilizado y otros. Como contraparte, servicios de correo como Gmail o Yahoo Mail, cuentan con aplicaciones web donde el usuario solo debe ingresar su nombre de usuario y contraseña.

El hecho de que NetLogo/HubNet pueda llegar a ser ejecutado, por lo menos a nivel de cliente, en la forma de aplicación web, permitiría que cualquier usuario con un mínimo esfuerzo pueda participar de las simulaciones, sin necesidad de descargar e instalar el software y lo que eso conlleva (derechos de administrador en el equipo, conocimiento básico), sin tener que preocuparse por reglas de "firewall" que podría estar instalado en cada equipo. Además, cualquier cambio que se realice a la aplicación web afectará automáticamente a todos los usuarios, por lo que el encargado tendría libertad de realizar modificaciones para personalizar la herramienta a sus necesidades y sin más, los usuarios se vean beneficiados por esos cambios. De igual forma, al ser una aplicación web permitiría que las simulaciones se vuelvan independientes de la plataforma, no importaría si es Windows, Linux, MacOS o dispositivos móviles, solo es necesario contar con un navegador web moderno. Por último, al ser una aplicación web, esta podrá ser instalada en un servidor en la nube y con ello las ventajas del cómputo en la nube como lo serían disponibilidad, escalabilidad, costos, entre otros.

¿Por qué NodeTortoise?

La plataforma propuesta se ha denominado NodeTortoise, relacionándola así con el proyecto llevado a cabo por Jason Bertsche y equipo, que permite generar código de NetLogo ejecutable desde cualquier navegador web, el cual sirve de base de base al proyecto actual, y NodeJS, tecnología utilizada para la implementación de la plataforma propuesta.

Este Trabajo Final de Investigación Aplicada plantea aún más que facilidad para el usuario. Se propone crear una plataforma que permita a otros estudiantes y personas en general a modificar y extender las funcionalidades. Esta herramienta será dependiente de Tortoise/Galapagos, pero únicamente en el proceso de generación de los modelos. El manejo de usuarios, interacción, interfaz gráficos y casi cualquier otra funcionalidad extra, es manejada por esta herramienta.

El código fuente de NodeTortoise se colocará en un repositorio público de GitHub, bajo licencia GNU[6], permitiendo así su libre descarga, distribución y modificación. De forma adicional, se propone crear y mantener de forma pública la documentación de usuario y técnica necesaria que permita sacarle el mayor de los provechos a la plataforma.

¿Por qué utilizar Tortoise/Galapagos como base?

No debemos reinventar la rueda. Esta es una metáfora comúnmente utilizada en el mundo de la computación e indica que si ya existe algo que tiene la funcionalidad que necesitamos y tenemos los permisos para utilizarla, por qué no utilizarlo en lugar de estar consumiendo recursos en hacerlo desde cero.

Tortoise/Galapagos es un proyecto serio, gestado desde la Northwestern University y que forma parte de los proyectos oficiales de NetLogo (Bertsche, 2012). El código fuente es de libre distribución y modificación (Bertsche, 2012) y tiene una actividad constante[7]. Se espera que el código generado y exportado de Galapagos sea de calidad, además de que estos contarían con un proceso oficial y constante de actualización con respecto a las funcionalidades de NetLogo.

[6] Se decide utilizar esta licencia, en primera instancia para mantener la misma licencia utilizada por los proyectos relacionados con NetLogo y debido a la libertad que brinda esta.

[7] Es posible ver cambios en el repositorio de Tortoise/Galapagos de forma constante. Se puede acceder al log del repositorio por medio de este URL: https://github.com/NetLogo/Galapagos

Adicionalmente se podría perfectamente utilizar el servidor oficial de Galapagos[8] y de acceso público, para generar los modelos, los cuáles serán importados a NodeTortoise, evitando así la instalación y configuración de Galapagos y tener que contar con un servidor con recursos considerables. De manera adicional, se documentará el proceso de instalación de Galapagos y se creará un script de Linux que facilite este proceso, en caso de que se desee mantener en un servidor propio.

Finalmente, es necesario tomar en cuenta que este proyecto nace dentro del marco de un Trabajo Final de Investigación Aplicada, el; cual debería ser concluido en un plazo cercano a los 5 meses, por lo que permite que el desarrollo se enfoque en funcionalidad no implementada en Galapagos, como la interacción entre distintos usuarios, y en agregar nuevas funcionalidades que se consideren útiles.

Los retos del proyecto

El desarrollo de este proyecto implica un gran proceso de investigación, ya que dada la escasa documentación existente para Tortoise/Galapagos, es necesario aprender cómo funciona y se le pueda sacar provecho para facilitar el desarrollo de la aplicación web. El desarrollo de la herramienta propuesta implica conocimientos en NodeJS, web sockets y otro tipo de tecnologías web como HTML 5 y CSS. Además, se debe cuidar detalles como la seguridad de la aplicación, dado que podría estar expuesta al mundo de Internet, y el uso de los recursos del servidor (procesador, memoria RAM, almacenamiento), ya que con el nuevo modelo de cómputo en la nube, en muchas ocasiones el costo depende de los recursos necesarios.

Por último, esta idea nace como un proyecto sin fines de lucro, donde tanto el código fuente, como la documentación estarán disponibles de forma pública y sin ningún costo al público en general, con la libertad de modificar la aplicación a sus necesidades e incluso en un futuro realizar aportes para extender esta herramienta.

[8] Se puede acceder a este servidor por medio del URL: http://li425-91.members.linode.com:9000/

Objetivo General

Desarrollo de una aplicación web para NetLogo, que permita al acceso por medio de internet, implementada de forma simple, utilizando una única tecnología, con código fuente y documentación de acceso público, para que pueda ser fácilmente utilizado y extendido a las necesidades, en el campo académico y científico.

Objetivos Específicos

1. Investigar y modificar el funcionamiento de Tortoise para adaptarlo a las necesidades del proyecto.

2. Documentar el proceso de instalación de Galapagos.

3. Crear un script de Linux que automatice la instalación y configuración de Galapagos.

4. Crear una aplicación web que permita el acceso a las simulaciones participativas de NetLogo/Hubnet desde una plataforma web.

5. Crear documentación de usuario y técnica de la aplicación web a implementar.

Implementación

Descripción de las arquitecturas de las aplicaciones base

A continuación, se describe la arquitectura de NetLogo/Hubnet y de Tortoise/Galapagos, las cuáles son importantes de comprender, ya que son las aplicaciones sobre las cuales se sustenta la arquitectura de NodeTortoise.

NetLogo/HubNet

NetLogo es el ambiente sobre el cual se desarrollan y se ejecutan las simulaciones. Por su parte, la extensión HubNet, añade una nueva dimensión a NetLogo, permitiendo la participación humana directa que las simulaciones y logrando la interacción entre distintas personas, por medio de una red LAN.

HubNet se basa en una arquitectura cliente/servidor. El líder de la actividad utiliza la aplicación NetLogo para ejecutar una actividad en HubNet, en este caso, nos referimos a esta como un servidor HubNet. Los participantes utilizan una aplicación cliente para conectarse e interactuar con el servidor HubNet. HubNet es una arquitectura diseñada para dar a los estudiantes la experiencia de participar como elementos en una simulación de un sistema dinámico complejo. Permite a muchos usuarios "nodos" controlar el comportamiento de los objetos o agentes individuales y ver los resultados agregados en un ordenador central, conocido como el "Hub" (Wilensky, 2001).

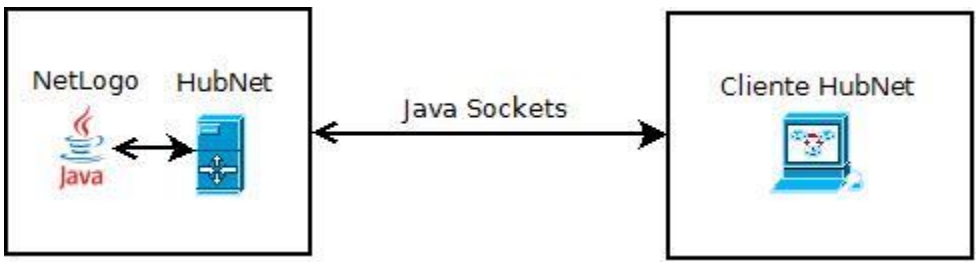

Diagrama 2: Arquitectura de NetLogo/HubNet

NetLogo/HubNet fue creado en el lenguaje de Programación Java.

La interacción de múltiples usuarios, se basa en la idea de sesiones. Para el funcionamiento de estas sesiones, los clientes se conectan a HubNet abriendo una conexión de socket (Java Sockets) a un puerto en particular (el puerto es elegido automáticamente por HubNet). Una vez establecida la conexión, los usuarios deben completar el siguiente *handshake* (Tisue, 2011):

1. El cliente envía un “java.lang.String” que contiene la versión de NetLogo que se están ejecutando.
2. El servidor envía un “java.lang.String” que contiene la versión de NetLogo que se están ejecutando.
3. El cliente envía un mensaje que contiene el nombre de usuario deseado y el tipo de cliente.
4. El servidor responde:

- Si la versión del cliente (enviado en el mensaje 1) coincide con la versión del servidor, a continuación, se envía un “LoginFailure”.
- Si el nombre de usuario del cliente es nulo o vacío, entonces se envía un “LoginFailure”.
- Si el nombre de usuario del cliente ya está en uso por otro usuario, a continuación, se envía un “LoginFailure”.
- De lo contrario, el servidor responde con un uno, o muchos mensajes:
    - En primer lugar, se envía un mensaje “HandshakeFromServer” que contiene el nombre del modelo, y la especificación de interfaz de cliente.
    - Si está activo el modo “vista” (*view mirroring*), el servidor enviará a cada cliente un mensaje “ViewUpdate” que contiene una actualización de la vista.
    - Si está activo el modo “trama” (*plot mirrirong*), el servidor enviará un montón de mensajes relacionados con la trama.

5. El cliente envía un “EnterMessage”. En este momento, el mensaje de entrar se pone en la cola de HubNet, y puede ser procesado con las primitivas HubNet.

Tortoise/Galapagos

Tal como se ha comentado en otras secciones del presente documento, Tortoise/Galapagos vienen a reemplazar a NetLogo/HubNet, utilizando tecnologías web (HTML5 y JavaScript) para visualizar y ejecutar las simulaciones.

Galapagos sirve de servidor web y es el encargado de recibir y responder peticiones de los clientes (navegadores web) y ejecutar funciones de Tortoise, cuando así lo amerite. Galapagos trabaja sobre el *framework* Play 2, utiliza Scala como lenguaje de programación principal y CofeeScript para crear archivos que finalmente serán compilados en código JavaScript para ser ejecutado por navegadores web (Bertsche y Tisue, 2013).

Tortoise está escrito en Scala, y al igual que NetLogo, se ejecuta sobre una máquina virtual Java. Es el encargado de convertir el código de las simulaciones (código de NetLogo) en código JavaScript que pueda ser ejecutado por un navegador web y haciendo uso de Canvas para el graficado 2D (Bertsche y Tisue, 2013). Tortoise es capaz de crea código "stand-alone" que permite ejecutar una simulación desde un navegador web, sin necesidad de un servidor o más tecnología que HTML5, JavaScript y CSS.

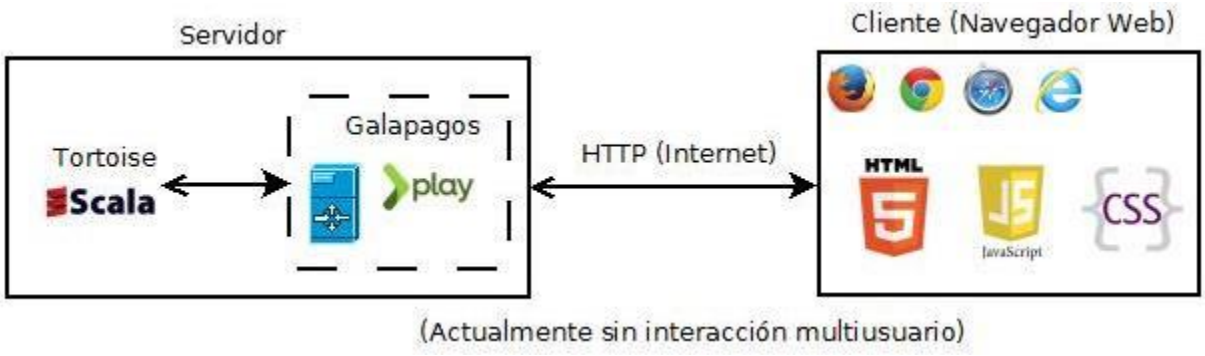

Diagrama 3: Arquitectura de Tortoise/Galapagos

Proceso de Investigación

El presente proyecto de Trabajo Final de Investigación Aplicada nace como una opción para crear una plataforma web para NetLogo/HubNet. Desde el inicio fue evidente la falta de documentación, sobre todo técnica, sobre el tema. Cuando se inició el proceso de investigación para poder sentar las bases de la implementación, se encontró información sobre Tortoise/Galapagos, el cual tiene un fin similar al planteamiento de este Trabajo Final de Investigación Aplicada. Con base en este panorama, se determinó que el proyecto planteado no tenía conflictos con Tortoise/Galapagos, si no que por el contrario, podíamos tomar este proyecto como base y así no tener que "reinventar la rueda".

El hecho de que Tortoise/Galapagos sea un proyecto activo, no cambió el panorama de la falta de documentación. Por el contrario, en comparación con NetLogo, el conocimiento es más reducido, por lo que la fuente de información disponible se reduce principalmente a consultas en el foro oficial, principalmente para esclarecer dudas, pero donde no existe documentación técnica que ayude a comprender la arquitectura y su implementación y con esto poder llevar a cabo modificaciones en el código.

Ejecución

Dada la escasa documentación, la ejecución fue un proceso basado en prueba y error, ejecutando cada vez pequeñas modificaciones para intentar lograr el objetivo principal, el cual consistía en comunicar una instancia de un modelo con otra instancia del mismo modelo.

La arquitectura se eligió desde un inicio. Se implementó en NodeJS, con ayuda de la librería de Sockets.io para la utilización de web sockets, además de HTML 5 y CSS, utilizando librerías como Bootstrap y jQuery, entre otras, para crear una interfaz de usuario atractiva y compatible con los navegadores web modernos. La selección de NodeJS se da por cuanto esta tecnología permite el uso de WebSockets de forma

sencilla, es rápida y liviana y permite mantener el código de la parte del servidor y del cliente en el mismo lenguaje de programación, JavaScript, siendo éste un lenguaje de programación ampliamente utilizado, permitiendo así en un futuro la fácil adopción de las tecnologías utilizadas y así extender sus funcionalidades.

El proceso de implementación inició con el proceso de instalación y configuración de Galapagos. Llevado a cabo este proceso, se procedió a descargar unos de los modelos generados por esta herramienta. Dado que el código generado está en único archivo y puede ocupar incluso más de 54000 líneas de código[9] y muchas de ellas minimizadas[10], dificultando la comprensión del código, por lo que fue necesario separarlos en distintos archivos, según la funcionalidad, de manera que fuera sencillo poder modificar el modelo.

La idea inicial era modificar el código de Galapagos, para agregar el código personalizado, necesario para el funcionamiento de la nueva plataforma. Así se hizo y se llevó a cabo una serie de modificaciones al código, que permitía que los modelos en NodeJS pudieran ser accedidos por distintos usuarios en forma de una única sesión. Luego de un proceso de actualización a la última versión de Galapagos, donde se fue necesario volver a realizar las modificaciones antes implementadas, se decidió que el código implementado debía ser menos invasivo y minimizar al máximo posible la cantidad de archivos de Galapagos modificados, por lo que se decidió sacar toda la lógica a distintas clases y que él único archivo modificado fuera la vista “stand-alone” de Galapagos, donde se importaría todo el código personalizado. No obstante, finalmente se logró el conocimiento suficiente del código de Tortoise para poder sobrescribir la lógica de las simulaciones, sin alterar el código original, por lo que actualmente no es necesario modificar Galapagos, sino que, el archivo exportado es el original y NodeTortoise se encarga de tomar este código y copiar en el archivo final utilizado por el sistema implementado.

[9] Este número de línea es variable entre modelos, pero se colocó a modo de referencia.

[10] La minificación de recursos se refiere a la eliminación de bytes innecesarios, como los espacios adicionales, saltos de línea y sangrías. Al minimizar los códigos HTML, CSS y JavaScript, es posible acelerar la descarga, el análisis y el tiempo de ejecución.

Concluido este proceso, se procedió a trabajar en la interfaz de usuario y funcionalidades extra como la librería de modelos, el manejo de usuarios y la posibilidad de mantener distintas sesiones de un mismo modelo.

Arquitectura de NodeTortoise

NodeTortoise cuenta una arquitectura cliente-servidor, comúnmente utilizada en las aplicaciones web. Está creada en NodeJS, haciendo uso del *framework* para NodeJS *Express 4*, el cual brindó una estructura inicial de archivos y configuración de la aplicación inicial, los cuales fueron modificados a conveniencia.

A nivel de arquitectura, la aplicación sigue un modelo cliente-servidor, con separación en tres capas: presentación, negocio y datos. Si bien es cierto, la aplicación se ejecuta en un mismo servidor y al usar SQLite está limitado a mantener la base de datos en el mismo servidor que la aplicación, existe una separación bien delimitada en la programación y si en un futuro se implementa un motor de base de datos más robusto, como MySQL, será posible llevar a cabo dicha separación física. Por otra parte, se implementó una capa intermedia de acceso a datos, de manera que la lógica de negocios es independiente del origen de datos.

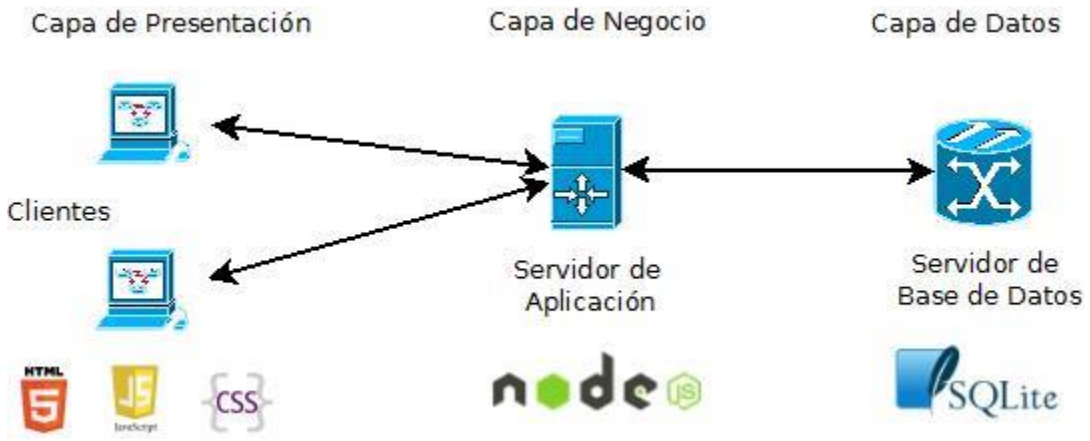

Diagrama 4: arquitectura cliente-servidor con separación de 3 capas utilizada en NodeTortoise

NodeTortoise utiliza el patrón de diseño Modelo-Vista-Controlador (MVC). Al estructurar NodeTortoise siguiendo este patrón, se separa los datos y la lógica de negocio de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones. Bajo este esquema, existe una capa de programación encargada de definir un modelo de datos y acceder al origen de estos (SQLite y WebSockets), otra capa de programación se encarga de controlar las peticiones al servidor, hacer las solicitudes necesarias al modelo de datos y responder con una vista, cuando así sea necesario. Una última capa se encarga de generar las vistas que serán enviadas al usuario, haciendo uso de un motor de plantillas (swig).

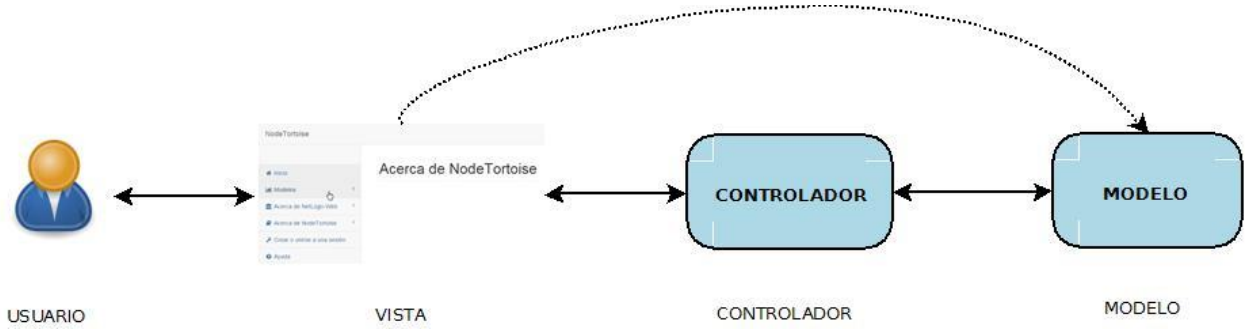

Diagrama 5: patrón de diseño Modelo-Vista-Controlador

La aplicación hace uso de otras librerías de terceros para fines específicos. Estas librerías son:

- **compression** (versión 1.5.2): permite configurar la compresión en las respuestas del servidor.
- **cheerio** (versión 0.19.0): implementación del núcleo de jQuery, diseñado específicamente para el servidor.
- **dateformat** (versión 1.0.11): librería para darle formato a las fechas.
- **morgan** (versión 1.6.1): permite mantener un log de las peticiones HTTP. Deshabilitado para producción.
- **multer** (versión 0.1.8): permite manejar el “multipart/form-data”, que se utiliza principalmente para la carga de archivos.
- **socket.io** (versión 1.3.5): framework para el uso de WebSockets.

- **sqlite3** (versión 3.0.8): framework para el uso de bases de datos SQLite.
- **swig** (versión 1.4.2): motor de plantillas.
- **yuidoc:** aplicación que genera un API de documentación, basado en los comentarios en el código fuente.

A nivel de vista, se hace uso de librerías de terceros, que ayuda a agilizar el desarrollo de una interfaz de usuario de una forma ágil y dinámica:

- **bootstrap:** es un framework para diseño de sitios y aplicaciones web. Contiene plantillas de diseño con tipografía, formularios, botones, cuadros, menús de navegación y otros elementos de diseño basado en HTML y CSS.
- **dataTables:** plugin de jQuery para mejorar las tablas HTML, añadiendo habilidades de clasificación, paginación y filtrado las tablas HTML.
- **fancybox:** es una herramienta para la visualización de imágenes y contenido HTML y multimedia en un "lighbox" al estilo de Mac, donde el contenido flota sobre la página web.
- **fontawesome:** biblioteca de CSS que permite añadir íconos vectoriales escalables.
- **jQuery:** biblioteca de JavaScript que permite simplificar la manera de interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción con la técnica AJAX a páginas web.
- **metisMenu:** plugin para jQuery para Bootstrap 3 que permite crear un menú desplegable con efectos de acordeón animados.

Para la estructura de archivos, se tomó como base la estructura raíz creada por *Express 4*, la cual consiste en:

- **bin:** contiene el archivo de inicio de la aplicación.
- **commons:** contiene archivos JavaScript utilizados tanto por los clientes, como por la aplicación a nivel de servidor. No existe previamente en la estructura de archivos creada por Express 4.
- **public:** contiene los archivos que pueden ser accedidos de forma pública. En este directorio están los archivos hojas de estilo (css), JavaScript (js) y otros que son accedidos por la navegadores web al cargar una página de la aplicación.

    - **apidcos:** contiene los archivos generados por el sistema de documentación.
    - **css:** contiene los archivos de hojas de estilo accedidos por los navegadores web.
    - **fonts:** contiene los archivos de *fontawesome* accedidos por los navegadores web.
    - **img:** contiene las imágenes accedidas por los navegadores web.
    - **js:** contiene los archivos de JavaScript accedidos por los navegadores web.
    - **simulation:** contiene los archivos de modelos que se cargan al sistema.
- **server:** contiene los archivos utilizados por la aplicación a nivel de servidor. No existe previamente en la estructura de archivos creada por Express 4. A esta carpeta se movieron otros directorios importantes en la lógica de la aplicación, como *routes* y *views*. Dentro de esta carpeta se sigue la lógica de la programación por capas, separando explícitamente las clases dedicadas a la lógica de negocio (bl), acceso a datos (da) y vistas (views).
    - **bl:** contiene las clases que se relacionan con la lógica de negocios y los controladores de la aplicación.
    - **da:** contiene las clases relacionadas con el acceso.
    - **db:** contiene al archivo de base de datos de SQLite.
    - **libs:** contiene librerías auxiliares, creadas para fines específicos.
    - **routes:** contiene los archivos de configuración de las rutas de la aplicación (URLs).
    - **tmp:** carpeta utilizada para generar archivos temporales.
    - **views:** contiene las plantillas de las vistas de la aplicación.
    - **config.js:** archivo con valores de configuración de la aplicación.
- **yuidoc:** contiene configuración de yuidoc.

Proceso de sincronización de las simulaciones

El objetivo principal de este proyecto es poder proveer una herramienta que permita ejecutar simulaciones de NetLogo de forma participativa, por medio de una plataforma web. Precisamente, el aporte principal de NodeTortoise consiste en dotar de capacidad multiusuario a los modelos exportados por Tortoise, razón por la cual se consideró que, a nivel técnico, la parte más importante radica en el proceso de sincronización de las simulaciones, entre los distintos participantes.

Durante el proceso de investigación y desarrollo se determinó ciertos requerimientos para llevar a cabo la sincronización de las simulaciones:

- Se creó el concepto de “sesión”, donde cada sesión representa una instancia distinta de una simulación.
- Un único usuario es el encargado de inicializar (función *setup*) y activar y detener (función *go*) la ejecución de una simulación. Este usuario se denominó “Usuario maestro” y se establece como tal al primer usuario que ingresa a una sesión de simulación.
- Se creó el concepto de “paquete de sincronización” en el cuál se envía la información necesaria para que las distintas instancias se sincronicen durante la ejecución de la simulación. El envío de este paquete coincide con la ejecución de la función de actualización del “mundo”.
- La instancia del usuario maestro es la encargada de enviar los paquetes de sincronización el servidor y este a los demás usuarios. El usuario maestro no recibe paquetes de sincronización, si no que la ejecución de esta se da completamente de forma local para este usuario.
- Cualquier usuario, maestro o no, puede modificar los controles de la simulación (*inputs*) y la velocidad (*speed*) y estos cambios afectarán a los demás usuarios, incluyendo al maestro.
- El proceso de sincronización se realizaría enviando el objeto “modelUpdate” de Tortoise, el cual contiene información de todos los cambios realizados al “mundo” en un momento dado. Este método de sincronización garantiza una sincronización

perfecta a nivel de “estado de la simulación” entre las distintas instancias. Este método de sincronización se denominó “sincronización exacta”. Tal como se explicará en la sección de pruebas, este método de sincronización tiene ciertos inconvenientes derivados de la alta transferencia de datos.

- Dado los problemas derivados del método de “sincronización de mundo”, se exploró con la implementación de un método de sincronización consistente en enviar en cada paquete de sincronización únicamente la señal de ejecución del comando “go” y dejar que sea cada una de las instancias la que determine el estado actual de la simulación. Este método se denominó “sincronización libre”. Tal como se explicará en la sección de pruebas, este método soluciona la alta transferencia de datos hacia y desde el servidor, pero el resultado final de cada instancia de la simulación varía debido a que toda simulación es un proceso computacional estocástico.

Pruebas

A continuación se presenta las diferentes pruebas que se realizaron para determinar el funcionamiento del sistema propuesta.

Propósito

Como se indica en los objetivos, la finalidad del presente proyecto es elaborar una aplicación web que permita la ejecución de simulaciones participativas, por medio de Internet.

Por lo tanto las pruebas realizadas tienen como finalidad comprobar que la aplicación funcione de forma ágil y confiable y que cumpla con el objetivo de permitir que dos o más personas puedan ejecutar una simulación de NetLogo de forma colaborativa.

Consideraciones

Dentro de la fase de pruebas, se consideró probar únicamente el proceso de simulación, excluyendo cualquier otra funcionalidad de NodeTortoise. Inicialmente, se planteó una única etapa cuya finalidad era probar la calidad de la sincronización de las simulaciones, en un ambiente participativo con múltiples usuarios y comprobar que el sistema planteado funcionaba tal como se esperaba.

No obstante, los resultados de esta primera etapa no fueron completamente satisfactorios, ya que las simulaciones se notaban con poca fluidez, sobre todo en las pruebas con cuatro personas. Es por esta razón que se implementaron cambios en la forma en que se realiza la sincronización de las distintas simulaciones, los cuales terminaron en el método sincronización que se denominó “sincronización libre”, esto con el objetivo de lograr tener una simulación participativa que cumpla con cierto nivel

de calidad. De esta forma, posterior a los cambios realizados, se dio una segunda etapa de pruebas, siguiendo la misma metodología de la primera etapa.

En la primera etapa de pruebas, se ejecutaron las simulaciones utilizando un método de sincronización el que se le denominó “sincronización exacta”, en el cuál, la sincronización entre usuarios consistía en enviar en cada paquete de sincronización, el objeto de sincronización utilizado por Tortoise.

En la segunda etapa de pruebas, se utilizó el método de sincronización denominado “sincronización libre”, en el cual, en cada paquete de sincronización se envía únicamente la señal de ejecución del comando “go”, dejando que sea la misma instancia la de que, de forma libre, ejecute la actualización.

En ambas etapas de pruebas, se determinó que una misma estación cliente sirviera de usuario maestro, por lo que, dado que las actualizaciones de las simulaciones se ejecutan para este cliente de forma local (en su equipo) y es quien envía al servidor las actualizaciones en el estado de la simulación, esta instancia de la simulación no se ve afectada por defectos en las simulaciones. En el caso de las otras estaciones cliente, los datos provienen del servidor, por lo que su simulación se ve afectada por el tiempo y la velocidad de la transferencia de datos.

Lugar y fecha

La primera etapa de pruebas se llevó a cabo el día sábado 19 de setiembre del 2015. En esta participaron cuatro personas, estando todas en distintas ubicaciones.

La segunda etapa de pruebas se realizó el sábado 10 de octubre del 2015, siendo los participantes las mismas cuatro personas de la primera etapa, estando cada uno en la misma ubicación que en dicha etapa.

Características técnicas

Las pruebas se ejecutaron utilizando la aplicación en un servidor con las siguientes características: instancia de Amazon EC2, con sistema operativo Ubuntu 14.04, 1 CPU, 1 GB de memoria RAM y 27 GB de disco duro SSD.

Con respecto a los participantes, se consideró importante tomar en cuenta la velocidad de la conexión a Internet. En este caso se obtuvo:

- Participante 1 (usuario maestro): 4 MB de bajada de datos y 1 MB de subida de datos.
- Participante 2 (usuario normal): 4 MB de bajada de datos y 1 MB de subida de datos.
- Participante 3 (usuario normal): 2 MB de bajada de datos y 1 MB de subida de datos.
- Participante 4 (usuario normal): 2 MB de bajada de datos y 1 MB de subida de datos.

Metodología de las pruebas

Tal como se explicó con anterioridad, en un inicio se planteó una única etapa de pruebas, que consistía en probar el funcionamiento correcto de las simulaciones en un ambiente participativo, con usuarios en distintas ubicaciones, interconectados a un servidor por medio de Internet, utilizando protocolos estándar como HTTP y WebSockets.

No obstante, dados los problemas encontrados durante la primera etapa de pruebas, se realizaron modificaciones para tratar de corregir dichos inconvenientes. Por esta razón, se planteó una segunda etapa de pruebas, utilizando la misma metodología de la primera etapa, con el fin de validar los resultados de los cambios implementados.

Las pruebas fueron diseñadas para ser evaluadas de manera cualitativa, donde es importante la percepción de los usuarios sobre el funcionamiento del sistema. Se toma como parámetro para determinar la correcta sincronización entre instancias, el estado de la simulación y de los valores de salida.

Cada etapa de pruebas se realizó en dos fases:

1. Se probaron las simulaciones sin realizar ningún tipo de manipulación de los controles (inputs). Con este fin se utilizaron cinco modelos distintos: 3D Solids, 3D Surface, Division, Rugby y Virus. Para cada una de estas ejecuciones se llevaron, se dieron seis repeticiones, tres de ellas con 2 usuarios y las tres restantes repeticiones con cuatro usuarios.

2. Se probaron las simulaciones manipulando los controles (inputs). Con este fin se utilizaron los mismos modelos del punto anterior. Cada una de estas ejecuciones se llevaron a cabo dos veces, una de ellas con dos usuarios y la otra con cuatro usuarios. En el caso de las ejecuciones con dos usuarios, el usuario maestro se encargó de modificar los controles en tres de los modelos utilizados, mientras que el usuario normal hizo lo mismo con los otros dos modelos. En las ejecuciones con cuatro usuarios, el usuario maestro modificó los controles en dos ocasiones, y los usuarios normales estuvieron encargados de modificar los controles en un modelo cada uno.

Todas las ejecuciones de los modelos se detuvieron después de transcurrido un minuto, o bien, hasta que la ejecución se detuviera, si esto sucedía antes del primer minuto.

Métodos de recolección de resultados

Los resultados fueron recopilados por el organizador, que a su vez fungió como usuario maestro, estableciendo comunicación con los demás usuarios por medio de *Skype*,

solicitando sus opiniones y capturas de pantalla, y compilando dicha información con los resultados obtenidos de su propia labor.

Resultados

Etapa 1: Sincronización exacta

Fase 1 (2 usuarios):

- Sincronización de la simulación: en todos los casos fue correcta y las simulaciones terminaron en el mismo estado cuando se detuvieron.
- Sincronización de outputs: en todos los casos, los valores de los outputs fueron iguales.
- Percepción del usuario maestro: reporta un funcionamiento idóneo del sistema, en todos los casos. El usuario reporta que es necesario presionar en dos ocasiones el botón de “setup”, para que este se ejecute en las instancias de los usuarios normales.
- Percepción del usuario normal: reporte un funcionamiento correcto del sistema. No obstante, indica que por momentos el movimientos dentro de las simulaciones no se veía fluido, no obstante, indica que no afectaba de gran manera. Indica que la simulación con mayores problemas fue Rugby.

Fase 1 (4 usuarios):

- Sincronización de la simulación: en todos los casos fue correcta y las simulaciones terminaron en el mismo estado cuando se detuvieron.
- Sincronización de outputs: en todos los casos, los valores de los outputs fueron iguales.
- Percepción del usuario maestro: reporta un funcionamiento idóneo del sistema, en todos los casos. El usuario reporta que es necesario presionar en dos ocasiones el botón de “setup”, para que este se ejecute en las instancias de los usuarios normales.

- Percepción de los usuarios normales: todos los usuarios normales reportan que las simulaciones se notaban poco fluidas. Los usuarios indican que se presentan mayores inconvenientes en las simulaciones Rugby y 3D Surface, las cuales corresponden a las simulaciones con de mayor complejidad a nivel de gráficos.

Fase 2 (2 usuarios)

- Percepción del usuario maestro: el sistema funciona correctamente cuando se modifican los controles estando la simulación detenida. No obstante, es difícil manipular controles cuando la simulación está en ejecución, ya que pareciera que éstos no responden.
- Percepción del usuario normal: reporta condiciones similares a las reportadas por el usuario maestro.

Fase 2 (4 usuarios)

- Percepción del usuario maestro: el sistema funciona correctamente cuando se modifica los controles estando la simulación detenida. Al igual que en las pruebas ejecutadas con dos usuarios, los controles responden de forma lenta cuando la simulación está en ejecución.
- Percepción del usuario normal: todos los usuarios reportan condiciones similares a las reportadas por el usuario maestro. Cuando se modificó la velocidad de ejecución, se mostró mayores inconvenientes en la fluidez de la simulación, problema relacionado con la fase 1, que tiene que ver con la calidad de las actualizaciones.

<u>Etapa 2: Sincronización libre</u>

Fase 1 (2 usuarios)

- Sincronización de la simulación: es fluida y la actualización es rápida. El estado de la simulación difiere entre ambas instancias.
- Sincronización de outputs: los valores de los “output” son distintos al terminar de ejecutar se la simulación.

- Percepción del usuario maestro: funciona correctamente.
- Percepción del usuario normal: funciona correctamente y se ve un proceso mucho más fluido y natural.

Fase 1 (4 usuarios)

- Sincronización de la simulación: es fluida y la actualización es rápida. El estado de la simulación difiere entre ambas instancias.
- Sincronización de outputs: los valores de los “outputs” son distintos al terminar de ejecutar se la simulación.
- Percepción del usuario maestro: funciona correctamente.
- Percepción de los usuarios normales: los usuarios coinciden en que la fluidez de las simulaciones se tornan agradables para el usuario, porque les de la percepción que funciona como debe ser.

Fase 2 (2 usuarios)

- Percepción del usuario maestro: el reporte es similar al de la fase 1.
- Percepción del usuario normal: el reporte es similar al de la fase 1.

Fase 2 (4 usuarios)

- Percepción del usuario maestro: el reporte es similar al de la fase 1.
- Percepción del usuario normal: el reporte es similar al de la fase 1.

Análisis de resultados

Etapa 1: Sincronización exacta

De los resultados obtenidos, podemos determinar que las simulaciones funcionan correctamente y que efectivamente se alcanza el objetivo de mantener dos o más instancias sincronizadas, permitiendo la ejecución de simulaciones colaborativas desde una plataforma web. No obstante se nota una clara disminución en la velocidad de la sincronización cuando empiezan a interactuar más usuarios.

Este problema se debe a la alta transferencia de datos, ya que bajo este modelo de sincronización se debe enviar todo el objeto de actualización. Otro de los factores que afecta es el tiempo de latencia en la transmisión, ya que se debe tomar en cuenta que las velocidad de la conexión a Internet, la cual usualmente es mucho más baja que en redes locales.

El usuario maestro no se ve afectado por ninguno de los inconvenientes comentados anteriormente, ya que las actualizaciones para este usuario se ejecutan desde el procesamiento local, mientras que para los demás usuarios, la actualización proviene del servidor y luego debe ser procesado por el navegador web. En lo que respecta a los valores de salida se obtiene un funcionamiento correcto y estos se mantienen sincronizados hasta el final de la simulación.

En el caso de las pruebas llevadas a cabo modificando los controles, se obtiene que éstos modifican de forma correcta la simulación. No obstante, se vuelve complicado manipular los controles cuando la simulación está en ejecución.

Etapa 2: Sincronización libre

Bajo este modelo de sincronización, se obtiene velocidad en la ejecución de las simulaciones para todos los usuarios y les da una impresión de fluidez. El aspecto negativo es que el ser simulaciones con estados aleatorios, el estado resultante al finalizar las simulaciones, tanto a nivel de gráficos, como de valores de salida, difieren entre los participantes.

El inconveniente reportado bajo este modelo de sincronización, tiene su fuente en la naturaleza estocástica de los procesos de simulación basados en NetLogo, por cuanto cada instancia de un proceso de simulación sigue su propia dinámica, a pesar de que los valores iniciales de los parámetros de ejecución sean los mismos y las señales de actualización se ejecuten sincrónicamente.

En el caso de las pruebas llevadas a cabo modificando los controles, al igual que en la Etapa 1, se obtiene que éstos modifican de forma correcta la simulación, pero la manipulación de los controles resulta muy lenta cuando la simulación está en ejecución.

Síntesis

A partir de las pruebas llevadas a cabo, se obtiene que NodeTortoise cumple con el objetivo de permitir la ejecución de simulaciones participativas por medio de Internet, a través de una plataforma web. Con este fin, se llevó a cabo dos métodos de sincronización, ambas cumpliendo el objetivo de sincronizar dos o más instancias de una simulación, pero ambas con diferentes ventajas y desventajas.

Por un lado utilizando el método de “sincronización exacta”, tenemos problemas en la fluidez en la actualización de la simulación, producto del tiempo de latencia existente entre la comunicación de la instancia que envía el mensaje de comunicación (instancia del usuario maestro), el servidor y su correspondiente procesamiento por las demás instancias. Esto en gran medida debido al tamaño del paquete de actualización, el cual comprende todo el “mundo”. Este problema de latencia es más claro en las simulaciones con mayor cantidad de gráfica e incluso fue posible observar cómo, después de detenida la simulación en la instancia principal, esta se sigue procesando en las demás instancias, hasta completar el proceso y quedar en el mismo estado que la instancia “madre”.

Por otro lado, el método de “sincronización libre disminuye el tamaño del paquete de actualización, reduciéndolo a una simple señal de “go”, pero dando resultado que las distintas instancias de las simulaciones concluyan en estados distintos.

Basado en los resultados anteriores, podemos determinar que, pese a que el proceso “sincronización exacta” tiene defectos, es el ideal para el fin que se busca en las

simulaciones participativas. No obstante, preocupa el hecho de que los defectos ya mencionados, podrían resultar en una pobre experiencia de usuario y esto, aunado a que entre más usuarios, más se agrava el problema.

En lo que respecta al proceso de manipulación de los controles, se obtiene que, sin importar el modo de sincronización de la simulación, este funciona correctamente, siempre y cuando se lleve a cabo con la simulación detenida. Cuando la simulación está en ejecución, los controles reaccionan de forma lenta a la manipulación del usuario, se nota como si estuvieran “trabados”. No obstante, esta característica es observable en Tortoise, donde ni siquiera existe proceso alguno de ejecución multiusuario, por lo que se puede achacar este inconveniente a que el proceso de actualización de la simulación necesita de mucha capacidad de procesamiento por parte del navegador web, por lo que el navegador se “congela” durante ciertos momentos. A raíz de esta característica, se decide deshabilitar la posibilidad de modificar los controles durante la ejecución de una simulación.

Conclusiones

¿Qué se logró con NodeTortoise?

- El proceso investigación y modificación sobre el funcionamiento de Tortoise fue el adecuado para alcanzar el objetivo del proyecto. Se alcanzó a comprender el sistema de tal forma, que fue posible sobrescribir la lógica para implementar el proceso de sincronización de las instancias, sin necesidad de alterar el código original de Tortoise o Galapagos.

- Se documentó el proceso de instalación de Galapagos, de manera que este pueda ser más accesible y comprensible por los usuarios.

- Se creó un script de instalación de Galapagos para Ubuntu y CentOS, pero este mismo script se puede utilizar de guía para implementarlo para otras distribuciones de Linux y sistemas operativos. Con este script, se facilita aún más la instalación de Galapagos.

- Se creó NodeTortoise, una aplicación web que cumple con el objetivo de permitir la ejecución de simulaciones participativas de NetLogo/HubNet por medio de Internet, a través de una plataforma web. Efectivamente, fue posible tomar un modelo generado por Tortoise y llevar a cabo la sincronización de este, tal y como funciona en la plataforma de HubNet. Esta aplicación fue sometida a pruebas y se comprobó un funcionamiento satisfactorio.

- Existe documentación de usuario y técnica sobre NodeTortoise, disponible para los usuarios. De igual forma, NodeTortoise cuenta con un script de instalación para Ubuntu y CentOS, que facilita el proceso de instalación.

Las ventajas de NodeTortoise con respecto a Tortoise/Galapagos

- A pesar de tener a Tortoise/Galapagos como base, NodeTortoise es multiusuario, permitiendo a dos o más personas interactuar en una misma instancia de simulación.

- NodeJS se presentó como una plataforma fácil de utilizar y bien documentada, y a pesar de que personalmente no se contaba con experiencia previa, el conocimiento de JavaScript fue suficiente para poder desarrollar cómodamente, caso contrario a cuando se tuvo que estudiar el código en Scala y Play Framework, en que está hecho Galápagos.

- NodeJS es liviano y el consumo de recursos de hardware es menor al percibido con Galapagos, con el que se experimentó que en etapa de compilación un equipo con 1 CPU a 3.3 GHz y 1 GB de memoria RAM, resultaba ser insuficiente.

- Por esta razón, el hecho de que NodeTortoise cuente con documentación técnica y de usuario de fácil acceso, permite que en un futuro otros estudiantes y personas en general puedan llevar a cabo modificaciones y extender las funcionalidades de forma sencilla.

¿Qué se puede mejorar en la versión actual de NodeTortoise?

- Se puede mejorar el tiempo de latencia en la comunicación entre las distintas instancias de una simulación. Con este fin, se podría descomponer el objeto que contiene el mundo y que es enviado en el paquete de actualización, quizás controlando que objetos han cambiado y enviando únicamente dichos objetos, en lugar de enviar el mundo completo.

- Se puede mejorar la reacción de los controles cuando la simulación está en ejecución.

Experiencias adquiridas

- Proyectos de corte académico/científico como NetLogo y Tortoise sufren de grandes problemas de documentación, sobre todo a nivel técnico, quedando el conocimiento encerrado en las personas que integran los equipos de trabajo.

- A pesar de las dificultades presentadas, es posible llevar a cabo proyectos que implican trabajar sobre plataformas de código abierto, ya que por lo menos en el caso del presente proyecto, se encontró disposición de las personas implicadas en el proyecto oficial a contestar las dudas, por medio del foro correspondiente, lo que facilitó el proceso.

- Las posibilidades de NodeJS como plataforma, son bastantes amplias. Dentro de las ventajas están: es rápida, fiable, ocupa pocos recursos de hardware, permite mantener el código de servidor y cliente en el mismo lenguaje de programación, existe gran cantidad de extensiones y muy buena documentación. Se podría considerar la aplicación de NodeJS en el plano académico y como parte del aprendizaje de los estudiantes de la carrera de Computación e Informática.

- Es necesario contemplar todo tipo de limitaciones técnicas antes de iniciar un proyecto de este tipo. Durante el desarrollo del presente proyecto, no se tomó en cuenta que, en un ambiente de ejecución real, el tiempo de latencia y la conexión a Internet podría afectar el desempeño del sistema. Lastimosamente, fue hasta la etapa de pruebas, que este tipo de problemas aparecieron.

Trabajo futuro

- El proceso de optimización de transferencia de datos entre las instancias de los modelos es una tarea que nunca debería acabar, buscando siempre que se transfiera la menor cantidad de datos posible, minimizando así la cantidad de ancho de banda necesaria y los recursos de hardware requeridos.

- Se debe optimizar la interfaz gráfica, para que sea 100% soportada por dispositivos móviles.

- Minimizar el riesgo de un posible impacto tras la actualización de Galapagos, realizando algún tipo de validación que permita verificar que los cambios implementados no afectan la lógica de NodeTortoise.

Bibliografía

- Cough, Josh. 2011. NetLogo/HubNet-Web-Extension. Recuperado el 7 de octubre de 2014 de http://github.com/NetLogo/HubNet-Web-Extension

- Bertsche, Jason. (2015, Mayo 4). Tortoise: where are the models compiled to JS [Mensaje en Foro]. Recuperado de https://groups.google.com/forum/#!topic/netlogo-devel/8lLk8qfq3hw

- Bertsche, Jason. 2012. NetLogo/HubNet-Web-Extension. Recuperado el 28 de octubre de 2014 de https://github.com/NetLogo/Galapagos/wiki

- Bertsche, Jason. 2015. NetLogo/HubNet-Web-Extension. Recuperado el 26 de junio de 2015 de https://github.com/NetLogo/NetLogo/wiki/NetLogo-Web

- Driver, Mark. 2014. Introducing the Gartner Programming Language Index for 2014. Recuperado el 26 de junio de 2015 de http://blogs.gartner.com/mark_driver/2014/10/02/gartner-programming-language-index-for-2014/

- San Francisco State University. Laboratorio 1 y 2 - Iniciando con NetLogo. Recuperado el 8 de noviembre de 2014 de http://online.sfsu.edu/jjohnson/NetlogoTranslation/laboratorio_0102.html

- The Center for Connected Learning (CCL) and Computer-Based Modeling. HubNet Guide. Recuperado el 28 de octubre de 2014 de http://ccl.northwestern.edu/netlogo/hubnet.html

- The Center for Connected Learning (CCL) and Computer-Based Modeling. NetLogo. Recuperado el 28 de octubre de 2014 de https://ccl.northwestern.edu/netlogo/

- Turner, Charlie. 2013. cjt8109/hubnet-web-extension. Recuperado el 7 de octubre de 2014 de http://github.com/cjt8109/hubnet-web-extension

- Wikipedia. Aplicación web. Recuperado el 28 de octubre de 2014 de http://es.wikipedia.org/wiki/Aplicaci%C3%B3n_web