CRECTEALC MÉXICO

Centro Regional de Enseñanza en Ciencia y Tecnología Espacial
para América Latina y el Caribe
Campus México

Proyecto de aplicación

Sistema multiagente para la simulación de desastres

Asesor(es)

PhD. Jesús A. González Bernal

PhD. Enrique Muñoz de Cote

Por

Lic. Francisco Vera Voronisky

Lic. Reynel Remón Martínez

Tonantzintla, Puebla, México, Agosto 2012

Índice de temas

Índice de imágenes

I Introducción

La modelación y simulación basada en agentes es un enfoque relativamente nuevo para generar sistemas complejos consistentes de agentes autónomos los cuales interactúan entre ellos.

Un modelo típico basado en agentes cuenta con tres elementos:

1. Un conjunto de agentes, sus atributos y su comportamiento.
2. Un conjunto de relaciones entre agentes y métodos de interacción
3. El ambiente del agente: Los agentes interactúan con su ambiente adicionalmente a la interacción entre los propios agentes.

El desarrollador debe identificar, modelar y programar estos elementos para crear un modelo basado en agentes.

Nuestro proyecto se enfocará en desarrollar extensiones a la plataforma Repast y al proyecto de simulación RepastCity.

Repast es una plataforma de simulación basada en agentes la cual integra datos GIS directamente dentro de su simulación sin necesidad de usar librerías externas.

Haciendo referencia a los elementos de un modelo basado en agentes, nuestro trabajo creará una plataforma de simulación basada en agentes la cual:

- permite crear agentes que sean capaces de actuar ante un fenómeno catastrófico y simular desastres con un comportamiento.
- define un modelo de comunicación entre agentes.
- crea un ambiente (ContextManager) el cual actuará como árbitro validando los movimientos de los agentes.

II Descripción del problema

Planteamiento del problema

Se desea crear una plataforma de simulaciones de desastres naturales mediante un software de simulación basado en agentes y observar el comportamiento de los diferentes actores que participan en la simulación.

Justificación

En el comportamiento de un fenómeno catastrófico pueden influir diversos factores que hacen que el comportamiento de estossea difícil de predecir. Estudiar el comportamiento en un escenario real es casi imposible, ya sea por los costos que conlleve o porque no es posible recrear el fenómeno en sí. El uso de agentes y de plataformas de software para simular fenómenos es una variante viable para realizar estudios de fenómenos catastróficos.

Objetivos

Obtener una plataforma que permita realizar la simulación de desastres utilizando agentes.

Alcance

Se creará una plataforma de simulación mediante agentes la cual permita la incorporación de diversos tipos de agentes (bomberos, ambulancias, etc.) y la creación de estrategias de comportamiento de estos agentes para modelar un fenómeno catastrófico específico. Esta plataforma debe permitir la incorporación de información geográfica contenida en archivos shapefile que represente el mundo sobre el cual los agentes deben de actuar.

Esta plataforma podrá utilizarse posteriormente con fines de docencia como aplicación práctica de uso de agentes en la simulación de un fenómeno catastrófico en particular. Además, esta plataforma puede ser utilizada en un ambiente científico ya que proporciona una herramienta que permite

enfocarse más en el desarrollo de estrategias y comportamientode los agentes que en el desarrollo de detalles de los movimientos de los mismos.

III Solución propuesta

Creación de una estrategia multiagente en la cual éstos sean capaces de realizar una acción en respuesta a un desastre determinado. Para esto se asimilará la tecnología presente en la plataforma Repast y RepastCity realizando modificaciones en esta última para cambiar su diseño e incorporar nuevas funcionalidades.

Etapas

1. Estudio de plataformas de simulación basadas en agentes.
   Se precederá a estudiar la plataforma Repast Simphony y RepastCity sobre la cual se realizarán modificaciones para el desarrollo de un sistema de simulación de desastres basados en agentes.
2. Rediseñar un Manejador de Contexto (ContextManager) que sea el encargado de controlar, regular y validar el comportamiento y acciones de los agentes de la plataforma.
   En el diseño original del Repast, los agentes tienen la libertad de ser ellos los que realicen las acciones y movimientos, sin que exista una entidad que controle la validez de sus acciones. Se procederá a la modificación de un Manejador de Contexto, el cual se encargará de validar las acciones y movimientos de los agentes.
3. Desarrollar un modelo de comunicación para permitir la comunicación entre los agentes.
   Se desarrollarán los siguientes modelos de pasos de mensajes entre los agentes:
   - Broadcast: se envía un mensaje a todos los agentes.
   - Multicast: se envía un mensaje solo a los agentes del mismo tipo.
   - Peer to Peer: se envía un mensaje a un agente en específico.
4. Extensión del proyecto RepastCity para permitir no solo el movimiento de agentes sobre una ciudad o sobre una red de caminos, sino a permitir el movimiento de agentes en un entorno rural.

En RepastCity, el movimiento de los agentes está restringido a estar sobre una red de vías. Nosotros pretendemos que el agente pueda salir de la ciudad y luego moverse libremente fuera de la ciudad sin que éste tenga que estar sobre un camino. El agente deberá ser capaz de detectar los obstáculos en su movimiento y modificar su trayectoria para evadir dichos obstáculos.

5. Definir el comportamiento de un desastre.
   Se implementará un desastre como un agente más, ya que en Repast todos los actores que intervienen en la simulación son modelados como agentes.
6. Realizar simulación.
   En este paso se observará el comportamiento de los diversos agentes que participan en la simulación.

IV Estado del arte

En esta sección mostraremos algunas plataformas ABMS (Agent Based Modeling and Simulation). La mayoría de las plataformas siguen el paradigma ‘framework y librería’, proveyendo un framework, un conjunto de conceptos estándar para diseñar y describir ABMs, junto con una librería de clases y un ambiente de simulación.

La primera de estas plataformas fue Swarm (enjambre), las librerías de ésta fueron escritas en Objective-C. Java Swarm es un conjunto de clases de Java que permiten el uso de las librerías Objective-C de Swarm desde Java. Repast comenzó una implementación de Swarm en Java pero ha cambiado significativamente a través del tiempo. Más recientemente, han sido desarrollados: MASON en la plataforma Java y Ecolab en la plataforma C++. Otros frameworks como meta ABM proveen un entorno de diseño visual para el modelado basado en agentes.

Estas plataformas han sido exitosas porque proveen una colección estandarizada de herramientas, pero también tiene varias limitaciones. Entre las debilidades podemos destacar que no son fáciles de usar, hay una insuficiencia de herramientas para construir modelos.Especialmente herramientas para representar datos espaciales, insuficientes herramientas para ejecutar y observar los experimentos de la simulación y una documentación pobre[Robert and Carole (2004)].

[Allan, et al. (2010)]realizaron un trabajo donde describe las principales plataformas de simluación ABMS que serán mencionadas a continuación.

AgentSheets

AgentSheets es una herramienta ABMS propietaria la cual está basada en uso de hoja de cálculo. En lugar de que las celdas de la malla de la hoja de

cálculo contengan números, están ocupadas por agentes. La simulación entonces toma lugar en la malla en la cual están los agentes.

AgentSheets está enfocada para usuarios no programadores y es muy simple de usar. Ha sido utilizadapara la enseñanza en estudios sociales y matemáticas. Usa un paradigma de programación visual por lo que no hay un código en texto y toda la programación es realizada mediante una interfaz gráfica. Tiene como desventaja que cuando el modelo de simulación se vuelve complejo aparecen las limitaciones: un agente no puede enviar mensajes a otro agente, un agente no puede modificar los atributos de otro agente.

AndroMeta

AndroMeta, formalmente DeX, Dynamic Experimentation Toolkit, es una plataforma orientada a objetos escrita en C++ para el desarrollo, análisis y visualización de agentes. DeX es un simulador de eventos discretos construido sobre un motor de alto rendimiento de simulación diseñado para manejar un gran número de entidades con alto nivel de comunicación. DeX incluye herramientas para la ejecución en lotes (batch), optimización, tareas distribuidas, controles visuales y ploteo en tiempo real y visualización 3D usando OpenGL. AndroMeta incluye una interfaz de diseño con un lenguaje de alto nivel. El software está disponible libremente para ser descargado en este enlace http://andrometa.net y puede ser ejecutado sobre Linux y Mac OS.

AnyLogic

AnyLogic es un software propietario que incorpora un rango de funcionalidades (eventos discretos y sistemas dinámicos basados en agentes) para el desarrollo de modelos basados en agentes. Por ejemplo, los modelos pueden leer y escribir dinámicamente datos a hojas de cálculo o

bases de datos durante la ejecución de la simulación así como dar como salida un gráfico del modelo. Además, es posible iniciar programas externos dentro de un modelo AnyLogic para la comunicación dinámica. Los modelos AnyLogic solo pueden ser creados sobre el sistema operativo Windows, aunque la simulación puede ser ejecutada usando la plataforma Java sobre cualquier sistema operativo que sea compatible con ésta una vez compilado. El sitio web de AnyLogic http://www.xjtek.com muestra varios ejemplos de modelos que han sido desarrollados para un rango diverso de aplicaciones incluyendo estudios sociales, dinámica de ecosistemas, planeamiento de sistemas de salud, redes de computadoras y telecomunicación, y la ubicación de servicios de emergencias. Sin embargo, los códigos fuente y la documentación de estos modelos no están disponibles.

Ascape

Ascape es una plataforma para el desarrollo y análisis mediante modelos basados en agentes, el cual ha sido desarrollado por Miles Parker del Institute Center on Social and Economics Dynamics.

Ascape sigue algunas ideas de Swarm, por ejemplo, los agentes existen dentro de escenarios, los cuales pueden ser tratados a su vez como agentes. Sin embargo, es más fácil desarrollar modelos dentro de Ascape que en Swarm. De hecho, su intención es que personas con poca experiencia en programación desarrolle situaciones complejas proveyendo un rango de herramientas para el usuario.Por ejemplo, herramientas para la recolección de estadísticas en la ejecución de la simulación, herramientas para la creación de grafos, etc.

Ascape está disponible en los siguientes enlaces: http://ascape.sourceforge.net y http://www.brook.edu/es/dynamics/models/ascape.

Breve

Breve es un software libre usado como una herramienta de enseñanza que provee un ambiente 3D para la simulación de sistemas de-centralizados y vida artificial. El usuario define el comportamiento de los agentes en un mundo 3D y observa como estos interactúan. Breve incluye un lenguaje de script basado en Python, simulación física y detección de colisiones para la simulación de criaturas realistas y el motor OpenGL para la visualización. Está disponible para las plataformas Mac, Linux y Windows. Disponible en el enlace http://www.spiderland.org.

Cormas

Cormas es una plataforma de simulación basada en el ambiente de programación VisualWorks, el cual permite el desarrollo de aplicaciones en el lenguaje Orientado a Objetos SmallTalk. Las entidades Cormas predefinidas son representadas como clases genéricas de SmallTalk de las cuales, por especialización y refinamiento, los usuarios pueden crear entidades específicas para su propio modelo.

Cormas ha sido aplicado al manejo de recursos naturales, es decir, en el estudio de la interacción de las sociedades humanas con los ecosistemas terrestres. Disponible en http://cormas.cirad.fr.

DEVS: Discrete Event System Specification

DEVS es un formalismo de B. P. Zeigler del Arizona Center for Integrative Modeling and Simulation. DEVS es una extensión de las máquinas de estado finito de Moore. Un número de herramientas de varios autores han sido diseñadas usando DEVS. Veáse http://www.acims.arizona.edu/SOFTWARE/software.shtml. El formalismo DEVS y sus variaciones han sido usados en muchas aplicaciones de

ingeniería, como diseño de hardware, diseño hardware-software, sistemas de comunicación, sistemas de manufacturas y en ciencias como la biología y sociología.

EcoLab

EcoLab es un ambiente de simulación orientado a objetos. Éste provee una serie de instrumentos que pueden ser acoplados con el modelo del usuario.Está escrito en C++, en tiempo de ejecución para visualizar el modelo, así como para dar soporte para distribuir agentes sobre una topología arbitraria de grafos, particionado sobre múltiples procesadores y soporte a reinicio de la simulación. EcoLab fue originalmente desarrollado para simular un modelo en particular – el modelo EcoLab de una ecología abstracta. Sin embargo, varios modelos diferentes han sido implementados usando el software, demostrando su naturaleza de propósito general.

EcoLab es desarrollado y mantenido por Russell Standish en University of Sydney, Australia, http://ecolab.sourceforge.net. Es semejante a Swarm, pero escrito completamente en C++ en lugar de Objective C.

FLAME: FLexible Agent Modelling Enviroment

FLAME es desarrollado principalmente por University of Sheffield (véase http://www.flame.ac.uk y http://www.flamegpu.com). FLAME ha sido desarrollado para permitir un rango de modelos de agentes y modelos sin agentes para ser llevados juntos a un solo ambiente de simulación. Su objetivo principal está en los dominios médicos y biológicos, por ejemplo, estudio de cultivos.

FLAME provee una especificación en forma de una plataforma formal, la cual puede ser usada por desarrolladores para crear modelos y herramientas que sean compatibles entre ellas. Los modelos nuevos, hechos bajo las

especificaciones, puedes ser incorporados fácilmente a los existentes, o las nuevas simulaciones, sin mucho esfuerzo. Métodos de paralelización usando MPI y técnicas de pruebas, permiten el desarrollo de simulaciones multiprocesos.

La metodología FLAME está basada en la definición de agentes como X-Machines, las cuales son extensiones de las máquinas de estados finitos con adición de memoria. Éstas leen los datos de una pizarra de mensajes y cambian de estado de acuerdo a las reglas codificadas a las funciones asociadas.

No existen restricciones del tipo de simulación que puedan ser adjuntadas. A pesar que el framework está diseñado alrededor de modelado basado en agentes, no es un requerimiento que los agentes sean incluidos en la simulación. Admitela incorporación de herramientas externas, permitiendo a los desarrolladores concentrarse en el desarrollo de los modelos y no re-inventando las herramientas.

La implementación paralela de FLAME usa la tecnología de pizarra de mensajes para la comunicación entre agentes. Ésta está basada en libmboard (http://www.softeng.rl.ac.uk/st/projects/libmboard).

FLAME no es de por si un simulador, pero las herramientas adheridas siguiendo las especificaciones del framework crean los paquetes de simulación requeridos a través de un proceso de compilación.

Los puntos clave de FLAME son:

- FLAME es un ambiente de modelación que permite una modelación de rendimiento alto basada en agentes en arquitecturas paralelas.

- Los modeladores no requieren conocimientos de especialistas de la arquitectura subyacente usada para la simulación, dado que los modelos son diseñados usando técnicas de especificación formal.
- Algoritmos eficientes para la comunicación inter-agente aseguran un rendimiento alto en la simulación.
- El sistema está basado en una sintaxis XML, la cual puede ser fácilmente entendible.

El sitio web brinda información de como usar FLAME basado en un par de ejemplos y también brinda acceso a varias herramientas compatibles con FLAME. Modelos compatibles para su uso dentro de esta plataforma también pueden ser descargados y luego modificados para su uso. El xparser convierte un archivo XML que contiene una especificación de un modelo FLAME usando plantillas a un código C, el cual puede ser compilado usando los makefile provistos. Las dependencias funcionales son representadas a través de grafos.

JAS: Java Agent Based Simulation Library

JAS es un proyecto italiano para desarrollar una herramienta específicamente diseñada para la simulación basada en agentes y modelación. JAS es un clon en Java de la librería de Swarm. El núcleo de la herramienta JAS es un motor de simulación basado en la simulación de eventos discretos, lo cual permite que el tiempo sea manejado con alta precisión y desde una perspectiva multi-escala. Varios aspectos de JAS están basados en librerías de terceros de código abierto. JAS es una herramienta libre disponible en http://jaslibrary.sourceforge.net.

LSD: Laboratory for Simulation Developmet

Éstees otro proyecto italiano activocon una plataforma escrita por Marco Valente de University of l'Aquila en C++[Andersen and Esben (2002)].Está

disponible en http://www.labsimdev.org/Joomla_1-3. Se ha enfocado en modelos económicos y en ciencias sociales. Las aplicaciones LSD toman un modelo de sistema dinámico (ecuaciones diferenciales) utilizando la dinámica de réplica en lugar de un enfoque ascendente basado en agentes, pero el uso de C++ sugiere que un enfoque más basado en agentes es posible.

MAML: Multi-Agent Modelling Language

El lenguaje MAML y el compilador xmc fueron desarrollados por Complex Adaptive Systems Laboratory de la Central European University en Hungría entre 1998 y 1999. Desde entonces, varias mejoras y parches han sido programados por Agent Lab Intelligent Systems, Research, Design, Development and Consulting Ltd. (http://www.aitia.ai/eng).

El lenguaje MAML, lenguaje orientado a aspectos, fue inicialmente desarrollado para ayudar a los estudiantes de ciencias sociales con conocimientos limitados de programación para crear modelos basados en agentes de forma rápida. La meta final del proyecto fue desarrollar una ambiente fácil de usar, con una interfaz gráfica. Sin embargo, la versión actual de MAML, es un lenguaje de programación y no un ambiente.

Como comentario sobre lenguajes orientados a aspectos, señalamos que en los lenguajes “tradicionales” (ya sea orientado a objetos o procedurales), seimplementan las funciones principales mediante objetos, métodos oprocedimientos. Los lenguajes orientados a aspectos agregan una nueva“dimensión”, encargada de concentrar la programación de conceptos tales comopersistencia, gestión de errores, registros de auditoria, sincronización,comunicación entre procesos, o, en forma más general, lo que se ha definidopreviamente como aspectos.

MAML de hecho estábasado sobre Swarm e intenta hacer a Swarm más fácil de usar proveyendo palabras claves que definan la estructura del simulador y el acceso a las librerías de Swarm. MAML trabaja a un nivel de abstracción más alto que Swarm. Sin embargo, además de aprender Swarm, el desarrollador necesita conocer Objective-C y Swarm, lo cual es una limitante para programadores con poca experiencia. De hecho, programadores experimentados prefieren agregar funcionalidades a Swarm. Programar en Swarm requiere que el desarrollador cree archivos de texto usando un editor ya que no existe una interfaz gráfica. Debido a que MAML accede a las librerías de Swarm, la interfaz del modelo de simulación desarrollado es muy similar a uno que hubiera sido desarrollado en Swarm.

El código escrito en MAML es convertido a una aplicación Swarm mediante el compilador MAML (llamado xmc). La aplicación resultante es luego compilada de la misma manera que un código normal Swarm mediante gcc. Actualmente xmc solo corre bajo el sistema operativo Linux.

Alguna documentación de MAML es provista: un tutorial con código fuente, un manual de referencia y un manual técnico, pero es necesario además conocimiento de Swarm. La página de MAML contiene algunos ejemplos simples de simulaciones comunes. Desafortunadamente, fuera de la página oficial de MAML existen muy pocas páginas que den ejemplos usando MAML.

MATSim

MATSim es una herramienta de simulación de agentes de sistema de transporte desarrollado en TU Berlin y EHT Zürich. Está enfocado en la simulación de manejo de flujo de tráfico y planificación urbana. También ha sido usado para escenarios de evacuación. MATSim usa un enfoque

modular siendo fácilmente personalizable. Existe un tutorial, una guía de usuario, FAQ y una lista de correos y brinda soporte a sistemas multi núcleos, véase http://www.matsim.org. Esta herramienta se puede descragar desde http://sourceforge.net/projects/matsim/files.

MASON: Multi-Agent Simulation of Neighborhoods

MASON fue diseñado como una alternativa más pequeña y rápida que Repast, enfocado a demandas computacionales de los modelos con muchos agentes ejecutados sobre varias iteraciones. Los desarrolladores de MASON permitieron el uso de herramientas de propósito general en lugar de herramientas de un dominio específico. Los modelos del núcleo se ejecutan independientemente de la visualización, la cual puede ser agregada. Se proporcionan la posibilidad de crear puntos de control de la simulación de los modelos y la posibilidad de migración. De hecho, la capacidad de separar y conectar la interfaz gráfica, parar la simulación y moverla entre computadoras es considerada como una prioridad para simulaciones largas.

En resumen, MASON es rápido, fácil de extender, simulación multi agente de eventos discretos en Java. Está diseñado para servir de base para un amplio rango de tareas multiagentes que va desde la robótica de enjambres (swarm robotics) a aprendizaje de máquinas para ambientes sociales complejos.

MASON contiene una librería de modelos y una suite opcional de visualización en 2D y 3D. Este sistema es de código abierto y es un desarrollo del Departamento de Ciencias de la Computación de George Mason University. MASON se puede descargar desde http://cs.gmu.edu/~eclab/projects/mason.

MASS: Multi-Agent Simulation Suite

MASS consiste en tres aplicaciones que ofrecen solución para aspectos diferentes de la modelación. Cada aplicación es desarrollada con la intención de proveer herramientas profesionales para programadores inexpertos. El software ofrece interfaces de usuario amigables para escribir los modelos, crear visualizaciones y análisis de simulación de datos.

El lenguaje relacionado Agent Based Languaje for Simulation (FABLES), es un lenguaje fácil de usar, especialmente diseñado para crear simulaciones basadas en agentes. Requiere pocas habilidades de programación y tiene un rango amplio de funciones que hacen que sea un lenguaje fácil de usar. En la primera publicación de MASS, el núcleo de la simulación es Repast-J, lo que significa que todos los modelos de FABLES son compilados con Repast-J. Modelos NetLogo también pueden ser ejecutados.

La extensión Participatory Extension (PET), es un ambiente Web para crear, administrar y participación en simulaciones basadas en agentes. El uso de PET se basa en mecanismos familiares a los usuarios que navegan en páginas Web.

El Módulo de Exploración de Modelos, Model Exploration Module (MEME), es una herramienta que permite organizar los experimentos, manejarlos y analizar los resultados. Permite a los usuarios ejecutar simulaciones por lotes con varios parámetros de configuración, almacenar, manejar y analizar los datos.

Estas herramientas pueden ser descargadas desde el sitio web http://mass.aitia.ai y pueden ser ejecutadas sobre sistemas Windows, Linux o Mac.

MetaABM

Es un sistema de modelación ABM desarrollado en Eclipse, véase http://www.metascapeabm.com. MetaABM define y soporta una arquitectura de alto nivel para el diseño, ejecución y estudios sistemáticos de modelos ABM. Comenzó como un componente del sistema Repast Simphony, pero su ámbito va más allá de una herramienta ABM. MetaABM no es un motor ABM o un ambiente de ejecución, pero como enfoque puede aprovechar estos ambientes en formas múltiples. Más allá, metaABM busca proveer un centro común que permita a los desarrolladores de herramientas ABM para evitar la duplicación de esfuerzos y centrarse en el valor que puede agregar al software en general. Los contribuidores están comprometidos con un desarrollo abierto.

MIMOSE

MIMOSE consiste en un lenguaje de descripción de modelo y un entorno de experimentación para la simulación de un modelo. El propósito principal del proyecto MIMOSE ha sido el desarrollo de un lenguaje que considera demandas especiales en la modelación en ciencias sociales, especialmente en la descripción de relaciones cuantitativas y cualitativas, influencias estocásticas, procesos de nacimiento y muerte, y modelos micro y multi nivel. El objetivo de MIMOSE es que al describir los modelos el modelador no se vea afectado con detalles de programación.

MIMOSE fue creado por Michael Möhring de la University of Koblenz-Landau, Alemania. Se requiere para su ejecución Sun Sparc, SunOS, Solaris, X11R5/6 or Linux, véase http://www.uni-koblenz.de/~moeh/projekte/mimose.html.

MobiDyc: Modélisation Basée sur les Individus pour la Dynamique des Communautés

Mobidyc es un proyecto que enfoca los ABMS en el campo de la ecología, biología y medio ambiente. Permite crear modelos y ejecutarlos sin necesidad de tener habilidades de programación. El software está escrito en Smalltalk. Varias restricciones son notadas:

- El espacio y tiempo discreto puede limitar la opción de pasos de tiempo, mallas y comunicación entre agentes.
- Precisión numérica
- Cálculos lentos limitando el tamaño cerca de 10000 celdas y 10000 agentes.

Esto sugiere que MobiDyc es apropiado por la enseñanza y construcción de modelos iniciales, pero limitado para simulaciones realísticas. Véase http://w3.avignon.inra.fr/mobidyc.

Modelling4all

Modelling4all es un proyecto de la Universidad de Oxford con soporte por Eduserv y JISC. Es posible crear y ejecutar un modelo de comportamiento simplemente desde un navegador Web. Usuarios no expertos puede crearcomponentes pre-construidos llamados micro-behaviors apoyando así una forma constructivista de aprendizaje. Modelos colaborativos también son soportados. La plataforma computacional subyacente es NetLogo. BehaviouralComposer está dispoible como código abierto bajo la licencia New BSD, véase http://modelling4all.nsms.ox.ac.uk.

NetLogo

NetLogo, (originalmente llamado StarLogoT), es una plataforma de alto nivel, la cual provee un lenguaje de programación simple y poderoso, interfaz gráfica incorporada y documentación comprensible. Los

modeladores pueden dar instrucciones a cientos o miles de agentes independientes, los cuales operan concurrentemente.

NetLogo refleja su herencia de StarLogo como una herramienta educacional, ya que su objetivo principal es hacer su uso fácil. Su lenguaje de programación incluye varias estructuras primitivas de alto nivel que reducen el esfuerzo de programación. Este lenguaje está basado en Logo, un dialecto de Lisp y contiene varias, pero no todas las estructuras de control del lenguaje estándar. NetLogo fue diseñado para un modelo específico – agentes móviles que interactúan concurrentemente en una rejilla dominados por interacciones locales en un corto tiempo. Si bien los modelos de este tipo son fáciles de implementar en NetLogo, la plataforma no está limitada a ellos.

NetLogo tiene una documentación extensa y tutoriales. También viene con una librería de modelos, la cual es una colección de modelos pre-escritos que pueden ser usados y modificados. Estas simulaciones están enfocadas a varias áreas de dominio en las ciencias naturales y sociales, incluyendo la biología y medicina, física y química, matemáticas y ciencias computacionales, economía y psicología social. Está disponible en http://ccl.northwestern.edu/netlogo.

Open StarLogo

StarLogo es un lenguaje de simulación basado en agentes desarrollado por Resnick, Klopfer y otros en el MIT Media Lab y MIT Teacher Education Program. Es una extensión del lenguaje de programación Logo. StarLogo fuediseñado para la educación y puede ser usado por los estudiantes para modelar el comportamiento de sistemas decentralizados.

StarLogo también está disponible en una versión llamada Open StarLogo desde junio del 2006. El código fuente de Open StarLogo está disponible en línea(http://education.mit.edu/openstarlogo).

RePast: Recursive Porous Agent Simulation Toolkit

RePast fue desarrollado en University of Chicago's Social Science Research Computing Lab específicamente para crear simulaciones de agentes en ciencias sociales [North, et al. (2006)]. Es muy semejante a Swarm, tanto en filosofía como en apariencia y similarmente provee una librería de códigos para crear, ejecutar, mostrar y recolectar datos de las simulaciones. De hecho, fue inicialmente una codificación en Java de Swarm, véase http://repast.sourceforge.net/repast_3.

El desarrollo de RePast parece haber sido impulsadopor varios objetivos. RePast no adoptó toda la filosofía de Swarm y de hecho no implementa swarms. RePast fue creado para soportar un dominio en particular, ciencias sociales, e incluye herramientas específicas a ese dominio. El objetivo adicional deconstruir más fácil los modelos para usuarios inexpertos es llevado a cabo por el proyecto Repast mediante varios enfoques. Estos enfoques incluyen un modelo simple built-in e interfaces a través de menús y se puede utilizar código Python para iniciar la construcción de un modelo.

RePast es una herramienta basad en Java y desarrollar una simulación requiere que se tenga habilidades en Java. RePast provee algunos modelos de simulación conocidos como SugarScape, Swarm's Heatbugs y MouseTrap.

Repast Simphony

Desde 1994, ha habido un avance constante de las herramientas de software y entornos de desarrollo que han reemplazado a Swarm. Repast-3 ha sido recientemente sustituido por un desarrollo significativo de nombre Repast Simphony, o Repast-S. Esta es una herramienta libre de código abierto desarrollada en Argonne National Laboratory. Contiene una herramienta para un desarrollo visual del modelo, ejecución visual del modelo, conexión automática a base de datos, salida automática a ficheros de registros (log files) y visualización de resultados, véase http://repast.sourceforge.net.

Si bien Repast-J, Repast.Net y Repast-Py reciben mantenimiento, estos ya han alcanzado la madurez y su desarrollo ya no continúa. Ellos han sido sustituidos por Repast Simphony (Repast-S), que proporciona toda la funcionalidad básica de Repast-J o Repast.Net, aunque se limita a la aplicación en Java. Repast-S fue lanzado como versión alfa a finales del 2006. Desde el punto de vista de usuario, las mejoras en Simphony comparadas con Repast-3 son:

- Una nueva interfaz gráfica de usuario para el desarrollo de modelos.
- El tiempo de ejecución de la interfaz gráfica de usuario ha mejorado.
- Adición de contextos y proyecciones.

Un contexto contiene una población de agentes pero no le brinda a los agentes ningún concepto de espacio o relación. Los contextos pueden ser ordenados jerárquicamente y contener subcontextos. Un agente en un subcontexto también existe en el contexto padre, pero en el caso inverso no tiene por quécumplirse.

Las proyecciones brindan a los agentes un espacio y pueden definir las relaciones entre ellos. Las proyecciones son creadas para contextos específicos y automáticamente cada agente en el contexto. Diferentes proyecciones pueden ser hechas para propiedades diferentes.

Repast-S probablemente tendrá la mayor funcionalidad que cualquier paquete ABMS. Soporta un amplio rango de herramientas externas para estadísticas, análisis de redes, visualización, minería de datos, hojas de cálculo, etc. Soportan modelos 2D y 3D. Los modelos pueden tener puntos de control escritos en varios formatos, incluyendo XML. El planificador de eventos discretos es concurrente y permite varios hilos. Están disponibles varias librerías numéricas, por ejemplo, para generación de números aleatorios y computación distribuida.

SimPack

Simpack, desarrollado por Paul Fishwick, es un directorio de herramientas diseñadas para la enseñanza y soporta simulación de eventos discretos, véase http://www.cise.ufl.edu/~fishwick/simpack.html. Simpack soporta una gran variedad de planificación de eventos y modelos de simulación continuos en tiempo. Hay modelos descritos en el libro de dicho autor [Fishwick (1995)] y ejemplos usando Procesamiento de lengujae natural, véase http://www.processing.org.

Simpack fue originalmente escrito en C y esta versión continúa disponible pero no es mantenida para actualizaciones. En las últimas versiones los modelos son ejecutados en Java. Simpack está disponible bajo la licencia GPL.

SimPy

Es un lenguaje de programación Orientado a Objetos basado en Python para la modelación basada en agentes, específicamente destinado para aplicaciones de ciencias sociales. Véase http://simpy.sourceforge.net y el blog de SimPy http://blog.gmane.org/gmane.comp.python.simpy.user. Hay un desarrollo activo y una comunidad de usuarios, http://pypi.python.org/pypi/SimPy/2.1.0.

SOARS: Spot Oriented Agent Role Simulator

SOARS del Tokyo Institute of Technology, es un proyecto en Java con un desarrollo activo, véase http://www.soars.jp.

En SOARS, un modelo está compuesto por un agente y 'puntos', siendo estos últimos un lugar para la interacción y brindar una descomposición espacial. Los eventos ocurren en pasos de tiempo regulares. Durante cada paso una serie de fases (descomposición temporal) son realizadas donde reglas de comportamiento son evaluadas. Una transición entre una fase debe ser independiente del orden delas reglasdentro de una fase. El ambiente consiste de componentes principales, incluyendo un lanzador, una consola de líneas de comandos, un simulador y un animador (visualizador). Las simulaciones pueden ser construidas en forma de juegos y ser ejecutadas en recursos distribuidos. Ejemplos de aplicaciones son una ciudad virtual, un modelo de mercado y actividades de oficina.

StarLogo

StarLogo es un modelo ambiente de modelación programable dirigido específicamente a explorar los sistemas descentralizados a través de la simulación. Es una versión especializada de Logo, la cual ha sido usada para la enseñanza. StarLogo permite al usuario crear y controlar el comportamientos de las 'turtles', un término usado en Logo. Las turtles se mueven dentro de un paisaje especificado por el usuario que está hecho de 'patches'. Los patches son similares a los 'puntos' en SOARS.

Si bien, se puede considerar StarLogo como un sistema basado en agentes, por ejemplo, un turtle, es un agente, el paradigma de programación es procedural, en lugar de Orientado a Objetos.

Proporciona un conjunto de comandos que el programador utiliza para crear y controlar a las turtles y los patches.

En la práctica, StarLogo es muy fácil de usar. Provee una interfaz gráfica para ayudar al desarrollo de simulaciones. Esta puede ser usada para crear gráficos de datos de la simulación y definir botones y barras deslizables con la cual controlar la simulación y definir los datos de entrada. Sin embargo, si bien es fácil graficar los datos existen algunos problemas asociados con la herramienta graficadora.

Hay unagran variedadde ejemplos de simulaciones de StarLogo disponibles en Internet y existe una lista de soporte de correos, véase http://www.media.mit.edu/starlogo. Una de las principales desventajas de StarLogo es su inflexibilidad. El conjunto de comandos ofrecidos por StarLogo es muy restrictivo si se desean realizar mecanismos sociales complejos.

Swarm

Swarm fue la primera herramienta de software reutilizable para la modelación basada en agentes y simulación. Fue desarrollado por Santa Fe Institute en 1994y fue específicamente diseñado para aplicaciones de vida artificial y estudios de complejidad.

Swarm fue originalmente desarrollado para la simulación multi-agente de sistemas adaptativos complejos. Hasta hace poco, el proyecto aún tenía su sede en Santa Fe Institute, pero su desarrollo y la gestión está ahora bajo control de Swarm Development Group, el cual tiene mayor número de miembros para mantener el software, véase http://www.swarm.org.

Swarm fue diseñado como un lenguaje general y como herramienta para ABMS, destinado para su uso generalizado en todos los dominios

científicos. Los desarrolladores iniciaron con un enfoque conceptual general de software de simulación basado en agentes. La clave de Swarm es el concepto de que el software debe implementar un modelo y, separadamente, proveer un laboratorio virtual para la observación y la realización de experimentos en el modelo. Otro concepto clave es el diseño de un modelo como una jerarquía de 'enjambres' (swarm). Un enjambre es un grupo de objetos y un planificador de acciones que el objeto ejecute. Esto es similar a los conceptos de contexto y proyección incluidos en Repast Simphony. Un enjambre puede contener enjambres a más bajo nivel, cuyos planificadores están integrados a enjambres de más alto nivel; modelos simples tienen un nivel menor en el 'modelo de enjambre' dentro de un 'enjambre observador' el cual adjunta herramientas de observación al modelo.

La filosofía de diseño esincluirsoftware que implementenconceptos de modelos Swarm,junto con herramientas generales que puedan ser útiles para varios modelos, pero no incluye herramientas específicas para un dominio particular.

Swarm fue diseñado antes de la aparición de Java como lenguaje. Una de las razones para la implementación de Swarm en Objective-C es que la falta de tipado fuerte en este idioma (a diferencia por ejemplo de C++), apoya la filosofía de los sistemas complejos de falta de control centralizado. Esto significa que el planificador de un modelo puede decirle a una lista de objetos que ejecuten una acción sin la necesidad de conocer el tipo de objeto que está en la lista. Swarm utiliza sus propias estructuras de datos y la gestión de memoria para representar los objetos del modelo. Una consecuencia es que Swarm es capaz de aplicar plenamente el concepto de 'probes' – herramientas que permiten a los usuarios monitorear y controlar cualquier objeto de simulación, no importa como esté protegido desde la interfaz gráfica o dentro del código.

Swarm proporciona un conjunto de bibliotecas que el desarrollador utiliza para la construcción de modelos y análisis, la visualización y el control de experimentos con esos modelos. Dado que las bibliotecas se han escrito en Objective-C, hasta hace poco la construcción de un simulador implicaba programar en una mezcla de Objective-C y Swarm. Sin embargo, ahora es posible usar Java con algunas llamadas a librerías Swarm. Java Swarm fue diseñado para proporcionar, con los menores cambios posibles, acceso a la librería de Swarm escrita en Objective-C desde Java. Esto fue motivado por una fuerte demanda entre los usuarios de Swarm para poder escribir modelos en Java. Java Swarm simplemente permite que Java envíe mensajes a la biblioteca de Objective-C con un procesamiento para acomodar el tipado fuerte de Java.

En el sistema Swarm el componente fundamental que organiza los agentes de un modelo de Swarm es un 'enjambre'. Un enjambre es una colección de agentes con un planificador de eventos sobre los agentes. El enjambre representa un modelo completo: contiene los agentes, así como la representación del tiempo. Swarm admite el modelado jerárquico mediante el cual puede ser un agente compuesto por enjambres de otros agentes en estructuras anidadas. En este caso, el comportamiento del agente de nivel superior está definido por los fenómenos emergentes de los agentes dentro de su enjambre. Este modelo de enfoque de múltiples niveles que ofrece Swarm es muy poderoso. Enjambres múltiples pueden ser utilizados para modelar agentes y que ellos mismos construyan modelos de su mundo. En Swarm, los agentes pueden contener sus propios enjambres, modelo que un agente construye para sí mismo para entender su propio mundo.

El modelo actual y la tarea de observar el modelo están claramente separados en el sistema Swarm. Existen agentes especiales 'observadores' cuyo propósito es tener en cuenta otros objetos a través

de la interfaz probe. Estos objetos pueden proporcionar en tiempo real la presentación de datos y el almacenamiento de datos para posterior análisis. Los agentes de observadores en realidad son enjambres como se señaló anteriormente. Combinando el enjambre de observador con el enjambre del modelo proporciona una plataforma de experimentación completa. Con otras herramientas de simulación la distinción entre el modelo actual y la necesidad de observar y recolectar datos del modelo es algo borrosa, resultando difícil de poder modificar una parte sin afectar a otra. Separar el modelo de su observador es un patrón de programación que permite al modelo mismo a permanecer inalterado si el código del observador es modificado.

Swarm probablemente continúe siendo la plataforma de simulación más potente y flexible. Sin embargo tiene un precio. En la práctica Swarm tiene una curva de aprendizaje muy empinada. Es necesario contar con experiencia de Objective-C y, posiblemente de Java, estar familiarizado con la metodología de programación Orientadoa Objetos y ser capaz de aprender algo de código de Swarm.

Swarm normalmente se ejecuta en máquinas Linux con GNU Objective-C y X-Windows El código fuente está disponible libremente bajo una licencia GNU.

Para el desarrollo de nuestro proyecto, hemos seleccionado la plataforma Repast Simphony por las siguientes razones:

- Es una plataforma de código abierto por lo que permite que sea modificada por nosotros.
- Es una plataforma escrita en lenguaje Java lo que permite que el proyecto se pueda ejecutar en diversos sistemas operativos: Linux, Mac OS, Windows.

- Incorpora el manejo de archivos shape de forma nativa sin necesidad de buscar herramientas adicionales para trabajar con este tipo de ficheros.
- Proporciona un visualizador integrado a la plataforma en el cual se puede observar la simulación de los agentes.
- Existencia del proyecto RepastCity como extensión de la plataforma Simphony, que nos permite enfocarnos más en diseñar la estrategia del agente que en detalles de como está implementado la comunicación con los archivos shapefile.

V Conceptos básicos

ABMS

Agent Based Modelling and Simulation es una técnica computacional que tiene sus orígenes en la simulación de eventos discretos, algoritmos genéticos y autómatas celulares. Es una técnica poderosa para simular sistemas dinámicos complejos y observar comportamientos emergentes.

Agente (inteligencia artificial)

Es una entidad capaz de percibir su entorno, procesar tales percepciones y responder o actuar en su entorno de manera racional, es decir, de manera correcta y tendiendo a maximizar un resultado esperado.

Archivo shapefile

Es un formato de archivo informático propietario de datos espaciales desarrollado por la compañía ESRI, quien crea y comercializa Sistemas de Información Geográfica como Arc/Info o ArcGIS. Originalmente se creó para la utilización con su producto ArcView GIS, pero actualmente se ha convertido en el formato estándar de facto para el intercambio de información geográfica entre Sistemas de Información Geográfica. Un shapefile es un formato vectorial de almacenamiento digital donde se guarda la localización de los elementos geográficos y los atributos asociados a ellos.

Framework

La palabra inglesa "framework" define, en términos generales, un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar.

En desarrollo de software se define como un framework o infraestructura digital, a una estructura conceptual y tecnológica de soporte definido, normalmente con artefactos o módulos de software concretos, con base a la cual otro proyecto de software puede ser más fácilmente organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas, y un lenguaje interpretado, entre otras herramientas, para así ayudar a desarrollar y unir los diferentes componentes de un proyecto.

Información geográfica

Se denomina como información geográfica (IG) a aquellos datos espaciales geo referenciados requeridos como parte de las operaciones científicas, administrativas o legales. Dichos geodatos poseen una posición implícita (la población de un censo, una referencia catastral) o explícita (coordenadas obtenidas a partir de datos capturados mediante GPS).

Sistema de Información Geográfica

Un Sistema de Información Geográfica (SIG o GIS) es una integración organizada de hardware, software y datos geográficos diseñados para capturar, almacenar, manipular, analizar y desplegar en todas sus formas la información geográficamente referenciada con el fin de resolver problemas complejos de planificación y gestión geográfica. También puede definirse como un modelo de una parte de la realidad referido a un sistema de coordenadas terrestre y construido para satisfacer unas necesidades concretas de información. En el sentido más estricto, es cualquier sistema de información capaz de integrar, almacenar, editar, analizar, compartir y mostrar la información geográficamente referenciada. En un sentido más genérico, los SIG son herramientas que permiten a los usuarios crear consultas interactivas, analizar la información espacial, editar datos, mapas y presentar los resultados de todas estas operaciones.

VI RepastSimphony y RepastCity

Introducción a Repast y RepastCity

Repast Simphony 2.0 es una plataforma basada en Java que puede ser ejecutada en Windows, Apple Mac OS X y Linux. Soporta el desarrollo de modelos flexibles de agentes. La plataforma Repast Simphony 2.0 viene ya incorporada con el ambiente de desarrollo Eclipse con los plugins y las configuraciones necesarias para desarrollar un proyecto en dicha plataforma.

Los modelos de Repast Symphony pueden ser desarrollados de varias formas incluyendo ReLogo que es un dialecto de Logo, diagramas de flujo, Groovy y Java, los cuales pueden ser intercambiados indistintamente. Es posible importar modelos NetLogo.

Una característica atractiva de Repast es la habilidad de integrar datos GIS directamente dentro de la simulación.

RepastCity es un programa pequeño de Repast Simphony que demuestra como crear una ciudad virtual y mover algunos agentes sobre una red de caminos. A pesar de que el código para mover un agente sobre la red de caminos es complicado, este programa permite que un programador se concentre en implementar el comportamiento de un agente.

RepastCity fue modificado por nosotros extendiendo el modelo para adecuarse a los requisitos pedidos en el desarrollo de la plataforma multi-agente. Este es el código que será distribuido como producto resultante de nuestro proyecto.

RepastCity – Estuctura de los modelos

En los modelos de RepastCity, los agentes están organizados en colecciones llamadas **Contexto**. Un contexto es básicamente un contenedor que puede ser usado para contener a un agente. Los contextos están organizados jerárquicamente y pueden contener subcontextos. Los agentes que existen en un subcontexto también existen en el contexto padre.

Las **Proyecciones** son usadas para darle un espacio a un agente y puede definir sus relaciones. Por ejemplo, las proyecciones 'GIS' le dan a cada agente una localización espacial (x,y) y la proyección Network permite definir relaciones entre agentes. Las proyecciones creadas para contextos específicos y automáticamente contendrán a cada agente dentro del contexto (por lo que si un agente es agregado a un contexto, automáticamente será agregado a cualesquiera de las proyecciones que han sido creadas en el contexto).

La Imagen 1muestra la organización del modelo de RepastCity. Cada contexto se ha asociado a una proyección GIS para almacenar la ubicación espacial de los objetos.

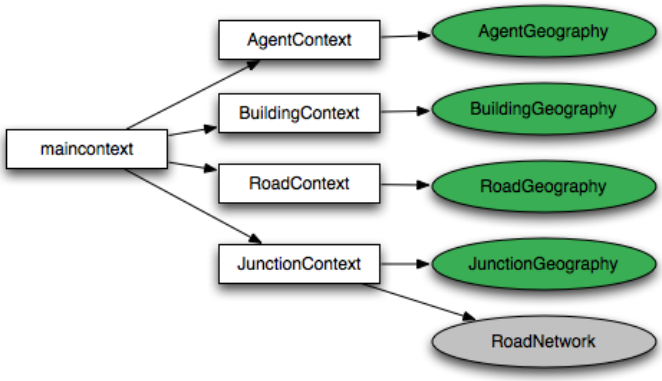

Imagen 1. Organización de contextos y proyecciones en RepastCity

Creando la ciudad – contextos y proyecciones

El contexto AgentContext contendrá todos los tipos de agentes que estén en la simulación.

El contexto BuildingContex contendrá a todos los edificios de la ciudad.

Existen dos tipos de objetos para representar la red de vías: Roads (caminos) y Junctions (intersecciones). El contexto RoadContext contendrá los caminos de la ciudad. El contexto JunctionContext contendrá las intersecciones entre dos caminos. Los Junctions son creados en los puntos

donde dos caminos se crucen y forman los nodos en la proyección de red de estos contextos. El contexto JunctionContext es usado para mover a los agentes. La proyección JunctionGeography es una proyección GIS usada para contener todas las intersecciones de caminos y la proyección RoadNetwork es una proyección de red que contiene los enlaces entre las diferentes intersecciones. Por lo tanto los agentes usan la red para moverse de un lugar a otro a lo largo de la red de vías.

Se usan dos contextos separados porque solo se desea obtener objetos Juntions en la proyección RoadNetwork para formar los nodos de la red de vías y esto no se pueda realizar si Roads y Juntions permanecen en el mismo contexto.

Las proyecciones GIS AgentGeography, BuidingGeograpahy y RoadGeography utilizarán archivos shapefile, los cuales contendrán la información geográfica que usarán los agentes en la simulación.

AgentGeography usará una capa de puntos (o varias capas, cada capa de puntos asociada a un tipo de agente en específico) para representar la ubicación inicial de los agentes en la escena.

BuidingGeograpahy usará una capa de polígonos en la cual cada polígono representa una edificación. Un agente puede partir inicialmente de una edificación, o puede partir de un camino.

RoadGeography utilizará una capa de líneas para representar los caminos de la ciudad. Un agente debe de moverse sobre un camino y solo saldrá de este cuando entre a una edificación. Como requerimiento especial para la capa de puntos debemos de destacar que

- No deben de existir caminos desconectados, o sea, la red entera debe estar conectada.

- Se requiere que si dos caminos se cruzan, entonces la línea debe de terminar en el punto en que se cruzan (se intersectan). Si este requerimiento no se cumple, probablemente el modelo se ejecute, pero el ruteado de los agentes no funcione bien. La función 'Planarize Lines' de ArcGIS puede convertir los datos a esta forma.

Las proyecciones JunctionGeography y RoadNetwork utilizarán la información de la capa de líneas para identificar los puntos de intersección.

En la Imagen 2se muestran las tres capas (shapefiles). Los puntos rojos muestran la ubicación inicial de los agentes, los polígonos muestran las edificaciones y las líneas verdes las vías de la ciudad.

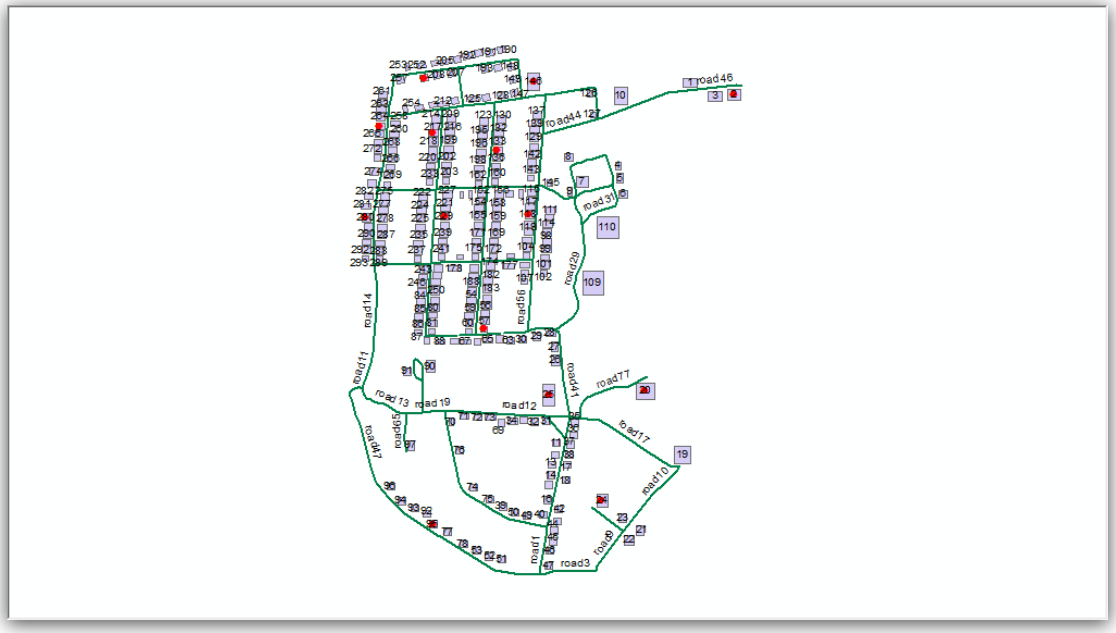

Imagen 2. Ciudad virtual en RepastCity

Como hemos podido observar, con estas capas hemos creado una ciudad virtual, la cual consiste en caminos y edificios. El directorio por defecto donde se deben de colocar los archivos shapefiles es data/gis_data/toy_city.

Extensiones realizadas sobre RepastCity

Una vez introducidos los modelos de Repast y RepastCity vamos a mencionar los cambios realizados a esta plataforma.

Extendiendo los movimientos de los agentes a un entorno de ciudad y un entorno rural.

Como se ha podido observar, en RepastCity el movimiento de los agentes está restringido a estar sobre una red de vías. Nosotros pretendemos que el agente pueda salir de la ciudad y luego moverse libremente fuera de la ciudad sin que este tenga que estar sobre un camino.

Para esto, es necesario agregar una nueva capa de puntos, en la cual un punto estará sobre un camino y va a representar la salida de la ciudad, una vez que el agente sobrepase este punto, podrá moverse libremente fuera de la ciudad. En la Imagen 3se muestran estos puntos con el símbolo .

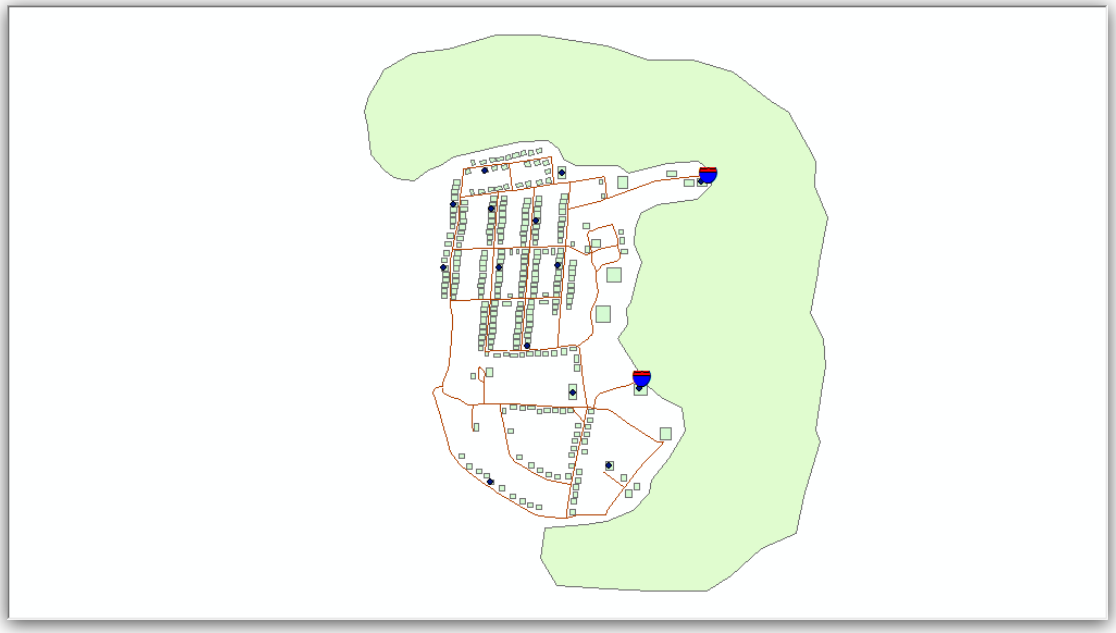

Imagen 3. Capas agregadas a la ciudad virual de RepastCity

Adicionalmente, agregamos un nuevo contexto llamado GateContext con una proyección del tipo GIS llamado GateGeography en la cual se tendrá la posición geográfica de los puntos de salida/entrada de la ciudad y poder consulta estas posiciones posteriormente.

Para ello creamos una nueva clase llamada GateContext la cual extiende la clase DefaultContext<Gate>.

```
publicclass GateContext extends DefaultContext<Gate> {
	public GateContext() {
		super(GlobalVars.CONTEXT_NAMES.GATE_CONTEXT);
	}
}
```

Además, debemos implementar la clase Gate, la cual hereda de FixedGeography, con el propósito de poder consultar las coordenadas de la ubicación de los puntos de salida de la ciudad.

```
publicclass Gate implements FixedGeography {
	private Coordinate coords;
	…
	@Override
	public Coordinate getCoords() {
		returnthis.coords;
	}

	@Override
	publicvoid setCoords(Coordinate c) {
		this.coords = c;

	}
}
```

VII Metodología de solución

A continuación mostraremos los materiales que usamos en nuestro proyecto

Materiales a utilizar:

- Java Development Kit (JDK)6 o superior.
  Dado a que nuestro proyecto está desarrollado en el lenguaje de programación Java, se debe de tener instalado el JDK. La versión 2 de RepastSimphony incluye elJDK en su instalador.
- RepastSimphony 2 con el ambiente de desarrollo Eclipse, disponible en http://repast.sourceforge.net/download.html
  RepastSymphony es una plataforma para realizar simulaciones basadas en agentes, sobre la cual se trabajará con el proyecto RepastCity 3.
- RepastCity 3 modificado. Este es el proyecto que hemos extendido el cual está basado en el programa RepastCity 3 disponible en http://code.google.com/p/repastcity/wiki/RepastCity3
  RepastCity 3 es un proyecto de la plataforma RepastSimphony escrito en Java al cual realizaremos modificaciones.
- ArcGIS
  ArcGIS es un Sistema de Información Geográfica, el cual usaremos para crear y/o editar las capas shapefile que formarán parte del mundo en el cual el agente deberá moverse en la simulación.

Ya citados los materiales que se usarán en el proyecto, mencionaremos los pasos necesarios para realizar una simulación usando agentes.

Pasos para realizar una simulación.

- Creación de una clase que represente la dinámica del agente, véase Agregar un nuevo agente.
- Creación de un nuevo fenómeno, véase Agregar un nuevo fenómeno

- Modificar capas shapefile que representen el ambiente donde se va a realizar la simulación, véase Agregando los archivos shapefile.

Creando nuevos agentes

La clase AgentState

Una de las extensiones realizadas a la plataforma RepastCity fue crear una nueva clase llamada AgentState, en la cual se representa el estado actual del agente. Los valores de estado de interés para los agentes son los siguientes:

private int healt;

representa la salud del agente

private int stamina;

representa la energía del agente

private int speedRoad;

representa la velocidad del agente sobre una vía

private int speedCountry;

representa la velocidad del agente cuando está en un terreno que no es vía

private boolean inCity;

devuelve verdadero si un agente está dentro de la ciudad, o está en el campo.

Una de las cuestiones que nos puede conllevar a dudas es la diferencia entre health y stamina. health se refiere a la salud propia del agente, es decir, es una medida para determinar si el agente sufre algún daño en su integridad, digamos, al chocar contra un obstáculo. stamina se refiere a la energía que el agente tiene para moverse, pudiera considerarse como el combustible que tenga un vehículo.

Adicionalmente, se incluyen estas variables

private int maxHealt;

representa la salud máxima del agente

```
privateintmaxStamina;
```

representa la energía máxima del agente

```
privateintmaxspeedRoad;
```

representa la velocidad máxima del agente sobre una vía

```
privateintmaxspeedContry;
```

representa la velocidad del agente cuando está en un terreno que no es vía.

La interfaz IAgent

Todos los agentes deben de implementar esta interfaz para que la plataforma Repast Simphony conozca que hacer durante un paso (step). Todos los agentes deben de implementar el método step(). En este método se pondrá la lógica de la funcionalidad del agente que debe realizar en un instante de tiempo. Este método esllamado por el planificador (scheduler) en cada iteración.

Adicionalmente a los métodos existentes en Repastcity 3, los cuales aparecen en la documentación que viene incluida en la distribución, esta interfaz la hemos extendido con el objetivo de incorporar un método AgentState GetState() el cual nos informará el estado actual del agente, véase La clase AgentState. Además, incorporamos los métodos ContextManager GetWorld() y **void**SetWorld(ContextManager world), los cuales nos permiten obtener la referencia al mundo y modificar la referencia al mundo, respectivamente. Se incluyen los métodos **void** Move(Point p),**void** MoveByVector(**double** d)y **void** MoveByVector(Coordinate coord)para mover a los agentes a un punto determinado, moverlo cierto ángulo o moverlo a cierta coordenada, respectivamente. Existen métodos para saber si el agente está dentro de la ciudad o en el campo, y para cambiar de estado de la ciudad a campo y viceversa, estos métodos son **boolean** IsInsideCity(),**void** InCity() y **void** OutCity(). Para el manejo de mensajes entre agentes están los métodos **void**

SendMessage(MessageType messageType,String message),**void** SendMessageTo(IAgent target, String message) y**void** ReceiveMessage(MessageType messageType, String message, IAgent sender); los cuales permiten enviar mensajes *broadcast* (a todos los agentes), *multicast* (a agentes de un mismo tipo)y *peer to peer* (a un agente en específico) y recibir un mensaje.

Agregar un nuevo agente

Para agregar un nuevo agente, primero es necesario crear una nueva clase, la cual debe de heredar de la interfaz **IAgent**, véase La interfaz **IAgent**.

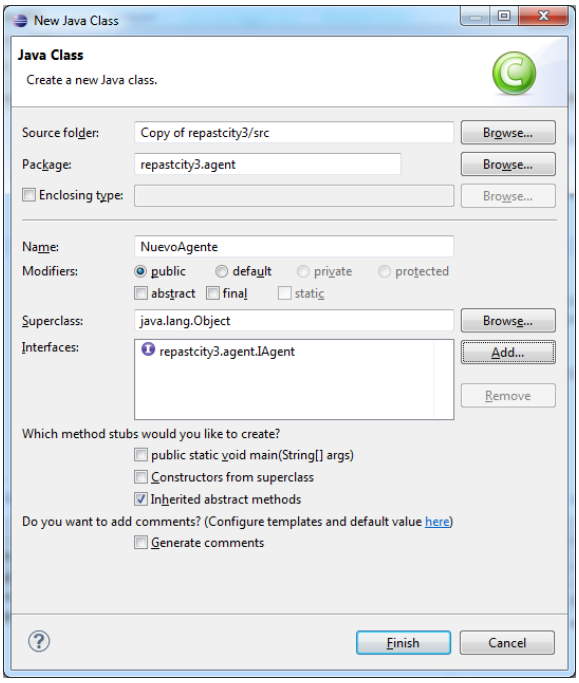

Imagen 4. Ventana de dialogo de Eclipse para agregar una nueva clase

El código que agregamos hará que el agente seleccione un edificio aleatoriamente, se mueve hacia él y luego regresa a su lugar de origen.

El código fundamental de la clase que implemente la interfaz IAgente es la que le da implementación al método step(). En el siguiente código se implementa la funcionalidad de movimiento antes mencionada.

```
publicvoid step() throws Exception {
        // Versi el agentetieneunobjeto Route
        if (this.route == null) { // No hay unobjeto route, crearunonuevo
        this.goingHome = false;
// Elegirunnuevoedificioparaviajar a él
Building b = world.GetRandomBuilding();
this.route = new Route(this, b.getCoords(), b);
}

// Versi el agente ha alcanzadosudestino, si no, debecontinuarsuviaje a
sudestino
if (!this.route.atDestination()) {
this.route.travel();
}
else {
// Se ha alcanzado el destino, ahoraregresa a sulugardeorigen
        this.goingHome = true;
this.route = new Route(this, this.home.getCoords(),
                        this.home);

}
}
```

Adicionalmente alhaber creado la clase para el agente, es necesario crear con ArcGIS una nueva capa de puntos, la cual reflejará la ubicación inicial de los agentes.

Para ello debemos de abrir ArcCatalog y crear una nueva capa de puntos, donde cada punto reflejará la ubicación inicial (el hogar) del agente. No es objetivo de nosotros explicar el trabajo con ArcGIS así que solo mencionaremos las operaciones principales que se realizarán para crear la capa de puntos. Creada a capa de puntos se procede con ArcMap a editar esta capa agregando los puntos dentro de los polígonos que representan a los edificios. Una vez terminada la edición, procedemos a salvar dicha capa. En la Imagen 5 mostramos la creación de la nueva capa de puntos con las ubicaciones iniciales de los agentes.

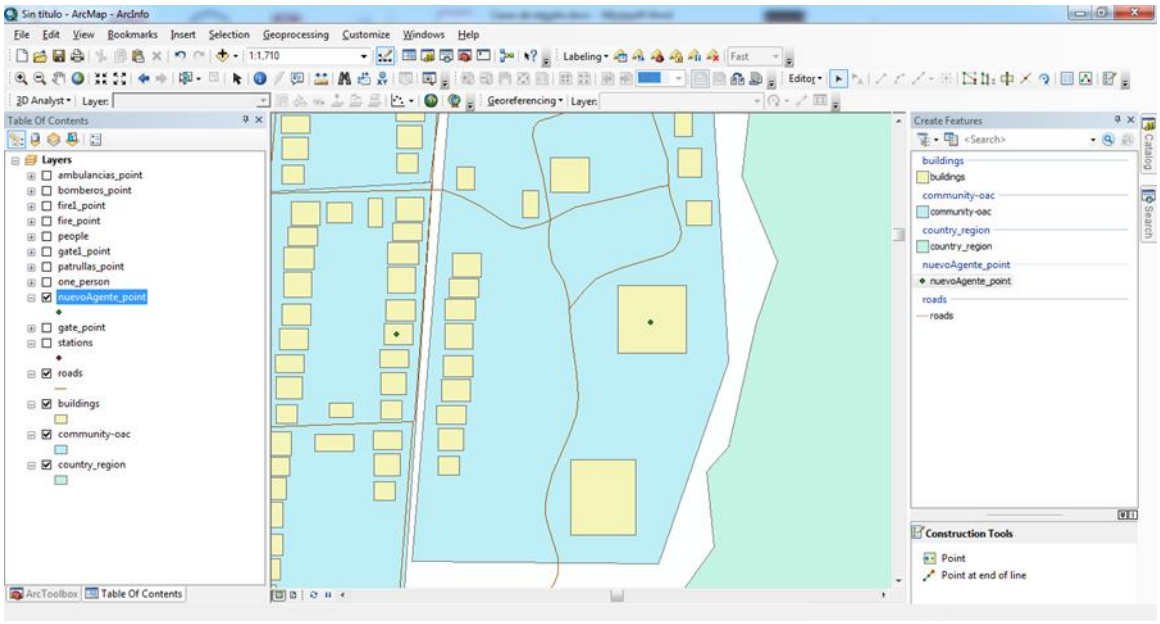

Imagen 5. Edición de capas en ArcGIS

Una vez creada la clase del nuevo agente y la capa de puntos donde están las ubicaciones iniciales del tipo de agente recién creado, se le debe de especificar al visualizador de la simulación integrado a RepastSimphony el nombre de la nueva clase de agente que recién implementamos. Para lograr esto debemos:

1. Iniciar el modelo (compilar la aplicación).
2. En la ventana dar clic en la parte inferior izquierda, en el tab llamado Scenario Tree.
3. En el panel de TreeView de la izquierda, seleccionar ‘AgentContext’ y luego ‘AgentDisplay’ (Imagen 6).

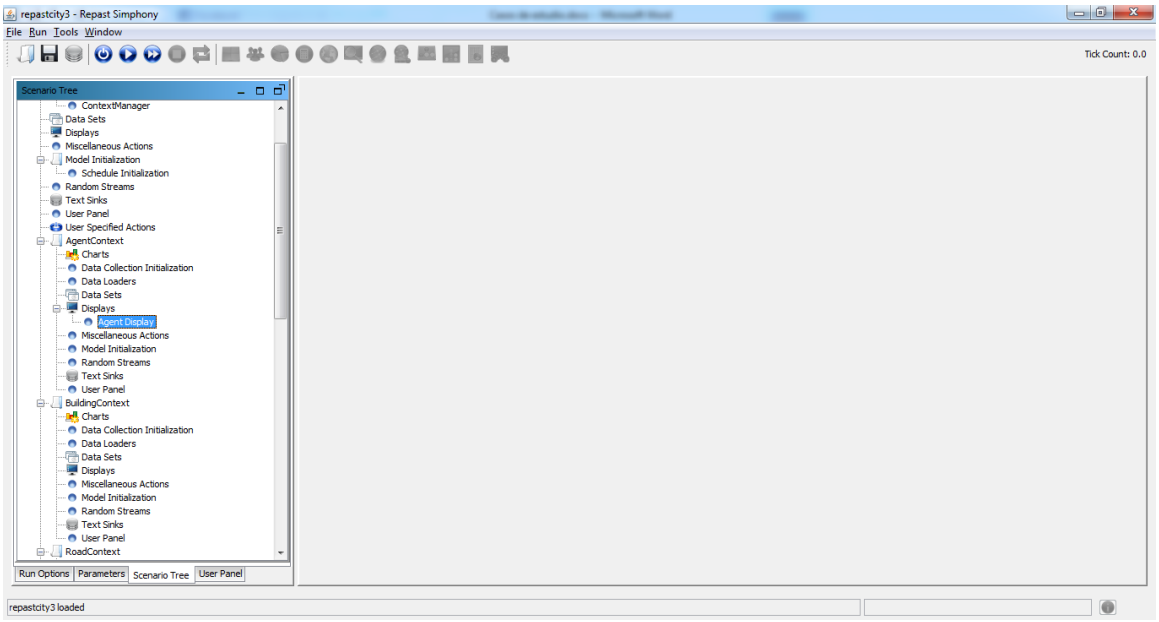

Imagen 6. Ventana de simulación de Repast

4. Luego en la ventana de diálogo, seleccionar ‘AgentStyle’ , damos clic sobre el botón + y seleccionamos la capa recién creada.

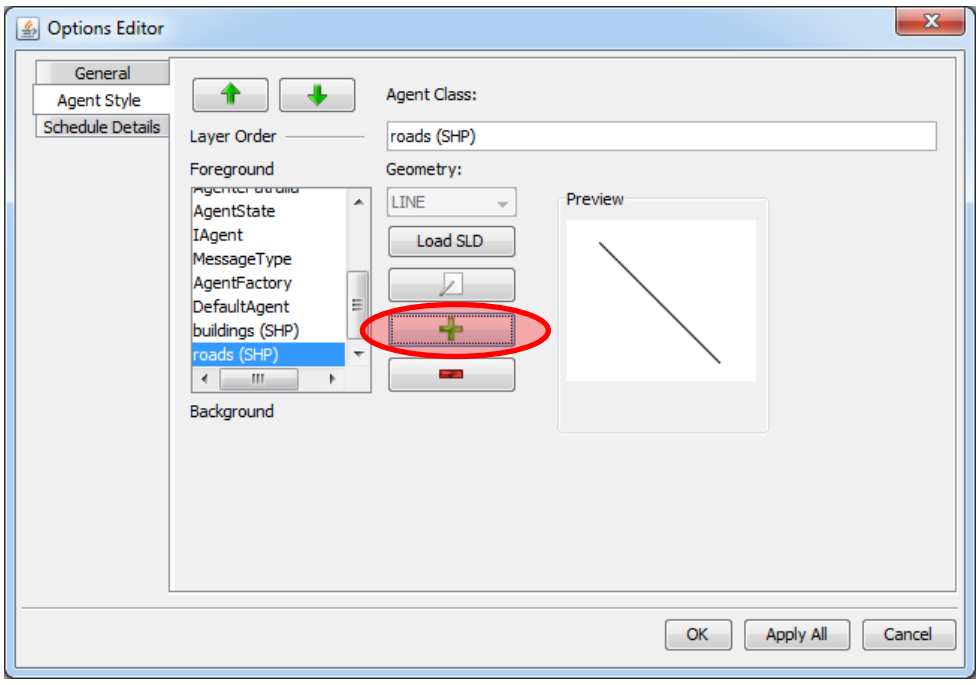

Imagen 7. Ventana para agregar una capa shapefile

5. En la ventana dar clic en la parte inferior izquierda, en el tab llamado Parameters

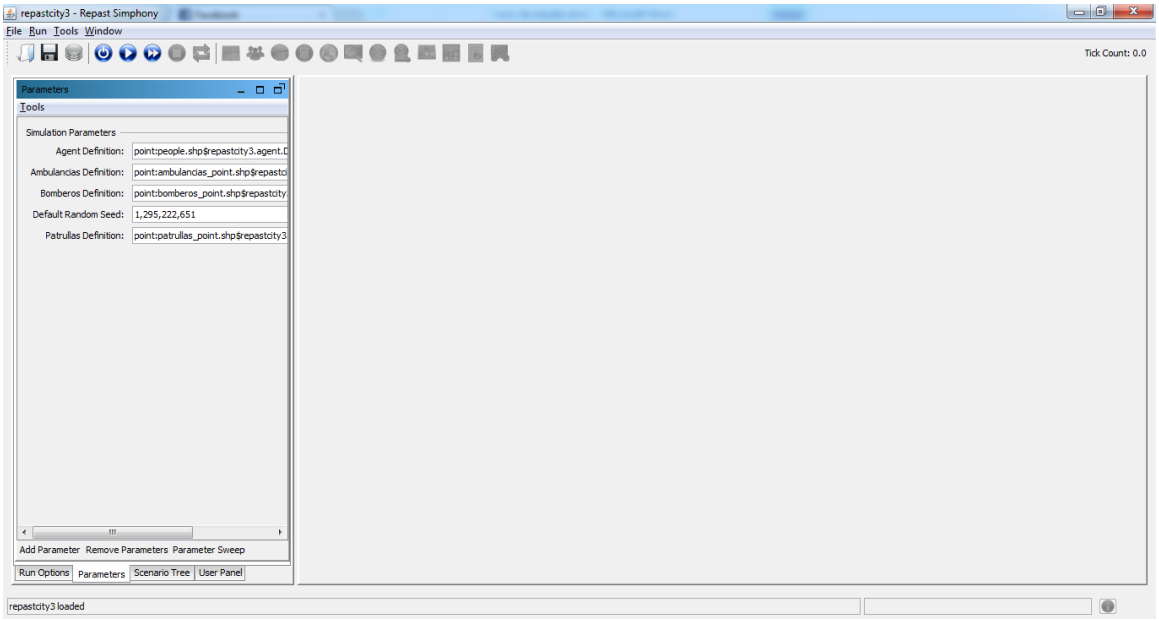

Imagen 8. Ventana para agregar la definición de un agente a la simulación

6. Dar clic en el botón Add Parameter y llenar el cuadro de diálogo que nos aparecerá con los siguientes valores para el nuevo parámetro.

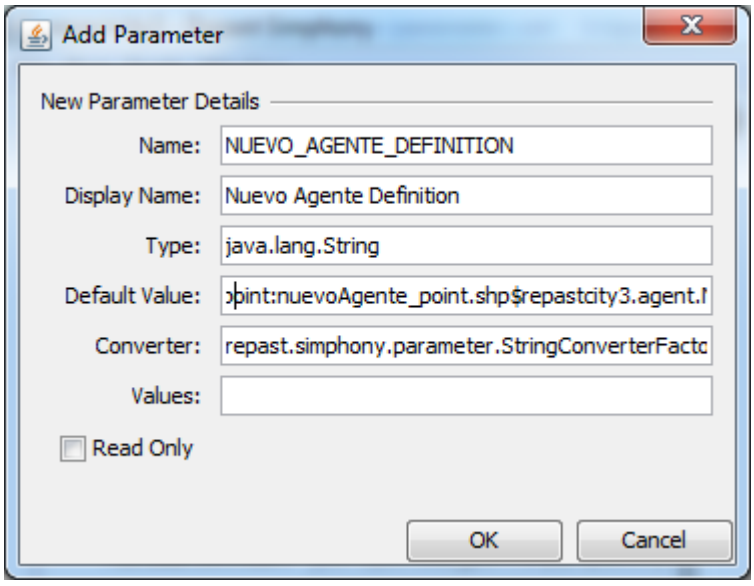

Imagen 9. Parámetros de la definición de un agente

Name: NUEVO_AGENTE_DEFINITION
Display Name: Nuevo Agente Definition
Type: java.lang.String
Default Value: point:nuevoAgente_point.shp$repastcity3.agent.NuevoAgente
Converter: repast.simphony.parameter.StringConverterFactory$StringStringConverter

Luego damos clic en OK. Si consultamos el fichero xml repastcity3.rs\parameters.xml podremos observar que se han incluido las siguientes líneas de código xml con los valores recién establecidos para el nuevo parámetro.

```
<parametername="NUEVO_AGENTE_DEFINITION"displayName="Nuevo Agente Definition"type="java.lang.String"

        defaultValue="point:nuevoAgente_point.shp$repastcity3.agent.NuevoAgente"
        isReadOnly="false"

        converter="repast.simphony.parameter.StringConverterFactory$StringStringConverter"

/>
```

Mediante la configuración de los parámetros hemos establecido la clase endonde se define el nuevo tipo de agente y el archivo shapefile donde están las ubicaciones iniciales de los agentes del nuevo tipo definido.

7. Agregar al enum MODEL_PARAMETERS del paquete repastcity3.main el nombre del parámetro establecido, en este caso es NUEVO_AGENTE_DEFINITION. Las siguientes líneas muestran como quedaría el enum.

```
publicenum MODEL_PARAMETERS {
/** A string definition of how agents should be created */
```

```
    AGENT_DEFINITION,PATRULLAS_DEFINITION,BOMBEROS_D
    EFINITION,
    AMBULANCIAS_DEFINITION,NUEVO_AGENTE_DEFINITION;
}
```

Esto nos va a permitir crear la geografía que se muestra en el código en el próximo paso.

8. Agregar las siguientes líneas de código Java a la clase ContextManager del paquete repastcity3.main, en el método **public**Context<Object> build(Context<Object> con), en la sección donde se crean los agentes, el código a agregar aparece marcado en fondo azul claro.

```
// Now create the agents (note that their step methods are scheduled
later
try {
        agentContext = new AgentContext();
        mainContext.addSubContext(agentContext);
        agentGeography                                          =
        GeographyFactoryFinder.createGeographyFactory(null).
                createGeography(GlobalVars.CONTEXT_NAMES.AGEN
                T_GEOGRAPHY,agentContext,
                new GeographyParameters<IAgent>(
                new SimpleAdder<IAgent>()));
        …

        String nuevoAgenteDefinition =
                ContextManager.getParameter(MODEL_PARAMETERS.
                NUEVO_AGENTE_DEFINITION.toString());
        AgentFactory nuevoAgenteFactory =
                new AgentFactory(nuevoAgenteDefinition, this);
        nuevoAgenteFactory.createAgents(agentContext);

        }
```

9. En la ventana de la simulación de Repast Simphony, dar clic en el botón Guardar  y luego dar clic en el Botón Start Run  para iniciar la simulación.

Modelo de comunicación entre los agentes

En la plataforma hemos agregado tres modelos de comunicación entre agentes:

- Broadcast: se envía un mensaje a todos los agentes.
- Multicast: se envía un mensaje solo a los agentes del mismo tipo
- Peer to Peer: se envía un mensaje a un agente en específico

Para enviar mensajes del tipo broadcast y multicast se debe de dar implementación al método SendMessage(IAgent sender, MessageType messageType,String message).El siguiente código muestra una posible implementación para enviar mensajes *broadcast* y *multicast*.

```
publicsynchronizedvoid SendMessage(IAgent sender, MessageType messageType,String message)
{
        if(messageType==MessageType.Broadcast)
        {
        for (IAgent agent : agentGeography.getAllObjects()) {
                agent.ReceiveMessage(messageType, message, sender);
                }
        }
        if(messageType==MessageType.Multicast)
        {
                for (IAgent agent : agentGeography.getAllObjects()) {

                        if(agent.GetType()==sender.GetType())
                                agent.ReceiveMessage(messageType, message,
                                        sender);
                }

        }
}
```

Para implementar los mensajes del tipo Peer to peer se debe implementar el método SendMessage(IAgent sender,IAgent target, String message) de la interfaz IAgent. El siguiente código muestra una posible implementación para enviar mensajes *peer to peer* a un agente en específico.

```
/**
* Para manejar mensajes del tipo PeerToPeer
* @param sender
* @param target
* @param message
*/
publicsynchronizedvoid SendMessage(IAgent sender,IAgent target, String message)
{
        for (IAgent agent : agentGeography.getAllObjects()) {

                if(agent==target)
                {
                        agent.ReceiveMessage(MessageType.PeerToPeer,mess
                        age,
                        sender);
                        return;
                }
        }
}
```

Adicionalmente, para recibir los mensajes enviados, se debe de implementar el método ReceiveMessage de la interfaz IAgent. Una posible implementación se muestra en el siguiente fragmento de código. La operación realizada imprime en el log queagente envió el mensaje y el contenido del mensaje recibido.

```
publicvoid ReceiveMessage(MessageType messageType, String message,
                        IAgent sender) {
        LOGGER.log(Level.WARNING, this.toString() + " received message
        from " + sender.toString()+ "and the message was " + message);
}
```

Agregar un nuevo fenómeno

La plataforma Repast considera como un agente cualquier elemento que participe en un modelo. Por lo tanto, los fenómenos también serán tratados como un agente normal. Para implementar un fenómeno se realizarán las

mismas operaciones que se trataron en la sección de Agregar un nuevo agente.

Como caso de estudio modelaremos el fenómeno incendio. El modelo consiste en una serie de puntos iniciales ubicados fuera de la ciudad que representan los focos de incendios iniciales. El incendio, a medida que pasa tiempo, va ganando en intensidad y crece (se crea un nuevo agente de tipo Fuego) cuando se sobrepasa un umbral de intensidad (stamina > umbral, véaseLa clase AgentState). También el agente envejece, lo cual le da un tiempo de vida finito (consume el combustible). Mientras que health > 0 el agente está “vivo”. En cada iteración, luego que el agente haya alcanzado su máxima intensidad, su salud comienza a decrecer hasta llegar a 0. Cuando esto ocurre se elimina el agente del contexto. La implementación de este modelo se hará en una clase AgenteFuego, la cual implementa la interfaz IAgent. Como se había comentado anteriormente, el método principal que refleja la lógica del modelo es step(). A continuación mostramos un fragmento de un código que haga estas operaciones.

```
publicvoid step() throws Exception {
        if(state.GetStamina()<1000)
        {
                //El incendio aumenta su fuerza
                state.IncrementStamina(1);
        }
        elseif(isGrowing)
        {
                //El incendio ya ha alcanzado su máxima fuerza, entonces crece
                isGrowing = false;
                int dx = dy = 0.001;
                Coordinate coord = new Coordinate(this.GetCorrdinates().x+dx,
this.GetCorrdinates().y+dy);

                //Se crea el nuevo agente fuego
                AgenteFuego nuevoFuego = new AgenteFuego();
                nuevoFuego.SetWorld(world);
                this.world.addAgentToContext(nuevoFuego);
```

```
                //Se mueve el nuevo agente a una posicion adyacente al fuego
                GeometryFactory geometryFactory=new GeometryFactory();
                nuevoFuego.Move(geometryFactory.createPoint(coord));

        }
        else
        {
                //El incendio está envejeciendo, decrementar su salud
                state.DecrementHealt(1);
                if(state.GetHealt()<=0)
                {
                        //El incendio consumió su combustible, se elimina
                        world.removeAgentToContext(this);
                }
        }

}
```

Visibilidad

Como caso de estudio consideremos un agente que se puede mover por la ciudad o por el campo. Dentro de la ciudad el agente se mueve solo por las calles, excepto cuando entra aun edificio de origen o destino. En el campo el agente puede moverse libremente y debe ser capaz de detectar la presencia de un obstáculo

El modelo de visibilidad de los agentes lo podemos dividir en dos regiones

- Dentro de la ciudad
- Fuera de la ciudad

Dentro de la ciudad

Dentro de la ciudad un agente se moverá usando las vías. El agente elegirá un destino a donde debe llegar. Este destino puede ser un edificio, o un punto de salida de la ciudad. Para moverse, el agente creará una ruta usando las vías de la ciudad y en cada iteración el agente actualizará la ruta que debe de seguir al interactuar con el contexto.

Fuera de la ciudad

Fuera de la ciudad, el agente no tienerestricción de movimiento, ya que el agente no debe de seguir un camino en específico, sino que puede moverse libremente. La única restricción que se tiene es cuando se topa con un obstáculo y debe de cambiar de dirección de movimiento para buscar una forma de franquear el obstáculo. El agente no conoce las coordenadas del punto de destino, por lo que realizará movimientos aleatorios de exploración para encontrar el objetivo de destino (el punto de destino).

Agregando los archivos shapefile

Repast brinda la posibilidad de trabajar con capas shapefile directamente dentro de la simulación. En el caso de Repastcity ya trae configurado por defecto un conjunto de capas shapefile que utiliza para mostrar la simulación de agentes.Entre estas capas podemos destacar la capa ‘buildings’, que es una capa de polígonos que representa a los edificios, una capa ‘roads’, que es una capa de línea que representa las vías por donde transita el agente. Adicionalmente, en la plataforma modificada se agregó una capa de puntos llamada fire_point, la cual representa mediante puntos los focos de incendio.

Para modificar los parámetros de estas capas shapefile, Repast nos da la opción de configurar el entorno visual de la simulación. Una vez compilada la aplicación, se muestra la ventana del entorno Repast (Imagen 6). En el panel de la izquierda podemos observar en ‘Scenario Tree’ que nos muestra un TreeView con las propiedades del proyecto. Una propiedad importante de los agentes es ‘Display’. Un display es la representación gráfica de los agentes y su ambiente. El Display que debemos configurar es el que se encuentra en AgentContext. En la Imagen 10,dicho display es el que se encuentra señalado.

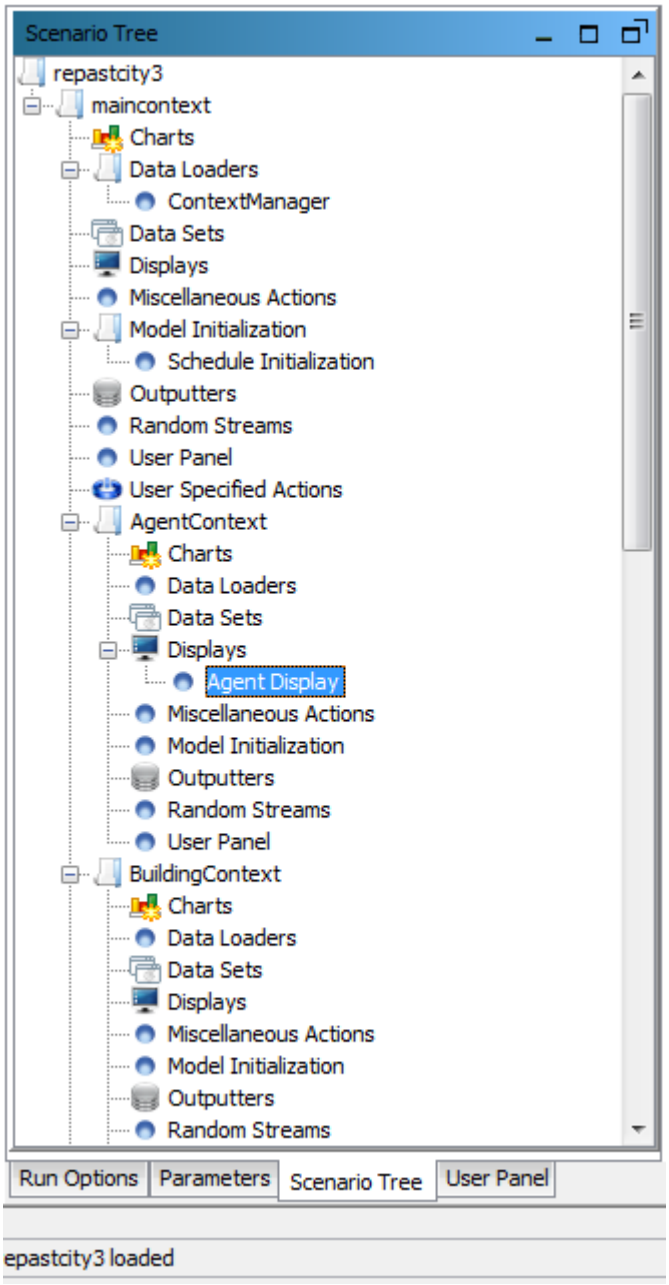

Imagen 10. Scenario Tree

Si queremos adicionar una capa extra, por ejemplo, la capa fire_point, damos doble clic sobre AgentDisplay (Imagen 10) y seguidamente nos aparece una ventana de diálogo para editar a los agentes (Imagen 7). Damos clic sobre el botón + que está señalado sobre una elipse roja y seleccionamos la capa fire_point.shp.

Configuración la ruta de acceso de los shapefiles.

Repastcity permite que podamos especificar los archivos shapefile que vamos a utilizar. Una vez agregadas las capas shapefile como se explicó en la secciónAgregando los archivos shapefile, podemos acceder a un fichero de configuración xml en donde especificamos la ruta en que se encuentran las capas. Este fichero nos permite poder cambiar de ubicación el proyecto y simplemente editar la ruta de las capas sin necesidad de recompilar el proyecto completo. Este fichero de configuración se llama **repast.simphony.action.dislay_1.xml** y se encuentra ubicado en la dirección relativa a la carpeta del workspace de Eclipse **'repastcity3\repastcity3.rs\'**. Usualmente en Windows está la dirección absoluta de este fichero y lo podemos encontrar en **'C:\RepastSimphony-2.0-beta\workspace\repastcity3\repastcity3.rs\repast.simphony.action.dislay_1.xml'**.

Las capas shapefiles, como convenio, debemos guardarlas dentro de la carpeta 'repastcity3\data\gis_data\toy_city'.

En el fichero repast.simphony.action.dislay_1.xml podemos escribir la dirección de las capas shapefile empleadas.

Abierto este fichero xml debemos de buscar dentro de la etiqueta

<layerOrder>
…
</layerOrder>
las secciones que contienen una ruta válida encerradas dentro de etiquetas <string> … </string>. En nuestro caso, las capas shape de nuestro interés son:

- La capa de edificios.
  <string>data\gis_data\toy_city\buildings.shp</string>

- La capa de caminos.
  <string>data\gis_data\toy_city\roads.shp</string>
- Capa de puntos de incendio. Esta es una capa extra incluida por nosotros. No está presente en la versión original de repastcity. El propósito de esta capa es representar un punto de incendio a donde deba de dirigirse el agente bombero.

```
<layerOrder>
<entry>
<string>data\gis_data\toy_city\buildings.shp</string>
<int>11</int>
</entry>
        …
<entry>
<string>data\gis_data\toy_city\fire_point.shp</string>
<int>13</int>
</entry>
        …
<entry>
<string>data\gis_data\toy_city\roads.shp</string>
<int>12</int>
</entry>
</layerOrder>
```

Adicionalmente a la etiqueta <layerOrder> debemos buscar y editar la dirección de los shapesfiles dentro de la etiqueta

<props>

…

</props>

editamos las secciones que contienen dentro de las etiquetas <string> … </string>con una ruta válida.

```
<props>
<entry>
<string>class
repast.simphony.visualization.gis.DisplayGIS.SHP_FILE_STYLE_PROP</stri
ng>
<map>
<entry>
```

```
<string>data\gis_data\toy_city\buildings.shp</string>
                …
</entry>
<entry>
        <string>data\gis_data\toy_city\roads.shp</string>
                …
        </entry>
<entry>
        <string>data\gis_data\toy_city\fire_point.shp</string>
        </entry>
</map>
</entry>
</props>
```

VIII Instituciones o empresas que aplican o participan en elproyecto

Instituciones participantes

INAOE, CRECTEALC-México

Instituciones en las que aplica el proyecto

INAOE

IX Resultados

Como resultado hemos modificado el proyecto RepastCity de la plataforma Repast Simphony para crear agentes para realizar simulaciones de desastres. En esta plataforma modificada permitimos las siguientes operaciones:

- Crear un nuevo agente para la simulación.
- Crear un nuevo desastre.
- Desarrollar modelos de comunicación entre los agentes.
- Modificación del mundo mediante el uso de archivos shapefiles.
- Desarrollo de un ambiente de simulación que permite la incorporación de diversos tipos de agentes y fenómenos catastróficos.

X Conclusiones

Como conclusión general podemos mencionar que en la plataforma RepastCity modificada es posible realizar simulaciones de desastres ya que hemos desarrollado una extensiónque cumple con los elementos que debe de tener un modelo basado en agentes mencionados en la Introducción. Estos elementos son el resultado de este proyecto:

- Posibilidad de creación de varios tipos de agentes.
- Posibilidad de modelar varios tipos de fenómenos o desastres.
- Se cuenta con tres modelos de comunicación entre agnetes, los cuales son el broadcast, multicast y el peer to peer.
- Se permite modelar el ambiente del agente mediante un contexto y usar información geográfica mediante archivos shapefiles, los cuales puede ser cambiados si se desea cambiar elmundo en el cual se moverán los agentes.

XI Trabajo futuro

El alcance de este proyecto es la creación de una plataforma para la simulación de desastres. Esta plataforma debe ser enriquecida con

- Implementación de varios tipos de agentes y de desastres naturales.
- Diseño e implementación varias estrategias de solución de los agentes ante un desastre
- Realizar varios modelos de comportamiento de los fenómenos catastróficos.
- Usar capas shapefile de un ambiente real para observar el comportamiento de la simulación en una localidad específica.
- Ya que la simulación solo se realizó con pocos tipos de agentes y con capas shapefile de tamaño pequeño, se propone realizar la prueba con varios tipos de agentes y desastres y con capas shapefile más grandes con el objetivo de probar que tan efectiva es la plataforma ante el uso de un gran volumen de información en la simulación.

XII Bibliografía

Allan, R.J., Science, and Technology Facilities Council. *Survey of Agent Based Modelling and Simulation Tools*. Science & Technology Facilities Council, 2010.

Andersen, Marco Valente, and Sloth Esben. "A Hands-on Approach to Evolutionary Simulation: Nelson and Winter Models in the Laboratory for Simulation Development." *The Electronic Journal of Evolutionary Modeling and Economic Dynamics* (2002).

Fishwick, Paul A. *Simulation Model Design and Execution: Building Digital Worlds*. Prentice Hall PTR, 1995.

North, Michael J., Nicholson T. Collier, and Jerry R. Vos. "Experiences Creating Three Implementations of the Repast Agent Modeling Toolkit." *ACM Trans. Model. Comput. Simul.* 16, no. 1 (2006): 1-25.

Robert, Tobias, and Hofmann Carole. "Evaluation of Free Java-Libraries for Social-Scientific Agent Based Simulation." *Journal of Artificial Societies and Social Simulation* 7 (2004).

Apéndices

Apéndice I

Pasos a seguir para instalar la plataforma

1- Instalar Repast Simphony 2.0.
2- Importar nuestro proyecto de simulación de desastre el cual es una modificación a RepastCity
3- Configurar archivos shapefile (véase Agregando los archivos shapefile).

Instalando Repast Simphony

Al instalar Repast Simphony 2.0, estaremos instalando el ambiente de desarrollo Eclipse configurado con una colección de clases y plugins necesarios para desarrollar un proyecto para la simulación de agentes. La ruta de instalación de Repast Simphony por defecto en Windows es en 'C:\RepastSimphony-2.0' (Imagen 11). Sobre esta carpeta se copiarán todos los archivos requeridos. Entre las carpetas más importantes podemos destacar la carpeta 'eclipse' la cual contiene el ambiente de desarrollo Eclipse y la carpeta workspace, la cual es la carpeta workspace del ambiente de desarrollo eclipse donde se almacenarán los proyectos desarrollados sobre este ambiente de desarrollo (Imagen 12).

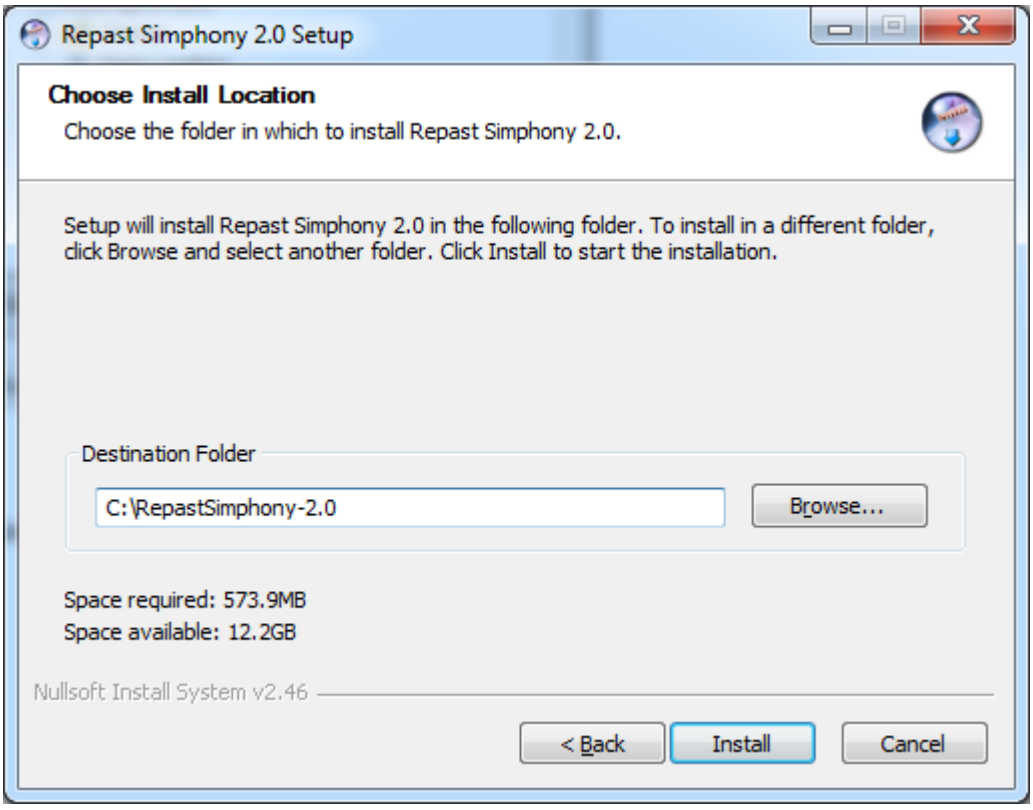

Imagen 11. Cuadro de dialogo de instalación de Repast Simphony 2.0

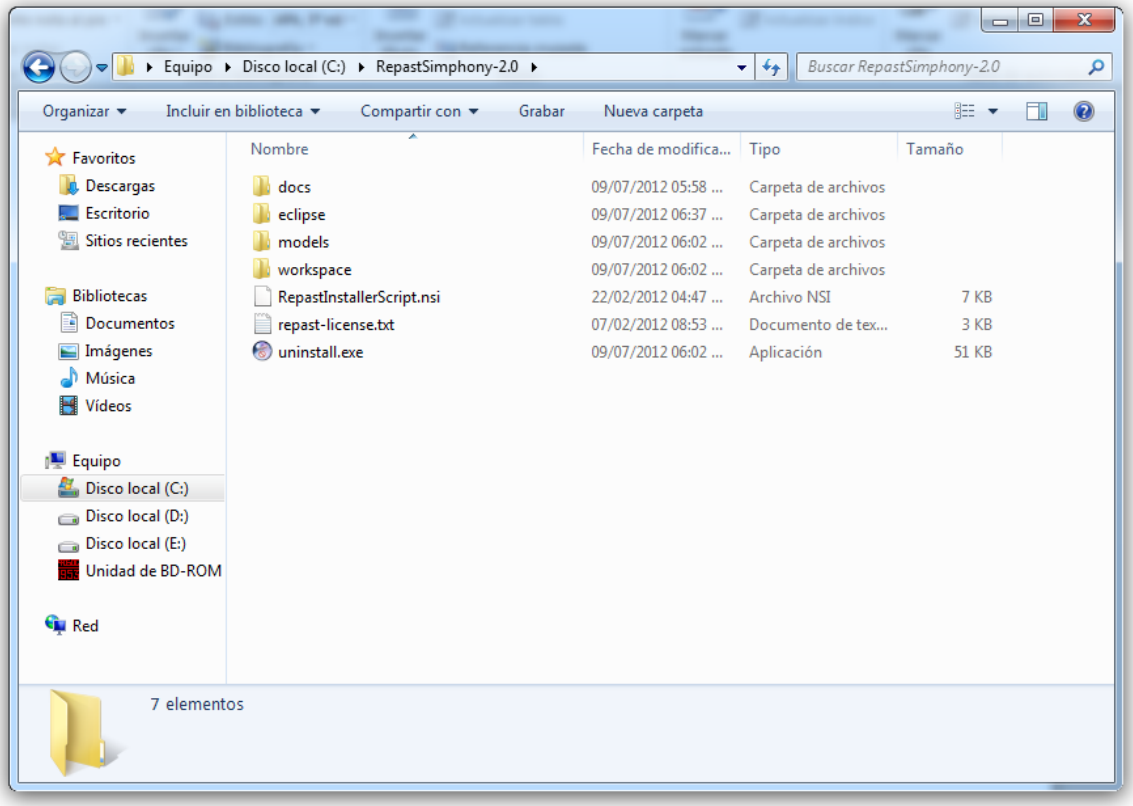

Imagen 12. Contenido de la carpeta de instalación de Repast Simphony 2.0

Importando el proyecto a la plataforma Repast Simphony.

Iniciamos Repast (Eclipse). Sobre el sistema operativo Windows podemos acceder mediante el botón Inicio de Windows -> RepastSimphony-2.0 -> Repast Simphony.

Vamos al menú File ->Import … y seleccionar 'Existing Projects IntoWorkspace' (Imagen 13) y dar clic en Next >. Luego en 'Select root directory', damos clic en 'Browse' y seleccionamos la ruta del proyecto. Nos aseguramos de que esté marcada la opción 'copy projects into workspace' para que se copie el código en el workspace de Eclipse. Finalmente damos clic en Finish para terminar (Imagen 14).

Una vez terminado se puedeverificar que se han copiado dentro de la carpeta 'workspace' los archivos del proyecto.

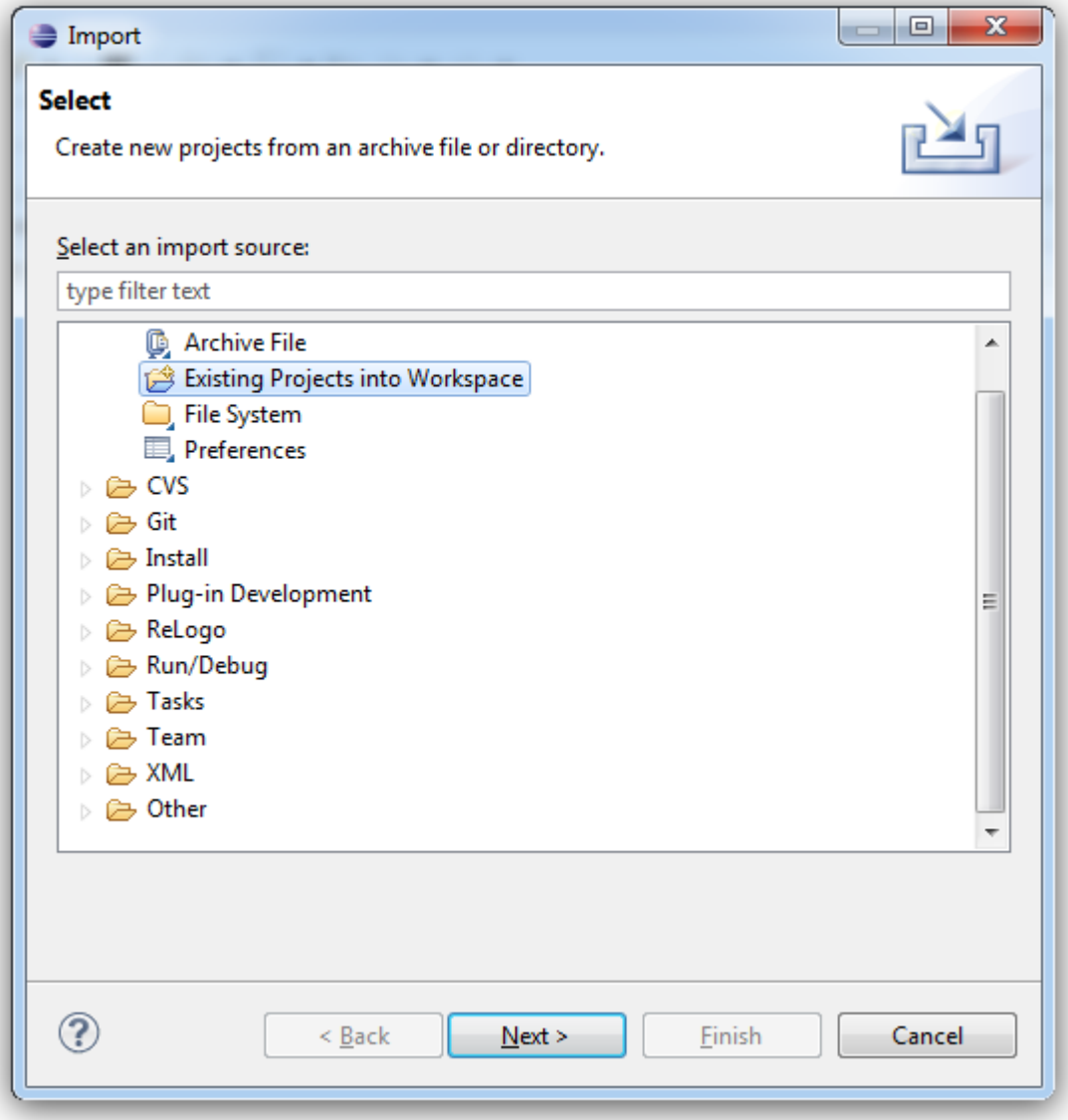

Imagen 13. Cuadro de diálogo para Importar el proyecto

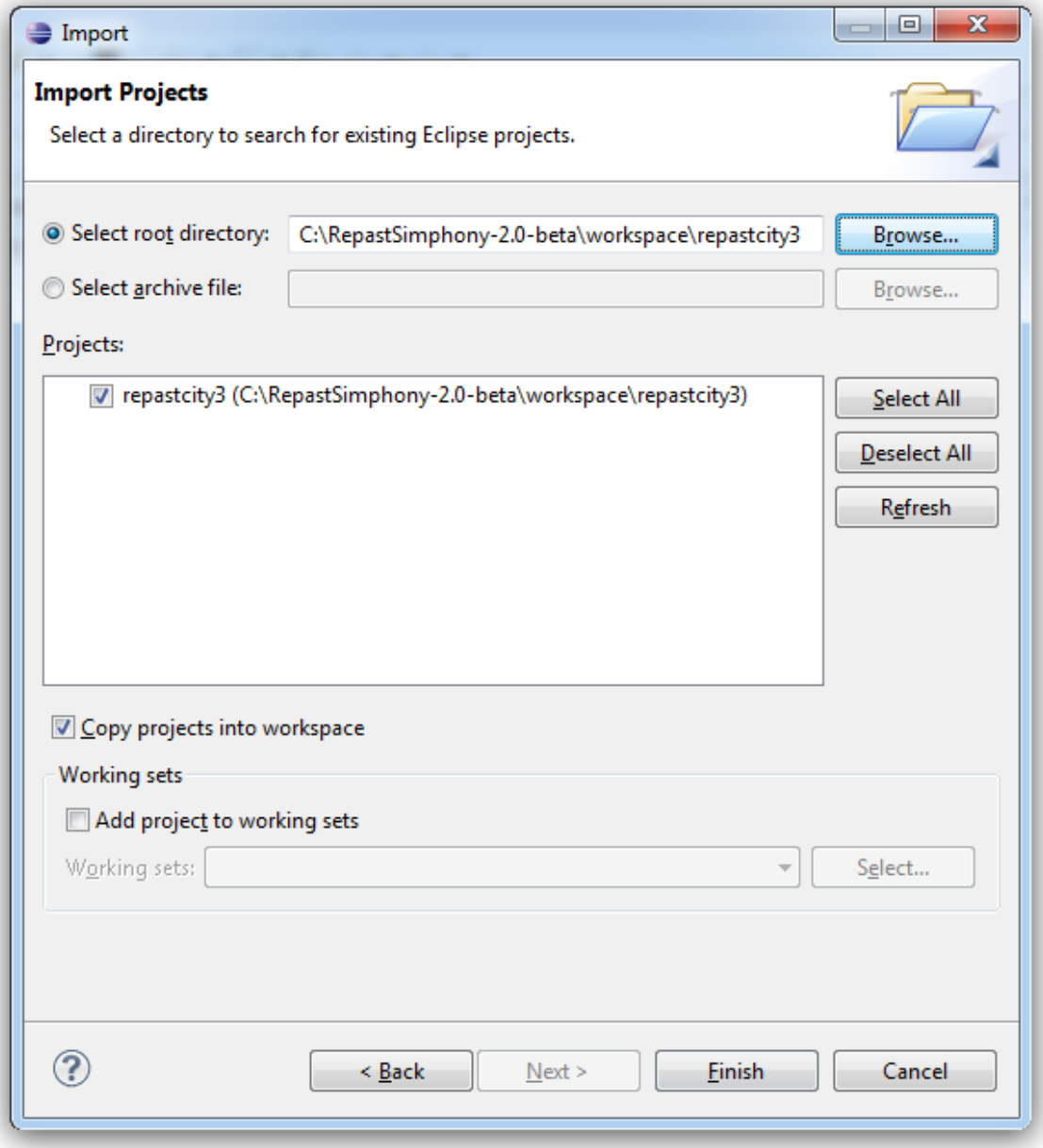

Imagen 14. Seleccionando el directorio raíz del proyecto

Agradecimientos

Realizar cualquier trabajo lejos delhogar es de por sí una tarea que para lograr cumplirla se debe de tener dedicación y convicción para realizarla y cumplirla. Además de esto, contar con el apoyo de personas sirve como base, soporte y apoyo moral que son elementos fundamentales para poder culminar una meta. Es por ellos que quisiera agradecer a:

- A mis padres por apoyarme a venir a México para recibir el curso del CRECTEALC.
- Al colectivo de profesores del CRECTEALC por las clases impartidas y por darnos siempre apoyo cuando lo he necesitado.
- Al Dr. Jesús Gonzalez Bernal y al Dr. Enrique Muñoz de Cote por asesorar el proyecto.
- Al Dr. Ansel Rodríguez González por brindar sus sugerencias y apoyar en la realización del proyecto.
- A mis compañeros de curso con los que he compartido un año no solo en clases, sino también un año de vida y de relación de amistad y compañerismo.
- A la familia Ramos Rayas por recibirme en Puebla y dar su apoyo cuando lo he necesitado.
- A Esther Meléndez Abrego quien no solo ha compartido la mesa en el comedor, sino que ha compartido su amistaddesinteresada.

A todos los que no he mencionado en esta lista, ya que si no sería interminable, han contribuido a que fuera posible la realización de este proyecto.