Simulación de Sistemas Sociales con Agentes

Juan Pavón Mestras

jpavon@fdi.ucm.es

Dep. de Ingeniería del Software e Inteligencia Artificial

Universidad Complutense Madrid

http://grasia.fdi.ucm.es

Ciencias sociales

- Las ciencias sociales estudian los aspectos humanos del mundo
- Abarcan múltiples disiciplinas interrelacionadas
  - Antropología, estudios culturales
  - Comunicación, lingüística
  - Derecho, normas sociales
  - Economía, producción, distribución, consumo
  - Historia, estudio e interpretación de fenómenos pasados
  - Demografía, dinámica de las poblaciones
  - Política, procesos de decisión, relaciones de poder
  - Psicología, comportamiento de los individuos
  - Sociología, comportamiento de las sociedades
- Y tiene relaciones con otros campos
  - Ciencias naturales: Sociobiología
  - Ciencias aplicadas: Ingeniería, TIC, Ciencias de la Salud, etc.

Ciencias sociales

- Estudio de fenómenos sociales
  - Análisis cuantitativo
    - Aspectos cuantificables, medibles
    - Series de números
    - Modelización matemática y modelos estadísticos
  - Análisis cualitativo
    - Investiga sobre las opiniones, comportamientos y experiencias de los sujetos
    - Análisis de contenidos del discurso
- Tradicionalmente confrontados, actualmente se tiende a buscar planteamientos más eclécticos
  - Los métodos cuantitativos pueden dar soporte al análisis cualitativo y viceversa

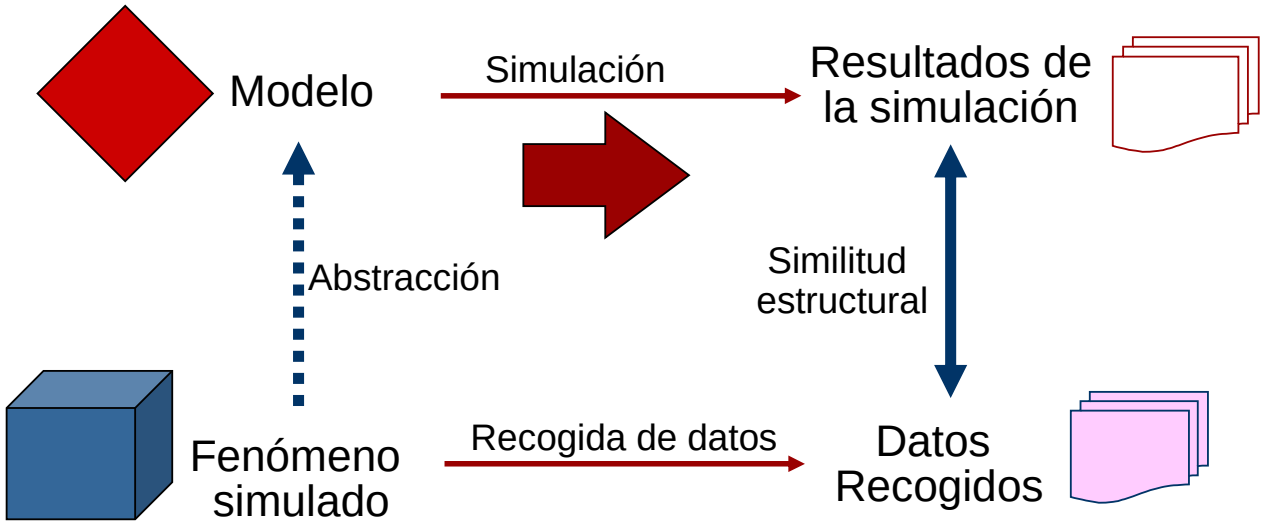
Simulación social

- Método de estudio que se basa en la construcción de un ***programa que modela el comportamiento de un fenómeno social***
  - Análisis del proceso
    - A lo largo del tiempo
  - Abstracción
    - No descriptivo
  - Macro y micro
    - No atomista
  - Experimental
    - No observacional

Simulación vs. Experimentación

- La experimentación consiste en aplicar pruebas o acciones a un grupo experimental objetivo para comparar el efecto que producen respecto a un grupo de control
  - A veces el grupo objetivo es demasiado complicado, demasiado inaccesible, o imposible por razones éticas o prácticas
    - ¿Se caerá un avión al modificar su estructura?
    - ¿Qué efecto tendrá en la población limitar el número de hijos?
    - ¿Cómo configurar la red de distribución de agua en una región?
- Por lo tanto, es más apropiado experimentar *sobre un modelo*
  - *Si el modelo es suficientemente bueno, reaccionará de manera similar al grupo objetivo*
  - El experimento se puede repetir muchas veces si el efecto varía aleatoriamente

Modelado y Simulación



De: Nigel Gilbert y Klaus G. Troitzsch, *Simulation for the Social Scientist*, Open University Press; 2 edition (1 Feb 2005)

Modelado y simulación con agentes

Sistema social

- Colectivo de individuos autónomos
  - Evolución autónoma
  - Motivados por sus propias creencias y objetivos personales
  - Y las circunstancias de su entorno
    - Todos estos factores evolucionan con el tiempo
  - Evolución demográfica
- Interactuando entre sí (directamente o a través del entorno)

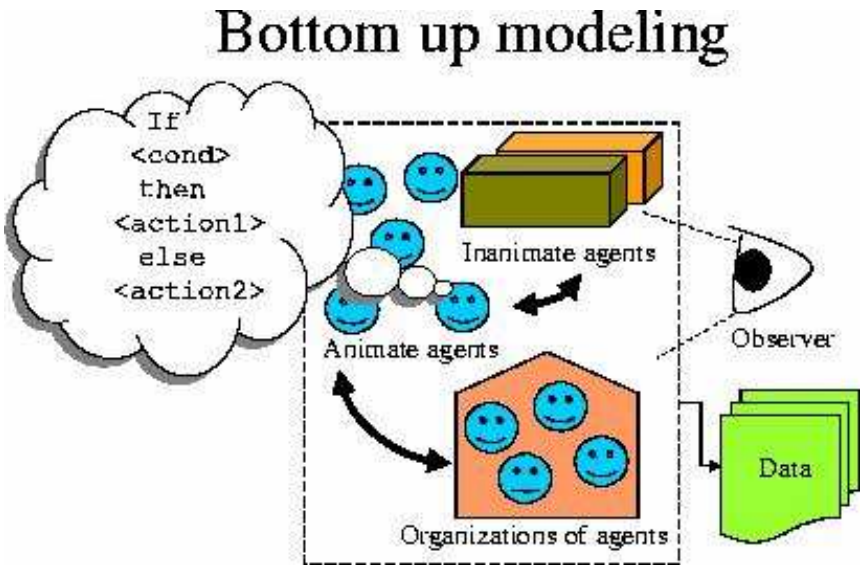
Sistema multi-agente

Simulación social basada en agentes

El modelo del programa es un sistema multi-agente

Simulación social basada en agentes

- Se modelan los elementos del sistema social: agentes
  - Planteamiento *bottom-up*: Agent Based Modelling (ABM)
- Se ejecuta para observar el comportamiento emergente



Simulación social basada en agentes

- La simulación consiste en la ejecución de agentes en un entorno de simulación
  - Los agentes modelan tipos específicos de comportamiento
  - Los agentes interaccionan
    - Directamente (mensajes)
    - A través del entorno (pizarra, feromonas, etc.)
  - Como resultado se ve la evolución de los agentes
    - Visualización de la simulación
    - Gráficas de resultados
    - Logs de la ejecución
- Los agentes tienen una percepción subjetiva
  - Los agentes no tienen un conocimiento global
  - Es más realista, flexible y sencillo si un agente sólo puede ver su vecindad

Simulación social basada en agentes

- Sobre esta base se han desarrollado herramientas de modelado y simulación basada en agentes
  - Ejecución de agentes en un entorno observable
  - Facilita la realización de estudios empíricos de sistemas sociales
- Retos
  - Modelo de los individuos
  - Modelo de las interacciones
  - Modelo de las normas sociales
  - Modelo del entorno
  - Interpretación de los resultados
- Limitación al estudio de procesos sociales concretos en un contexto sistémico y activo

Simulación social basada en agentes

- Como la simulación orientada a objeto, permite modelar la realidad simulada preservando la estructura
- Además tiene varias ventajas:
  - Control distribuido (permite computaciones paralelas en varias máquinas)
  - Permite simular comportamiento pro-activo
  - Permite añadir o quitar entidades durante la simulación
  - Es fácil intercambiar un agente con la entidad simulada correspondiente (p.ej., una persona o una máquina física) haciendo los escenarios de simulación muy dinámicos
  - Facilita la simulación de comportamiento en grupos en situaciones altamente dinámicas
    - Especialmente interesante con comportamientos emergentes

Aplicaciones de la simulación social

- Mejor entendimiento de los fenómenos sociales
  - Observando su evolución
  - Diagnóstico
- Descubrimiento de comportamientos emergentes
- Formalización y validación de teorías sociales
  - Del texto informal al modelo computacional
- Predicciones
  - Determinar cómo va a evolucionar una sociedad en aspectos específicos
- Entrenamiento
  - Modelos económicos: http://www.bized.co.uk/virtual/
- Entretenimiento
  - Juegos

A los científicos sociales les interesa más entender que predecir

Modelos de simulación con agentes

- Modelos conceptuales
  - Investigan patrones de comportamiento
  - Son en general más simples y permiten un análisis más riguroso del comportamiento
  - Tratan de responder a la pregunta: ¿Qué determina la cooperación?
- Modelos realistas
  - Prototipos del mundo real
  - Se centran en sistemas específicos y son por lo general más complejos
  - Se validan con datos reales

Modelado con agentes reactivos o cognitivos

Modelo de Mercado: equilibrio de Walras

- Los productores y los consumidores son los participantes del mercado
- La única información intercambiada es el precio y la demanda
- La información está centralizada –todas las tiendas publican sus ofertas
- Todos los consumidores publican sus demandas en función del precio (relación fijada como curva de precio-demanda)
- El mercado tiende hacia un equilibrio perfecto donde la oferta y demanda tienden a equilibrarse y los beneficios tienden a cero
- Existen herramientas analíticas para tratar el equilibrio del mercado y las condiciones para conseguirlas

Mercado con agentes reactivos ..

- Consumidores
  - Estado – grado de necesidad, presupuesto (infinito)
  - Action de comprar:

IF necesidad < límite de satisfacción
THEN escoge una tienda al azar (Escoge tienda X)

IF precio (pregunta precio, tienda X) < límite permitido
THEN compra 1 unidad AND incrementa el límite permitido AND adjusta el grado de necesidad AND reduce presupuesto.
ELSE reduce el límite permitido.

Mercado con agentes reactivos

- Productores
  - Estado – stock de bienes, precio, ventas en un periodo, presupuesto (limitado)
  - Action :

IF ventas < ventas periodo anterior
THEN bajar precio
ELSE subir precio

IF stock de bienes < ventas último periodo AND presupuesto > precio unidad
THEN comprar unidad AND decrementar presupuesto AND incrementar stock

Mercados con agentes cognitivos

- Consumidores
  - Estado – demanda, presupuesto (infinito), conocimiento acerca de compras anteriores
  - Compara las ofertas de diferentes tiendas
  - Pero recolectar información es consumir recursos
    - Por tanto, debe haber una regla para parar esta acción (satisfacer p.ej. un umbral junto con un número máximo de tiendas)
  - Las tiendas no se eligen de manera aleatoria sino que se intercambia información con otros agentes sobre las tiendas más baratas

Mercado con agentes cognitivos

- Productores
  - Estado – stock de bienes, precio, ventas en un período, beneficio, presupuesto (limitado)
  - Calcula los beneficios esperados y establece el precio para maximizar el beneficio (p.e. podría basarse en su experiencia anterior entre precio y ventas)
  - Intenta minimizar el stock de bienes – el almacenaje cuesta dinero
  - Puede comparar sus resultados con otros productores, mediante comunicacion, e imitarlos

Posibles motivaciones de los agentes

- Los agentes pueden maximizar sus beneficios o perseguir objetivos individuales
- Los agentes pueden obedecer normas sociales y reglas compartidas colectivamente
- Los agentes pueden imitar a otros agentes

Movimiento e interacción en el espacio

- Relevancia de las interacciones locales
  - Las interacciones humanas están influidas por su lugar en el espacio
  - Las interacciones locales son más importantes que las distantes
- Los agentes están localizados en el espacio, con capacidad para moverse
  - Existen reglas para decidir sobre dicho movimiento
- Reconocen otros agentes, si son similares o no
  - Son capaces de exibir diferente tipo de comportamiento basado en el grado de similaridad de otros agentes
  - Se pueden establecer ***redes sociales***, que determinan relaciones entre grupos de agentes

Metodología de la simulación social

1. Definir el tema y el problema
2. Especificar las hipótesis
3. Listar supuestos
4. Diseñar y construir un modelo
5. Verificar el modelo
6. Validar el modelo
7. Extraer conclusiones

De: Nigel Gilbert y Klaus G. Troitzsch, *Simulation for the Social Scientist*, Open University Press; 2 edition (1 Feb 2005)

Ejemplos

Etnocentrismo

Teoría del etnocentrismo

- *Etnocentrismo* es un síndrome universal de actitudes y comportamiento que implica cooperar con los miembros del grupo y no cooperar con los que no son del grupo
  - Favoritismo hacia los del mismo grupo
- Axelrod y Hammond (2005) sugieren que el comportamiento etnocentrista puede evolucionar bajo una amplia variedad de condiciones
  - Emergencia de la predisposición a cooperar con miembros del grupo
  - Incluso con poca capacidad cognitiva

Ejemplo con Netlogo

- Modelo de Axelrod y Hammond (2005)

http://ccl.northwestern.edu/netlogo/models/Ethnocentrism

- Para aislar los aspectos fundamentales del favoritismo inter-grupal se define un modelo de cooperación y competición lo más simple posible (principio KISS)
- Los agentes compiten por un espacio limitado por medio de iteraciones del *dilema del prisionero*
  - Los agentes etnocéntricos tratan mejor a los agentes de su grupo que al resto
    - La similitud se determina por alguna característica, como el color
  - El sistema incluye un sistema de herencia (genética o cultural) de comportamientos

El dilema del prisionero

- Problema de teoría de juegos
  - La policía arresta a dos sospechosos. No hay pruebas suficientes para condenarles, y tras haberles separado, les visita a cada uno y les ofrece el mismo trato: "Si confiesas y tu cómplice continúa sin hablar, él será condenado a la pena total, 10 años, y tú serás liberado. Si él confiesa y tú callas, tú recibirás esa pena y será él el que salga libre. Si ambos permanecéis callados, todo lo que podremos hacer será encerraros 6 meses por un cargo menor. Si ambos confesáis, ambos seréis condenados a 6 años.
  - Cada jugador está incentivado para traicionar al otro
    - Confesando siempre se tiene una pena considerable
  - Incluso aunque pudieras hablar con el otro, ¿te fiarías?

	Tú lo niegas	Tú confiesas
Él lo niega	**6 meses** para ambos	**10 años** para él y tú sales libre
Él confiesa	**10 años** para ti y él sale libre	Condena de **6 años** para ambos

Simulación del etnocentrismo

- Modelo de los agentes con tres características:
  - Color (grupo al que pertenece)
  - Estrategia con los de su mismo color (colabora o no)
  - Estrategia con los de un color diferente (colabora o no)
- Comportamiento:
  - Un agente etnocéntrico colabora con los del mismo color pero no con el resto (CD)
  - Un agente altruista coopera con todos (CC)
  - Un agente egoísta no coopera con nadie (DD)
  - Un agente cosmopolita coopera con los de otro color pero no con los suyos (DC)

C: cooperate, D: defect

Simulación del etnocentrismo

- Ciclos de la simulación
  1. Inmigración
     - Aparecen nuevos agentes de características aleatorias en posiciones aleatorias
       - Cada nuevo agente tiene un potencial de reproducción (PTR) inicial de 12%
  2. Interacción
     - Basada en el dilema del prisionero
       - Si ayudan pierden un 1% de PTR : la cooperación tiene un coste
       - Si reciben ayuda mejoran el PTR en 3%
  3. Reproducción
     - De manera aleatoria cada agente puede reproducirse:
       - Tiene que haber un espacio libre
       - El descendiente hereda las características del padre con una tasa de mutación del 5%
  4. Muerte
     - Todos los agentes tienen una probabilidad de morir (10%)
       - Para dejar espacio a nuevos agentes

Simulación del etnocentrismo

- Simulado con netlogo:
  - http://ccl.northwestern.edu/netlogo/models/Ethnocentrism

Los agentes aparecen como:

- Círculos si cooperan con los del mismo color, relleno si son altruistas o vacío si son etnocentristas
- Cuadrados si no cooperan con los del mismo color, relleno si son cosmopolitas o vacío si son egoístas

Ejecución

- El favoritismo aparece aunque no esté implementado en el modelo
  - Tras experimentar 2000 veces, tras 100 rondas, una media de 76% de agentes tienen una estrategia etnocéntrica
  - Y también el 74% de las operaciones son de cooperación
  - El etnocentrismo es un fenómeno robusto: aún cambiando muchos parámetros del modelo acaba manifestándose

Simulación basada en agentes

Herramientas

Herramientas de simulación basada en agentes

- Basadas en Java
  - Swarm (www.swarm.org)
    - Herramientas que ha influido en otras (Ascape, Mason, RePast)
    - Inicialmente en Objective-C, ahora también Java
  - **RePast** (repast.sourceforge.net)
  - **Mason** (cs.gmu.edu/~eclab/projects/mason/)
  - SeSAm (www.simsesam.de)
- Otras
  - Strictly Declarative Modeling Language, SDML (sdml.cfpm.org)
  - Multi-Agent Simulation Suite (mass.aitia.ai)
  - **NetLogo** (ccl.northwestern.edu/netlogo/)
    - Descendiente de StarLogo
    - Basado en el lenguaje Logo, fácil de usar
- Una buena lista en: http://www.econ.iastate.edu/tesfatsi/acecode.htm

SDML

- a Strictly Declarative Modelling Language
  - Manchester Metropolitan University (Steve Wallis, Scott Moss, Bruce Edmonds)
- Escrito en SmallTalk
- Representación declarativa del conocimiento (bases de datos y reglas)
- Composición de agentes para formar otros agentes
- Encadenamiento hacia delante o atrás de las reglas

  - Interfaz y lenguaje de modelado no triviales
  - No ha evolucionado mucho en los últimos años (parado?)

SeSAm

- **S**h**e**ll for **S**imulated **A**gent Syste**m**s
  - Universität Würzburg (Franziska Klügl)
- Permite el modelado visual de los agentes
  - Los agentes tienen un cuerpo que contiene varias variables de estado y un comportamiento que se especifica como un diagrama de estados tipo UML
  - Hay un conjunto de componentes primitivos que se pueden asociar

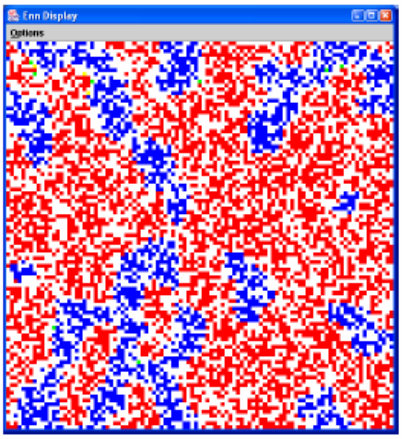
RePast

- RePast (*Recursive Porous Agent Simulation Toolkit*)
  - Inicialmente desarrollado por Sallach, Collier, Howe, North et al. en la Universidad de Chicago
  - Recientemente Repast Simphony (o Repast S)
  - Actualmente su evolución la gestiona una organización de voluntarios llamada ROAD (Repast Organization for Architecture and Development), y recibe aportaciones tanto del mundo académico como de la industria
    - Proyecto de código líbre que se distribuye con licencia BSD
    - http://repast.sourceforge.net
- Evolución de Swarm
  - Proporciona implementaciones puras en varios lenguajes
  - Nuevas características como algoritmos genéticos o regresión

RePast

- Arquitectura orientada a objetos
  - Plantillas de agentes y del entorno
- Interfaz gráfica de usuario
  - Control de la simulación
  - Gestión de parámetros del modelo
  - Visualización del comportamiento (gráficos 2D)
  - Gráficas estadísticas
- Utilidades y librerías
  - Espacios de interacción
  - Generación de números aleatorios y utilidades matemáticas
  - Algoritmos genéticos, redes neuronales
- Redes sociales (network models)
- Integración con Sistemas de Información Geográfica (GIS)
- Multilenguaje y multiplataforma
  - Java, C#, Managed C++, Visual Basic.Net, Managed Lisp, Managed Prolog, Python
  - Hay versiones para Windows, Linux y MacOS
- Muchos ejemplos y documentación

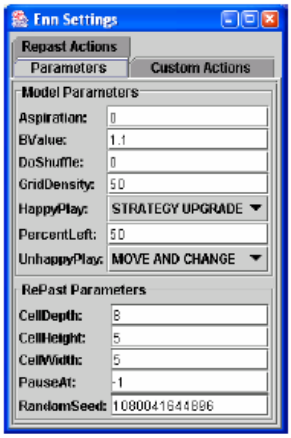
Repast framework



Displaying behavior

Controlling simulations

Charting



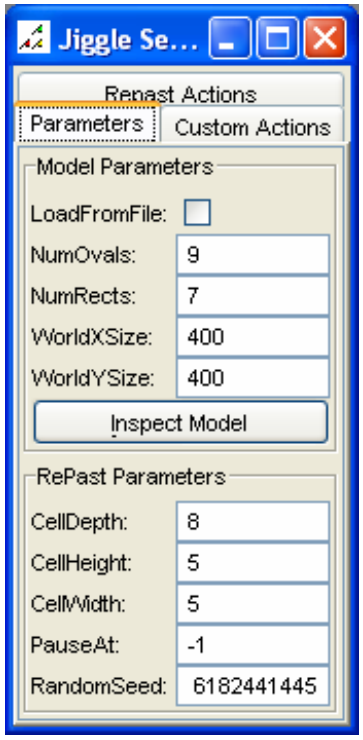
Managing parameters

RePast: herramientas

- Ejecución de simulaciones
  - Modo visual o interactivo
    - Permite ver cómo evoluciona el modelo, pausar la simulación, leer o modificar propiedades del modelo o de los agentes, etc.
  - Modo no visual
    - La simulación se ejecuta sin la sobrecarga de un entorno visual y almacena los datos deseados en ficheros
  - Control de la ejecución:

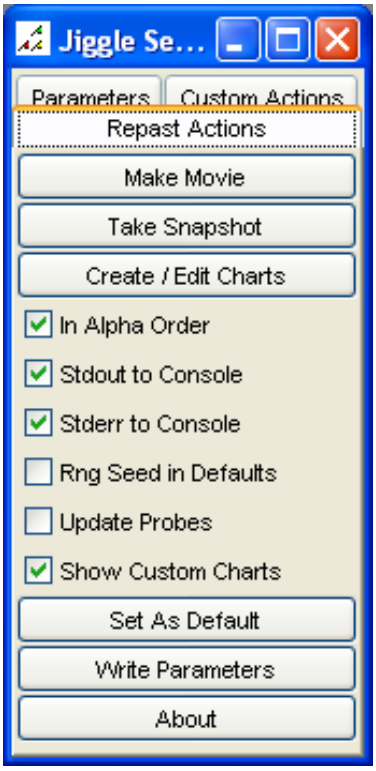
RePast: herramientas

- Parámetros del modelo
  - Propiedades configurables del modelo



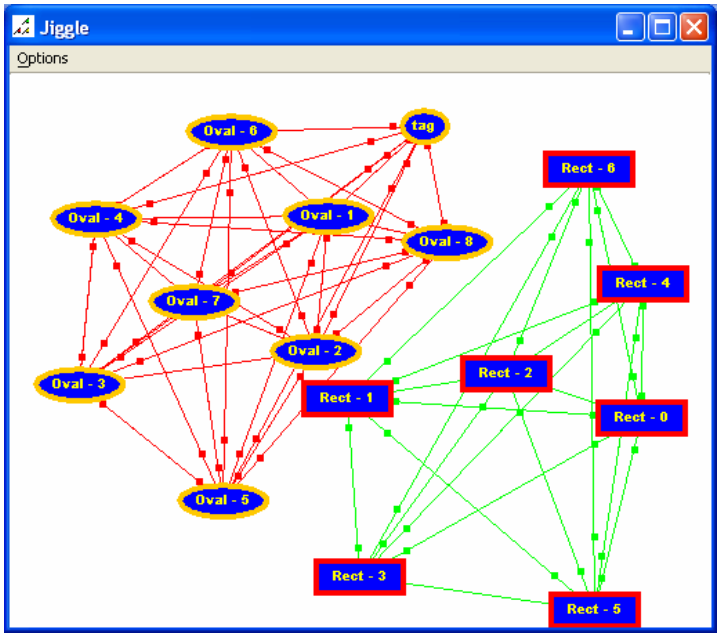
RePast: herramientas

- Acciones estándar de Repast
  - Tomar una foto (snapshot)
  - Grabar una película
  - Editar y crear diagramas
  - Salvar parámetros
- Acciones propias del modelo



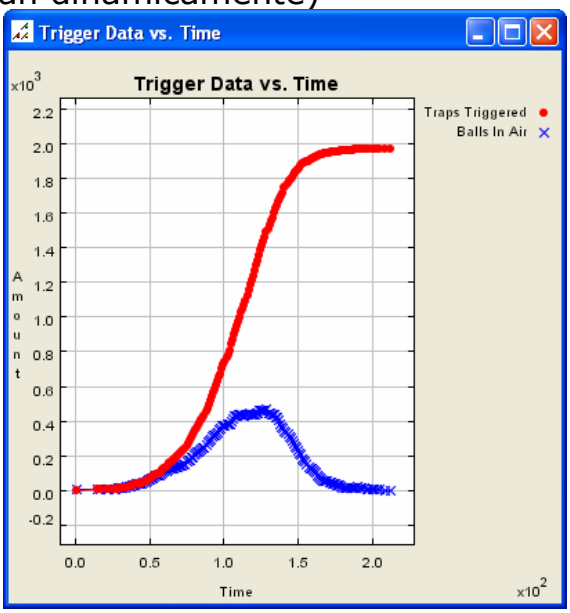
RePast: herramientas

- Display
  - Muestra la ejecución de la simulación



RePast: herramientas

- Diagramas
  - Muestran la evolución de ciertos parámetros a lo largo del tiempo (varían dinámicamente)



RePast: Construcción de un modelo

- La simulación avanza por pasos de tiempo (ticks)
  - Hay un primer paso en el que se inicializa el sistema (setup)
  - Cada tick sucesivo (step) se ejecuta una acción basada en resultados de acciones previas
- Normalmente un modelo RePast requiere tres tipos de clases:
  - Agente
    - Una clase por cada tipo de agente: describe el comportamiento del agente y las características que lo definen
  - Modelo
    - Coordina el inicio y ejecución de la simulación, la interfaz visual, etc.
  - Espacio
    - Controla el entorno en el que tienen lugar las acciones

RePast: El modelo

- El modelo
  - Extiende uchicago.src.sim.engine.SimpleModel
    - Gestiona el entorno y la simulación
    - Es el encargado de reaccionar a los botones de la barra de comandos
      - Por ejemplo, el método begin() es el que se invoca al iniciar la simulación
    - Proporciona los métodos setup() y buildModel() para inicializar la simulación
  - La forma más sencilla de crear un modelo es extender SimpleModel

    public class MyModel extends SimpleModel { ... }

    - Implementando los métodos setup(), buildModel() y step()

RePast: El modelo

```
import uchicago.src.sim.engine.SimpleModel;


public class MyModel extends SimpleModel {
    public String getName(){    return “Nombre del modelo";  }
    // ...
    public void setup() { ... };   // Inicializar el modelo: asignar valores iniciales
    public void step() { ... };    // Se ejecuta cada tick (unidad de tiempo) de la simulación
    // ...
    public void begin(){
       buildModel();
       buildSchedule();
       buildDisplay();
     }
    public void buildModel(){ ...  }
    public void buildSchedule(){ ...  }   // define un planificador para la simulación (opcional)
    public void buildDisplay(){ ...  }    // visualización de la simulación (opcional)
```


RePast: El modelo

```
public class MyModel extends SimpleModel {
  public static final int TIT_FOR_TAT = 0;
  public static final int ALWAYS_DEFECT = 1;
  private int  p1Strategy = TIT_FOR_TAT;
  private int p2Strategy = ALWAYS_DEFECT;
  private DisplaySurface dsurf;


 public void setup() {          // se llama una vez, al inicio (al pulsar el botón setup)
   super.setup();               // inicialización por defecto de los modelos
   p1Strategy = TIT_FOR_TAT;        // inicialización específica para el problema
   p2Strategy = ALWAYS_DEFECT;
 }
```



```
 private void buildDisplay() {  }  // Prepara el display donde dibujar la simulación
```

RePast: El modelo

- Para crear los objetos de la simulación (agentes y entorno) se utiliza el método buildModel() que se ejecuta tras setup()

```
public void buildModel() {
    // se ejecuta cuando empieza la simulación con los botones run, step o initialize
    // permite tener en cuenta los valores de los parámetros que ponga el usuario
    Player p1 = new Player(p1Strategy);
    Player p2 = new Player(p2Strategy);
    p2.setOtherPlayer(p1);
    p1.setOtherPlayer(p2);
    agentList.add(p1);
    agentList.add(p2);
    buildDisplay();
 }
```


RePast: El modelo

- En el método step() se suele iterar sobre todos los agentes para invocar un método que defina su comportamiento

```
public void step() {                    // Lo que ocurre en cada tick
   int size = agentList.size();
   for (int i = 0; i < size; i++) {
     Player p = (Player)agentList.get(i);
     p.play();
   }
 }
}
```

- Hay métodos preStep() y postStep() que definen lo que se hace antes y después
  - Por ejemplo para que cada agente reajuste su estrategia después de haber realizado una jugada

RePast: El modelo

- Otra posibilidad es que se itere automáticamente (auto step) por todos los agentes, para lo cual
  - Los agentes deben implementar el método step() de la interfaz Stepable
    - No se utiliza el método step() en el modelo
  - En el modelo se debe poner la variable autoStep a true

RePast: Ejecución del modelo

- La ejecución se puede controlar desde la barra de control
  - Al pulsar el botón setup se ejecuta el método setup()
  - Al pulsar el botón initialize se ejecuta el método buildModel()
  - Al pulsar el botón step se ejecuta el método buildModel() y una vez la secuencia de métodos preStep(), step() y postStep()
  - Al pulsar el botón start se ejecuta el método buildModel() y la secuencia de métodos preStep(), step() y postStep() se ejecuta repetidamente hasta que se pulse el botón stop o pause

RePast: Parámetros de un modelo

- Permite establecer parámetros para configurar la ejecución de un modelo
  - Métodos *setParametro()* y *getParametro()*
  - Se declaran en la variable *params* (de la clase SimpleModel)
  - Y aparecerán en el panel de parámetros

```
public class MyModel extends SimpleModel {
  public void setP1Strategy(int val) {  p1Strategy = val;  }
  public int getP1Strategy() { return p1Strategy;  }
 ...
 public MyModel() {
   params = new String[] {"P1Strategy"};
 }
 ...
```


RePast: Otros métodos de la clase SimpleModel

- setStoppingTime(long time)
  - indica cuantos ticks durará la simulación
- setRngSeed(long seed)
  - semilla para generación de números aleatorios (por defecto, 1)
- getNextIntFromTo(int from, int to)
- getNextDoubleFromTo(double from, double to)
  - devuelve un número aleatorio entre from y to
- atPause()
  - Se ejecuta al hacer pausa de la simulación
- atEnd()
  - Se ejecuta al acabar la simulación

RePast: Los agentes

- Deben implementar un método con el comportamiento del agente para cada paso de ejecución de la simulación
  - Se invoca desde el método step() del modelo
  - O implmentan el método step() en el modo auto step
- Para visualizarlo tiene que implementarse una interfaz Drawable
- Para ver su estado en ejecución debe implementar el patrón de método Accessor
  - Métodos get y set para las variables de estado
- No hay practicamente soporte para comunicación entre agentes

RePast: Planificador de tareas

- Todos los modelos utilizan un objeto planificador (Schedule)
  - Responsable de todos los cambios de estado durante la simulación
  - La planificación trata de ejecutar métodos sobre objetos en momentos determinados
- Para definir una planificación:

1. Definir las acciones planificables
   - Heredan de BasicAction o de ActionGroup, y ejecutan el método execute()

```
class MyAction extends BasicAction {
  public void execute() {
    agent.step(); // Acción que se desea ejecutar: invocar el método step en agente
  }
}
```

2. Definir el momento en que se ejecutarán las acciones

RePast: Planificador de tareas

- Ejemplo

```
public void step() { ... };

private void buildSchedule() {

  class MyAction1 extends BasicAction {
     public void execute() { ... }
  }

  class MyAction2 extends BasicAction {
     public void execute() { ... }
  }

   schedule.scheduleActionBeginning(1, new MyAction1());
   schedule.scheduleActionAtInterval(1, this, “step”);
   schedule.scheduleActionAtInterval(100, new MyAction2(),  Schedule.LAST);
}
```


RePast: El mundo

- RePast proporciona varios tipos de mundos
  - Se pueden utilizar directamente o construir otros heredados de ellos
- Spaces
  - Definen el mundo en el que van a “vivir” los agentes
  - Son contenedores de agentes que definen las posiciones relativas entre ellos y el resto de entidades del mundo
  - Todos los espacios proporcionan métodos para insertar y eliminar elementos utilizando las coordenadas correspondientes. También permiten otras operaciones útiles como obtener una lista de los elementos de las celdas vecinas, etc.

RePast: El mundo

- Espacios discretos:
  - Object2DGrid: cada celda puede contener un objeto.
  - Multi2DGrid: cada celda puede contener varios objetos (sin orden).
  - OrderedMulti2DGrid: cada celda puede contener varios objetos (FIFO).
- Espacios discretos toroidales:
  - Diffuse2D: cada celda puede contener un double. Aproximación discreta a la difusión 2D
  - Object2DTorus: cada celda puede contener un objeto.
  - Multi2DTorus: cada celda puede contener varios objetos (sin orden).
  - OrderedMulti2DTorus: cada celda puede contener varios objetos (FIFO)

RePast: El mundo

- Espacios con celdas hexagonales:
  - Diffuse2DHexagonal: cada celda puede contener un double. Aproximación discreta a la difusión 2D.
  - Object2DHexagonalGrid: cada celda puede contener un objeto.
  - Multi2DHexagonalGrid. cada celda puede contener varios objetos (sin orden).
  - OrderedMulti2DHexagonalGrid: cada celda puede contener varios objetos (FIFO).
- Espacios toroidales con celdas hexagonales:
  - Object2DHexagonalTorus: cada celda puede contener un objeto.
  - Multi2DHexagonalTorus: cada celda puede contener varios objetos (sin orden).
  - OrderedMulti2DHexagonalTorus. cada celda puede contener varios objetos (FIFO).
- RasterSpace
  - Espacio 2D que permite representar datos geográficos. Cada celda puede contener un objeto. El movimiento en este espacio puede expresarse en distancia, coordenas o celdas discretas.

RePast: El mundo

- Ejemplo

```
void buildModel() {
  ...
  Object2DGrid space = new Object2DGrid(spaceWidth, spaceHeight);

  for (int i = 0; i < numAgents; i++) {
    int x, y;
    do {
      x = Random.uniform.nextIntFromTo(0, space.getSizeX() - 1);
      y = Random.uniform.nextIntFromTo(0, space.getSizeY() - 1);
    } while (space.getObjectAt(x, y) != null);

    MyAgent agent = new MyAgent(x, y, space);
    space.putObjectAt(x, y, agent);
    agentList.add(agent);
  }
  ...
}
```


RePast: Displays

- Representan gráficamente los agentes y su entorno
- Se crean mediante 3 tipos de clases:
  - Spaces: representan el mundo.
  - Displays: correspondientes al space elegido. Controla como se va a visualizar.
  - DisplaySurface: representa la superficie donde se dibuja.
- Todos los elementos visuales (tanto agentes como elementos del entorno) deben implementar ciertas interfaces para que puedan ser representados gráficamente

```
public void buildDisplay() {
  Object2DDisplay agentDisplay = new Object2DDisplay(world);
  agentDisplay.setObjectList(agentList);

  dsurf.addDisplayableProbeable(agentDisplay, "Agents");
  addSimEventListener(dsurf);
}
```

  - "world" es un space de tipo Objet2DTorus poblado con agentes, "agentList" es una lista de agentes, y "dsurf" es la DisplaySurface

RePast: Displays

- Normalmente el planificador (schedule) invocará el método updateDisplay del objeto DisplaySurface cada cierto tiempo para actualizar la escena:
  - El DisplaySurface indica a todos los displays que contiene que deben pintarse
  - Cada display recorre la lista de objetos de su space obteniendo cierta información (coordenadas, etc) y prepara la superficie de dibujo.
  - Cada display recorre la lista de objetos de su space y les pide que se dibujen
  - Cada objeto se dibuja a si mismo
- El objeto DisplaySurface proporciona facilidades para sacar "fotos" o grabar videos de la simulación.

```
dsurf.setSnapshotFileName(some_fileName);
schedule.scheduleActionAtInterval(100, dsurf, "takeSnapshot");

dsurf.setMovieName(movie_name, DisplaySurface.QUICK_TIME);
schedule.scheduleActionAtInterval(10, dsurf, "addMovieFrame");
schedule.scheduleActionAtEnd(dsurf, "closeMovie");
```


RePast: Recogida de datos de la simulación

- Durante la simulación puede ser útil anotar datos en ficheros de texto, para posteriormente procesarlos. Para hacerlo se utiliza el objeto DataRecorder que puede recoger datos de varias fuentes y escribirlos en ficheros.
- Se pueden añadir distintas fuentes de datos al DataRecorder que indiquen de donde recoger los datos deseados. Las fuentes de datos pueden ser de dos tipos: numéricas (int, float, double, etc) u objetos (strings, etc).

```
public class MyModel extends SimModelImpl {

  private int numAgents;
  private Space space;
  private DataRecorder recorder;
  ...
  public int getNumAgents() {  return numAgents; }

  public String getSpaceData() {  return space.getData(); }
```

RePast: Recogida de datos de la simulación

```
private void buildModel() {
    recorder = new DataRecorder("./data.txt", this);
    recorder.createNumericDataSource("numAgents", this, "getNumAgents");
    recorder.createObjectDataSource("SpaceData", this, "getSpaceData");
     ...
  }

  private void buildSchedule() {
     ...
    class CollectData extends BasicAction {
    public void execute() {
      dRecorder.record();
    }
  }

    schedule.scheduleActionBeginning(0, new CollectData());
    schedule.scheduleActionAtEnd(dRecorder, "writeToFile");
  }
}
```


RePast: Recogida de datos de la simulación

- Los ficheros de datos comienzan con una descripción de los parámetros iniciales de la simulación y después contienen los datos recogidos:

```
RngSeed: 948816295132
numAgents: 100
maxAge: 120

"tick","Avg. Age"
0.0,33
1.0,33
2.0,32
3.0,40
...
```

RePast: Números aleatorios

- La clase Random permite generar secuencias aleatorias que sigan distintas distribuciones estadísticas. Esta clase encapsula la parte de generación de secuencias aleatorias de la librería colt (Open Source Libraries for High Performance Scientific and Technical Computing in Java) [10]

- Distribuciones disponibles: beta, binomial, exponencial, logaritmica, gamma, normal, poisson, uniforme, etc.

```
Random.setSeed(1L);
Random.createUniform();
int index = Random.uniform.nextIntFromTo(1, 10);
```

- Los números aleatorios se utilizan con mucha frecuencia en las simulaciones por lo que tiene gran importancia la distribución elegida y sus propiedades estadísticas.

RePast: Network Models

- Modelan sociedades en forma de redes compuestas por nodos y aristas direccionales. Normalmente los nodos representan los agentes y las aristas relaciones entre ellos

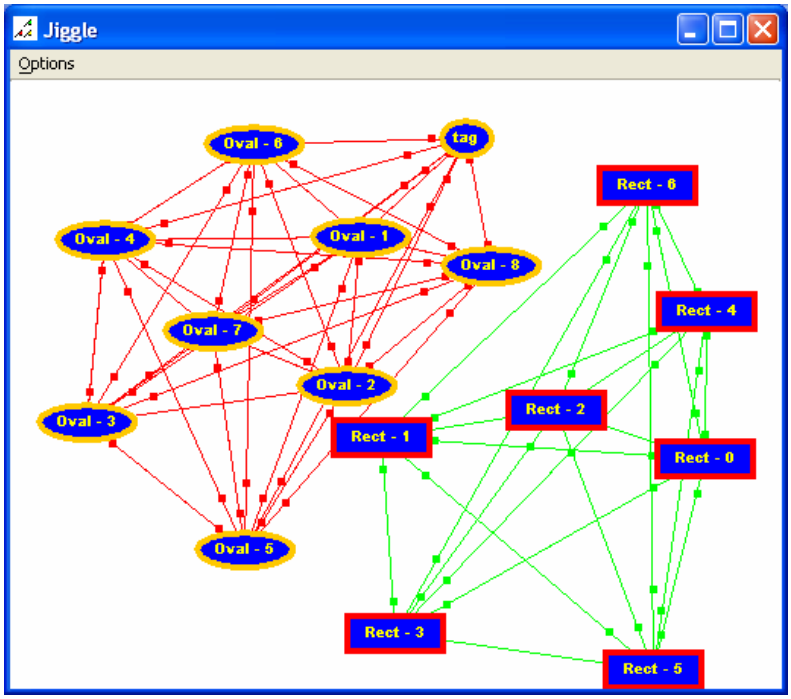
- La clase NetUtilities implementa varias operaciones comunes sobre redes. Sin embargo en la documentación advierten que aún no está muy madura y los datos obtenidos pueden no ser del todo precisos

- Para obtener datos más precisos recomiendan usar otros paquetes de software especializados como UCINET o Pajek

- Representar redes puede resultar bastante complicado si se desea obtener visiones representativas e intuitivas. ResPast proporciona varias clases especializadas en representar este tipo de redes (GraphLayout) que actúan de intermediarias entre la red y el Network2DDisplay

RePast: Network Models

- RePast proporciona métodos para escribir y leer la estructura de una red en ficheros ASCII, Excel o UCINet's dl.
- NetworkRecorder permite guardar una red en un fichero.
- NetworkFactory permite reconstruir una red desde un fichero.



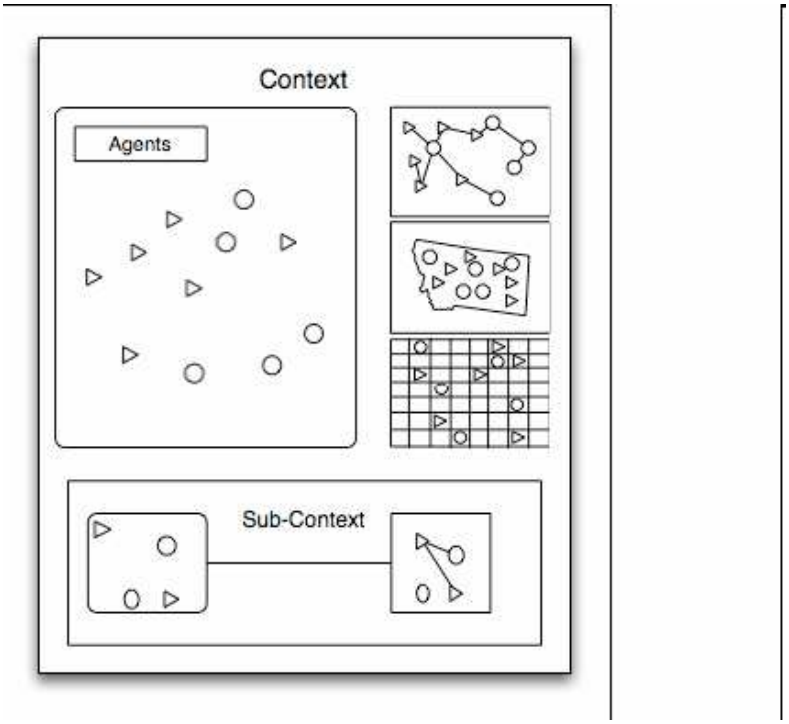
RePast: Integración con GIS

- RePast puede integrase con dos sistemas GIS: ESRI ArcMap y OpenMap
- Define dos clases principales, una para leer y escribir datos en el formato del sistema GIS y otra para presentar la información en el display.
- La interacción se realiza utilizando el siguiente esquema:
  - Leer datos en formato GIS y convertirlos a la representacion de RePast.
  - Ejecutar un perido de la simulación.
  - (Escribir los datos a un fichero de comunicación.)
  - Indicar al sistema GIS que debe actualizar el display con la nueva información.

El framework Repast Simphony

- Proto-agentes
  - Entidades de modelado con un conjunto de propiedades y comportamiento
  - Pero que no tienen capacidad de aprendizaje
    - Para ser agentes se les tendría que definir esta capacidad
- Contexto
  - Proto-espacio (contenedor) de la población de agentes
  - Tienen Data fields (campos n-dimensionales de valores) para la interacción entre los agentes
  - Pueden tener comportamientos asociados
- Proyecciones
  - Definen relaciones entre los proto-agentes en un contexto

El framework Repast Simphony



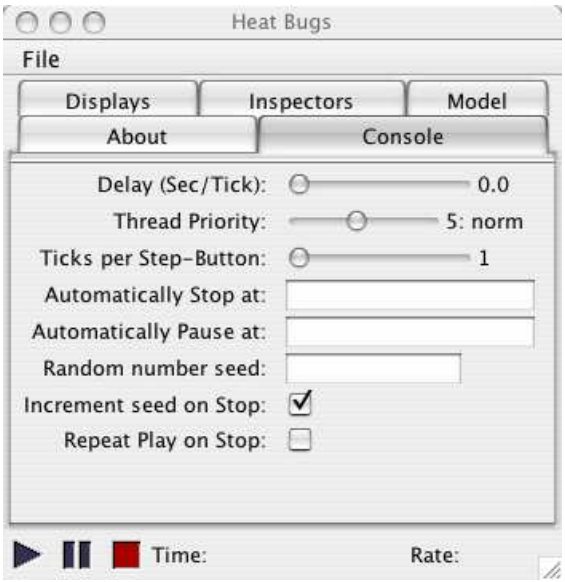
MASON

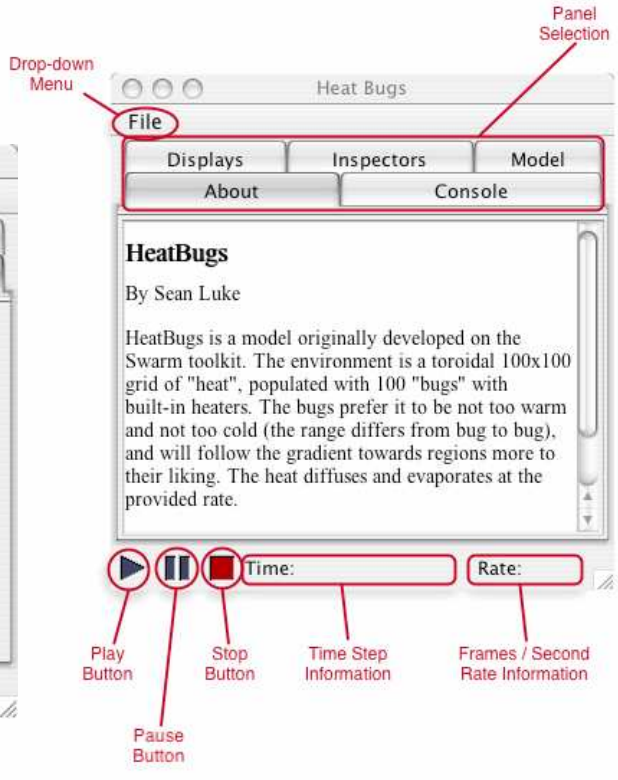
- **M**ulti-**A**gent **S**imulator **O**f **N**eighborhoods... or **N**etworks... http://cs.gmu.edu/~eclab/projects/mason/
  - Desarrollado conjuntamente por George Mason University's ECLab Evolutionary Computation Laboratory (C. Cioffi-Revilla) y GMU Center for Social Complexity (S. Luke)
- Librería de simulación multiagente de eventos discretos en Java
  - Orientada a simulaciones Java de pequeña y gran escala
  - Separa los modelos de las visualizaciones
  - Contiene una librería de modelos y un conjunto opcional de herramientas de visualización en 2D y 3D
  - Puede generar fotos (PNG) o películas (QuickTime)
  - Bastante ligero
    - El siguiente applet, que incluye la librería, tiene 800 KB

    http://cs.gmu.edu/~eclab/projects/mason/applet/

MASON

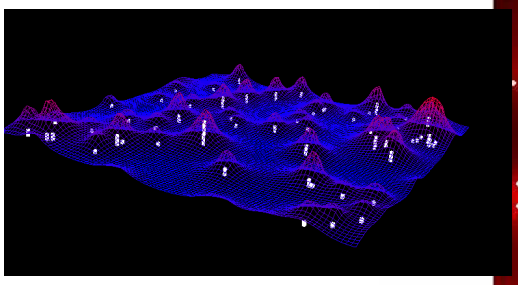
- Consola

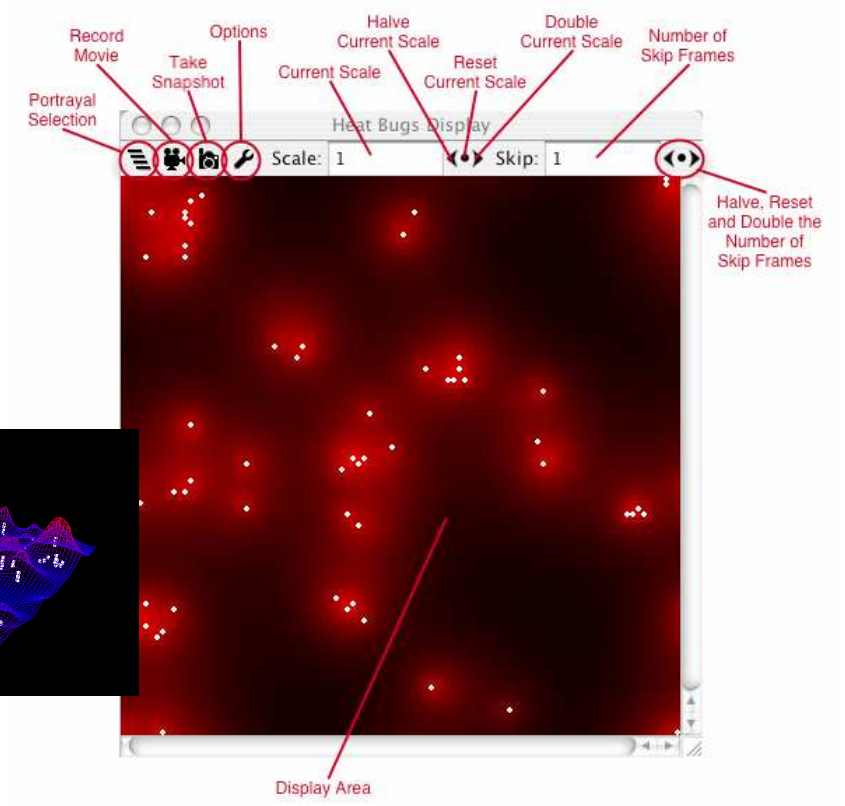




MASON

- Display





MASON vs. RePast

- MASON
  - Separación del modelo y la visualización
  - Modelos 3D y displays
  - Más rápido (?)
  - Más pequeño (tamaño del código)
- RePast
  - GIS
  - Importación y exportación de tablas Excel
  - Gráficas
  - SimBuilder
  - Permite manipular los objetos en la simulación (por ejemplo, moverlos de sitio)

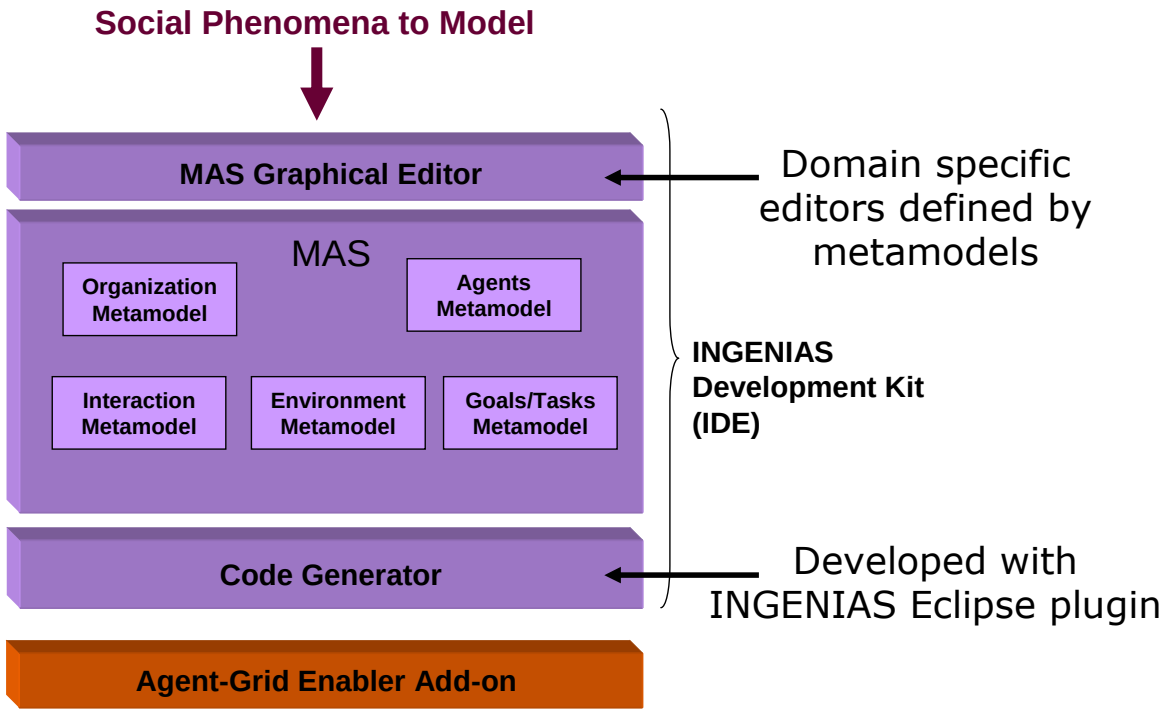
Cuestiones a investigar: Complejidad

- Complejidad de los modelos
  - Todos los modelos hacen una abstracción (ignoran) algunas características de la realidad
  - Cuanto más complejo sea el modelo
    - Más complicado será construirlo y validarlo
    - Más cercano será a la realidad
- ¿Qué nivel de abstracción adoptar?
  - KISS: Kit It Simple, Stupid (R. Axelrod, 1997)
  - KIDS: Kit It Descriptive, Stupid (B. Edmonds and S. Moss, 2004)
  - …

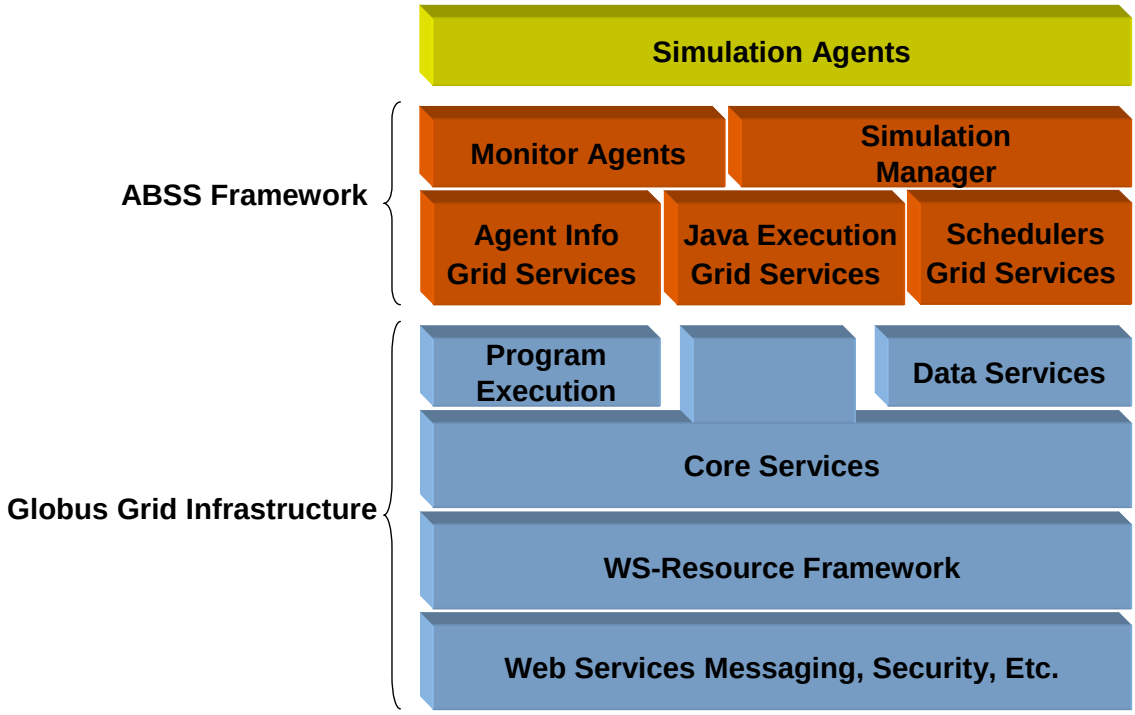
Cuestiones a investigar: Modelado vs. programación

- Herramientas de modelado
  - Los expertos en ciencias sociales
    - No son expertos en informática
    - Dificultad para utilizar las herramientas existentes
      - Requieren conocimientos de programación (p.ej. Java)
  - Repast ofrece con Repast Simphony facilidades para modelar sin codificar
  - Netlogo ofrece muchas facilidades
    - Modelos sencillos
  - MASS
    - Lenguaje funcional para describir modelos (FABLES)
    - Wizards para visualizar y analizar resultados
  - INGENIAS intenta adaptar el lenguaje de modelado de agentes a dominios de simulación social
    - Y luego generar código en plataformas de simulación usando técnicas de Model Driven Development

1: Facilitar el modelado (usando INGENIAS)



2: Ejecución en múltiples plataformas



Cuestiones a investigar: Escalabilidad

- (Cioffi-Revilla 2002) demuestran empíricamente que
  - El tamaño depende del tiempo en muchos sistemas sociales
  - Y el tamaño de un grupo o sistema influye en la evolución de sistemas o procesos con acción colectiva
    - Por ejemplo, el concurso del dilema del prisionero iterado
- Por esta razón es necesario poder escalar los modelos, sin limitar su tamaño

- Hemos hecho pruebas con RePast para un modelo de simulación de la bolsa (IBEX35) y soporta hasta 13000 agentes aprox.
  - Es importante la definición de un planificador avanzado para mejorar la eficiencia
  - Pero hay limitaciones de gestión de memoria

- Conclusión: es necesaria la distribución de la simulación
  - Una posibilidad es usar grid computing

Bibliografía

- Básico:
  - N. Gilbert y K.G. Troikzsch (2005). *Simulation for the Social Scientist*. Open University Press.
- Referencias:
  - R. Axelrod (1997). Advancing the art of simulation in the social sciences. Complexity, 3(2):16-22
  - B. Edmonds and S. Moss (2004). From KISS to KIDS - An 'Anti-simplistic' Modelling Approach. In P. Davidsson, B. Logan, and K. Takadama, editors, MABS, Lecture Notes in Computer Science 3415, Springer Verlag, 130-144.
  - C. M. Macal y M. J. North (2005). *Tutorial on Agent-Based Modeling and Simulation.* Proc. 2005 Winter Simulation Conference, pp. 2-14
  - Ross A. Hammond and Robert Axelrod (2005). *The Evolution of Ethnocentrism*, http://www-personal.umich.edu/~axe/research/Hammond-Ax_Ethno.pdf