RELEVAMIENTO DE LENGUAJES DE SIMULACIÓN

Sebastián Alaggia
18/3/2005

Índice

Metodología de selección de bibliotecas

1. Búsqueda de bibliotecas Open Source
2. Ficha técnica de las mismas
3. Preselección
4. Prueba de las bibliotecas

Posibles bibliotecas

- OMNet++
- C++SIM
- Adevs
- UCLA parsec
- DESMO-J
- DeX
- Ptolemy
- Java Sim
- Simkit
- simjava
- DSOL

Fichas técnicas

OMNet++

- **Página Web:** http://www.omnetpp.org/
- Descripción General: Es un ambiente para la simulación a eventos discretos. Esta más enfocado a la simulación de redes, pero ha sido utilizado para otros dominios de simulación (sistemas de colas, sistemas de trafico, seguir viendo…).
- **Lenguaje que utiliza:** C++
- **Entornos en los que corre:** Corre en sistemas tipo Unix y en Win32 (Windows 2000, XP).
- **Licencias:** El programa es gratuito para fines educacionales y de investigación. Se permite la modificación, distribución y copia de este producto, siempre que se adjunte a cada copia la licencia correspondiente.
- Estilos de simulación: Esta fuertemente orientado a simulación de redes de computadoras:
- **Documentación:**
- **Forma de distribución:**

C++SIM

- **Página Web:** http://cxxsim.ncl.ac.uk/
- **Descripción general:** Es una biblioteca de simulación a eventos discretos en C++. Maneja el enfoque de interacción de procesos, con una interfaz muy similar a SIMULA. Provee de un núcleo con rutinas de simulación parecidas a SIMULA (generadores de números aleatorios, algoritmos de colas y un paquete de interfaz de threads); manipulación de entidades y conjuntos; no maneja eventos tipo interrupciones; contiene histogramas y demás clases para estudiar los outputs de la simulación, y también clases para debuggear.
- **Lenguaje que utiliza:** Esta programada en C++ y utiliza threads. Soporta Sun's lightweight processes (lwp) library, Solaris threads, Posix threads, NT threads, C++ Task Library, C threads y Gnu Rex lightweight processes library
- **Entornos en los que corre:** Corre en SunOS 4, Solaris 2, HP9000s300 & HP9000s700 workstations que usan HPUX, Windows 95 y Windows NT (3.5.1 y 4.0), y Linux.
- **Licencias:** Se permite el uso, la copia, modificación y distribución del paquete para enseñanza e investigación en forma gratuita. La biblioteca es Open Source.
- **Estilos de simulación:** C++SIM tiene el enfoque de interacción de procesos.
- **Documentación:** Se tiene un manual de usuario y API. Estos están en la página, el primero en formato pdf, y el API en un formato tipo man.
- **Forma de distribución:** La biblioteca se encuentra para bajar en la página. Esta viene en un tar.gz, que contiene el código fuente, la documentación tipo man (API, hay que ver como anda) e instrucciones de instalación (probarla).

Adevs

- **Página Web:** http://www.ece.arizona.edu/~nutaro/
- Descripción general: Es una biblioteca de simulación a eventos discretos, su diseño esta basado en una variante del Parallel DEVS formalism
- **Lenguaje que utiliza:** Esta construida en C++.
- Entornos en los que corre: Aparentemente en Unix y Windows (no especifica que versión).
- **Licencias:** Es una biblioteca Open Source.
- **Estilos de simulación:** Maneja el formalismo DEVS.
- **Documentación:** Se tiene un manual de usuario y API. Estos están en la página, el primero todo esta en html.
- **Forma de distribución:** La biblioteca se encuentra para bajar en la página. Esta viene en un tar.gz, que contiene el código fuente, la documentación completa e instrucciones de instalación (probarla).

UCLA parsec

- **Página Web:** http://pcl.cs.ucla.edu/projects/parsec/
- **Descripción general:** Es una lenguaje de simulación a eventos discretos

desarrollada en C. Una característica especial de esta biblioteca es que puede ejecutar en arquitecturas paralelas y utilizando diferentes protocolos asincrónicos de simulación.

- **Lenguaje que utiliza:** Este paquete es un lenguaje de simulación que incorpora una gramática y sintaxis propia a C.
- **Entornos en los que corre:** Corre en Windows y en Linux
- **Licencias:** Es gratis para enseñanza e investigación universitaria
- **Estilos de simulación:** Interacción de procesos. Básicamente define entidades que se comunican por medio de mensajes.
- **Documentación:** Se tiene la documentación completa tanto online, en pdf, y se tienen también unas transparencias online.
- **Forma de distribución:** El compilador del lenguaje se encuentra disponible para bajar en la página. Se tienen instrucciones para la instalación de este también en la página.

DESMO-J

Página Web: http://asi-www.informatik.uni-hamburg.de/themen/sim/forschung/Simulation/Desmo-J/index.html

- **Descripción general:** Es un framework de desarrollo de simulación a eventos discretos. Está desarrollado en Java, soporta los enfoques de orientación a eventos e interacción de procesos. Provee de componentes como colas, muestreo de distribuciones basados en generadores de números aleatorios, y contenedores de datos; se dan clases abstractas como ser modelo, entidad, evento y procesos de simulación; y también se da la infraestructura necesaria para simular (scheduler, lista de eventos y clock) encapsulado en la clase Experiment.
- **Lenguaje que utiliza:** Java
- **Entornos en los que corre:** Debido a que esta desarrollado en Java (se supone que) es portable
- **Licencias:** GNU Lesser GPL
- **Estilos de simulación:** Soporta tanto el enfoque de interacción a procesos como orientación a eventos. En cuanto al enfoque a eventos, una simulación orientada a eventos (aparentemente) es de 2 fases, y la forma de programarla es parecida (aunque con una óptica O.O.) a Pascal Sim.
- **Documentación:** Se tiene tanto un tutorial del framework en la página, y además el API esta disponible para bajar en formato zippeado.
- **Forma de distribución:** De la página se puede bajar un el API, el código fuente, y un jar con la aplicación.

Amino

- **Página Web:** http://amino.sourceforge.net/
- **Descripción general:** Es una biblioteca de simulaciones que esta en desarrollo. Aparentemente esta implementada para modelos de Ingenieria mecánica, pero tratará (en futuras ediciones) de adaptarse a otros dominios.

No esta mencionado si esta biblioteca es para simulaciones a eventos discretos. Hay que tener en cuenta que Amino no tiene una distribución no oficial es un proyecto PRE-ALFA.

- **Lenguaje que utiliza:** Esta desarrollado en Java.
- **Entornos en los que corre:** Debido a que esta desarrollado en Java (se supone que) es portable.
- **Licencias:** Es un framework Open Source. Se permite la modificación y redistribución simple y cuando se verifiquen las condiciones de licencia del producto, como ser reconocer el copyright del autor).
- **Estilos de simulación:** no queda muy claro este punto.
- **Documentación:** Se posee tanto el API, el manual de usuario y manual de diseño de la biblioteca.
- **Forma de distribución:** De la página se puede bajar tanto el código fuente, como la librería compilada y la documentación. En el manual del usuario se tienen instrucciones de instalación del paquete.

DeX

- **Página Web:** http://dextk.org/
- Descripción general: Esta biblioteca se centra en dar una plataforma de simulación sencilla y visualmente potente para simulaciones con múltiples cuerpo. Se define en su núcleo mecanismos de sincronización y comunicación de eventos formando un motor de simulación continuo y discreto híbrido. Destaca la visualización 3D utilizando OpenGL.
- **Lenguaje que utiliza:** C++
- **Entornos en los que corre:** Linux con un procesador x86
- **Licencias:** Se da permiso de obtener copiar, modificar y distribuir esta biblioteca sin cargo siempre que se mantenga el permiso de copyright del producto.
- **Estilos de simulación:** híbrido continuo-discreto. En el enfoque discreto utiliza orientación a procesos.
- **Documentación:** Esta se encuentra en la página, y se muestran varios ejemplos.
- **Forma de distribución:** Se puede bajar de la página un tar.gz que contiene la librería además de varios ejemplos. Se tiene en la página un instructivo de instalación.

Ptolemy

- **Página Web:** http://ptolemy.eecs.berkeley.edu/
- Descripción general: Esta biblioteca esta especializada en crear ambientes (dominios). Estos se definen a partir de clases C++. Los dominios pueden ser de tipo simulación. Aparentemente se pueden simular sistemas de colas, además de redes de computadoras entre otros.
- **Lenguaje que utiliza:** C++
- **Entornos en los que corre:** Solaris2.5.1, HP-UX10.20, no ha sido probado

para otras plataformas.

- **Licencias:** Se da permiso de copia, modificación y distribución de Ptolemy, siempre que aparezcan el copyright del producto.
- **Estilos de simulación:** No se especifica claramente.
- **Documentación:** Se ofrece el manual de usuario, manual del programador y manual del kernel en diversos formatos.
- **Forma de distribución:** se puede bajar los binarios o el "código" desde la página. Se dan allí instrucciones de instalación.

Java Sim

- **Página Web:** http://javasim.ncl.ac.uk/
- **Descripción general:** Es una biblioteca de simulación a eventos discretos en Java, es predecesora de C++Sim (descrito arriba). Es orientada a interacción de procesos, con una interfaz muy similar a SIMULA.
- **Lenguaje que utiliza:** Esta programada en Java
- **Entornos en los que corre:** Corre en Solaris, Windows 95/NT 4.0 y Linux.
- **Licencias:** Se permite el uso, la copia, modificación y distribución del paquete para enseñanza e investigación en forma gratuita. La biblioteca es Open Source.
- **Estilos de simulación:** JavaSIM es orientado a procesos con el enfoque de interacción de procesos.
- **Documentación:** Se tiene un manual de usuario y API. Estos están en la página, el primero en formato pdf, y el API en un formato tipo man.
- **Forma de distribución:** La biblioteca se encuentra para bajar en la página. Esta viene en un tar.gz, que contiene el código fuente, la documentación tipo man (API, hay que ver como anda) e instrucciones de instalación (probarla).

Simkit

- **Página Web:** http://diana.gl.nps.navy.mil/Simkit/
- **Descripción general:** Esta biblioteca desarrollada en Java, está fuertemente relacionado con el modelo de grafo de eventos. Hay una traducción entre Simkit y estos grafos.
- **Lenguaje que utiliza:** Java
- **Entornos en los que corre:** Windows, Mac OS X, Linux, Unix.
- **Licencias:** GNU Lesser GPL
- **Estilos de simulación:** Extremadamente relacionado con grafo de eventos (investigar).
- **Documentación:** Se tiene el API de Simkit en http://diana.gl.nps.navy.mil/Simkit/doc/ Además esta librería es utilizada por una universidad y esta para bajarse los apuntes.
- **Forma de distribución:** De la página se baja un instalador de Simkit para la plataforma correspondiente.

Simjava

- **Página Web:** http://www.dcs.ed.ac.uk/home/hase/simjava/
- **Descripción general:** Esta es una biblioteca de simulación a eventos discretos, maneja el enfoque a interacción de procesos, y esta basada en Jade's Sim++ (?)
- **Lenguaje que utiliza:** Java
- **Entornos en los que corre:** Unix, Windows 95/NT
- **Licencias:** Simjava es Open Source.
- **Estilos de simulación:** interacción de procesos.
- **Documentación:** Toda la documentación esta en la página, y también en la distribución.
- **Forma de distribución:** Simjava es gratis, y su código fuente puede ser bajado desde la página. Primero se debe llenar un formulario, y luego se va a una página donde las distribuciones están disponibles. En dicha distribución se tiene la documentación completa que se encuentra en la página, (incluyendo el manual del diseñador), ejemplos y el código

DSOL

- **Página Web:** http://www.simulation.tudelft.nl/dsol/dsol/overview/goals.html
- **Descripción general:** Esta biblioteca desarrollada en Java se basa en el concepto de proveer las herramientas y técnicas de la simulación en entornos loosely-coupled, web-enabled services. Como caracteristicas adicionales soporta: animaciones 2D y 3D distribuidas, muestreo de distribuciones continuas y discretas, varios generadores de números aleatorios. En cuanto a la simulación a eventos discretos soporta simulación a eventos discretos distribuida e interacción a procesos en una sola thread. También soporta simulación continua.
- **Lenguaje que utiliza:** Java
- Entornos en los que corre: Debido a ser desarrollada en Java al concepto de su desarrollo debería ser portable.
- **Licencias:** Es Open Source.
- **Estilos de simulación:** Soporta interacción a procesos (no se si otro más).
- **Documentación:** Se tiene un tutorial que se baja de sourceforge.
- **Forma de distribución:** La librería se distribuye en un .jar que se baja de sourceforge.net. Allí se ofrece además un tutorial y ejemplos.

Sobre DesmoJ

De las bibliotecas de simulación que se encontraron, DesmoJ implementa varias de las funcionalidades que son requeridas para el curso "Simulación a Eventos Discretos". Debido a esto haremos aquí un estudio más detallado de dicha biblioteca.

Descripción General

DesmoJ es un framework para el modelado e implementación de simulaciones a eventos discretos. Se provee de dos de los enfoques más utilizados en este campo: Orientación a Eventos e Interacción de Procesos. DesmoJ esta bajo la licencia GNU Lesser GPL, por tanto se dispone de su código fuente y se permite hacerle modificaciones.

Esta biblioteca fue elegida pues fue la única encontrada que soportaba el enfoque de eventos, que es el enfoque seguido por el curso "Simulación a Eventos Discretos".

Modelado de sistemas en DesmoJ

El modelado de un sistema en DesmoJ, utilizando el enfoque de eventos, se realiza implementando clases derivadas de `Model`, `Event` y `Entity`.

`Model` es una clase abstracta que contiene el modelo del sistema a simular. En este se definen las colas de espera, las entidades y eventos, las distribuciones que rigen los aspectos aleatorios del sistema. Las operaciones que se deben implementar son:

- Constructor
- `init`: en esta operación se inicializan los aspectos estáticos del modelo, se deben crear instancias de las distribuciones, las colas y las entidades globales del sistema.
- `doInitialSchedules`: en esta operación realizan los primeros pasos para poder comenzar la simulación. O sea que se agendan los eventos que deben ponen en funcionamiento la simulación (los feeders, por ejemplo).
- `main`: Es el método principal, es aquí donde se realizan la corridas. Este método puede estar en esta clase o en otra aparte.

`Entity` es la superclase de todas las entidades del sistema. Se debe implementar un constructor y manejo de los atributos de la entidad (si es que tiene alguno).

Event es la superclase de los eventos de un sistema. En DesmoJ se definen 2 tipos de eventos:

- Internos al sistema: son aquellos que necesitan de una entidad para ser invocados. Este tipo de eventos heredan directamente de `Event`. Para este tipo de eventos se debe implementar la operación abstracta `eventRoutine(Entity)`, que define las acciones que se toman en el evento en cuestión.

- Externos al sistema: son aquellos que definen acciones externas al sistema. Son los eventos que generan los inputs del sistema. Estos heredan de la clase `ExternalEvent`. En estos eventos se debe implementar la operación abstracta `eventRoutine` que es equivalente al descrito anteriormente, solo que no necesita de una entidad para ser invocado.

Ejecución de una Simulación

Para correr una simulación se necesitan de dos clases, un Modelo (`Model`) y un Experimento (`Experiment`). En el modelo se encapsula el comportamiento del sistema a simular. O sea, en esta clase se definen las colas del sistema, las distribuciones que modelan la aleatoriedad del sistema, las entidades globales al sistema y (aunque no directamente) las entidades y eventos del sistema.

En el experimento se definen las herramientas que controlan la ejecución correcta de la simulación. Estas son el ejecutivo (`Scheduler`), el calendario (`EventList`) y el manejo de las distribuciones aleatorias (`DistributionManager`).

En el calendario se tienen objetos del tipo `EventNote`, que contienen una entidad y un evento. En el caso de orientación a eventos se tiene una entidad y un evento o una entidad y un proceso para la orientación a procesos (la superclase de `SimProcess` es `Entity`).

El proceso de ejecución de la simulación se realiza de la siguiente manera:

1. Se crea una instancia del modelo a simular
2. Se crea una instancia de experimento
3. Se conecta el modelo con el experimento por medio de la operación `conectToExperiment`. En este paso se ejecuta el método `init` definido en el modelo a simular.
4. Desde el modelo se invoca la operación `start` del experimento. En este método se prosigue de este modo:
    i. El experimento invoca el método `doInitialSchedules` del modelo.
    ii. Se invocan varios métodos intermedios (`start(SimTime)` y `proceed`) del experimento para poder comenzar con la simulación.
    iii. En el método `proceed` se le invoca al `Scheduler` el método `processNextEventNote`, que procesa el siguiente evento o proceso de la simulación mientras sea posible.

El método `processNextEventNote` del `Scheduler` realiza las siguientes acciones:

1. Se invoca la operación `firstNote` al calendario. Esta retorna una instancia `EventNote` que es la que debe ejecutar.
2. Luego se realizan los chequeos necesarios para saber si se trata de una entidad y un evento o de un proceso, y se invocan las operaciones abstractas `eventRoutine` o se despierta al proceso.

Manejo de números aleatorios

Todos los generadores de números aleatorios (incluidas las distribuciones) implementan la interfaz `UniformRandomGenerator`. DesmoJ provee la clase `DefaultRandomGenerator` que es el generador por defecto. Si se quisiera utilizar otro se puede reemplazar adecuadamente (y en los constructores las clases que sean necesarias) el `DefaultRandomGenerator` con la clase deseada. O también se puede cambiar la implementación del generador por defecto.

Manejo de distribuciones

El manejo de la aleatoriedad en DesmoJ se realiza por medio de 2 clases básicas: `Distribution` y `DistributionManager`. Las instancias de la primera se definen y utilizan en el `Model`. Forman parte del sistema que se quiere modelar. Pero las instancias de `DistributionManager` se encuentran dentro de un `Experiment`. Forman parte de la infraestructura necesaria para la ejecución de una simulación.

En la clase `Distribution` se encapsula el comportamiento de una distribución probabilista (real o discreta). Es la superclase de todas las distribuciones que provee DesmoJ. Las instancias de `Ditribution` contiene dentro un generador de números aleatorios, con el cual muestrean una variable aleatoria.

En cambio el `DistributionManager` es utilizado por un `Experiment` para manejar el conjunto de las distribuciones utilizadas. Contiene generador de números aleatorios (`seedGenerator`) y el conjunto de todas las distribuciones definidas en un modelo. Con el generador inicializa a las distribuciones del modelo con una semilla.

Cuando una instancia de `Distribution` (esto se hace en el método `init` del `Model`) se crea, esta se registra en el `DistributionManager` y se le asigna una semilla. Esto significa que se tienen tantos streams como distribuciones declaradas en el modelo.

El `DistributionManager` es un control central de todas las distribuciones del modelo. Si quiere repetir una corrida, se le asigna al manejador la misma semilla que en la corrida anterior (por medio de la operación `setSeedGenerator` del `Experiment`).

Orientación a procesos

En DesmoJ se manejan 2 visiones del mundo: el enfoque a eventos y a proceso. El primero ya ha sido discutido en profundidad. Sobre la orientación a procesos en DesmoJ se puede decir que utiliza interacción a procesos.

Para crear una simulación a procesos se debe implementar una clase derivada de `SimProcesss`. En ésta se debe implementar el método abstracto `lifeCycle` donde se define el ciclo de vida de la entidad. Se brindan las primitivas de `activate` y `passivate` para activar o suspender un proceso.

Prueba de implementación

Para ello se realizó la implementación del ejemplo del Hospital simple que se encuentra descrito en el libro del curso en el capitulo 1 sección 1.3.1.

En cuanto al entorno que brinda DesmoJ, se puede decir que es fácil de aprender a utilizarlo, ya que no llevo más de 3 horas implementar el programa completo y libre de errores. Las interfaces que se manejan son claras y simples.

Fortalezas de DesmoJ

- Está desarrollado en Java, por tanto corre en un amplio conjunto de plataformas.
- Se dispone del código fuente. Esto es una fortaleza por dos motivos: se puede realizarle modificaciones y se pueden sacar ideas de su diseño.
- Ha sido desarrollado desde 1999, no es un producto abandonado y fuera de uso. De hecho la versión con la que se experimento es la 2.0.1 liberada en Febrero de 2005, es un producto que se actualiza en forma más o menos regular.
- Ofrece interfaces claras y fáciles de aprender. Se tienen los conceptos de:
    - `Model`: En esta clase se encapsula el modelo a simular. Esta clase posee las colas de espera, distribuciones y entidades globales al sistema.
    - `Entity`: Es la clase abstracta de la cual descienden todas la entidades de un modelo
    - `Event`: Los hay de dos tipos eventos externos al modelo (aquellos que generan eventos sin estar relacionados directamente con una entidad, son utilizados como feeders) y los eventos internos (aquellos que actúan sobre una entidad)
    - `Experiment`: Esta construcción encapsula la experimentación con el modelo. Contiene el calendario, el ejecutivo y el manejo de distribuciones. Para hacer una corrida, se debe conectar una instancia de esta clase con una instancia del modelo.
- Los calendarios heredan de la clase abstracta `EventList`. La biblioteca provee un calendario por defecto (`EventVector`). Para cambiarlo se crea una clase nueva se cambia una línea en la clase `Scheduler`.
- Se provee de la interfaz `UniformRandomGenerator` para la generación de números aleatorios y un generador por defecto `DefaultRandomGenerator`. Para utilizar un nuevo generador, se implementa y se cambia la línea correspondiente.
- Se tiene la documentación completa de usuario (tutorial online y API)
- Luego de cada corrida se generan 4 archivos html:
    - Report: brinda un reporte completo de la corrida (investigar que pasa con las estructuras estadísticas)
    - Trace: brinda un reporte con todos los eventos (que han sido previamente seleccionados en el constructor)
    - Debug: (investigar como funciona)
    - Errors: brinda un reporte de los errores cometidos en el manejo interno de DesmoJ (sacar elementos de colas a las que no pertenecen, mal manejo de estructuras, etc.)

Debilidades de DesmoJ

- Si bien se tiene la documentación completa de usuario, no disponemos de documentación técnica sobre el diseño de la biblioteca. Si desea saber como funciona se debe adquirir el conocimiento por medio de lectura del código y el API.
- Se maneja el enfoque a eventos pero solo con 2 fases (en la clase eventos no parece haber ninguna construcción que soporte eventos condicionados).
- Como maneja 2 posibles visiones del mundo (eventos y procesos), el manejo del calendario y el ejecutivo son un poco confusos, ya que dentro del calendario se maneja una estructura que contiene un evento, una entidad y un proceso (`EventNote`).
- No maneja salida visual. Si se desea tenerla, se debe implementarla.
- Aunque se proveen herramientas avanzadas para el modelado de sistemas, estas están implementadas solo para el enfoque a procesos y no pueden ser utilizadas en el enfoque a eventos.