Simulación a eventos discretos

Clase nro 4.

Curso 2010.

Implementación de Sistemas de S.E.D.

Conceptos Importantes

- Modelo
- Evento
  - Fijo
  - Condicionado
- Entidad
- Recurso
- Calendario
- Ejecutivo

Implementación en Pascal SIM

- Entidades: Se utilizan las brindadas por Pascal SIM. Para especializarlas, se debe cambiar el código.
- Recursos: Brindados por Pascal SIM.
- Eventos: Son procedimientos, tanto los eventos B como los C.
- Modelo: Implícito.

Pascal SIM (1)

- Entidades:
  - Poseen atributos para indicar su disponibilidad, clase, identificación, próximo evento fijo y su tiempo.

```
entity = ^an_entity;
an_entity = packed record
              avail            :boolean;
              class            :class_num;
              col              :color;
              attr, next_B     :cardinal;
              time             :real;
end;
```

  - Creación:

```
function new_entity (c :class_num; a :cardinal) :entity;
procedure make_class (var c :queue; n, size :cardinal);
```

  - Eliminación:

```
procedure dis_entity (e :entity);
```

Pascal SIM (2)

- Recursos:
  - Los recursos son de tipo renovables, una vez utilizados deben ser devueltos.

```
bin = record
          number, num_avail :cardinal;
end;
```

  - Creación

```
procedure make_bin (var from :bin; n :cardinal);
```

  - Adquisición

```
procedure acquire (var from :bin; n :cardinal);
```

  - Retorno

```
procedure return (var from :bin; n :cardinal);
```


Pascal SIM (3)

- Colas:
  - En Pascal SIM las colas son listas circulares doblemente encadenadas.
  - Sus elementos son entidades.
  - No manejan prioridades.

```
procedure make_queue (var q :queue);
procedure give_top (q :queue; i :entity);
procedure give_tail (q :queue; i :entity);
function take_top (q :queue) :entity;
function take_tail (q :queue) :entity;
function empty (q :queue) :boolean;
function count (q :queue) :cardinal;
```

Pascal SIM (4)

- Eventos:
  - Son procedimientos definidos por el modelador.
  - No tienen ninguna forma particular de definición.
  - Para los eventos fijos: la entidad sobre la cual actúan se encuentra almacenada en la variable global `current`.
  - Programación de eventos (agendar)

```
procedure cause (nb :cardinal; e :entity; t :real);
```


Pascal SIM (5)

- Modelo:
  - El modelo del sistema a simular está implícito. Es el archivo .pas, y la memoria que este ocupa en el tiempo de ejecución.
  - Para inicializar un modelo se utiliza el procedimiento `initialize` (convención).
  - Aquí se crean las colas, las entidades globales, los recursos, los histogramas y los torrentes de números aleatorios.

Pascal SIM (6)

- Calendario:
  - Es una lista global de entidades ordenada por el campo `time`.
  - Contiene todas las entidades agendadas a algún evento fijo.
  - Se insertan elementos con el procedimiento `cause`.
  - Para tomar la entidad que se debe procesar, se utiliza `calendar_top`. Esta función deja la entidad a procesar en la variable global `current`. El calendario se crea con el procedimiento `make_sim`.

Pascal SIM (7)

- Ejecutivo:
  - Algoritmo que corre una simulación.
  - Debe ser implementado por el usuario. Por convención se utiliza el procedimiento `run`.
  - Pasos:
    - Fase A: avance del tiempo
    - Fase B: ejecución de los eventos fijos
    - Fase C: ejecución de los eventos condicionados

Pascal SIM (8)

- Ejecutivo:
  - Fase A: avance del tiempo
    - Se chequea que la simulación siga corriendo.
    - Se chequea que el calendario no este vacío y que no se halla llegado al tiempo de terminación.
    - Se avanza el tiempo (`tim`) al tiempo de la entidad a procesar.

Pascal SIM (9)

- Ejecutivo:
  - Fase B: ejecución de los eventos fijos
    - Mientras que halla entidades agendadas cuyo tiempo sea igual al actual:
      - Se las quita del calendario.
      - Se ejecuta el evento al cual están asignadas. Esto se hace por medio de números de eventos que son definidos por el usuario (`case` statement).

Pascal SIM (10)

- Ejecutivo:
  - Fase C: ejecución de los eventos condicionales
    - Luego de procesar todos los eventos fijos cuyo tiempo es igual al actual, se procesan los eventos condicionales.
    - Se recorre la lista de eventos condicionales, y se los ejecuta a todos.
  - Eficiencia se puede mejorar con simulación celular (más adelante).

Pascal SIM (11)

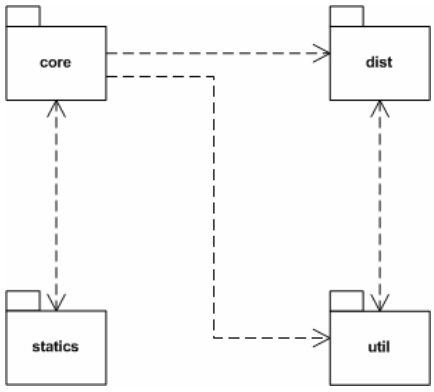
Estructura general del programa

```
bound events
conditional events
procedure run;
procedure initialize;
procedure report;
   begin
        …
        initialize;
        run;
        report;
        …
   end.
```

Implementación en EOSimulator

- Entidades: Se pueden utilizar las brindadas por defecto, o se crea una nueva subclase.
- Recursos: Brindados por EoSimulator.
- Eventos: Son clases abstractas.
- Modelo: Es una clase abstracta que contiene todos los atributos relevantes del sistema que se quiere modelar.

Arquitectura de EOSimulator



EOSimulator (1)

- Entidades:
  - Poseen atributos para indicar su próximo evento fijo y su tiempo.

```
class Entity {
private:
   BEvent* bEv;
   double clock;
public:
   Entity ();
   virtual ~Entity ();
   void setBEvent (BEvent* bEv_);
   void setClock (double clock_);
   double getClock ();
   void processEvent ();
};
```

  - Las entidades solo deben crearse en forma dinámica (`new`).

EOSimulator (2)

- Recursos:
  - `Renewable` y `NonRenewable`. Interfaz similar a Pascal SIM.

```
class Bin {
public:
   virtual ~Bin();
   void acquire (double amount_);
   bool isAvailable (double amount_);
};

class Renewable: public Bin {
public:
   Renewable (double cuantity_, double max_);
   ~Renewable ();
   void returnBin (double amount_);
};

class NonRenewable: public Bin {
public:
   NonRenewable (double quantity_);
   ~NonRenewable ();
   void addBin (double amount_);
};
```

EOSimulator (3)

- Colas:
  - En EOSimulator las colas de entidades siguen la misma interfaz, `EntityQueue`.
  - Hay implementaciones de colas fifo, colas lifo y de prioridad. La prioridad está dada por un comparador (ver man y manual de usuario).

```
class EntityQueue {
public:
   EntityQueue() {};
   virtual ~EntityQueue() {};
   virtual void push(core::Entity* ent_) = 0;
   virtual core::Entity* pop() = 0;
   virtual void remove(unsigned int i_) = 0;
   virtual bool empty() = 0;
   virtual core::Entity* operator[] (unsigned int i_) = 0;
   virtual unsigned int size() = 0;
};
```


EOSimulator (4)

- Eventos:
  - Son clases abstractas. Pertenecen a un Modelo y poseen una referencia al mismo (`owner`). Hay dos tipos de eventos, `BEvent` y `CEvent`. Se debe implementar el método `eventRoutine`.

```
class BEvent {
protected:
     Model& owner;
public:
     BEvent (std::string name_, Model& owner_);
     virtual ~BEvent ();
     std::string getName();
     virtual void eventRoutine (Entity* who_) = 0;
};
```

```
class CEvent {
protected:
     Model& owner;
public:
     CEvent (Model& owner_);
     virtual ~CEvent ();
     virtual void eventRoutine () = 0;
};
```

  - El nombre de los BEvent es el identificador utilizado para agendar las entidades. Es una std::string

EOSimulator (5)

- Modelos:
  - Es una clase abstracta. Define el modelo del sistema que se quiere simular. Posee como atributos: eventos (B y C), recursos, entidades globales, colas, histogramas, distribuciones.
  - Se deben implementar 2 operaciones: `init` y `doInitialSchedules`.

```
class Model {
private:
   Experiment* exp;
   utils::BEventMap bEvs;
public:
   Model();
   virtual ~Model();
   virtual void init () = 0;
   virtual void doInitialSchedules () = 0;
   void connectToExp (Experiment *exp_);
   void registerBEvent (BEvent* bEv_);
   void registerCEvent (CEvent* cEv_);
   void registerDist (dist::Distribution* dist_);
   void registerHistogram (statics::Histogram* hist_);
   void schedule (double offset_, Entity* who_, std::string what_);
   double getSimTime();
};
```


EOSimulator (6)

- Calendario:
  - Dos tipos de calendarios: eventos B y eventos C. El primero contiene las entidades agendadas a un evento fijo, y el segundo contiene todos los eventos C registrados.

```
class BCalendar {
private:
     double simTime, endSim;
     utils::EntityQueueOrdered ents;
public:
     BCalendar ();
     ~BCalendar ();
     void bPhase ();
     bool isStopped();
     void schedule (double offset_, Entity* who_);
     void setEndTime (double when_);
     double getSimTime();
};
```

```
class CCalendar {
private:
     utils::CEventVector cEvs;
public:
     CCalendar ();
     ~CCalendar ();
     void cPhase ();
     void registerCEvent (CEvent* cEv_);
};
```

EOSimulator (7)

- Ejecutivo:
  - Clase `Experiment`. Contiene calendarios B y C, y los torrentes.
  - Fases A y B usan `Bcalendar`; fase C usa `CCalendar`.

```
class Experiment {
private:
   bool running;
   dist::DistManager distMan;
   BCalendar bCal;
   CCalendar cCal;
   Model* currModel;
public:
   Experiment ();
   ~Experiment();
   void run (double simTime_);
   void setModel (Model* model_);
   void schedule (double offset_, Entity* who_);
   void setSeed (unsigned long seed_);
   void registerDist (dist::Distribution* dist_);
   void registerCEvent (CEvent* cEv_);
   double getSimTime();
};
```

  - Para correr la simulación: se conecta el `Model` con el `Experiment`, y luego se ejecuta.

Próxima clase:

Muestreos.