Breve introducción al GLIDER

GLIDER es un lenguaje de simulación desarrollado en la Universidad de Los Andes y permite modelar sistemas discretos, continuos o mixtos (discretos y continuos).

En GLIDER el sistema se modela como una red de nodos (grafo dirigido). Hay un conjunto de nodos predefinidos que tienen cierto comportamiento implícito, que depende del tipo de nodo, y un comportamiento explicito que el usuario describe mediante el código que asocia al nodo. Los tipos de nodo predefinidos en GLIDER son: *Input*, *Resource*, *Exit*, *Gate*, *Line*, *Decision*, *Autonomous*, *Continuous* y *General*. El nombre del lenguaje se debe a los nodos que se implementaron en su primera versión: ***G**ate*, ***L**ine*, ***I**nput*, ***D**ecison*, ***E**xit* y ***R**esource*. En versiones posteriores se agregaron los otros nodos.

Entre los nodos circulan mensajes (registros) que representan aquello que se mueve en el sistema: clientes, personas, vehículos, información, insumos, etc. Los mensajes se alojan en listas asociadas a los nodos. Las listas que puede tener un nodo son: *lista externa* (*EL*) que aloja aquellos mensajes que aun no han sido procesados por el nodo, y *lista interna* (*IL*) que aloja los mensajes procesados o siendo procesados. Los mensajes necesariamente deben estar en alguna lista asociada a algún nodo. En realidad los mensajes son registros que se crean dinámicamente y lo que pasa por las listas son sus apuntadores.

Un nodo se define indicando su nombre y opcionalmente su tipo, multiplicidad, sucesores y código asociado.

- *Nombre*: cualquier identificador valido.
- *Tipo*: permite hacer explicito el tipo de nodo. Para esto se coloca entre paréntesis la primera letra del tipo de nodo: *G*, *L*, *I*, *D*, *E*. *R*, *A*, *C* para *Gate*, *Line*, *Input*, *Decision*, *Exit*, *Autonomous* y *Continuous* respectivamente, y cualquier otra letra, aparate de estas, para los *General*. Si no se hace explicito el tipo de nodo, la primera letra del nombre indicara su tipo.
- *Multiplicidad*: permite definir varios nodos del mismo tipo. Para esto definimos entre corchetes la multiplicidad. Ejemplo: `NodoMultiple (R) [1..4]` en donde indicamos que el nodo de nombre `NodoMultiple` es de tipo *Resource* y su multiplicidad es 4. En realidad lo que tenemos son 4 nodos de tipo *Resource*: `NodoMultiple[1], NodoMultiple[2], NodoMultiple[3] y NodoMultiple[4].` Los nodos tipo *C* y *D* **no** pueden tener multiplicidad.
- *Sucesores:* esta lista indica cuales son los nodos a los que se puede dirigir un mensaje después de abandonar este nodo. Si no se indican sucesores, se asume que el siguiente nodo en la red que define el sistema es el nodo sucesor. Ejemplo:

```
Llegadas (I) :: IT := Expo(2);
Taquilla (R) :: STAY := Expo(1);
Salida (E) ::
```

  En este ejemplo como no se hacen explícitos los sucesores, tenemos que el sucesor de `Llegadas` es `Taquilla`, y de `Taquilla` es `Salida`. `Salida,` por ser un nodo tipo *Exit*, no requiere sucesores. `Llegadas` genera los mensajes (llegadas al sistema) los cuales son pasados a `Taquilla`. Los mensajes si no pueden ser atendidos en `Taquilla` (esta ocupada),

deben esperar en cola. Si pueden ser atendidos, pasan cierto tiempo en `Taquilla` (siendo atendidos) y después la abandonan pasando a `Salida`. En `Salida` son eliminados los mensajes (abandonan el sistema). Nótese que el tipo de nodo se hizo explicito ya que de otra forma `Llegadas` seria tipo *Line* los otros dos tipo *General*, no siendo esta la intención.
Si invertimos el orden de los nodos, entonces debemos hacer explícitos los sucesores ya que la opción por omisión ya no nos sirve:

```
Salida (E) ::
Taquilla (R) Salida :: STAY := Expo(1);
Llegadas (I) Taquilla :: IT := Expo(2);
```

- *Código*: esta formado por instrucciones en GLIDER y/o Pascal. Este código indica que procesamientos se le debe hacer a los mensajes que llegan al nodo. En los ejemplos anteriores las instrucciones `IT := Expo(2)` y `STAY := Expo(1)` representan el código asociado a los nodos `Llegadas` y `Taquilla` respectivamente (más adelante se dirá que hacen estas instrucciones).

Activación de los Nodos

Cuando se ejecuta el código de un nodo se dice que este se activa. La activación se puede dar:

- *Por evento*: cuando se procesa el próximo evento de la lista de eventos futuros (en GLIDER la *FEL*: *Future Event List*), este debe estar relacionado con uno de los nodos de la red, haciendo que dicho nodo se ejecute o active (por evento).
- *Por revisión*: después de haberse activado un nodo por evento, GLIDER entra en un proceso de revisión. Este proceso es imprescindible para completar las consecuencias de dicha activación por evento. Lo que se hace es revisar, en el orden en que están definidos los nodos en la red y comenzando por el nodo que le sigue al activado por evento, nodo por nodo para ver si es posible mover mensajes de una lista a otra. El proceso de revisión culmina una vez que se ha pasado por todos los nodos y no se ha podido mover mensaje alguno. Para ilustrar la necesidad de esta revisión, consideremos el ejemplo anterior. Cuando `Llegadas` genera un mensaje, este pasa a la *lista externa* de `Taquilla`. Si `Taquilla` esta desocupada, este mensaje debe pasar a ser atendido (a su *lista interna*). Si solo se activa `Llegadas` y no se revisa `Taquilla`, no hay forma de que dicho mensaje se atendido y por lo tanto las consecuencias de la llegada son completadas a medias. Solo nodos con listas asociados son revisados. Los otros nodos (*I*, *A* y *C*) nunca son revisados ya que no alojan mensajes.

Mensajes

Los mensajes son registros (estructuras de datos) que se crean dinámicamente. Tienen ciertos campos fijos y opcionalmente el usuario puede agregar otros que considere conveniente. Siempre están alojados en una lista asociada a alguno de los nodos de la red. Dentro de los campos fijos tenemos:

- *GT* (generation time): contiene el tiempo (de simulación) en que fue generado.
- *NODE*: indica el nodo en donde fue generado el mensaje.
- *NUMBER*: es un número secuencial que indica el orden de generación. El primer mensaje es el 1, el segundo el 2, etc.

- *USE*: permite definir cuanto recurso requiere el mensaje al momento de ocupar un nodo tipo *R*. Por omisión la capacidad de un nodo tipo *R* en 1 y por omisión el campo *USE* tiene el valor 1.
- *ET* (entry time): registra el tiempo en el que un mensaje entro en la lista en que esta actualmente.
- *XT* (exit time): registra el tiempo en el que el mensaje esta saliendo de la lista que recién lo alojo. La diferencia entre *ET* y *XT* permite calcula el tiempo de permanecía en la lista.

Eventos

Como en todo simulador por eventos discretos, estos son los que indican cuando se da un cambio en el sistema, los que conducen o guían la simulación, los que permiten avanzar el tiempo de simulación. Son generados en forma dinámica (al igual que los mensajes) y se almacenan, en orden cronológico, en la ***F****uture* ***E****vent* ***L****ist* (*FEL* o lista de eventos futuros). Dentro de la estructura de datos que los componen tenemos los siguientes campos:

- *EN*: nodo que se activará cuando se procese el evento.
- *IEV*: subíndice del nodo a activar, si lo hay. Este campo se usa cuando el nodo tiene multiplicidad > 1.
- *T*: tiempo de ocurrencia del evento.
- *EVP*: 1 byte que el usuario puede manipular en caso de que requiera manejar explícitamente los eventos en la *FEL*. Es usado en circunstancias muy particulares.
- *SU*: indica si el evento esta suspendido.
- En GLIDER un evento como por ejemplo una llegada al sistema o una culminación de servicio en una taquilla, está asociado implícitamente con el nodo a ejecutar (activar). Si se activa un nodo tipo *I*, por lo general se da una o más llegadas. Cuando se activa un nodo tipo *R*, se da una o más culminaciones de servicio.

Secciones

Un código en GLIDER incluye varias secciones. Entre las más comunes están:

- *Title*: sección al comienzo que usualmente comenta el código que se describe en las secciones siguientes.
- *Network*: es la sección más relevante en donde se definen los nodos que conforman el modelo del sistema que se esta simulando.
- *Init*: esta es la sección de inicialización de variables, definición del tiempo a simular, generación de los primeros eventos, etc.

  Nota: es importante resaltar que si no se generan explícitamente los primeros eventos (al menos uno), la simulación no podrá efectuarse (habrá un error) ya que la *FEL* estará vacía.
- *Decl*: aquí se declaran variables a usar, campos adicionales relacionados con los mensajes, funciones, tablas, nodos de los que se quieren estadísticas, etc.

Tipos de Nodos

Input (I)

- Genera un mensaje (puede generar más de uno) y lo pasa de una vez a un nodo sucesor.
- No tiene ni *EL* ni *IL*, por lo tanto nunca es revisado (durante el proceso de revisión) ya que no tienen mensajes que puedan ser movidos. Solo se activa mediante eventos.
- Por lo general incluye una instrucción para generar en forma automática la próxima activación. La instrucción es de la forma: `IT := <exp>` que indica que el lapso hasta la próxima activación será a lo que evalué <exp>.
- Ejemplo:

```
Llegadas (I) Taquilla :: IT := Expo(2);
```

Line (L)

- Ordena mensajes según el código asociado al nodo.
- Tiene *EL* e *IL*.
- Pasa los mensajes de la *EL* a la *IL* y en la *IL* se ordenan según el código asociado.
- La *IL* del nodo tipo *L* es la misma *EL* del nodo sucesor.
- Se puede activar por evento o por revisión.
- Ejemplo:

```
Cola (L):: ORDER(NUMBER, D);
```

  El nodo `cola` ordena los mensajes en forma descendente (`D`) de acuerdo al valor del campo `NUMBER`, es decir; los que fueron generados más tarde (tienen `NUMBER` más alto) están de primeros.

Gate (G)

- Solo tiene *EL*.
- Retiene los mensajes y los deja pasar en forma selectiva (de acuerdo a lo que establezca el código asociado).
- Se puede activar por evento o por revisión.
- Puede contener una instrucción `STATE`. Si tiene esta instrucción, la misma solo se ejecutará si el nodo se activa por evento (nunca durante una revisión). Esta instrucción permite cambiar en forma controlada el estado de nodo *G*. Si esta instrucción esta presente, debe ser la primera en el código asociado al nodo.
- Ejemplo:

```
Puerta (G):: State begin
                    IT := 20;
                    Abierta := not Abierta
             end;
             If Abierta then SendTo(Taquilla)
```

  El estado de `Puerta` esta controlado por la variable (booleana) `Abierta`. Si esta variable tiene el valor `true`, los mensajes que llegan a su *EL* son enviados al nodo `Taquilla`. En

caso contrario (tiene el valor `false`), son retenidos en la *EL* hasta que cambie de valor. La instrucción `state` tiene por objetivo generar la próxima activación por evento del nodo `Puerta` (dentro de 20 unidades de tiempo) y cambiar el valor de `Abierta` (si estaba en `false` lo cambia a `true` y viceversa). La intención es que `Puerta` esta 20 unidades de tiempo abierta y 20 unidades de tiempo cerrada.

Resource (R)

- Tienen *EL* e *IL*.
- Tienen capacidad.
- Los mensajes que están en la *EL* son pasados a la *IL* siempre y cuando exista suficiente capacidad disponible en el nodo (la capacidad disponible en el nodo es mayor o igual al valor del campo *USE* del mensaje). El pase de menajes de la *EL* a la *IL* usualmente se da por revisión del nodo. El abandono de un mensaje de la *IL* se da cuando el nodo se activa por evento.
- Por lo general incluye una instrucción para generar el tiempo de permanencia del mensaje cuando este ocupa el nodo (tiempo de servicio). La instrucción es de la forma: `STAY := <exp>` que indica que el tiempo de servicio será a lo que evalué <exp>. En este caso GLIDER genera un evento asociado al nodo *R* que representa la salida de dicho mensaje.
- Ejemplo:

```
Taquilla (R) Salida :: STAY := Expo(1);
```

- Tal como en los nodos tipos *G* hay una instrucción (opcional) que se ejecuta solo cuando hay una activación por evento, en los nodos *R* también hay una. Es la instrucción `RELEASE` (si aparece debe estar de primero) e indica que hacer con los mensajes que abandonan la *IL* del nodo.
- Ejemplo:

```
Taquilla1 (R) Salida, Taquilla2::
            Release If Ber(0.7) then SendTo(Salida)
                    Else SendTo(Taquilla2);
            STAY := Expo(1);
```

  Los mensajes que ocupan `Taquilla1` son atendidos con una duración de servicio exponencial con media 1. Una vez concluido este tiempo, los mensajes abandonan la *IL* de `Taquilla1` y se decide que hacer con ellos. Un 70% (`Ber(0.7)` es un experimento Bernoulli con probabilidad de éxito 0.7, es decir, aproximadamente el 70% de las evaluaciones devuelve `true` y el 30% `false`) se va a `Salida`, mientras que un 30% debe ir a `Taquilla2` (algún otro tramite).

Exit (E)

- Solo *EL*.
- Toma los mensajes de la *EL*, los procesa según el código asociado (si lo hay), y los elimina.
- Se ejecuta por revisión.

Decisión (D)

- Tiene una *EL* por cada nodo antecesor. No tiene *IL*.

- Es una suerte de generalización de nodo tipo *G*.
- Toma los mensajes de las *EL* y los envía en forma selectiva a los sucesores.
- Se ejecuta por revisión o por evento.

Autonomous (A)

- No tienen ni *EL* ni *IL*. No son revisados.
- Se activan explícitamente por eventos.
- Comúnmente se usan para incorporar funcionalidades no relacionadas con el sistema modelado como por ejemplo, imprimir ciertas variables a medida que transcurre la simulación, imprimir estadísticas que GLIDER no genera, producir graficas, etc.

Continuous (C)

- No tienen ni *EL* ni *IL*. No son revisados.
- El código asociado esta formado por ecuaciones diferenciales y algebraicas que se resuelven numéricamente.
- Se usan para simulación continua.
- Se activan o desactivan explícitamente mediante los procedimientos *ACT* o *DEACT*, o automáticamente cada *DT*.
- Las activaciones no se propagan a la red (no causan una revisión).
- Ejemplo:

```
NETWORK
  RF (C) ::  R':= 0.18*(1-R/20000)*R-0.00015*R*F;
             F':=-0.2*F+0.0001*R*F;
INIT TSIM:=650; ACT(RF,0);
     DT_RF:=0.125;
DECL
END.
```

En el nodo continuo *RF* se definen dos ecuaciones diferenciales interrelacionadas de la forma $\frac{dR}{dt} = f(R,F)$ y $\frac{dF}{dt} = g(R,F)$, las cuales se resolverán numéricamente. En la sección `Init` se fija el tiempo de simulación en 650 (`TSIM=650`), se indica que el nodo `RF` se activa en el tiempo 0 (`ACT(RF,0)` - el primer evento), lo que implica que el intervalo de solución es [0,650] , y se fija el paso de solución en 0.125 (`DT_RF:=0.125`) lo que hace que `RF` se active automáticamente cada 0.125 unidades de tiempo.

Ejemplo 1

Ilustraron del uso de nodos con multiplicidad. Se tienes 3 taquillas, una detrás de la otra, cada una con un tiempo de atención exponencial con media 2. En la primera taquilla llegan clientes según una distribución exponencial con media 3. Los clientes que salen de la primera taquilla pasan a la segunda y los de la segunda a la tercera. Los que salen de la tercera taquilla abandonan el sistema. Como el comportamiento de las 3 taquillas es similar, se implementan como un nodo con multiplicidad 3. Lo que hay que tener cuidado, es que los mensajes pasen por las taquillas en el orden estipulado. Para esto usamos dentro del nodo con multiplicidad la variable predefinida `INO` que contiene el subíndice del nodo que se esta procesando en un momento dado. Si abandonamos `Taquilla[1]` vamos a `Taquilla[2]`, si abandonamos `Taquilla[2]` vamos a `Taquilla[3]` y finalmente si abandonamos `Taquilla[3]` vamos a `Salida`.

```
Ejemplo Red1 con multiplicidad

NETWORK

Llegadas (I) Taquilla[1]  ::
    IT:=EXPO(3.0);

Taquilla (R) [1..3] Taquilla[INO],Salida ::
    RELEASE
      IF INO<3 THEN BEGIN
        I := INO + 1;
        SENDTO(Taquilla[I])
      END ELSE SENDTO(Salida);
    STAY:=EXPO(2.0);

Salida   (E)   ::

INIT TSIM:=10000;
     ACT(Llegadas,EXPO(3.0));

DECL VAR I: integer;
     STATISTICS ALLNODES;

END.
```

Ejemplo2

Ilustración del uso de varios de los nodos. Los mensajes generados en `Llegadas` (que se activa en el tiempo `Expo(3.0)`) son enviados a `Cola`. En `Cola` son ordenados en forma descendente según su campo `NUMBER`. La *IL* de `Cola` (`IL_Cola`) es la misma *EL* de `Puerta` (`EL_Puerta`). Puerta comienza cerrada y después alterna cada 10 unidades de tiempo entre abierta y cerrada. En `Taquilla` los mensajes son atendidos según una exponencial con media 2.

```
Ejemplo Red con varios tipos de nodos

NETWORK

  Llegadas (I) :: IT:=Expo(3.0);

  Cola (L) :: Order(NUMBER,D);

  Puerta (G) :: STATE BEGIN
                  IT := 10;
                  Abierta := not Abierta;
                END;
                IF Abierta then SendTo(Taquilla);

  Taquilla (R) :: STAY:=Expo(2.0);

  Salida   (E)   ::

INIT TSIM:=10000; ACT(Llegadas,EXPO(3.0)); ACT(Puerta,10);
     Abierta := False;

DECL
  VAR Abierta: Boolean;
  STATISTICS ALLNODES;

END.
```

La salida que genera GLIDER es (nótese que las estadísticas de la *IL* de `Cola` y la *EL* de `Puerta` son las mismas ya que son las mismas listas):

```
Basic Experiment
Time   10000.00 Time Stat.  10000.00  Replication    1   0/ 0/   0  0h  0m  0s
Elapsed time  0h  0m  0.0s
Nod/Ind Ant/Li #Ent Lgth   Max    Mean    Dev   MaxSt   MeanSt    Dev    T.Free
LLEGADAS
   1 # Gen.    3283
COLA
   1 EL        3283    0     1     0.0     0.0     0.0     0.0     0.0 10000.00
     IL        3283    0     9 0.84190 1.38834 9.99503 2.56442 3.29177  6351.56
PUERTA
   1 EL        3283    0     9 0.84190 1.38834 9.99503 2.56442 3.29177  6351.56
TAQUILLA
   1 EL        3283    2    18 1.96315 2.65402 46.6037 5.97791 6.52353  4680.91
     IL        3281    1     1 0.65816 0.47433 18.9231 2.00632 2.03382  3418.39
SALIDA
   1 EL        3280    0     1     0.0     0.0     0.0     0.0     0.0 10000.00
  Time in System: Mean  10.5565   Dev.  7.53044   Max.  54.9571   Min.  0.01443


Var/Ind                Mean(t) Dev.(t)    Max.    Min. Mean(v) Dev.(v)  Actual
  U_TAQUILLA  1        0.65816 0.47433 1.00000     0.0 0.50008 0.50000 1.00000
```