Salida visual

Clase Nro 13

CURSO 2010

Motivación

En qué situaciones es útil la salida visual:

- Validación.
- Análisis visual.
- Comunicación a los usuarios.

Simulación interactiva: permite cambiar parámetros durante el transcurso de la simulación.

Métodos de visualización

- Imprimir valores de parámetros.
- Imprimir histogramas durante el transcurso de la simulación.
- Diagramas animados: despliegan el estado de las entidades y recursos.
- Gráficos: reproducen el escenario que se modela (*dynamic iconic display*).

Diseño de la salida visual

Etapas:

a) Diseño del fondo estático: elementos que permanecen constantes durante la simulación.

b) Diseño de los componentes dinámicos: elementos que cambian con cada evento.

c) Integración con la simulación:
   - Codificación explícita.
   - Información para un visualizador.

Salida visual en PascalSim

- Basada en caracteres.
- Iconos son letras del alfabeto.
- Primitivas de salida visual son independientes de las que gobiernan la simulación.
- Componentes dinámicos se actualizan:
  - Cuando ocurren los eventos, en forma instantánea.
  - Después de cada avance del tiempo.

Salida visual en PascalSim

- Movimiento de entidades debe programarse en los eventos, porque es donde se accede a la entidad *current*.
- Ejemplo:

```
procedure patient2_arrives;
begin
  <add current to tail of q2>;
  <move current towards q2>;
  <display the new instance of q2>;
  <create a new entity>;
  <cause the arrival of the new entity>;
end;
```

Salida visual en PascalSim

- En el enfoque de 3 fases, los recursos deben actualizarse al finalizar la fase C, dado que pueden devolverse y tomarse en el mismo instante de tiempo.

Salida visual en PascalSim

Procedimientos:

- `picture`: inicialización del escenario estático.
- `display`: actualización de los elementos al final de cada time beat.

Primitivas de salida visual en PascalSim

Pantalla

```
procedure make_screen;
procedure clear_screen;
procedure gotoxy(x,y : cardinal);
```

Colores

```
procedure set_foreground(c : color);
procedure set_background(c : color);
procedure reset_colors;
```


Primitivas de salida visual en PascalSim

Fondo estático

```
procedure write_block(x1,y1,x2,y2 : cardinal;
                      b : color);
```

Inicialización de íconos de entidades

```
procedure make_class_table;
procedure enter_class(n : class_num; l : char
                      c : color);
```

Primitivas de salida visual en PascalSim

Escenario dinámico

```
procedure write_entity(x,y : cardinal; e : entity);
procedure write_queue(x,y : cardinal; b : color;
                      q : queue;
                      max_length : cardinal);
procedure move_v(x,y1,y2 : cardinal; e : entity;
                 b : color); {donde y2 > y1}
procedure move_h(y,x1,x2 : cardinal; e : entity;
                 b : color); {donde x2 > x1}
```


Primitivas de salida visual en PascalSim

Velocidad de la simulación

```
procedure delay; {utilizado en procedimientos move}
const delay_num = {valor apropiado};
```

Visualización realista

```
procedure display;
begin
  {actualización del escenario estático}
  for i := 0 to trunc(tim - old_tim) do delay;
  old_tim := tim;
end;
```

Simulación interactiva

Interacción:

- Determinada por el modelo: mecanismos complejos de decisión, se transfiere el control de la simulación al usuario.
- Determinada por el usuario: permite alterar partes del sistema durante la simulación.

Simulación interactiva

Interacción determinada por el usuario:

- Implementación:

```
procedure display;
begin
  {update display};
  if keypress (o readkey) then begin
   {read input};
   {take appropriate action};
  end;
end;
```

- Consecuencias:
  - Invalida análisis estadístico.
  - Altera estado estacionario.

Casos de estudio

- Hospital
- Taller de reparaciones

Hospital

- Escenario estático (página 174): diagrama de la disposición física de los elementos. Similar al diagrama de actividades.
- Elementos:
  - Colas q1 (pacientes para estadía), q2 (pacientes para camas) y q3 (pacientes para sala).
  - Número de camas en uso.
  - Estado de la sala de operaciones (abierto/cerrado, disponible/ocupado).

Hospital

Representación de las entidades:

```
enter_class(1,'s',blue); {stay}
enter_class(2,'o',blue); {operation}
current^.col : = yellow; {ready for
                           operation}
```


Hospital

Detalles del programa:

- Colas son actualizadas (`write_queue`) cada vez que se modifican.
- Estado de la sala es actualizado cada vez que cambia.
- Número de camas en uso es actualizado en el procedimiento `display`.

Taller de reparaciones

- Escenario estático (página 175): máquinas permanecen en el mismo lugar.
- Máquinas:
  - Se agregan las coordenadas x,y al tipo `an_entity` para identificar su posición.
  - Tres estados: working (blue), waiting to be mended (yellow), being mended (red).

Taller de reparaciones

- Detalles del programa:

```
enter_class(1,'m',blue);
```

- Cuando una máquina se rompe:

```
current^.col := red;
write_entity(current^.x, current^.y,
             current);
```

Taller de reparaciones

- Número de mecánicos y equipamiento en uso se actualizan en el procedimiento `display`.
- Si se pretende mostrar el movimiento de los mecánicos y equipamiento, deben modelarse como entidades.

Salida visual en EOSimulator

- Basada en íconos.
- Íconos son imágenes.
- Primitivas de salida visual son independientes de las que gobiernan la simulación.
- Componentes dinámicos se actualizan:
  - Cuando ocurren los eventos, pero no instantáneamente.
  - Después de cada avance del tiempo.

Salida visual en EOSimulator

- Para la salida gráfica se definen 3 clases:
  - Display: Esta clase es la que define la ventana donde se desplegará la salida visual.
  - Sprite y TextSprite: Estas clases definen el comportamiento de los objetos que se desplegan en la salida visual.
  - RefreshGraphic: Es un evento C auxiliar cuya responsabilidad es mantener la salida visual sincronizada con el modelo de simulación.

Salida visual en EOSimulator
Display

- `Display` crea la ventana de fondo, dibuja y mueve los objetos que se encuentran en ella. Debe ser un atributo del modelo.

```
class Display : boost::noncopyable {
public:
  Display(int width, int height);
  ~Display();
  void setTitle(const char* title);
  void setBackground(const char* file);
  void setSpeed(double sp);
  double getSpeed() const;
  int width() const;
  int height() const;
  void advance(double offsetTime);
};
```

- Para tener sincronizada la salida visual y la simulación se debe invocar `Display::advance` en cada iteración (del ejecutivo) con un tiempo igual al time beat.

Salida visual en EOSimulator Sprites

- `Sprite` y `TextSprite` son clases que definen los objetos dinámicos de la salida gráfica. Pueden ser creados, destruidos, cambiar de "forma" y moverse libremente por la ventana de salida visual.

```
class Sprite {
public:
   Sprite();
   Sprite(Display& dis, const char* file, double x, double
   y);
   void setImage(const char* file);
   void setMoves(MoveAction m);
   void setVisible(bool v);
   bool getVisible() const;
};
```

- Un sprite desaparece de la ventana de visualización cuando la única referencia a dicho objeto es aquella perteneciente a su display.

Salida visual en EOSimulator Sprites

- `Sprite` y `TextSprite` pueden cambiar de forma con las operaciones `Sprite::setImage` y `TextSprite::setText`

```
spr.setImage("pepito.png"); // cambia el icono del sprite
   por la imagen pepito.png

tspr.setText("Soy una simulación visual"); // cambia el
   texto por el asignado
```

- Para mover un sprite por la ventana se utiliza `Sprite::setMoves`:

```
spr.setMoves(move(2,4,6.0));
spr.setMoves(move(2,4,6).move(0,0,8));
```

- Si se invoca cuando aún hay movimientos asignados, estos se pierden, y el sprite comienza su nuevo movimiento desde su posición actual.

Salida visual en EOSimulator RefreshGraphics

- Como `Display` debe ser sincronizado con el modelo invocando `Display::advance` con el time beat, una forma de hacerlo es en la fase C

```
class RefreshGraphic: public eosim::core::CEvent {
private:
   Display& d;
   double lastTime;
public:
   RefreshGraphic(eosim::core::Model& model, Display&
   disp);
   ~RefreshGraphic();
   void eventRoutine();
};
```

- Este evento C debe ser un atributo del modelo, y debe ser registrado último.