

Prozeßorientierte Simulation

Johann Christoph Strelen

Rheinische Friedrich–Wilhelms–Universität Bonn

Römerstr. 164, 53117 Bonn, Germany

E-mail: strelen@cs.uni-bonn.de

Juli 2004

Bonn

Prozeßorientierte Simulation

Bevorzugt in Simulationssoftware-Paketen

- Realität natürlicher wiedergeben
- Räumliche Komponente deutlicher modelliert
- Graph, Entitäten
- Animation

Top-down-Entwurf

- Graph zuerst grob
- Dann Komponenten genauer

Programmierung: Koroutinen, nicht in C, C++, Fortran und Java ...

Simula

Prozeßorientierte (vorgangsorientierte) Simulation

Spezialfall ereignisorientierter Simulation

Entitäten (mobile Komponenten, Transaktionen, Aufträge)

Netz von *Stationen (Blöcken)*

Komponenten haben Zustände

In jeder Station: Einen oder mehrere *Prozesse*

- Empfängt Entitäten
- Verändert deren Zustand und seinen eigenen
- Wartet auf Ereignisse
- Speichert Entitäten in Warteräumen
- Erzeugt Entitäten
- Fügt sie zusammen oder splittet
- Sendet Entitäten zu anderen Stationen

Spur der Vorgänge, bei jeder Aktivität einen „Tritt“

Prozeßroutinen beschreiben den Ablauf der Prozesse, Schleife

Modell: Mehrere Stationen mit je einem oder mehreren Prozessen

Alle Prozesse laufen parallel ab: Starten, warten, wieder aktiv, terminieren → Koroutinen

Entitäten

- Kunden (Aufträge) in einem Warteschlangennetz
- Nachrichten in einem Kommunikationssystem
- Werkstücke in einer Fabrik
- Ein Monteur, der zu Maschinen geht, um sie zu reparieren
- Gedacht: Marken, deren Anwesenheit bei einer Maschine bedeutet, daß diese defekt ist

Eigenschaften (*Attribute*), zum Beispiel

- Typ
- Eintrittszeitpunkt in das System
- Priorität

Veränderlicher Zustand, zum Beispiel Alter

In Objekten einer Klasse gespeichert

Stationen

- Quellen und Senken für Entitäten
- Auftragsverzögerer
- Warteräume
- Ganze Warteschlangensysteme
- Wegverzweigungen
- Steuerelemente, die bedingt blockieren (Ampel)
- Statistische Zähler

Eigenschaften (*Attribute*), zum Beispiel

- Ankunftsprozeß
- Eigenschaften der Aufträge einer Quelle
- Kapazität eines Warteraumes
- Verteilung der Bediendauern eines Bedieners
- Vorhandene Anzahl von Bedienern
- Bedienstrategie in einem Warteschlangensystem
- Wahrscheinlichkeiten, mit denen abzweigende Wege in einem Verzweigungsknoten eingeschlagen werden

Eigenschaften und Zustände (Variablen) in Objekten einer Klasse

Ein Quellprozeß, zwei Parameter: Zielstation und Rate λ der Erzeugungen

do

```
{  erzeuge einen Kunden;
    // Datenverbund
lege seine Eigenschaften fest;
    // zum Beispiel Erzeugungszeitpunkt,
    // Nummer der Quelle
if (es wird eine Spur aufgezeichnet)
    hinterlasse einen Spurtritt mit den wichtigen Daten;
samme statistische Daten;
    // zähle zum Beispiel die erzeugten Kunden
sende den neuen Kunden zum Eingang der Zielstation;
warte eine Zwischenankunftszeit;
}
while die Simulation ist nicht zu Ende;
```

Station und Prozesse eines Warteschlangensystems

Zwei Arten von Prozessen:

Ein Ankunftsprozeß empfängt Kunden von einer Quelle oder von einem anderen Warteschlangensystem und reiht sie in eine Warteschlange ein

Ein Bedienprozeß entfrent ihn daraus, bedient ihn, und schickt ihn weiter

Strategie

Die Bediendauer kann durch eine Zufallsvariable definiert sein und von anderen Größen abhängen

```
do
{
  warte auf einen Kunden;
  aktualisiere dessen Zustand und den der Station;
    // zum Beispiel Ankunftszeitpunkt,
    // Anzahl der Kunden in der Station, in der Warteschlange
  füge den Kunden in die Warteschlange ein;
  if (es wird eine Spur aufgezeichnet)
    hinterlasse einen Spurtritt mit den wichtigen Daten;
  sammle statistische Daten;
  if (der Bediener ist frei)
    aktiviere diesen;
}
while die Simulation ist nicht zu Ende;
```

Ein Ankunftsprozeß

do

```
{ warte bis ein Kunden in der Warteschlange ist;  
  entnimm der Warteschlange einen Kunden;  
  aktualisiere den Zustand;  
    // zum Beispiel Bediener belegen,  
    // Anzahl der Kunden in der Warteschlange - 1  
  if (es wird eine Spur aufgezeichnet)  
    hinterlasse einen Spurtritt mit den wichtigen Daten;  
  sammle statistische Daten;  
  beginne die Bedienung und warte deren Ende ab;  
  sende den Kunden zur Zielstation;  
  aktualisiere den Zustand;  
}  
while die Simulation ist nicht zu Ende;
```

Ein Bedienprozeß

Station und Prozeß einer Senke

do

```
{ warte auf einen Kunden;  
  löse seinen Datenverbund auf;  
  sammle statistische Daten;  
  if (es wird eine Spur aufgezeichnet)  
      hinterlasse einen Spurtritt mit den wichtigen Daten;  
}
```

while die Simulation ist nicht zu Ende;

Ein Senkenprozeß

Implementierung mit C++

Da es in C++ keine Koroutinen gibt, zerlegen wir die Prozeßroutinen in einzelne Aktivitäten, und diese werden als Funktionen realisiert, die *Aktivitätsroutinen*. Alles, was eine solche Aktivitätsroutine tut, passiert zur selben Modellzeit. *Spontane Aktivitätsroutinen* werden direkt von Aktivitätsroutinen aufgerufen und sofort ganz ausgeführt. Die andere Art von Aktivitätsroutinen sind Ereignisroutinen, die immer vom Simulator aus aufgerufen werden, nachdem ihr Aufruf vorher von einem Prozeß geplant worden war, und die immer in den Simulator zurückkehren.

Stationen mit ihren Attributen, Zuständen und Prozeßroutinen, bestehend aus Aktivitätsroutinen, werden als Klassen programmiert. Diese sind parametrisiert, sodaß sich die Objekte der Klasse, also die Stationen dessen Modells, dann in den Werten der Parameter unterscheiden können.

```

// Prozessroutine, aus Aktivitätsroutinen zusammengesetzt
/*do*/

    Anweisungen;
    /*resume(Simulator);*/

    void Eingang(Entitaet* Kunde) {

    Anweisungen;
    /*resume(Simulator);*/

    }

    void Ereignisroutine(Entitaet* Kunde) {

    Anweisungen;
    /*resume(Aufrufer);*/

    }

    void andere_Aktivitätsroutine() {

/*while(true)*/

```