Vorlesungsreihe Simulation betrieblicher Prozesse

Aufbau diskreter Simulationsmodelle mit SLX

Prof. Dr.-Ing. Thomas Wiedemann
email: wiedem@informatik.htw-dresden.de

HOCHSCHULE FÜR TECHNIK UND WIRTSCHAFT DRESDEN (FH)
Fachbereich Informatik/Mathematik

Diskrete Simulation mit SLX

- Zufallszahlengenerierung
- Ergebnisanalyse
- Modellierung mit SLX
  - Typische Modellstrukturen
  - Debuggen (Pucküberwachung)
- Ausführung von GPSS-Modellen mit SLX
  - Das Prinzip der Compilererweiterung
  - Portierung / Ausführung von GPSS-Modellen
- Anbindung von SLX an andere IT-Systeme
  - Schnittstellen
  - Animation
  - Einbindung von DLL´s

Zufallszahlengenerierung mit SLX

- SLX stellt die Basisklasse rn_stream zur Generierung von gleichverteilten Zufallszahlen in Entsprechung zur Methode von Greenberger bereit
- Die Klasse verfügt über folgende Attribute:
  - seed - aktueller Wert des Generators
  - start - Startwert des Zufallszahlengenerators (Default: 100.000)
  - count – Anzahl der bisher gelieferten Zufallszahlen
  - antithetic – Option für Generierung antithetischer Zufallszahlen der Form (1-X)
  - title - Name für die Ausgabe in Reports
  - histo[0..15] - intern für statistische Zwecke

- Die Deklaration und Initialisierung eines SLX-Zufallszahlengenerator erfolgt mit
  rn_stream *name* [**seed**= *Startwert*] [**antithetic**] [**title** = *Generatorname*]
  wobei die kursiven Texte durch modellspezifische Werte zu ersetzen sind.
- Bsp.: **rn_stream** Ankunft **seed**= *1000* **title** = Ankunftsintervall
- Alle rn_stream-Objekte werden in das System-Set rn_stream_set eingeordnet und stehen dort zentral zur Verfügung.

Erzeugung beliebiger Zufallzahlenverteilungen mit SLX

- Auf der Basis der gleichverteilten Zufallszahlengeneratoren kann eine große Anzahl mathematischer Verteilungsfunktionen generiert werden.
- Die Generierung einzelner Zufallszahlenwerte erfolgt mit einem Aufruf der Form:
  x = Name_Zufallszahlentyp (rn_stream Generatorname, Parameterliste **...)**
- Die Ausprägung der Parameterliste entspricht in der Regel in Typ und Anzahl der mathematischen Vorgabe.

Beispiele mit Syntax der Parameterliste :

- float rv_uniform(rn_stream r, double mean , double spread) // Gleichverteilung
- int rv_discrete_uniform(rn_stream r, int leftpoint , int rightpoint) //diskrete GV
- float rv_normal(rn_stream r, double xmean , double xstd) // Normalverteilung
- float rv_expo(rn_stream r, double mean) // Exponentialverteilung
- float rv_triangular(rn_stream r, double x1 , double x2 ,double x3) // Dreieckvert.
- float rv_gamma(rn_stream r, double alpha, double beta) // Gammavert.
- float frn(rn_stream r) - liefert direkt den generierten Wert von r aus [0,1]
- Beispiele zur Anwendung und weitere Verteilungen sind in der Datei rvtest.slx aufgeführt.

Beispiel zur Anwendung von Zufallszahlen

Verschiedene Zufallsprozesse in einem Modell sollten mit unabhängigen (also mit verschiedenen Startwerten initialisierten) Zufallszahlengeneratoren erzeugt werden.

```
// Variablendeklaration
rn_stream gen_ankunftsrate; double  ankunftsabstand = 100;
rn_stream gen_bedienrate;   double  bedienzeitmean = 50, bedienabw = 30;
...
// Modellierung der Ankünfte
   ankunft= rv_expo(gen_ankunftsrate,  ankunftsabstand ); // Exponentialvertl. ;
   advance (ankunft); // Warten bis zum Generieren
   //  Erzeugung eines neuen Objektes mit new .. activate / fork

        ... // in der Action-Methode des Objektes
        advance ( rv_normal(gen_bedienrate, bedienzeitmean ,bedienabw );
        ... //
```


Modellierung empirischer Verteilungen

- **SLX unterstützt die Realisierung empirischer Verteilungsfunktionen**
- **analoge Vorgehensweise für diskrete und kontinuierliche Verteilungen:**

<u>Generierung über direkt vorgegebene Wertepaare</u>

- **discrete_empirical** *Name_der_Vert* **(** *x1 y1 , x2 y2 , ... Xn yn* **);**
- **continuous_empirical** *Name_der_Vert* **(** *x1 y1 , x2 y2 , ... Xn yn* **);**
- **Die X-Werte müssen aufsteigend von 0 bis 1 definiert werden !**

Alternativ kann auch eine Datei mit den Werten vorgegeben werden:

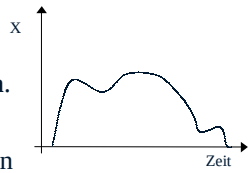
- **continuous_empirical** *Name_der_Vert* **file =** *Dateiname* **;**
- **Die Anweisungen generieren eine Funktion mit dem vorgegebenen Namen, welche als Parameter einen rn_stream erwartet:**
- **Procedure** *Name_der_Vert (rnstream r);*

Bsp.: **discrete_empirical** *Bedienzeit* **(** *0.1 6 , 0.3 10 , 0.6 15, 0.8 20 , 1* **);**

Erlaubt Aufruf mit : **z = BedienZeit(BedienzeitStream);**

Weitere Funktionen zru Modellierung zufälliger Prozesse

Kurvenbasierte Approximation



- zur noch besseren Modellierung von empirischen Verteilungsfunktionen verfügt SLX über die **Bezierfunktion**.
- Bezierfunktionen dienen auch im CAD-Bereich zur numerischen Definitionen von Kurven durch eine Anzahl von Stützpunkten.
- SLX-Funktion: rv_bezier(rn_stream r, bezierdat b);
  wobei b die Menge der Stützpunkte vorgibt (analog zu _empirical)

Hilfsfunktion zur Adaption von Verteilungsfunktionen an vorgegebene Grenzen:

- Häufig sind die praktischen Werte einer Zufallsgröße nur innerhalb von betimmten Grenzen zulässig (z.B. sollten Normalverteilte Bedienzeiten > 0 sein)
- **random_input** t=rv_normal(rstream1, 10, 5) accept (0, 40)
  liefert nur die Werte im Intervall [0,40] als Ergebnisse der Normalverteilung
- weitere Einschränkungen der Gültigkeitsbereiche können natürlich auch mit den Standardbefehlen if-then und while() realisiert werden

Ergebnisdokumentation und –analyse

Bei der Verwendung zufälliger Modellgrößen sind zur korrekten Simulationsauswertung auch entsprechende Statistiken zu erstellen.

Typische Aufgaben und Vorgehensweisen

- Sammlung von Ergebnisdaten zu einer Größe
- Verdichtung der Daten im Zeitmaßstab bis hin zu Werten wie Mittelwert und Streuung
- Speicherung der Ergebnisdaten für mehrere Läufe
- Berechnung der statistischen Daten für alle Läufe

- In SLX und auch allen anderen modernen, diskreten Simulationssystemen werden diese Aufgabe meist durch Standardfunktionen unterstützt.

Definition von Einzelstatistiken

- **SLX stellt eine Klasse random_variable zur Sammlung von Statistiken für eine einzelne Größe bereit**
- **Definition einer random_variable :**

  random_variable [(time | weight)] rv_name

  [histogram start = startwert **width=**Breite **count =** AnzahlIntervalle]
- **[histogram ...] definiert ein optionales Histogramm.**
- Der optionale Parameter [(time | weight)] dient zur "Wichtung der Meßwerte.
  - Werte wie Bedienzeiten benötigen keine Wichtung.
  - Werden Wertveränderungen über die Zeit gemessen, z.B. die Anzahl der Objekte in einer Wartschlange, so müssen die Werte jeweils mit der Zeit gewichtet werden.
  - Der Anwender kann auch eigene Wichtungen, z.B. die Priorität eines Kunden oder Auftrags als Wichtung verwenden
- Jedes Objekt vom Typ random_variable wird in das Set random_variable_set aufgenommen. Die Methoden report und clear können darüber aufgerufen werden.

Beispiel zur Erzeugung eines Warteschlangenhistogramms :

random_variable (time) Warteschlangenstatistik

histogram start = 0 width = 100 count = 20 ;

Aufzeichnung von Einzelstatistiken

- **Bei vorhandener random_variable** rv_name **erfolgt die Aufzeichnung mit**

  tabulate rv_name = Wert [weight = Wichtung]

Bsp.: **tabulate** Warteschlangenstatistik = WS1_Warteschlangen_Laenge;

Der Abruf der statistischen Daten erfolgt mit :

- **x= sample_count(rv_name); // Anzahl der Meßwerte**
- **x= sample_min(rv_name); // Minimum der Meßwerte**
- **x= sample_max(rv_name); // Maximum der Meßwerte**
- **x= sample_sum(rv_name); // Summe der Meßwerte**
- **x= sample_mean(rv_name); // Mittelwert der Meßwerte**
- **x= sample_variance(rv_name); // Varianz der Meßwerte**
- **x= sample_stdev(rv_name); // Standardabweichung der Meßwerte**

Auswertung von Histogrammen

- **Bei vorhandenem Histogramm können außer der tabellarischen Ausgabe auch direkte Ergebniswerte ermittelt werden.**
- Innerhalb einer random_variable kann dazu mit dem pointer histo auf die Attribute des Histogramms zugegriffen werden:

  - das Feld double frequency [0... Class_count-1] liefert die Häufigkeiten

  - die Attribute lower_bound, upper_bound, class_width, class_count, count liefern Hilfsgrößen zur Durchführung von Berechnungen

Beispiel :

- **Bestimmung der Wahrscheinlichkeiten für eine bestimmte Wartezeit**
- **Berechnung der gesuchten Histogrammspalte durch**

  i = (Wartezeit-lowerbound) / class_width
- **Aufsummierung der Einzelhäufigkeiten bis zu i. Spalte**
- **Summe / count ergibt die gesuchte Wahrscheinlichkeit**
- **Achtung: Durch die diskretem Intervalle ist die Genauigkeit abhängig von der Anzahl der Histogrammklassen.**

Weitere Funktionen zur Simulationsauswertung in SLX

Zur Auswertung komplexer Experimente sind weitere Funktionen verfügbar:

- Schätzung von Konfidenzintervallen (build_mean_ci , report_mean_ci)
- Ein vorhandenes Konfidenzintervall kann auch zur Bestimmung der Ergebnisqualität verwendet werden:
  - Durchführung einer ersten Anzahl von Simulationsläufen
  - Bestimmung der Konfidenzintervalle für alle relevanten Größen
  - Falls das Minimum der Quotienten aller Konfidenzintervalle / Mittelwerte eine bestimmte Schranke (z.B. 0.05= 5% Irrtumswahrscheinlichkeit) unterschreitet, kann die Simulation beendet werden, ansonsten sind weitere Experimente durchzuführen!#

- Vergleich von Systemvarianten eines Modells
  - Der Vergleich soll nur die Unterschiede bzgl. des geänderten Modells aufdecken !
  - Dazu sind die Zufallgeneratoren mit gleichen Anfangswerten zu starten (seed)
  - Ergebnisse der Simulation beider Modelle werden in Dateien geschrieben
  - Die SLX-Funktion build_common_main_ci(...) können dann zum Vergleich der Ergebnisdaten eingesetzt werden und liefern den Mittelwert der Differenzen.

Aufbau von Simulationsmodellen mit SLX

- Modellobjekte können durch aktive oder passive SLX-Objekte nachgebildet werden
- SLX-Objekte bestehen aus einem Datenteil und aus einem oder mehreren zugeordneten Pucks
- Die Zuordnung zu dynamischen oder stationären Systemobjekten kann in Abhängigkeit von deren Verhalten erfolgen:
  - relativ selbständige dynamische Objekte (Kunden, Fahrzeuge, individuelle Produkte) können mittels einer Prozeßsicht modelliert werden – sie bewegen sich selbst innerhalb des Modell´s
  - Bei einem komplexen Verhalten der stationären Objekte (automatisierte Fertigungsanlagen, Sachbearbeiter von Drucksachen mit komplexen Entscheidungsverhalten) können auch diese als aktive SLX-Objekte modelliert werden.
  - Auch Mischformen sind möglich, d.h. die Modellierung sowohl der stationären wie dynamischen Systemobjekten mittels aktiver SLX-Objekte.

Ein Beispielmodell

Aufgabe: Modellierung einer Schwimmhalle

Ergebnis der Systemanalyse

- Schwimmhalle läßt aus organisatorischen Gründen nur jede halbe Stunde ein
- Es kommen in der Stoßzeit etwa exponential
- die Gesamtzahl der Schwimmer in der Halle darf 100 nicht überschreiten
- jeder Schwimmer darf maximal zwei Stunden in der Halle bleiben.
- Erfahrungsgemäß verteilt sich die Aufenthaltsdauer wie folgt:
- 60% der Schwimmer bleiben die vollen zwei Stunden +/- 5 min
- 40% der Schwimmer verlassen die Halle bis zu 45 min eher (angenäherte Gleich-verteilung)

Untersuchungsaufgaben

- Welche Kapazität in Personen/Stunde hat die Schwimmhalle ?
- Welche mittlere Wartezeit ergibt für die Warteschlange vor der Schwimmhalle ?
- Welchen Effekt hätte der Verzicht auf den getakteten Einlaß ?

Lösung: siehe Tafellösung + Programm im Internet

Erstellen und Übersetzen von SLX-Programmen

Arbeitsfenster bei Erstellen von SLX-Programmen

- Arbeitsfenster ist dreigeteilt in Quelltext, Ausgabefenster und Debuggfenster
- Editor kann einen Quelltext auch gleichzeitig in mehreren Fenstern darstellen
- unter Options->Font kann die Schriftgröße angepaßt werden
- Im Editor sind alle Standardbefehle (Cut&Paste,F3-Suche,Einrücken etc.) verfügbar.
- Bei rechter Maustaste erscheint ein Kontextmenü zum Einfügen von Breakpoints und zur Anzeige eventuell zugrundeliegender Systemfunktionen (... Expansion)

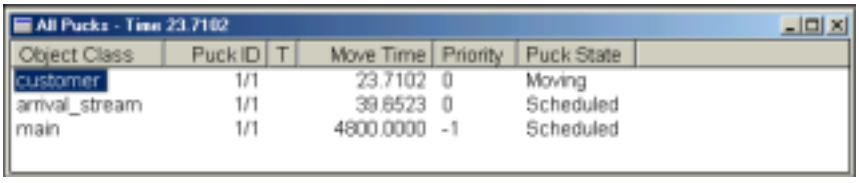
Übersetzen des Modellquelltextes

- über Menübefehl Compiler oder 
- Die spezielle Option->Output->Generated Code zeigt bei der Compilierung den erzeugten Assemblercode an !
- Syntaxfehler werden mit roter Markierung innerhalb des Quelltextes angezeigt (Achtung : Der eingefügte rote Text ist keine permanente Erweiterung)
- Bei der Compilierung werden die syntaktischen Elemente farblich hervorgehoben (fett – SLX-Schlüsselwörter, pink-SLX-Prozeduren, dunkelblau – normaler Quelltext, Hellblau –SLX-Macros, rot - > Fehler)

Ausführen und Testen von SLX-Programmen

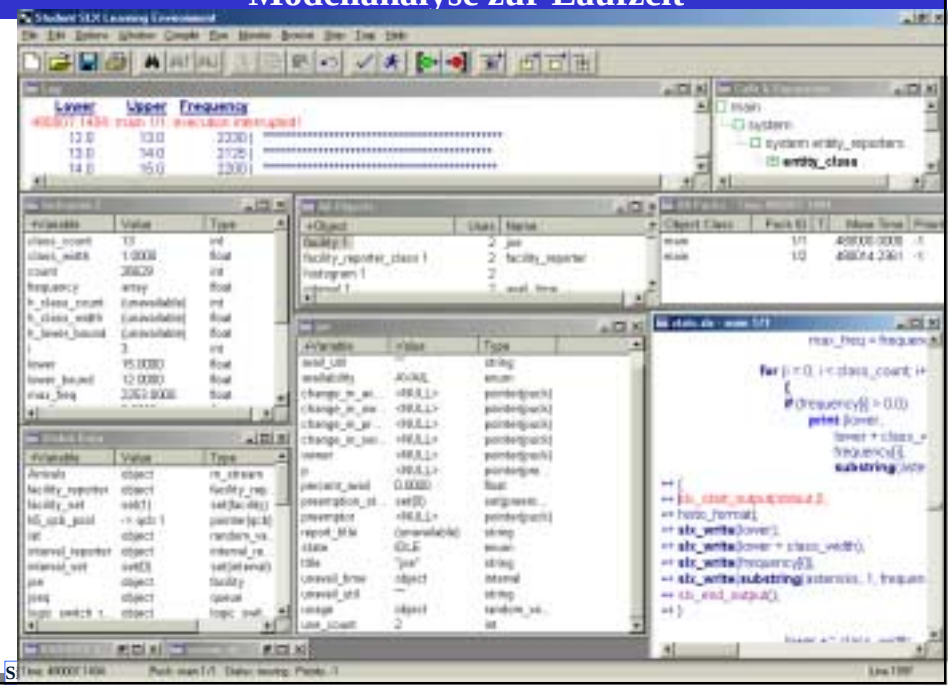
Starten von SLX-Programmen

- Bei fehlerfreier Compilierung kann mit dem Menüpunkt Run oder  gestartet werden.
- Während der Simulation kommt es in der Regel im Log-Window zu Ausgaben.
- Eine Animation oder umfangreichere Ergebnispräsentationen bedürfen spezieller Zusatzbibliotheken oder weiterer Anwendungen (z.B. PROOF für die Animation)
- Abbruch oder Unterbrechung der Simulation durch erneute Betätigung der Starttaste
- Bei angehaltener Simulation können alle Objekte genau untersucht werden:
- Unter Menüpunkt -> Monitor sind alle Puckliste aufgeführt (siehe Puck state)
- Erkennbar sind auch die geplanten Zeiten der nächste Aktivierung



- Ein Beispiel zur Analyse der übrigen Datenstrukturen mit der Option -> Basic Debugging Option ist auf der Folgeseite aufgeführt.

Modellanalyse zur Laufzeit

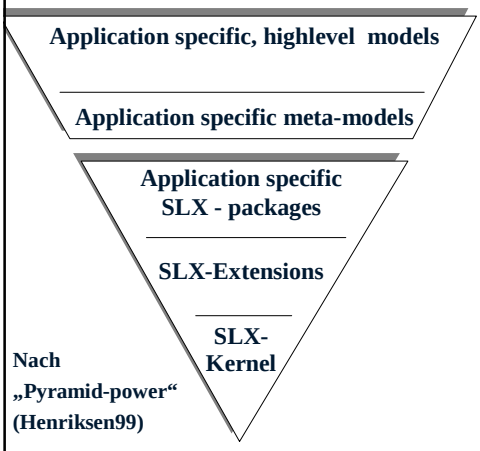


Das Schichtenmodell des SLX-Compilers

Das Prinzip der Compilererweiterung

- SLX ist im Gegensatz zu anderen Simulationssprachen und auch universellen Programmiersprachen in **mehreren Schichten** aufgebaut

The SLX pyramid



Nach „Pyramid-power“ (Henriksen99)

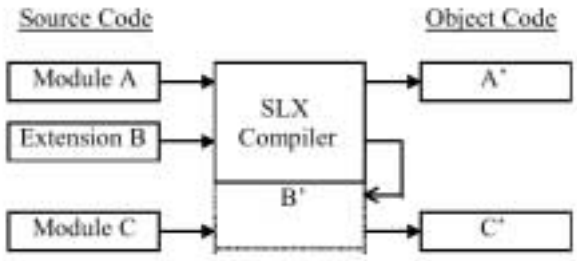
- Auf der höchsten Ebene kann eine völlig andere Syntax und Semantik verwendet werden.
- Das Metamodell definiert eine völlig neue Klasse von Modellen.
- Diese Ebene entspricht der bisherigen Modellierung.
- Die SLX-Extensions bauen aus den Kernel-befehlen komplexere Befehle zusammen (z.B. wird der print-Befehl durch Parsen der Parameter in eine Reihe von Kernelbefehlen überführt.
- Der SLX-Kernel stellt eine kleine Anzahl von Grundfunktionen bereit. (sichtbar über ...expansion)

Der erweiterbare SLX-Compiler

- Bei traditionellen grundlegenden Compilern ist der Befehlssatz in der Regel fixiert.
- Der SLX-Compiler kann durch spezielle Extensions um neue Grundregeln und Syntaxformen erweitert werden.
- Bei der nachfolgenden Übersetzung weiterer Module können die neuen Regeln dann bereits eingesetzt werden.

Vorteile:

- Ausgezeichnete Anpassung an fachspezifische Modellierungsanforderungen
- Effiziente Notation des Quelltexte



Ausführung von GPSS-Modellen mit SLX

- Ein Beispiel für eine Compilererweiterung stellt die GPSS-Erweiterung h5.slx für SLX dar. Nach dem import von h5.slx sind GPSS-Anweisungen verwendbar.
- Von Simulationsanwendern auch schon vorhandene GPSS-Programme damit auf das modernere SLX-System portiert

```
import    <h5>
module barb13 {    facility    joe;
                   queue       joeq;
                   rn_stream Arrivals  seed=100000;
                   rn_stream Service   seed=200000;

 ...
                   {
                   enqueue  joeq;
                   seize    joe;
                   depart   joeq;
                   advance  rv_uniform(Service, 12.0, 18.0);
                   release  joe;
                   terminate;
                   }
...
```

Aufbau der SLX-Extensions

- Die SLX-Extensions bestehen aus einem Teil mit Standard-SLX-Code und einem Teil zum Parsen und Umsetzen des GPSS-Quelltextes auf die SLX-Funktionen.

```
// Definition des Speicherbetretens mit ENTER als SLX-Prozedur
procedure ENTER(inout storage s, float amount_needed)
{ forever  { if (s.remaining < amount_needed)
                    { wait list=s.change_in_too_full; // sleep
                      continue; // and try again
                    }
             if (s.availability != AVAIL)
             { wait list=s.change_in_availability; // sleep
               continue;  // and try again
             }

...  //  Definition des neuen Befehls enter mit Parameterparsing
statement enter #storage [units=#units];
   definition { if (#units == "")
                    expand(#storage)   "ENTER(#, 1.0);\n";
                    else expand(#storage, #units)  "ENTER(#, #);\n";
              }
```


Anbindung von SLX an andere IT-Systeme

Die An- oder Einbindung von SLX in andere Systeme ist wie folgt möglich:

- universeller Datenaustausch über **formatierte Textdateien** (sehr schnell)
- Datenbankmodul für **ODBC-Datenbank** von der UNI-Magdeburg
- Universelles **DLL-Interface** (C-DLL´s können eingebunden werden und erlauben eine Erweiterung der Kernelfunktionen)
- **HLA-Interface** vom Fraunhofer-Institut (HLA ist eine simulations-spezifische Schnittstelle analog zu CORBA, definiert vom amerikanischen Verteidigungsministerium)
- Kopplung mit dem **Optimierungssystem** ISSOP
- Die oben aufgeführten Kopplungsoptionen wurden zum Aufbau eines Kundenspezifischen Simulators genutzt. SLX ist dabei für den Endkunden nicht mehr direkt sichtbar. (Vorführung am Ende des Semesters)

Allgemeine Bewertung von SLX

Vorteile

- SLX ist gegenwärtig das weltweit schnellste Simulationssystem (Performance teilweise bis zu 100 mal besser als bei anderen kommerziellen Produkten)
- Diese Eigenschaft ist insbesondere bei der Optimierung mit einer großen Anzahl von Einzelläufen relevant.
- Keine Grenzen bei der Flexibilität durch beliebige Syntaxkonstruktionen und Ergänung um externe DLL´s

Gewisse Nachteile liegen in der Natur der Quelltextprogrammierung

- Bei größeren Modellen schnell unübersichtlich – Ausweg durch automatisierte Codegenerierung analog zu Bausteinsystemen (Eigenentwicklung).
- Animation und grafische Ergebnisdarstellungen müssen explizit programmiert werden

Fazit:

- bei Abstrichen am Komfort sehr gutes Simulationssystem ohne Grenzen bei der Modellierung