Diskrete Simulation (Masterkurs)

Einführung in das Simulationssystem SLX

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

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

Das Simulationssystem SLX

SLX im Überblick

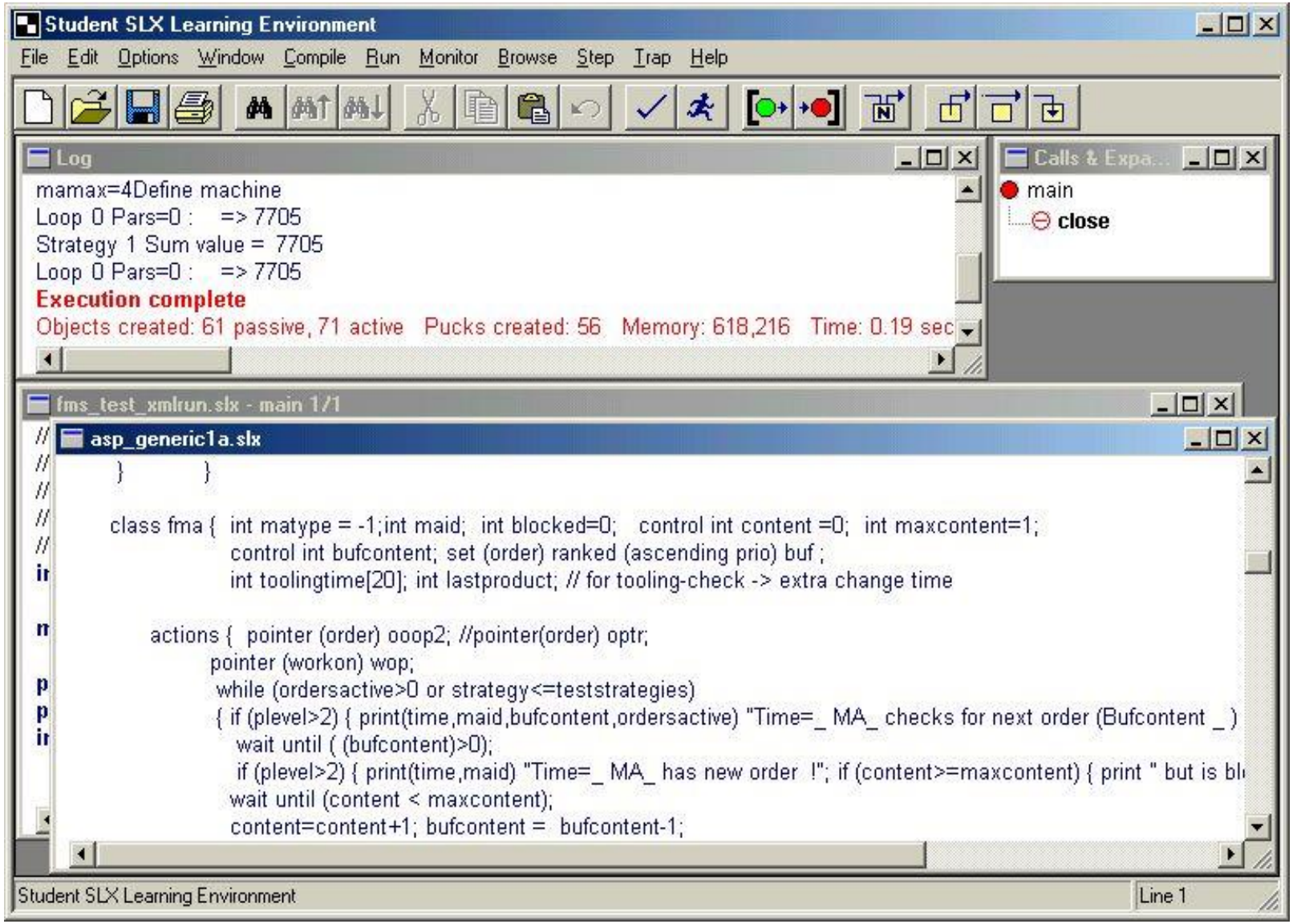
- SLX steht für **S**imulation **L**anguage with e**x**tensibilities
- Hersteller ist die Wolverine Software Corporation USA
- von der gleichen Firma stammen auch das leistungsfähigste GPSS-System GPSS/H und das Animationssystem PROOF
- alle Grundkonzepte der sehr effizienten Ereigniskalenderverwaltung von GPSS/H wurden übernommen
- weiterhin beruht SLX auch auf Konzepten aus SIMULA, C und C++
- SLX ist objektbasiert, aber nicht vollständig objektorientiert

Grundaufbau der Sprache

- alle nicht simulationsbezogenen Funktionen entsprechen C !
- Simulationsbezogene Funktionen verwenden sogenannte Pucks mit frei definierbarer Datenstruktur als dynamische Objekte
- größere Quelltexte können in Module gegliedert werden

Die Bedienoberfläche von SLX

- Dreigeteilt in Quelltext, Ausgabefenster und Debuggfenster



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)

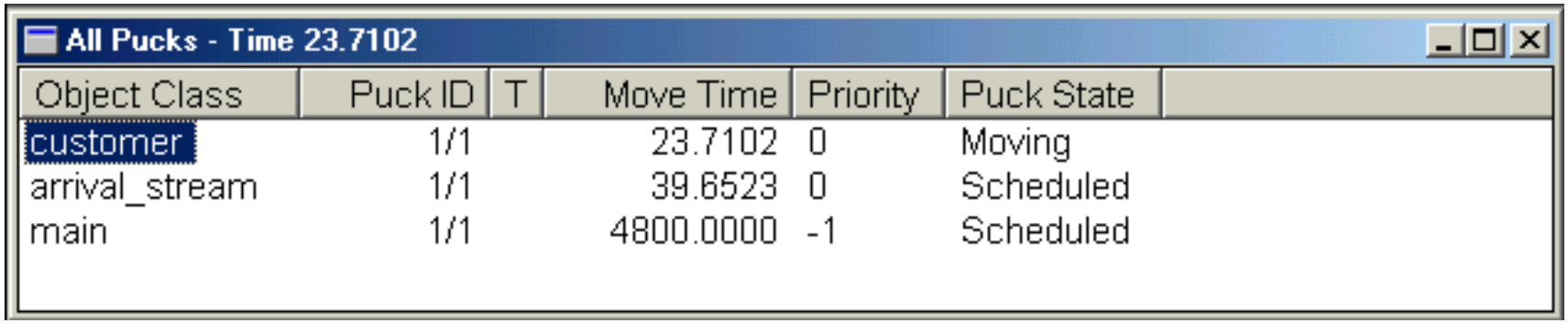
Übersetzen des Modellquelltextes

- 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.

Nicht simulationsbezogene Funktionen von SLX

```
// Example
module basic
int i;

{ // Main proc
  procedure main()
  { int j;

    ...
    print "Hello\n";
  }
}
```

- Falls nachfolgend nicht anders gezeigt, gelten im Zweifelsfall immer die Regeln von C++
- Kommentare mit /* ... */ oder //
- Namen von Funktionen und Variablen beginnen mit Buchstaben und haben sonst eine beliebige Menge von Buchstaben, Ziffern und Unterstriche, alle Zeichen sind signifikant
- Globale Variabeln außerhalb von Prozeduren
- Lokale innerhalb von Prozeduren
- Anweisungen sind generell mit ; abzuschließen
- eine produzedur main() muss generell existieren, diese wird beim Start aufgerufen

SLX-Datentypen und Operatoren

Anfangsinitialisierung

- int - ganzzahlig 32 bit 0
- Float, double - Gleitkomma (identische Speicherung mit 8 byte) 0.0
- boolean - logische Zustände FALSE
- string(anzahl) - Zeichenketten ""
- pointer - Zeiger, zur Vermeidung von häufigen Pointerfehlern können SLX-Pointer im Gegensatz zu C nur auf Objekte verweisen
- Enumeration: typedef name **enum (wert1, wert2, wert3 .... Wertn)**
- Felder: datentyp name[dim1][dim2]...[dimN] Index läuft von 1 bis N !!!

Arithmetische Operatoren (analog zu C)

+ , - , * , \ , % (Modulodivisionsrest), auch Kombination mit Zuweisung wie +=

Relationale Vergleichsoperatoren und logische Operatoren

< , > <= , >= , == , != ! NOT && ||

Inkrement und Dekrement wie bei C a++ --c

SLX-Steuerkonstrukte

Die SLX-Steuerkonstrukte entsprechen denen von C.

- **If-then-else**

  If (boolescher Ausdruck) Einzelanweisung oder Block

  else Einzelanweisung oder Block ;

- **Fallunterscheidung:**

switch (Ganzahliger_Ausdruck)

case Ausdruck : { Anweisung oder Block break; }

- **FOR-Schleife**

  for (Anfangszuweisung; Laufbedingung; Laufzuweisung) Anweisung oder Block ;

  Bsp.: for (i=1; i< 20 ; i++) show(i);

- **WHILE –Schleife**

  While (boolescher Ausdruck) Einzelanweisung oder Block ;

- **DO WHILE –Schleife**

  do Einzelanweisung oder Block while (boolescher Ausdruck) ;

- **Endlos –Schleife**

  forever Einzelanweisung oder Block ;

SLX-Klassen

- Klassen und Objekte in SLX im Sinne der objektorientierten Programmierung
- allerdings keine Polymorphie (Laufzeitprobleme) und Vererbung

Definition:

```
[prefix] class Klassenname  [(parameter_für_Initialisierung ]
{       Attributdefinitionen;   //   datentyp name
        Methodendefinitionen;  //
}
```

- In SLX sind nur vordefinierte Methoden innerhalb von Klassen zulässig. Weitere Methoden sind als normale Prozeduren zu schreiben.

Vordefinierte Methoden mit vorbestimmten Aufrufstatus :

- initial - zur Initialisierung
- **actions - Anweisungen zur eigentlichen Simulationszeit (wichtigster Bereich)**
- clear - Löschen (z.B. zum Simulationsneustart)
- final - beim Beenden der Simulation
- report - zur Ausgabe von Informationen

SLX-Objekte

Bei vorhandener Klassendefinition können Objekte wie nachfolgend definiert werden:

- Statische Definition — Beispiel für Klasse class customer () { ...})

  klassenname objektname; **class person1;**

- Definition eines Pointers und dynamische Generierung mittels new

  pointer (customer) pcustomer;

  pcustomer = new customer( );

  Beispiel für den Aufruf der Objektmethoden

  report (* pcustomer); // ruft report-Methode auf

  clear (* pcustomer); // ruft clear -Methode auf

  destroy(* pcustomer); // zerstört und ruft final-Methode auf

Ein- und Ausgabe in SLX

Ausgabe auf das LOG-Fenster

- print "Demo von Textausgaben";
- print "text _ text " varname; // Wert wird an der Stelle des _ eingefügt, es entscheidet allein die Reihenfolge der Variablen (kein %i oder ähnliches)
- Die Ausgabe in das Logfenster kann per Optionseinstellungen auch in einen Listingfile umgelenkt werden.

Eingabe von Daten:

- Über Tastatur (mit Dateihandler über "stdin")

```
Filedef keyboard name="stdin"
Read file = keyboard;
 prompt = "Please enter a number ";
 end = endinput; // muß weiter unten als Sprungmarke definiert sein
err = error_handler; // muß weiter unten als Sprungmarke definiert sein
I;  // liest diese Variable i ein
```

- Die Ausgabe in beliebige Dateien erfolgt analog zur Tastatureingabe mit der Anweisung write file = ...

Das SLX-Simulationskonzept

- SLX realisiert ein ereignisorientiertes, prozeßbasiertes Konzept,
- Modell besteht aus einer Menge sich gegenseitig beeinflussender Prozesse
- Simulationsuhr ist eine FLOAT-Größe und stellt sich auf den jeweils nächsten Ereigniszeitpunkt ein
- aktueller Wert der Modellzeit kann mit time abgefragt werden
- Bei den meisten diskreten Simulatoren kann ein dynamisches Objekt immer genau einmal als sich im Modell bewegende Einheit verwendet werden.
- kann bei sich beeinflussenden, unterschiedlichen Typen von Faktoren problematisch sein (z.B. gleichzeitige Abhängigkeit von informatorischen und materiellen Größen)

- SLX erlaubt daher eine 1:N –Relation zwischen dem eigentlichen Modellobjekt und den Referenzen im Modell
- Die Referenzen werden als **PUCK** bezeichnet (wie im Eishockey)

Attribute der SLX-Pucks

Die Pucks sind selbst SLX-Objekte und verfügen über folgende Attribute:

- Name - Bezeichner des Pucks
  - **wird gebildet aus dem Namen der generierenden Prozedur oder Klasse**
  - **Instanznr. / lfd. Nummer innerhalb einer Instanz**
  - **Bsp.: main 1/1 ist der erste Puck, welcher automatisch von main() erzeugt wird**
  - **Die Bezeichner sind hilfreich bei der Unterscheidung von Pucks während des Debuggens.**
- poised - Verweis auf ausführbare SLX-Anweisung
- priority - Priorität (durch Anwender modifizierbar und wirkt sich auf Reihenfolge der Abarbeitung bei gleicher Zeit aus))
- mark_ time - Erzeugungszeitpunkt
- moving_time - nächster Bewegungszeitpunkt (falls bekannt)
- puck_state - Status des Pucks – eine der nachfolgenden Zustände:
  - **MOVING - ist aktiv und wartet auf Abarbeitung**
  - **SCHEDULED – ist um eine definierte Zeit verzögert**
  - **WAITING – wartet auf den Eintritt einer Bedingung**
  - **INTERRUPTED – unterbrochen durch spezielle Anweisung**
  - **TERMINATED – beendet, aber infolge existierende Referenzen noch nicht freigegeben**

Simulationsspezifische Anweisungen

- Definierte Verzögerungen

  advance ausdruck; // Ausdruck muß eine Zeit >0 ergeben (float)

- Unbestimmte Verzögerungen

  wait until (Bedingung) ; // wartet solange, bis Bedingung wahr ist

  Innerhalb von Bedingung muß mindestens eine Variable existieren, welche den SLX-Präfix control hat ! Bei einer Änderung dieser Variable wird die wait until – Anweisung automatisch neu berechnet.

  control weist den SLX-Compiler an, spezielle Verwaltungsstrukturen für diese Variable zwecks effizienter Auswertungen zu erzeugen!

  Bsp.; control int inhalt;

  wait until (inhalt ==0); // wartet auf freien "Platz"

  inhalt++; advance 10; inhalt--;

- Totale Blockierung

  wait ; // Prozeß wird vollständig deaktiviert

  reactivate pcustomer; // Reaktivierung muß durch anderes Objekt erfolgen

Realisierung einfacher Modelle

Bei sehr einfachen Modellen kann direkt mit dem ersten Puck und davon abgeleiteten "Kindern" gearbeitet werden:

```
procedure main()
{  print "Start Simulation";
   while (time < 10000)      // Der erste Puck dient zur Generierung der Modellobjekte
        fork  {  // erzeugt eigenständigen Kindsprozess
              advance 50;            // zeitverbrauchende Aktion
              wait until ( inhalt <1 ) ;
               ...   // führe weitere Aktionen aus (auch erneutes Fork)
             terminate;  // vernichte Kindspuck
            }
      advance 100;  // wartet auf neuen Generierungszeitpunkt
}
```

Vorteil: **relativ einfaches Simulationsprogramm**

Nachteil: **keine eigenen Objektattribute möglich**

Modellierung mit Objekten

- Bei komplexeren Modellen reicht die einfache Aufspaltung des ersten Pucks nicht mehr aus. Eine klassenbasierte Modellierung ist dann notwendig.
- Voraussetzungen: Objektklasse mit der Methode actions

```
class customer  (int cusid ) {  int customerid;
                initial   {          customerid= cusid;  }
                actions {            print "Start customer _ " customerid ;
                                     advance 100;
                                     print "Stop customer _ "  customerid;
                        }
                }
```

- Erzeugung des Objektes und Aufruf von actions mit activate **:**

```
activate   new customer();
```

oder mit extra Pointer für späteren Zugriff

```
pointer (customer) pcus1;
pcus1 = new customer();
activate pcus1;
```

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)

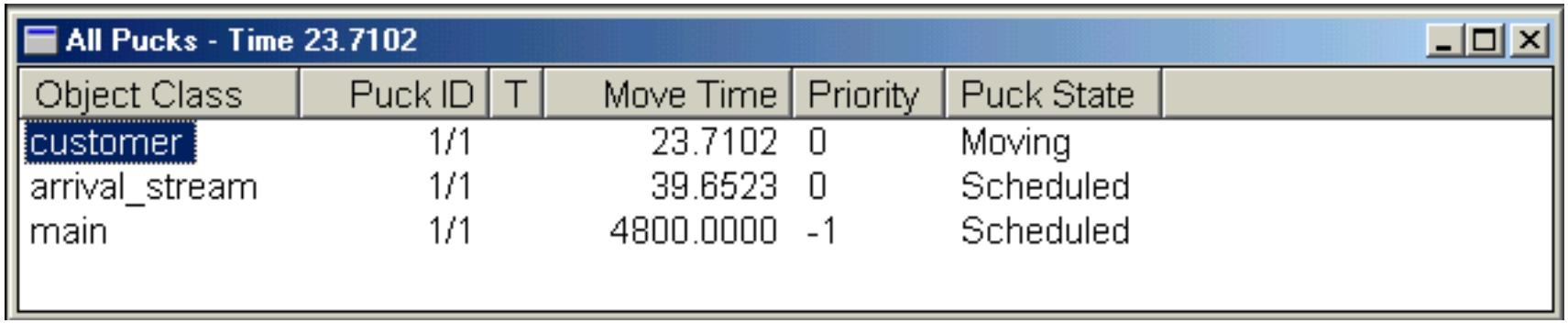
Übersetzen des Modellquelltextes

- 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

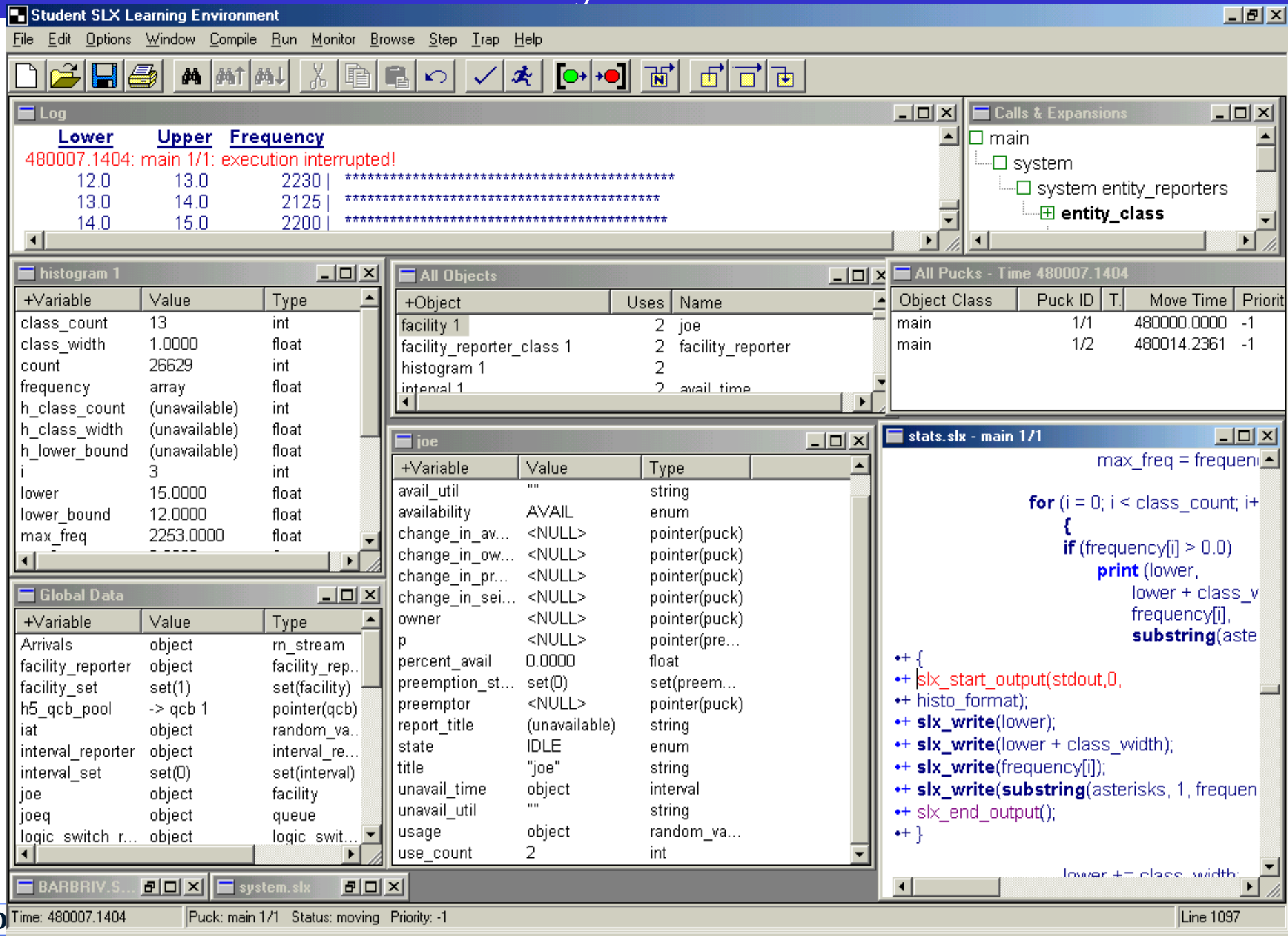
Starten von SLX-Programmen

- 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



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 Ergänzung 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