

4. Simulationssoftware

GLIEDERUNG

- 4.1 Anforderungen an Implementierungssprachen in der Simulation**
 - » Allgemeine programmiersprachliche Anforderungen
 - » Simulationsspezifische Anforderungen
 - » Höhere Programmiersprachen für die Simulation (Delphi, C++, Java)
- 4.2 Beispielhafte Implementierung eines Simulationsmodells in einer höheren Programmiersprache**
 - » Computer-Modell
- 4.3 Typisierung der Simulationssoftware**
 - » Entwicklungsphasen der Simulationssoftware
 - » Animation

4. Simulationssoftware

GLIEDERUNG, FORTS.

- 4.4 Anforderungen an moderne Simulationssysteme**
- 4.5 Auswahlkriterien für Simulationssoftware in der Praxis**
- 4.6 Ausgewählte Beispiele für Simulationssoftware**
 - 4.6.1 Simulationspaket DESMO C++**
 - 4.6.2 Simulationssystem EXTEND**
 - 4.6.3 Übersicht über weitere Softwareprodukte für die Simulation**

4. Simulationssoftware

• Literatur:

- J. Kosturiak/ M. Gregor:
Simulation von Produktionssystemen, Springer 1995, Kap. 7
- L. Oakshott:
Business Modelling and Simulation, Pitman 1997, Kap. 13
- B. Page:
Diskrete Simulation, Springer- Lehrbuch 1991, Kap. 6
- ORMS 1997 SIMULATION SOFTWARE SURVEY, Institute for
Operations Research and the Management Sciences, 1998
<http://lionhrtpub.com/orms/>
- Simulation News Europe Software Comparisons, EUROSIM,
<http://www.eurosim.org/>

4.1 Anforderungen an Implementierungssprachen in der Simulation

- **Anforderungen auf programmiersprachlicher Ebene**
 - Spezielle Anforderungen an einzusetzende Programmiersprachen in der Simulation
- **Bereitstellung von bestimmten Verwaltungs- und Statistikfunktionen zur Entlastung der Programmierung**
 - » Spezielle Simulationssprachen
 - » Allgemeine höhere Programmiersprachen mit Funktionserweiterungen in Form von Programmkomponenten (Bibliotheksmodule/-packages), die dem Anwender angeboten werden
- **Unterscheidung zwischen allgemeinen programmiersprachlichen und simulationsspezifischen Anforderungen**

Allgemeine programmiersprachliche Anforderungen

- **Strukturiertheit und Lesbarkeit**

- Wichtig bei komplexen Simulationsprogrammen
- Selbstdokumentierende Notation oder gute Unterstützung zur Dokumentation unmittelbar während der Implementierung
- Einfache, überschaubare, leicht erlernbare Sprachkonzepte, natürlichsprachliche Schlüsselworte, geeignete Datenstrukturen / -typen

= > Gute Lesbarkeit und Verständlichkeit der Simulationsprogramme

- **Effizienz**

- Simulationsprogramme umfangreich und rechenzeitaufwendig
- Zunehmender PC - Einsatz in der Simulation

=> Effiziente Übersetzer und effizienter Programmcode wichtig

Allgemeine programmiersprachliche Anforderungen

- **Interaktivität**

- Entwicklungs- / Testphase von Simulation-programmen
 - » Interaktive Debug - Unterstützung
- ggf. interaktive Simulationsexperimente

- **Grafikfähigkeit**

- Grafische Ausgabe der Simulationsergebnisse
 - » Nach Simulationsende
 - » Animation während eines Simulationslaufes
- Grafische Modellierung
- Grundfunktionalität für Grafikprogrammierung
 - » Grafik-Bibliothek

Allgemeine programmiersprachliche Anforderungen

- **Modularität**

- Klare, übersichtliche, gut lesbare Simulationsprogrammstruktur durch *Modularisierung* bzw. *Kapselung* (in objektorientierten Sprachen) erreichbar
- Basis für Trennung zwischen speziellem Modell und allgemeinen Simulationsfunktionen
- Grundlage für arbeitsteilige Implementation von komplexen Simulationsmodellen

Simulationsspezifische Anforderungen

Typische Simulationsprogrammkomponenten:

- **Zustandsvariablen**

- Beschreibung des Gesamtzustandes des modellierten Systems zu einem bestimmten Zeitpunkt über Modellvariable zur Darstellung der temporären und permanenten Modellobjekte
 - » *Records / Arrays / Objekte*

- **Temporäre Objekte**

- Dynamische Erzeugung neuer Modellobjekte und deren Vernichtung beim Verlassen des Systems
- **Dynamische Generierung/ Freigabe von dynamischen Datenobjekten**
 - » *NEW* und *DISPOSE* (Delphi)
 - » *new* und *delete*, sowie Konstruktoren und Destruktoren (C++)
 - » *new* mit Konstruktoren und automatisches *Garbage Collecting* (Java)

Typische Simulationsprogrammkomponenten

- **Prozess-/ Ereignisprozeduren (-methoden)**
 - In Prozessen/ Ereignisse i.d.R. komplexe Aktivitäten, Ablaufsteuerungen, Entscheidungsregeln zu implementieren
 - Gute algorithmische Ausdrucksfähigkeit der Implementations-sprache erforderlich
 - Imperative Sprachkonzepte hilfreich
- **Simulationsuhr**
 - Aktueller Stand der Simulationszeit
 - » Automatische Fortschreibung
 - Globale Variable mit speziellem Typ oder global sichtbares Objekt
 - » Typ / Klasse "Simulationszeit"
 - » evtl. aktiv, d.h. mit eigenem Thread
 - Alle zeitbezogenen Variablen von diesem Typ/ Klasse abgeleitet

=> *Benutzerdefinierte Datentypen*

Typische Simulationsprogrammkomponenten

- **Ereignisliste mit Zeitführungs- und Listenmanipulationsroutinen**
 - Alle Simulationsereignisse geordnet nach Zeit und Priorität in *Ereignisliste*
 - *Zeitführungsroutine*
 - » Ermittlung nächstes Ereigniss in Ereignisliste
 - » Vorstellen der Simulationsuhr
 - Spezielle Listenoperationen
 - » Einfügen bzw. Verschieben / Entfernen bereits angesetzter Ereignisse

=> *Dynamische Datenstrukturen*

 - » Pointer - / Record - Typen (Modula und C++)
 - » Klassen mit dynamischen Elementen (z.B. die Klasse Vector), statt Pointer (Java)

Typische Simulationsprogrammkomponenten

- **Warteschlangen mit Verwaltungsroutinen**
 - Wichtige Basiskomponenten für Bedienungssysteme
 - Neben FIFO ggf. andere Warteschlangendisziplinen
 - Routinen zur Listenverarbeitung:
 - » Einfügen / Entfernen / Suchen von Objekten

=> *Dynamische Datenstrukturen*
- **Zufallszahlengeneratoren und Verteilungsfunktionen**
 - Mehrere unabhängige Zufallszahlenströme für Zufallsprozesse in Simulationsmodell nötig
 - » 0/1- Zufallszahlen
 - » Andere Verteilungstypen:
Gleich-, Normal-, Exponentialverteilung, etc.

=> *Mathematische Standardfunktionen*

 - » Sprach-Bibliothek

Typische Simulationsprogrammkomponenten

- **Statistische Auswertung der Ergebnisse**
 - Funktionen für Mittelwert, Standardabweichung,, automatische Ablaufverfolgung (Trace)

=> *Mathematische Standardfunktionen*
- **Protokoll der Simulationsergebnisse**
 - Einfache Format- und Tabellierungsfunktionen

=> *Leistungsfähige Ein-/Ausgabefunktionen*
- **Parallele Prozesse**
 - Prozessorientierter Ansatz (quasi-)parallele Prozesse, die miteinander interagieren
 - Koroutinen-Konzept / Threads als Basis
 - » Abbildung von Prozessen auf Programmebene durch Prozeduren mit wechselseitiger Aktivierung ("Prozeßumschaltung")
 - » Threads als eigene aktive Objekte

Höhere Programmiersprachen für die Simulation

• Beispiel: Delphi als "Simulationssprache"

- Hohe Lesbarkeit, gute Strukturierung
- Effizienz
 - » Im Vergleich zu klassischen Simulationssprachen
- Interaktive Debug-Unterstützung
 - » compilerabhängig
- Modularität
- Dynamische Datenstrukturen
- Benutzerdefinierte Datentypen
- Mathem. Standardfunktionen in Bibliotheksmodul
- Threads für parallele Prozesse
- Nachteil:
 - » Unzureichende E/A-Funktionen im Sprachkern

Höhere Programmiersprachen für die Simulation

• Beispiel: C++ als "Simulationssprache"

- Sehr hohe Effizienz:
 - » direkte Verwendung von schnellem C-Code möglich
- Interaktive Debug-Unterstützung
 - » compilerabhängig
- Modularität
- Dynamische Datenstrukturen, Pointer
- Benutzerdefinierte Datentypen und Klassen
- Mathem. Standardfunktionen in Bibliotheken
- Koroutinen nicht als Sprachelement vorhanden, aber leicht implementierbar in einem neuem Modul
- E/A-Funktionen:
 - » umfangreiche Bibliotheken, aber aufwendige Programmierung

Höhere Programmiersprachen für die Simulation

- **Beispiel: C++ als "Simulationssprache", Forts.**
 - Nachteile:
 - » Schlechte Lesbarkeit
 - » Strukturierungsmöglichkeiten begrenzt.
 - » Kapselung kann durch globale Variablen und Funktionen sowie Freunklassen umgangen werden.
- **Beispiel: Java als "Simulationssprache"**
 - Gute Lesbarkeit (im Vergleich zu C++),
 - Dokumentationserstellung parallel zur Implementierung (*javadoc*)
 - Gliederungsmöglichkeiten in *packages*
 - Modularität durch Kapselung der Objekte
 - Dynamische Datenstrukturen nur durch abstrakte Datentypen
 - » z.B. die Klasse *Vector*, keine Pointer

Höhere Programmiersprachen für die Simulation

- **Beispiel: Java als "Simulationssprache", Forts.**
 - Benutzerdefinierte Klassen (abstrakte Datentypen)
 - Koroutinen nicht als Sprachelement vorhanden, aber leicht implementierbar in einem neuem Modul
 - » aktive Objekte durch Threads realisierbar
 - vielfältige E/A-Funktionen in Standardpackages
 - Nachteile:
 - » Mathem. Standardfunktionen in Bibliotheken noch unzureichend
 - » Interaktive Debug-Unterstützung:
compilerabhängig, meist noch unausgereift
 - » Mäßige Effizienz
 - » Aber:
 - » Verwendung von schnellem C-Code möglich
 - » Compilerentwicklung bietet noch Potential

4.2 Beispielimplementierung eines Simulationsmodells in einer höheren Programmiersprache

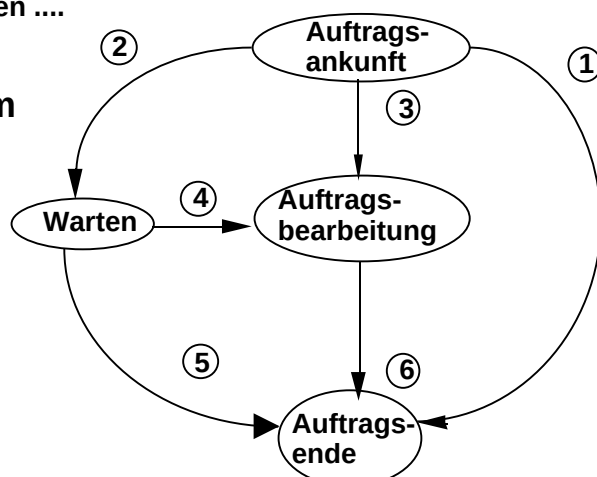
• Beispiel: Rechensystem-Modell

- Rechensystem als exemplarisches Bedienungssystem mit mehreren Service-Stellen und begrenztem Warteraum
 - » s. Vorlesung, Kap. 2.5
- Systemspezifikation:
 - » 3 identische Compute-Server
 - » Benutzeraufträge von dezentralen PCs über Wählleitungen
 - » 11 Wählleitungen verfügbar
 - » 3 Benutzerklassen mit unterschiedlicher Priorität
 - Diplomanden (Prio 1) mit 20%-Anteil
 - Lokale Übungsteilnehmer (Prio 2) mit 40%-Anteil
 - Dezentrale Übungsteilnehmer (Prio 3) mit 40%-Anteil
 - » Unterschiedliche Bearbeitungszeiten der Benutzeraufträge
 - Diplomanden: Exponentialverteilt mit Mittelwert =
 - Übungsteilnehmer: Exponentialverteilt mit Mittelwert =

Beispielimplementierung eines Simulationsmodells in einer höheren Programmiersprache

- Aufgabenstellung:
 - » Abschätzung folgender Leistungsgrößen:
 - Mittlere Benutzerwartezeit (nach Benutzergruppen)
 - Mittlere Warteschlangenlänge
 - Auslastung der Server
 - Auslastung der Wählleitungen

– Zustandsübergangsdiagramm



Implementierung des Rechensystem-Modells in einer höheren Programmiersprache

- **Ereignisorientierte Modellversion des Rechensystems**
 - “Implementationsnahe Modellformulierung” dicht an interner Simulationsablaufsteuerung
 - Identifikation möglicher Ereignisse an Hand des Zustandsübergangsdiagramms
- **Identifikation von 4 Ereignistypen**
 - 1) **Auftragsankunft :**
 - » Generierung neuer Auftrag mit zugehörigem Objekt, wenn freie Wählleitung vorhanden.
 - » Zuteilung eines Servers oder Einreihen in Warteschlange
 - 2.) **Berechnungssende :**
 - » Freigabe des Servers
 - » Zuweisung nächster Auftrag nach Prioritäten

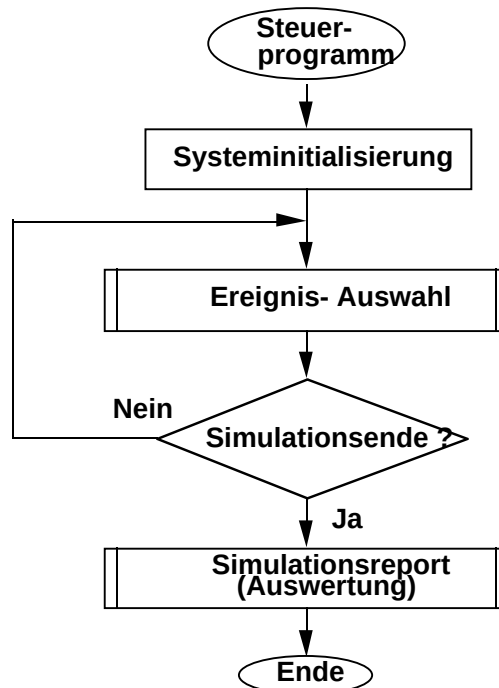
Implementierung des Rechensystem-Modells in einer höheren Programmiersprache

- **Identifikation von 4 Ereignistypen, Forts.**
 - 3) **Ende der Wartebereitschaft :**
 - » Löschen des Auftrages mit abgelaufener Wartebereitschaft aus der Warteschlange.
 - 4) **Simulationsende :**
 - » Ende der Simulation mit Berechnung der Simulationsergebnisse

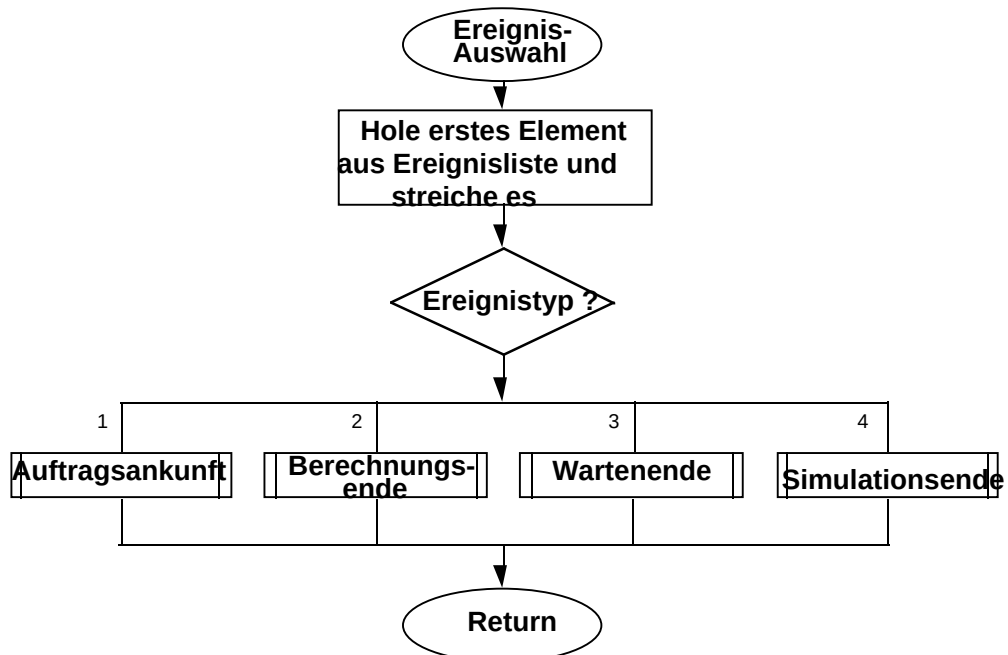
=> Ereignisprozeduren/ -methoden
- **Organisationsprozeduren des Simulationsprogramms**
 - » Simulationssteuerung mit Ereignisauswahl
 - » Zeitführung mit Aktivierung der zugehörigen Ereignisroutine für einen Ereignistyp
 - » Initialisierungsprozedur/ -methode
 - » Auswertungs- und Reportprozedur/ -methode

Implementierung des Rechengsystem-Modells in einer höheren Programmiersprache

**Steuerprozedur/
-methode**



Implementierung des Rechengsystem-Modells in einer höheren Programmiersprache

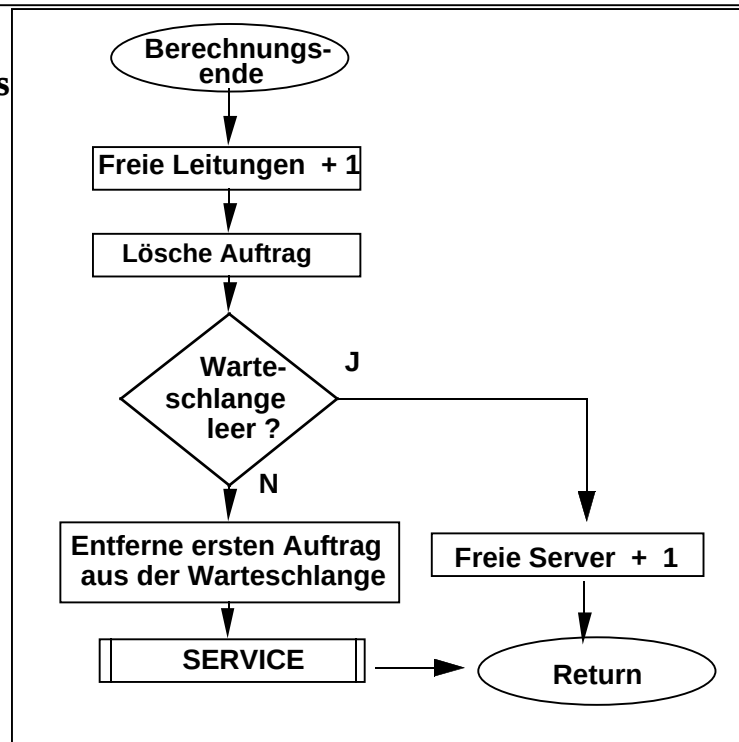


Flussdiagramm für Ereignisauswahlprozedur/-methode

Implementierung des Rechensystem-Modells in einer höheren Programmiersprache

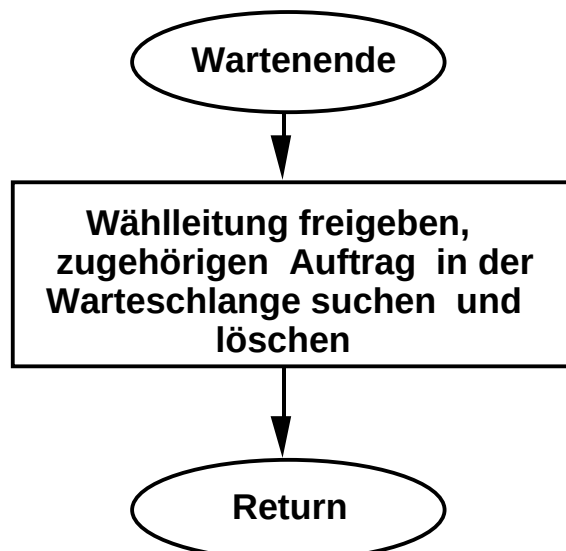
Flussdiagramm Ereignis
"Auftragsankunft",
s. Längsfolien

Flussdiagramm
des Ereignisses
"Berechnungsende"



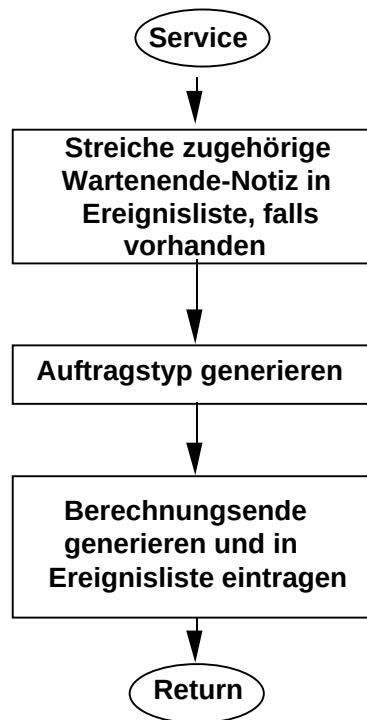
Implementierung des Rechensystem-Modells in einer höheren Programmiersprache

Flussdiagramm Ereignis "Ende der Wartebereitschaft"



Implementierung des Rechengesystem-Modells in einer höheren Programmiersprache

Flussdiagramm für
Prozedur/ Methode
"Service" für Auftrags-
bearbeitungsvorgang



Strukturierung des Simulationsprogramms

- Aufteilung des Simulationsprogramms nach simulationsspezifischen und modellspezifischen Modulen/ Prozeduren (Packages/ Methoden)
- Simulationsspezifische Module (Packages)
 - Simulationsverwaltung mit *Eventchain*:
 - » Eintrag in Ereignisliste mit *Schedule*
 - » Auswahl des nächsten Ereignisses mit *NextEvent*
 - Bereitstellung von Zufallszahlen bestimmter Verteilungen mit *Distributions*:
 - » *Normal, Exponential,...*
 - Bereitstellung von Warteschlangenfunktionalität mit *Queue*:
 - » *Insert, Find, Remove...*

Strukturierung des Simulationsprogramms

- **Modellspezifische Module (Packages)**
 - Modul *Rechensystem*
 - 4 Ereignisprozeduren / -methoden
 - » *Auftragsankunft*
 - » *Berechnungsende*
 - » *Wartenende*
 - » *Simulationseende*
 - Prozedur/ Methode *Service* für den Auftragsbearbeitungsvorgang
- **Trennung simulations-/ modellspezifische Programmteile Grundlage für Wiederverwendbarkeit**
 - » Abbildung Modularisierung eines Simulationsprogramms,
 - s. Längsfolien
 - » Einstieg in Simulationssoftware
 - » Basiskomponenten für ereignisorientiertes Simulationspaket