6 Simulationssoftware

Basisfunktionalität für die ereignisdiskrete Simulation:

- Generierung von ZZs aus vorgegebener Verteilung
- Simulationszeitverwaltung
- Listenverarbeitung und dynamische Speicherverwaltung
- Statistische Verfahren zur Auswertung

Wünschenswert und notwendig für komplexere Probleme

- Prozessorientierte Beschreibungsmöglichkeit in Form von Threads, Koroutinen etc.

Nach heutigem Stand der Technik weitergehende Forderungen

- Graphische Spezifikation (modellieren nicht programmieren)
- Graphische Ausgaben inkl. Animation
- Komponentenbibliotheken
- Integration von Methoden zur Datenmodellierung
- Experimentsteuerung

Lernziele dieses Kapitels:

- Kennen lernen der historischen Entwicklung
- Übersicht über heute verwendete Ansätze erhalten
- Einblick in einige Beispiele für reale Softwarewerkzeuge erhalten
- Aktuelle Entwicklungen kennen lernen

Literatur

- Law Kap. 3
- Banks et al Kap. 4
- Diverse Originalartikel

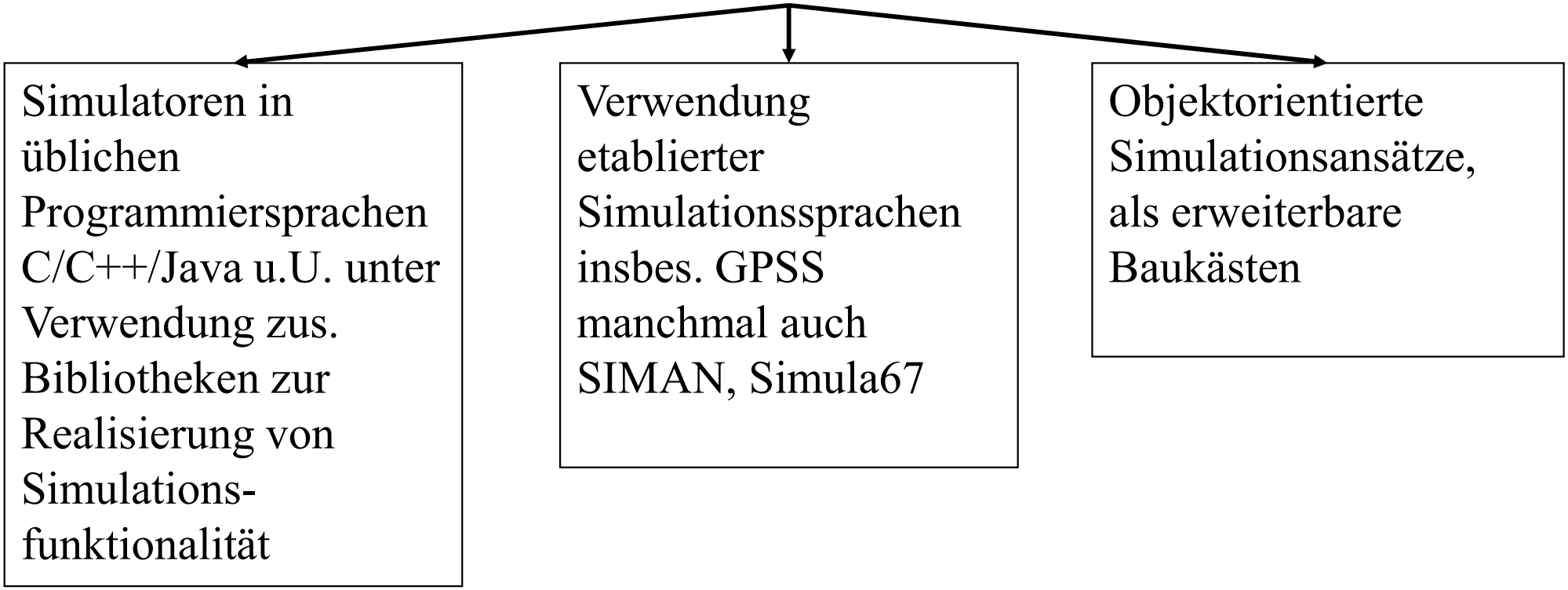
Historische Entwicklung

- seit 1960 erste Simulationsprogramme in FORTRAN
- 1961-1965 Erweiterung von Programmiersprachen um Simulationsfunktionalität

  Beispiele
  - GPSS (General Purpose Simulation System)

    IBM-Entwicklung, Blockdiagramm-Sicht auf Systeme
  - SIMSCRIPT als FORTRAN-Erweiterung 1965
  - GASP als ALGOL später FORTRAN-Erweiterung
- 1966-1975 Entwicklung von Simulationssprachen
  - Simula67 als erste „objektorientierte“ Sprache
  - GPSS/H als Erweiterung von GPSS
- 1976-1985 Simulation auf Mikro-Rechnern und PCs
  - SIMAN erste Sprache unter DOS auf IBM PC mit Realisierung event scheduling in FORTRAN
- seit 1986 Entwicklung integrierter Simulationsumgebungen

Programmiersprachen zur Simulation



Heute

- nur noch für kleinere Beispielmodelle genutzt
- in wenigen Ausnahmefällen für reale Systemsimulationen

Heutige Situation

- Integrierte Simulationsumgebungen für spez. Anwendungsgebiete
  Hauptanwendungsgebiete
  - Produktionssysteme
  - Computer- und Kommunikationsnetze
  - Service-Systeme
- Agentenorientierte Simulation
  Modelle betehen aus einer Menge autonomer interagierender Agenten
  - Modellierung autonomer Systeme (oft mit menschlichen Akteuren)
- Integration von Simulations-funktionalität in Spezifikationsansätze
  Beispiele
  - Spezifikationssprachen
  - Prozessketten

Wir betrachten etwas detaillierter

- GPSS als historische Simulationssprache
- Simulation von Warteschlangennetzen
- Arena als kommerzielles Werkzeug mit den Anwendungsgebieten Fertigungstechnik, Geschäftsprozesse
- OMNeT++ als C++ basierter Netzwerksimulator
- AnyLogic als Simulationsumgebung mit mehreren Paradigmen (Details dazu in den Übungen)

GPSS

Sprache+System zur Simulation warteschlangennetzartiger Modelle

- Geschichte reicht bis in die 50er Jahre zurück
    - Entwicklungsschritte GPS/GPSS/GPSS II/ III .., IBM geprägt
- Aufgrund von Alter (und Steinzeit-Erscheinungsbild) kaum noch genutzt

Heute in unterschiedlichen Versionen verfügbar

- Mit z.T. großen Unterschieden, aber identischer Weltsicht (i.d.R. material-orientiert)
    - Einheiten (hier transactions) bewegen sich über
    - Ressourcen (hier equipment entities)
- Modellspezifikation
    - Beschreibung verschiedener Verkehrsmuster (transactions)
    - Generierung neuer Objekte im Objekt selbst
    - Benötigte Ressourcen werden in Verkehrsobjekten genannt und nicht eigens deklariert

GPSS als Sprache stark von Lochkarten-/Flussdiagrammen geprägt

Entwurf von transactions empfohlen in Form

- bestimmten Typen von Flussdiagrammen
- bestehend aus (unterschiedlichen) Kästchen blocks und
- Pfeilen zur Darstellung der Kontrollflussrichtung

Flussdiagramme ursprünglich als „Abbild eines Lochkartenstapels“ gedacht, heute initiale Form einer graphischen Modellspezifikation

Jede GPSS-Version definiert durch

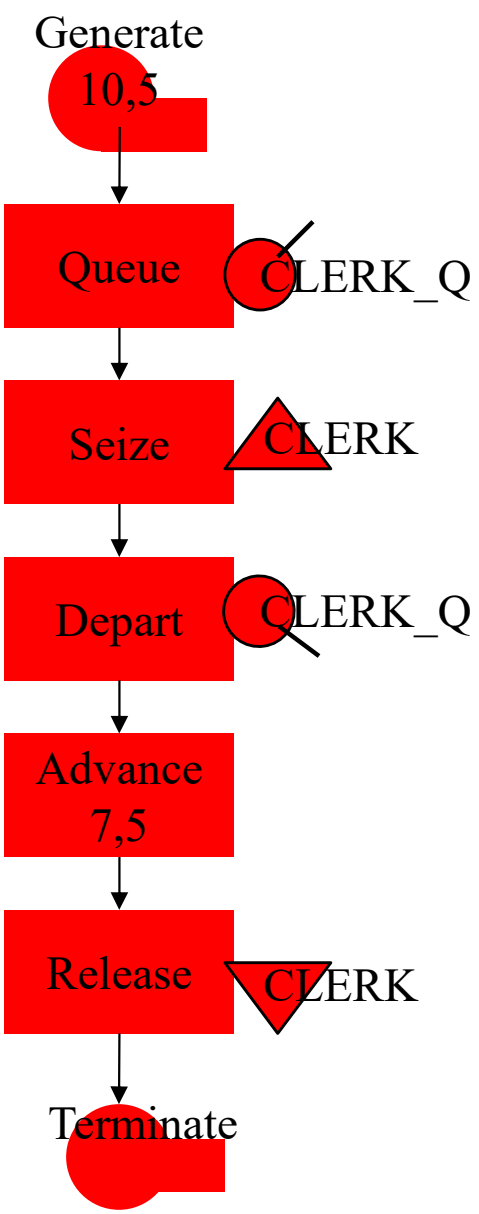
- ihre entities (Einheiten)
- deren attributes (Eigenschaften: Konstanten, Variablen)

Jedes entity (Typ) besitzt (per default) charakteristische attributes

- die im Verlauf der Simulation automatisch aktualisiert werden
- auf deren Werte zugegriffen werden kann

Identifikation von entities über Nummern ,
von attributes über Standardnamen

Beispiel Bankschalter



Kunden treffen ein in Intervallen [5,15]-gleichverteilt diskret

warten in „queue“

Angestellten belegen

verlassen „queue“

bedienen lassen für Intervall [2,12]

Angestellten freigeben

- Generelles wait_until wird ersetzt durch Warten auf Ressource
- Implizite Deklaration von CLERK und CLERK_Q als Bediener und Warteschlange
  Explizite Belegung-Freigabe von Bedienern durch SEIZE und RELEASE
- Zusätzlich vorhanden Strukturen zur Datensammlung
  (queues (!), frequency table)

GPSS-Programm

```
SIMULATE
GENERATE    10,5
QUEUE       1
SEIZE       1
DEPART      1
ADVANCE     7,5
RELEASE     1
TERMINATE   1
START       500
END
```

Start + Dauer Simulation

- Einfache Notierbarkeit einfacher Modelle
- Automatische Statistiken
- Gute Debugging-Hilfen
- Wenn Compiler vorhanden hinreichend schnell (oft nur interpretiert)
- Echte Ein-/Ausgabe erst in neueren Versionen (vorher Umweg über FORTRAN)
- GPSS/H mit Anbindung an Animationstool PROOF

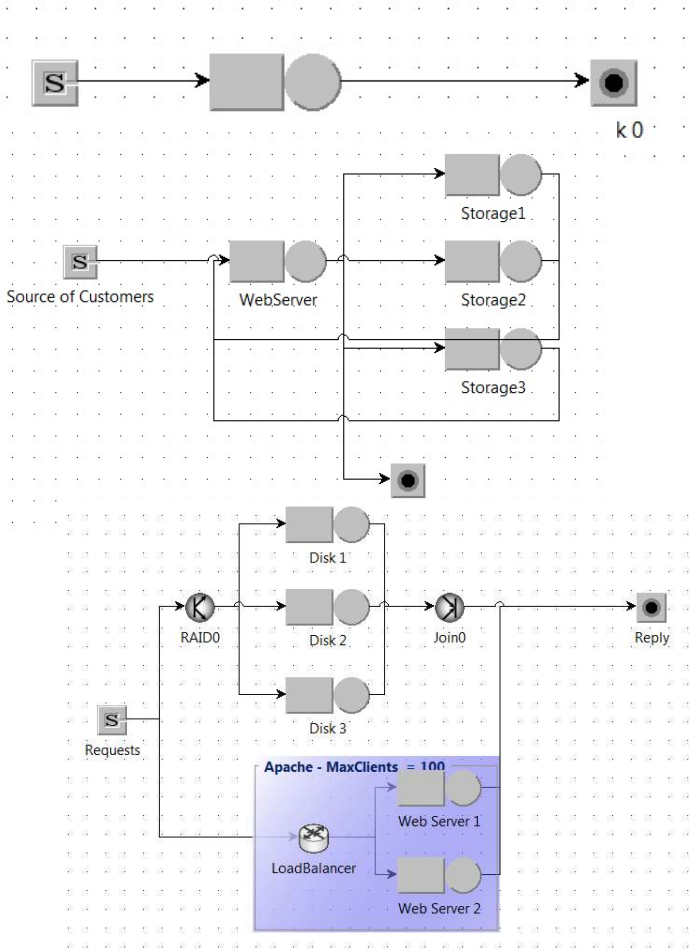
Insgesamt Konzepte überholt!

Warteschlangennetze

Weitverbreitete Modellklasse: Kunden verlangen Bearbeitung an Ressourcen

Unterschiedliche Klassen existieren:

- Einzelne Warte- oder Verlustsysteme
- Offene oder geschlossene Netze
- Erweiterte Warteschlangennetze



Vorteile von Warteschlangennetzen:

- Einfache und intuitive Modellierung von Bedienprozessen
- Diverse Erweiterungen der Basismodellklasse etabliert
- Graphische Darstellung (inkl. Animation)
- Gut fundierte, über die Simulation hinausgehende, Theorie
- Werkzeuge (wie JMT http://jmt.sourceforge.net/) verbinden Simulation, Animation und mathematische Analysetechniken

Grenzen von Warteschlangennetzen:

- Modellwelt begrenzt die Modellierungsmächtigkeit (simultane Ressourcenbelegung, Synchronisation, ….)

Oft dienen Warteschlangennetze als Basis komplexerer Modellierungsformalismen (siehe Arena)

Arena

Umfangreiche Simulationsumgebung für Produktionssysteme und Geschäftsprozesse

- Modelle werden graphisch spezifiziert und
- in SIMAN-Programme umgewandelt

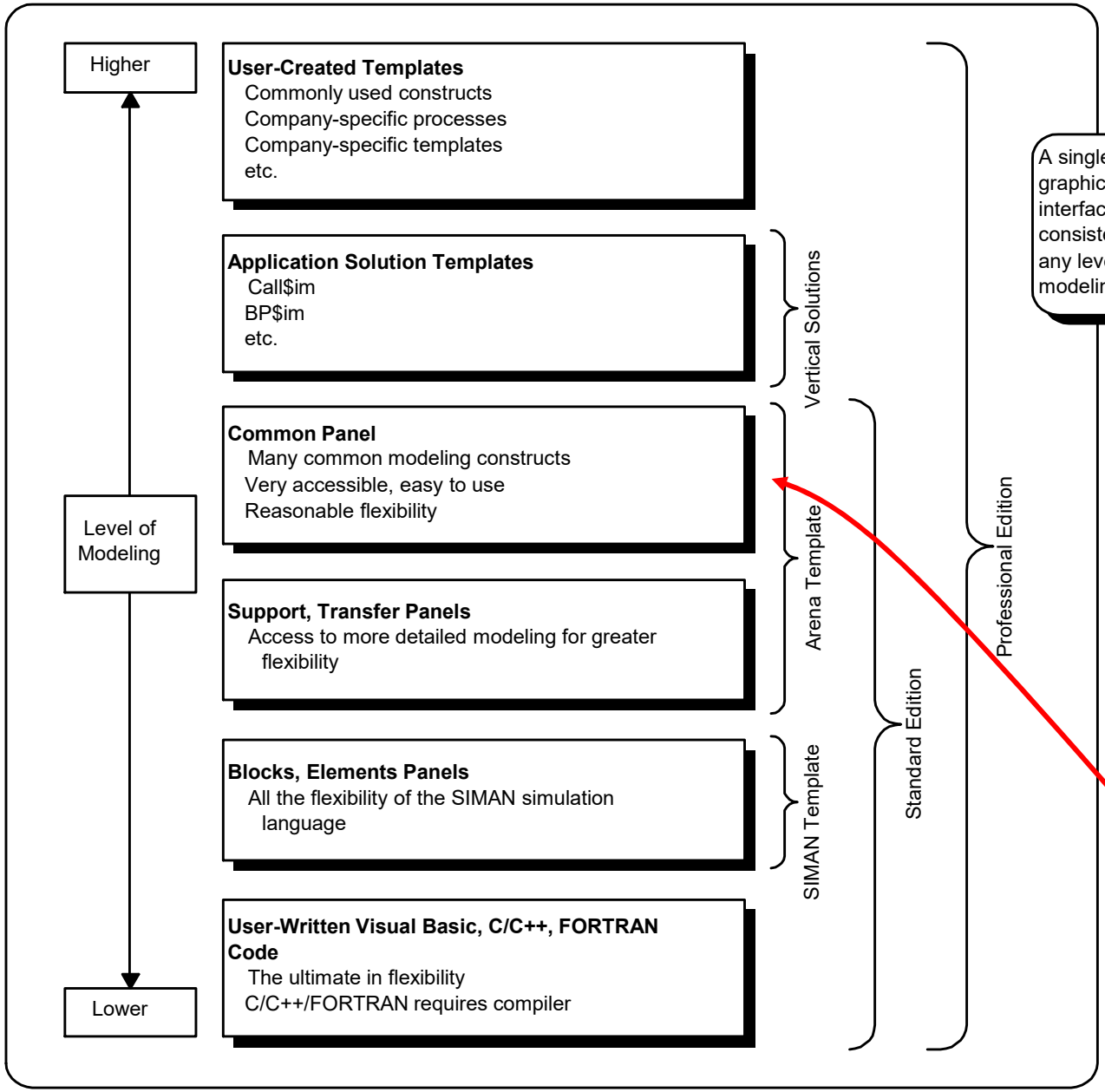
Bausteinorientierter und modularer Modellaufbau

- Vordefinierte Bausteine
- Möglichkeit der Definition von templates für spezielle Anwendungen
- Komplexe Strategien erfordern u.U. Rückgriff auf SIMAN und Verwendung globaler Variablen

Arena ist ein kommerzielles Werkzeug

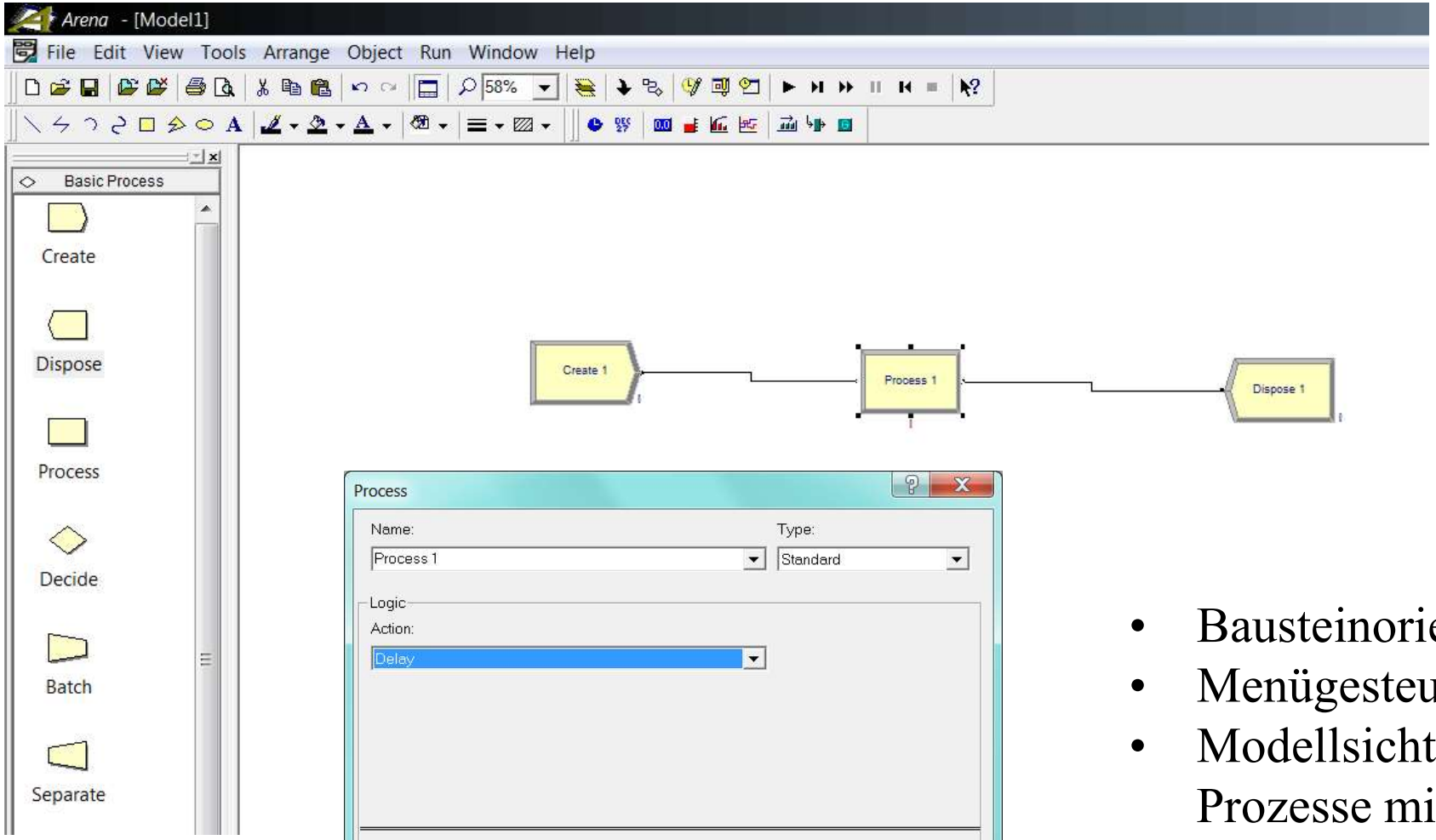
- Freie Studentenversion
  (mit eingeschränkter Funktionalität, Modellgröße)

Modellierungsebenen in Arena



- Mischung der Ebenen in einem Modell ist möglich
- Üblicherweise Modellierung im common panel mit einer Art von Flussdiagrammen

Aus W.D. Kelton, R. P. Sadowski, D. A. Sadowski. Simulation with Arena, Mc Graw Hill 2002



- Bausteinorientiert
- Menügesteuert
- Modellsicht: Prozesse mittels Flussdiagrammen beschrieben
- Nutzung von Ressourcen

Modellspezifikation Mischung aus graphischer Beschreibung und Menüeingaben

Möglichkeit der Definition verschiedener Sichten auf ein Modell inkl. Animationssicht (2D, neuere Version auch 3D)

Zahlreich Zusatzwerkzeuge

- input analyzer zur Modellierung von Eingabedaten
  - Anpassung unterschiedlicher Verteilungen inkl. Parameterschätzung
  - Ausführung von Anpassungstests (inkl. K.S. und χ^2 – Test)
  - Direkte Übertragung der Verteilungen in ein Modell inkl. empirischer Verteilungen
- output analyzer zur Analyse von Simulationsresultaten
  - Konfidenzintervallberechnung (auch online während der Simulation)
  - Schätzung Autokorrelationskoeffizienten
  - Graphische Aufbereitung

Insbesondere im Bereich der Simulation von Produktionssystemen, Materialflusssystemen,… existieren viele weitere Softwarewerkzeuge, z. B:

AutoMod, Extend, QUEST, ProModel, Taylor, Witness, ..

Alle

- kommerziell (und damit teuer)
- mit graphischer Benutzeroberfläche (oft 3D Animation)
- mit Modellbibliotheken für spezifische Anwendungen

Aber durchaus mit spezifischer Schwerpunktsetzung

Aber klare Orientierung am Anwendungsgebiet

$\Rightarrow$ nur eingeschränkt geeignet für die Modellierung in anderen Anwendungsgebieten

Netzwerksimulation

Weiteres großes Anwendungsgebiet für Simulatoren sind Computer- und Kommunikationsnetze

- Es existieren mehrere kommerzielle Werkzeuge z.B. Opnet
- Aber auch sehr gute freie Werkzeuge wie NS-3 oder OMNeT++

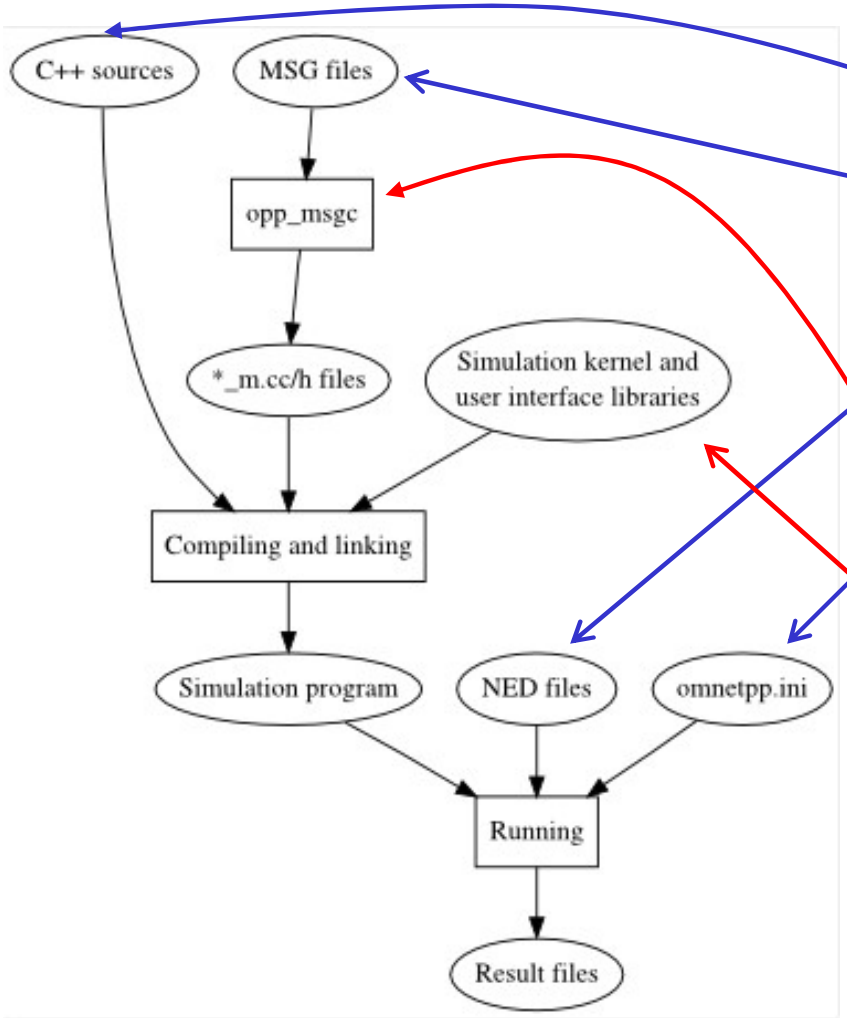
Anforderungen an Netzwerksimulatoren:

- Spezifikation von Protokollautomaten
- Hierarchische Spezifikation der Protokollebenen
- Spezifikation von Übertragungskanälen
- Basisprotokolle sollten in Bibliotheken vorhanden sein

OMNeT++ als Beispiel für einen Netzwerksimulator

- Objective Modular Network Testbed in C++ $\Rightarrow$ OMNeT++
- Frei verfügbar für Forschung und Lehre
- Entwicklung basiert auf der Dissertation von A. Varga in Budapest
  - Wird heute als open source-Projekt von einer wachsenden Gruppe von Anwendern weiter getrieben
  - Kommerzielle Ausgründung aus der Universität Budapest existiert
- Werkzeug mit
  - objektorientierter Struktur
  - graphischer Eingabeunterstützung
  - reichhaltiger Protokollbibliothek
  - …

Aufbau und Ablauf von OMNeT++-Simulationen

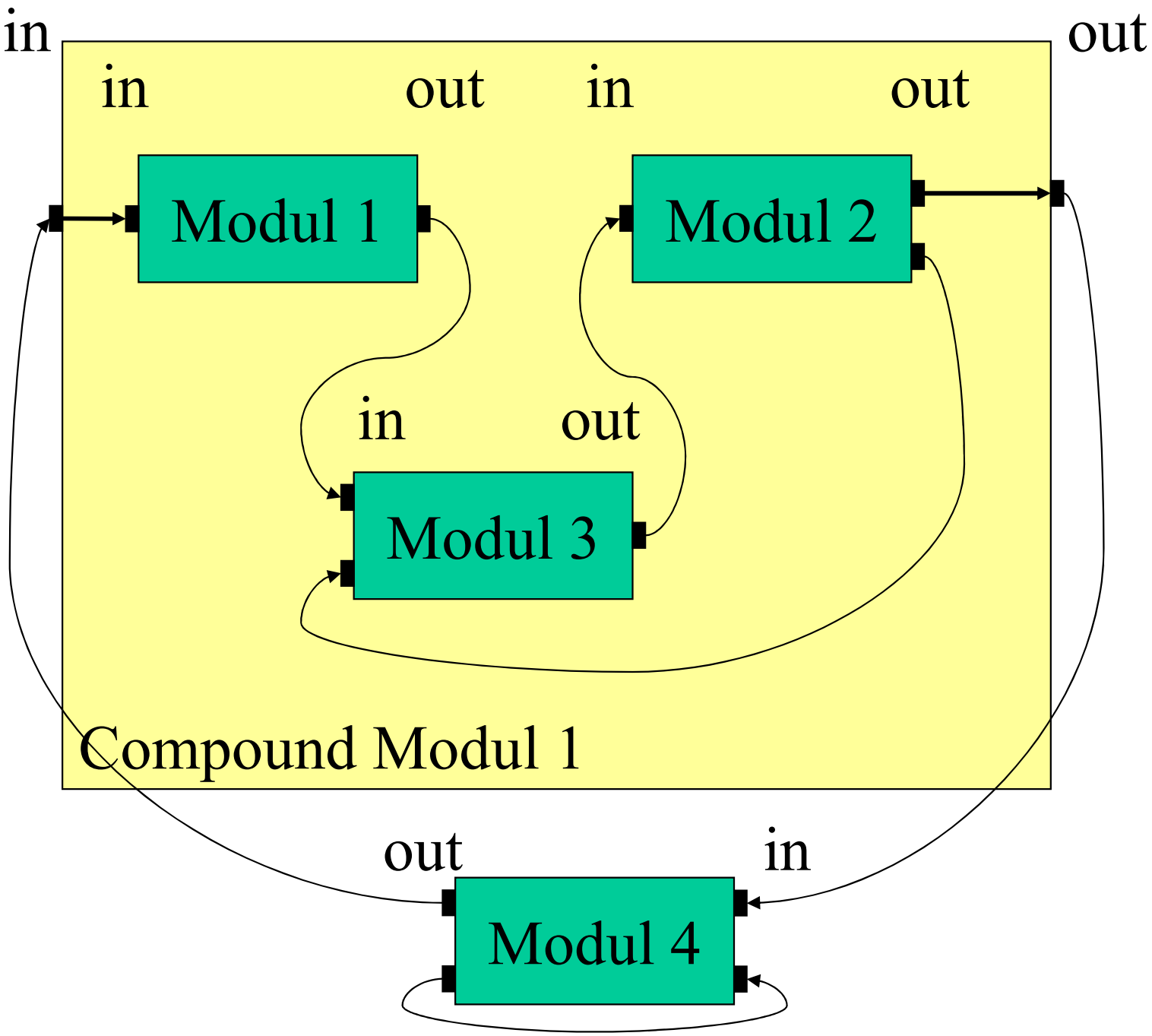


© OMNeT++-Manual

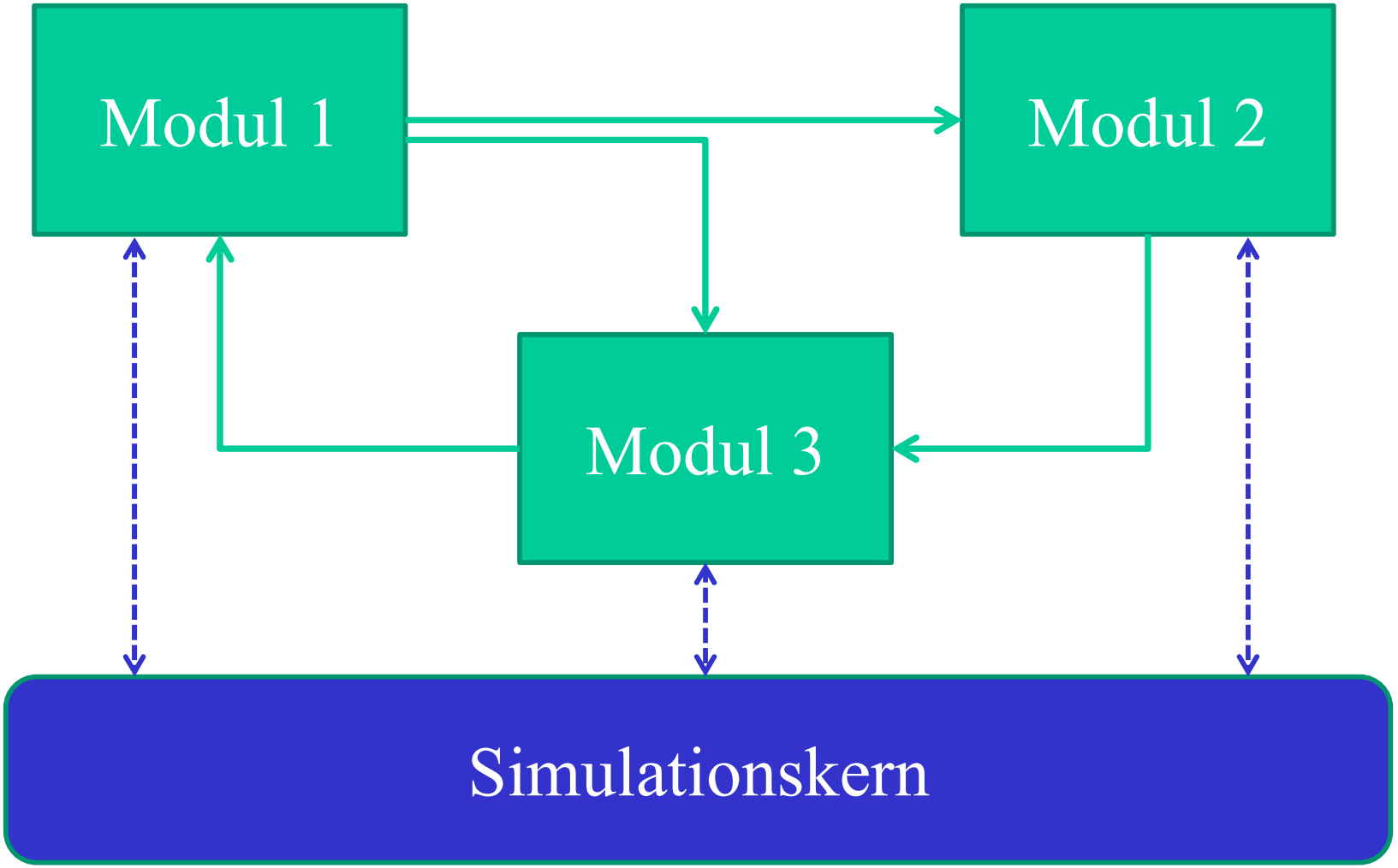
- Verhaltensbeschreibung der Module
- Beschreibung der Nachrichtenformate
- Beschreibung der Kommunikationswege
- Initialisierung der Parameter (Experiment, Simulation, Ausgabe…)
- Basisklasse für Nachrichten
- Simulationskern

Modellaufbau:

- Module implementieren die Funktionalität in C++ mit Hilfe von Basisfunktionen
  - Alle Module werden von der Klasse cSimpleModule abgeleitet
- Module werden über unidirektionale Verbindungen verbunden und kommunizieren durch Nachrichtenaustausch
- Modelltopologie wird durch eine spezielle Spezifikationssprache beschrieben
  (GUI zur Spezifikation existiert)
- Module können zu so genannten compound modules zusammengefasst werden (hierarchische Modellierung)


in
out
in
out
Modul 1
Modul 2
in
out
Modul 3
Compound Modul 1
out
in
Modul 4

Realisierung der Simulationsfunktionalität durch kommunizierende Module



Ereignisstrukturen in OMNeT++:

- Jede Nachrichtenankunft ruft `handleMessage()` auf
- `handleMessage()` bearbeitet die Nachricht und generiert dabei u.U. neue Nachrichten, die durch Ausgangsports verschickt werden per `send()` oder verzögert per `scheduleAt()` (Partner am anderen Ende ist auf dieser Ebene nicht bekannt!)
- Module können damit auch Nachrichten an sich selbst senden
- Nachrichtenbearbeitung hängt vom internen Zustand ab (über lokale Variablen)
- Simple Module sind C++-Klassen
- Compound Module werden wie simple Module behandelt, sind aber keine C++-Klassen

Aufbau und Verbindung von simple Modulen:

```
//
// Ethernet CSMA/CD MAC
//
simple EtherMAC {
    parameters: string address; // others omitted for brevity
    gates: input phyIn; // to physical layer or the network
           output phyOut; // to physical layer or the network
           input llcIn; // to EtherLLC or higher layer
           output llcOut; // to EtherLLC or higher layer
}
```

(OMNeT++ Wiki)

```
//
// Host with an Ethernet interface
//
module EtherStation {
        parameters: ...
        gates: ...
                input in; // for connecting to switch/hub, etc
                output out;
        submodules:
                app: EtherTrafficGen;
                llc: EtherLLC;
                mac: EtherMAC;
        connections:
                app.out --> llc.hlIn;
                app.in <-- llc.hlOut;
                llc.macIn <-- mac.llcOut;
                llc.macOout --> mac.llcIn;
                mac.phyIn <-- in;
                mac.phyOut --> out;
        }
```

Definition des Verhaltens durch C++ Funktionen

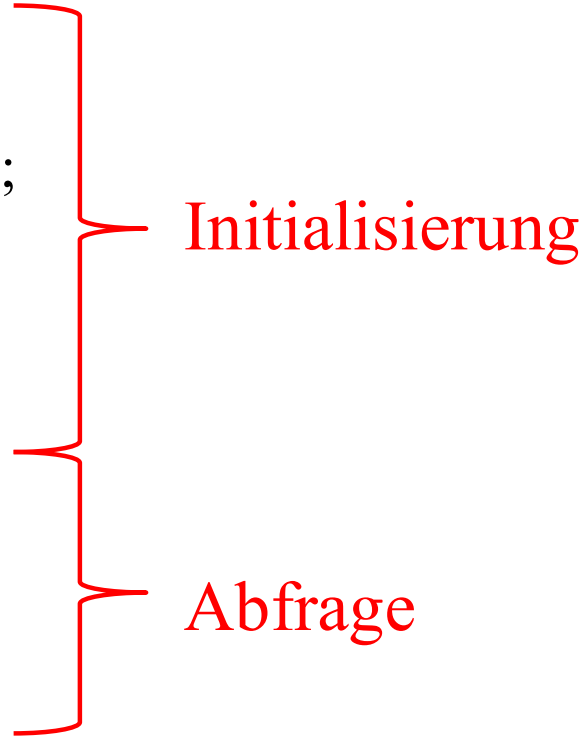
- Klasse wird abgeleitet von cSimpleModule
- Ersetzen der virtuellen Funktionen
  initialize(), handle_message() (oder activity)), finish()

```
// file: HelloModule.cc
#include <omnetpp.h>
class HelloModule :
public cSimpleModule {
        protected: virtual void initialize();
        virtual void handleMessage(cMessage *msg);
};
//  register module class with `\opp`
Define_Module(HelloModule);
void HelloModule::initialize() {
        ev << "Hello World!\n"; }
void HelloModule::handleMessage(cMessage *msg) {
        delete msg; // just discard everything we receive
} ….
```

```
// file: HelloModule.ned
simple HelloModule {
        gates: input in;
}
```

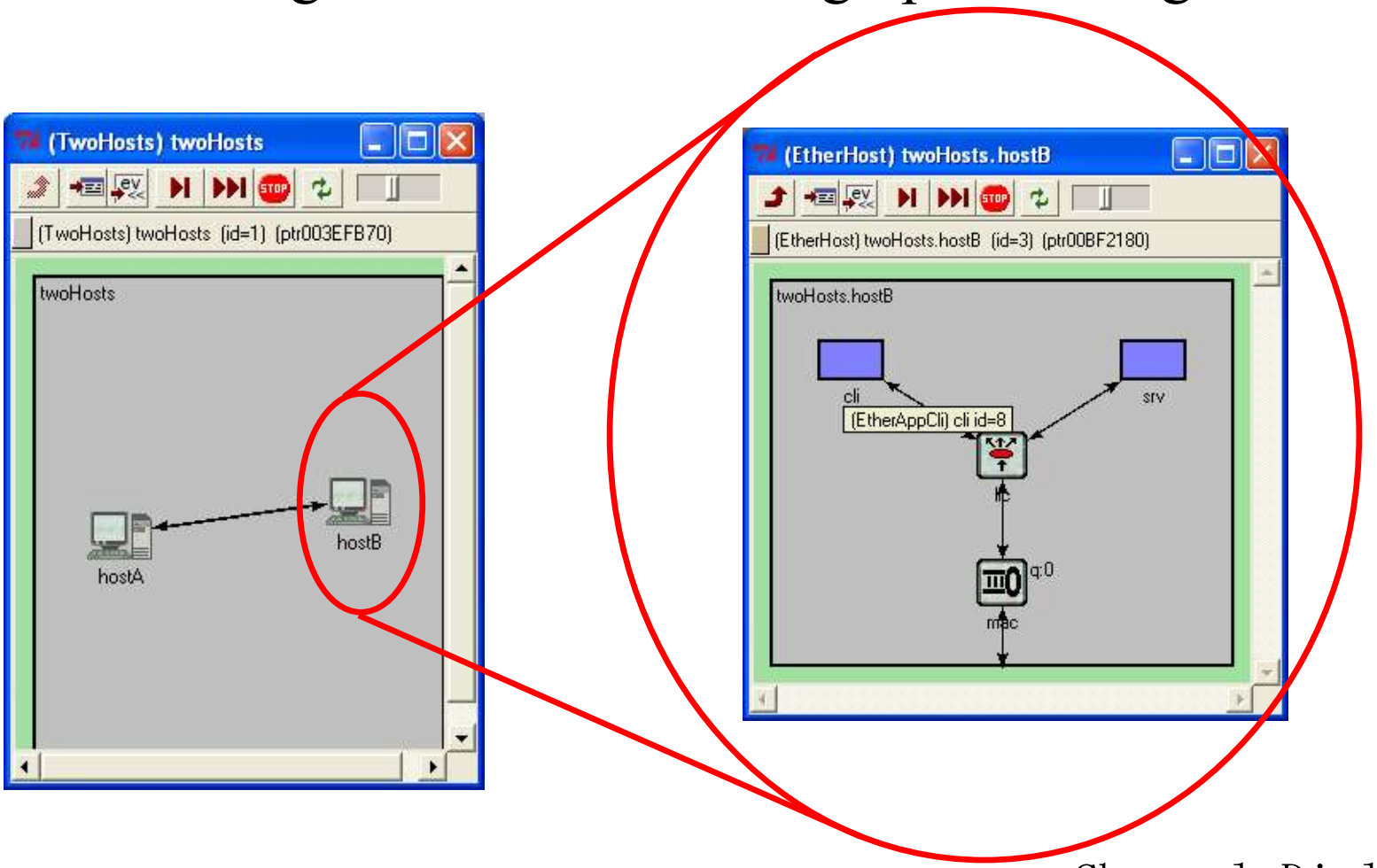
Kommunikation über Nachrichten

```
cMessage *msg = new cMessage("MessageName", msgKind);
msg->setKind( kind );
msg->setBitLength( length );
msg->setByteLength( lengthInBytes );
msg->setPriority( priority );
msg->setBitError( err );
msg->setTimestamp();
msg->setTimestamp( simtime );
int getSenderModuleId();
int getSenderGateId();
int getArrivalModuleId();
int getArrivalGateId();
```



+ viele weitere Möglichkeiten (z.B. zusammengesetzte Nachrichten)

Verbindung von Modulen auch graphisch möglich:



Screenshots aus dem OMNeT++-Inet Framework

Verbindung über Kanäle mit Verzögerung, Datenrate, Verlustrate

```
Channel DialUpConnection
  delay normal (0.004, 0.0018)
  error 0.00001
  datarate 14400
endchannel
```

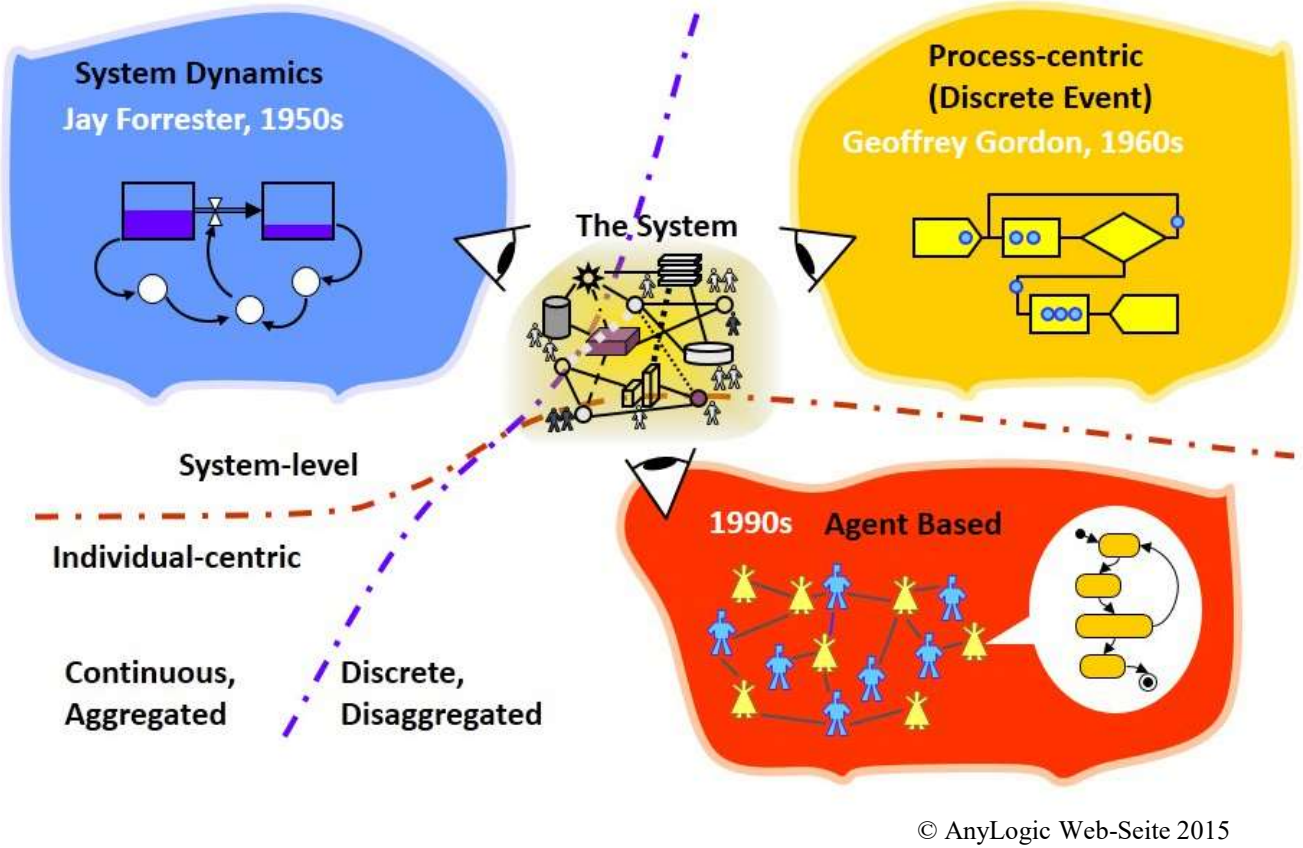
Weitere Eigenschaften

- 32 verschiedene Zufallszahlengeneratoren (erweiterbar)
- Seedtool um Zufallszahlenströme zu definieren
- Zufallszahlen aus vielen Verteilungen sind direkt generierbar
- Unterstützung zur graphischen Eingabe und Erstellung von Animationen
- Unterstützung zur graphischen Repräsentation von Resultaten
- Unterstützung beim Debugging
- Bibliothek mit Implementierungen vieler Standardprotokolle inkl. Protokolle für mobile Systeme
- Standardmäßig keine statistische Auswertung, aber Statistikklasse existiert inzwischen
- …

AnyLogic

- Entwicklung seit den 90er Jahren durch die Firma xj-technology später AnyLogic Comp.
  (Ausgründung der Universität St. Petersburg)
- Basis der Entwicklung: Analyse paralleler Prozesse in der Informatik
  (entwickelte sich zum allg. Modellierungssystem)
- Graphische Modellierungssprache + Erweiterungen in Java
  (System ist Java-basiert und damit plattformunabhängig)
- Mehrformalismen Modellierung
  (Mischung von Formalismen in einem Modell, hybride Modellierung)
- Animation (2D- und 3D)
- Viele Basisbibliotheken (Fußgängerströme, Supply-Chains, Fertigungssysteme)

Unterstützte Sichten auf ein System

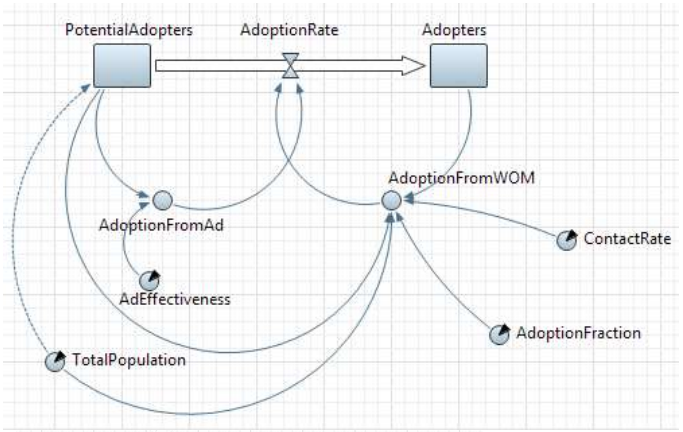


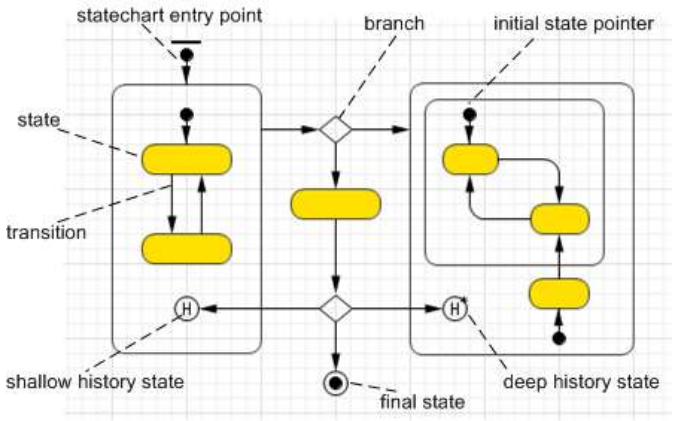
© AnyLogic Web-Seite 2015

- Prozessorientierte und agentenorientierte Sicht sind ereignisdiskret
- *System dynamics* ist eine kontinuierliche Sicht
- Kombination der Modelltypen in einem Modell ist möglich (hybride Modelle, unterschiedliche Abstraktionsebenen etc.)

Beschreibungsmöglichkeiten für dynamische Abläufe

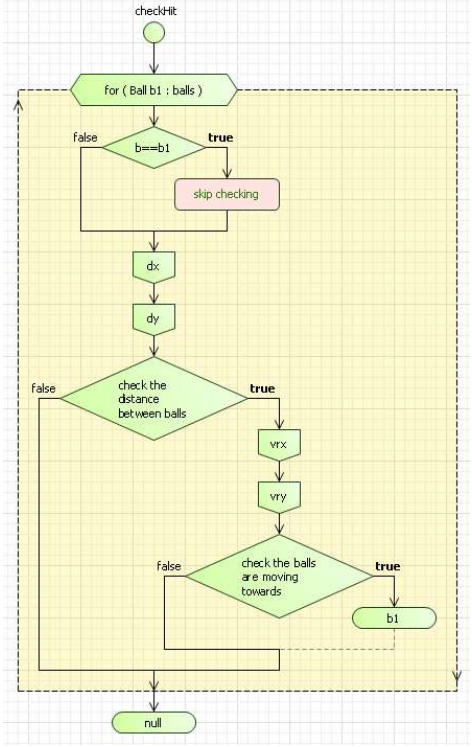
Lager- & Fluss-diagramme



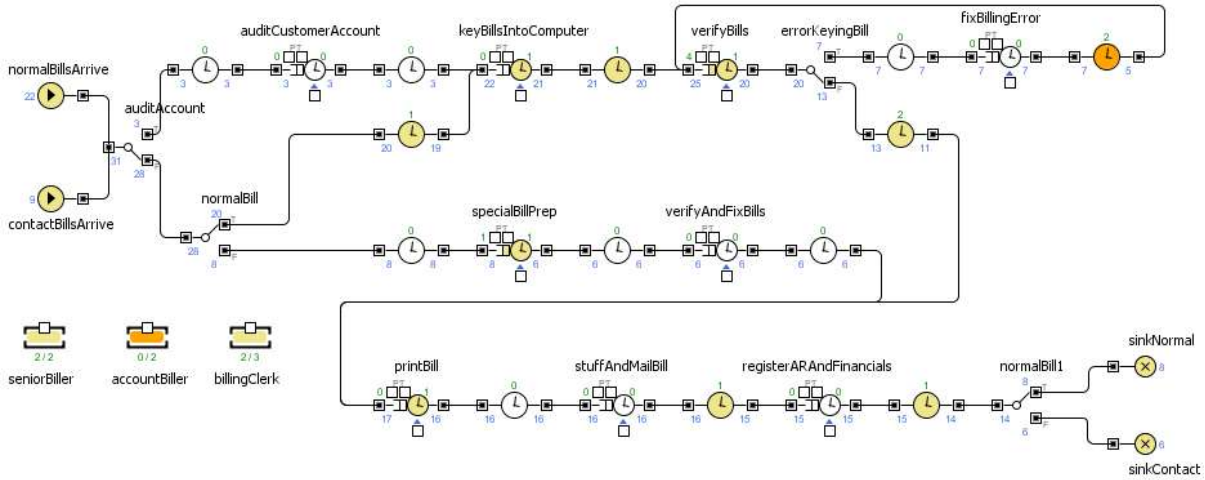


Zustands-diagramme

Funktions-diagramme

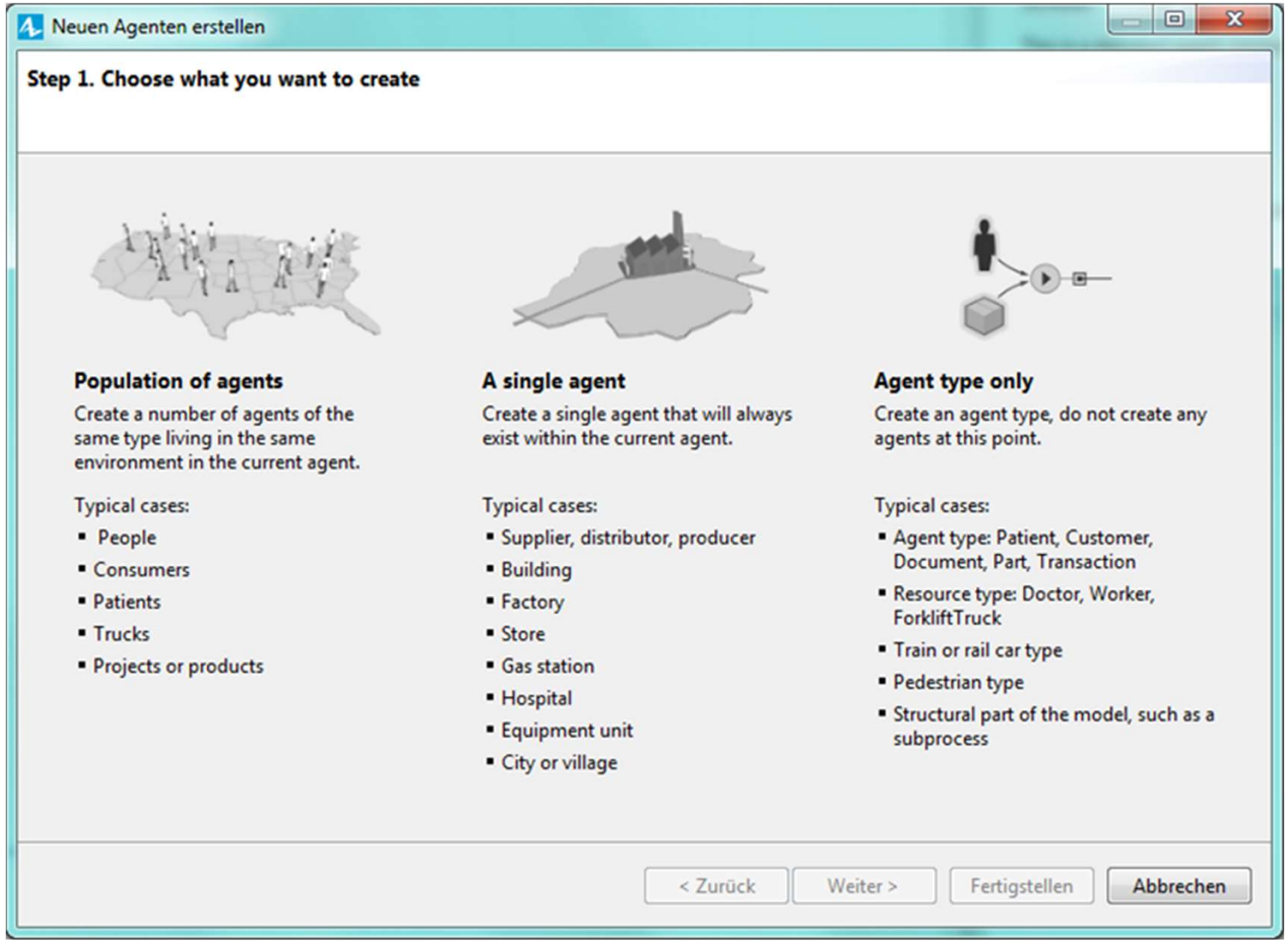


Prozessablaufdiagramme

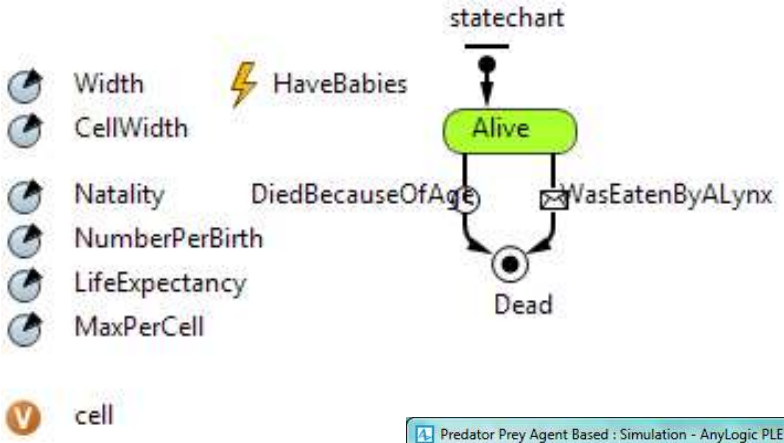


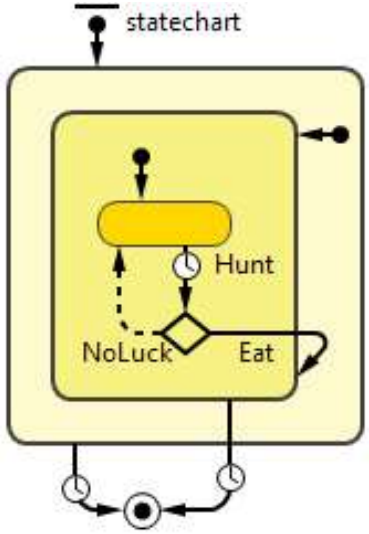
Agentenbasierte Modellierung:

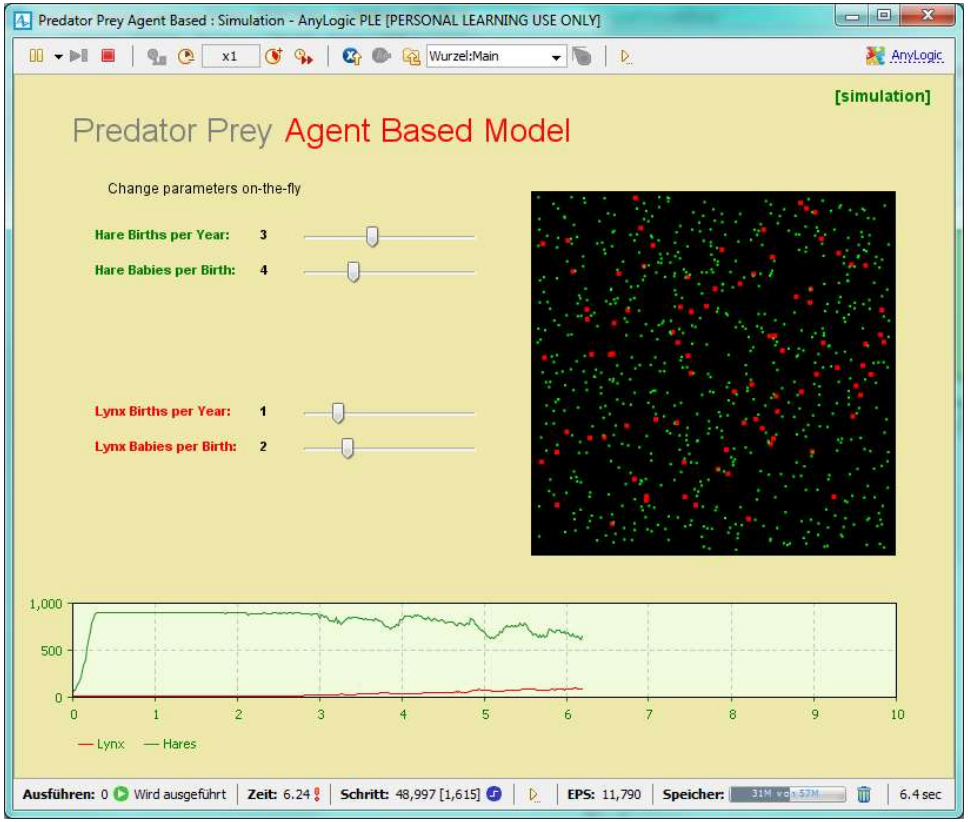
- Pragmatische Sicht auf Agenten
  (ohne spezifische Anforderungen)
- Spezifikation durch eines der Diagramme (oft statecharts) oder regel-basiert
- Agentenverhalten ereignisdiskret (oder auch kontinuierlich)
- Modell besteht aus mehreren (oft sehr vielen) Agenten
- Interaktion der Agenten direkt oder mit der Umgebung
  (Umgebung ist auch ein Agent)
- Agenten bewegen sich im Raum
- Agentenverhalten kann um Animation ergänzt werden
- Simulation der Agentenpopulation
  (lokal sequentialisiert)
- Beobachtung einzelner Agenten oder ganzer Populationen


Neuen Agenten erstellen
Step 1. Choose what you want to create
Population of agents
Create a number of agents of the same type living in the same environment in the current agent.
Typical cases:
People
Consumers
Patients
Trucks
Projects or products
A single agent
Create a single agent that will always exist within the current agent.
Typical cases:
Supplier, distributor, producer
Building
Factory
Store
Gas station
Hospital
Equipment unit
City or village
Agent type only
Create an agent type, do not create any agents at this point.
Typical cases:
Agent type: Patient, Customer, Document, Part, Transaction
Resource type: Doctor, Worker, ForkliftTruck
Train or rail car type
Pedestrian type
Structural part of the model, such as a subprocess
< Zurück
Weiter >
Fertigstellen
Abbrechen

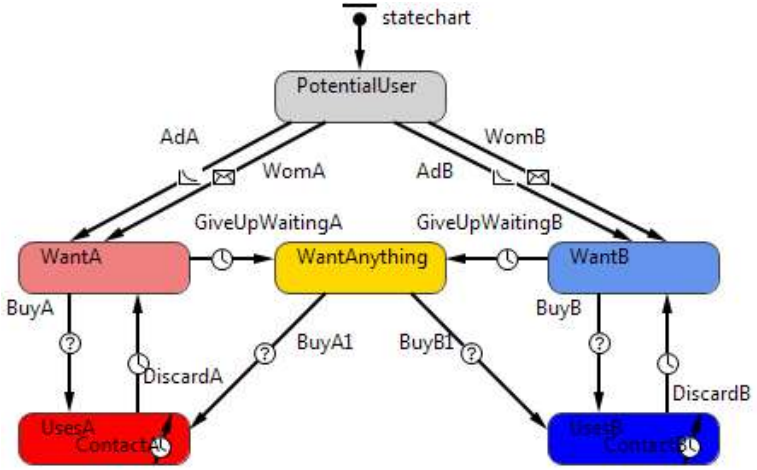
Diskrete Modellierung

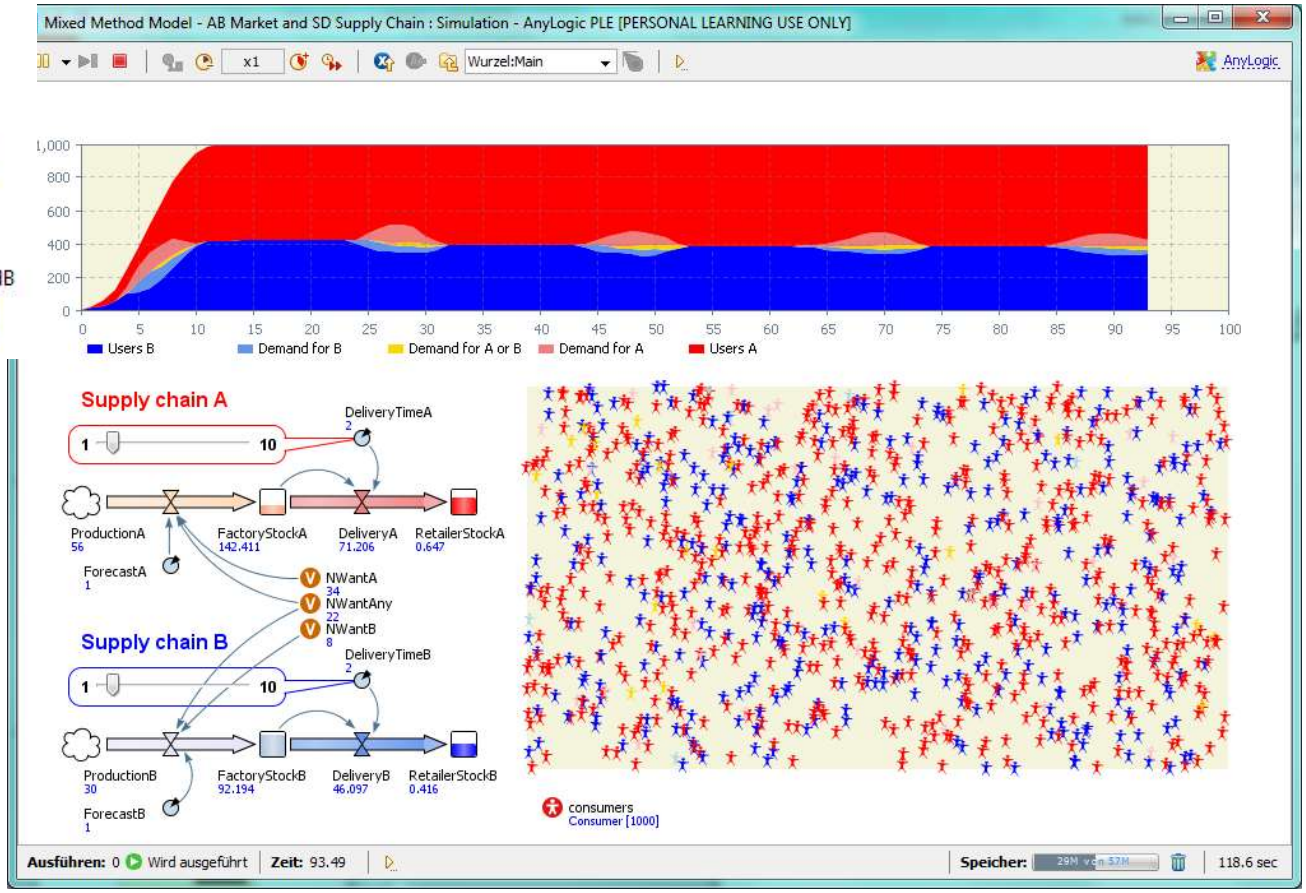






Gemischt kontinuierlich-diskrete Modellierung

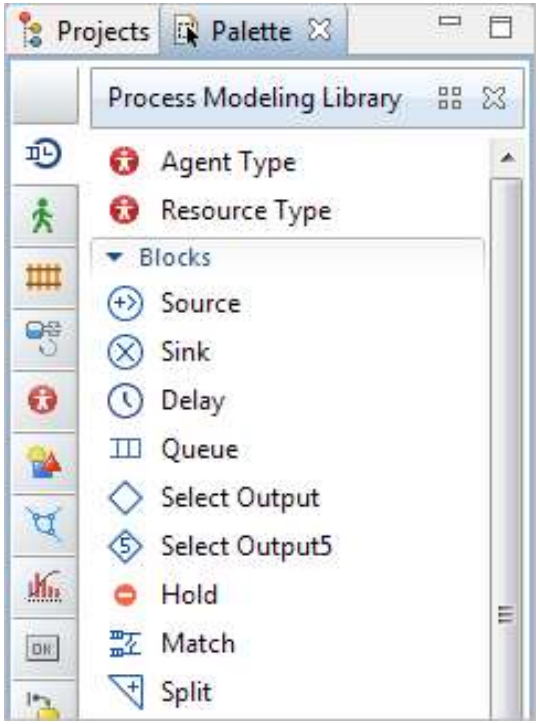




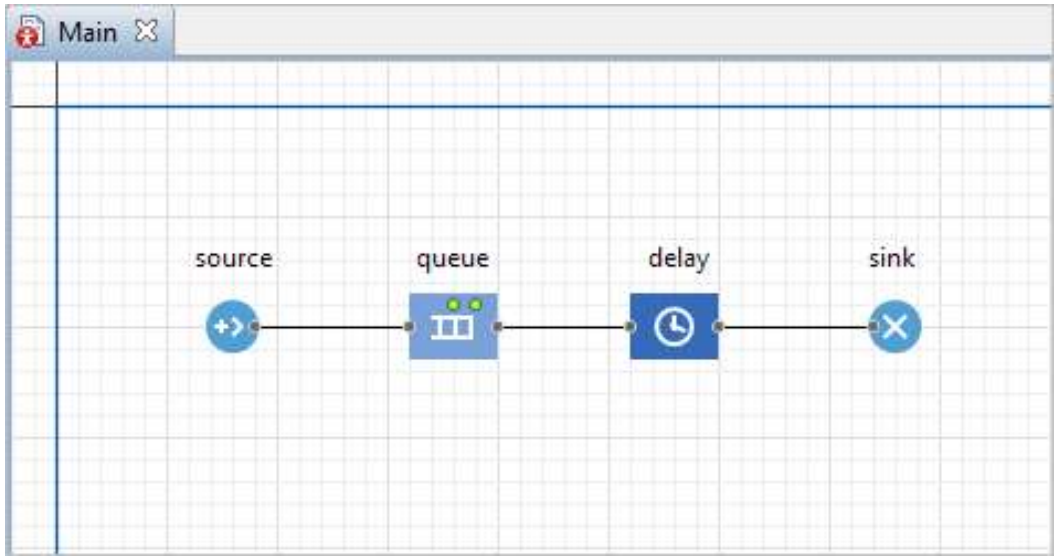
Ereignisdiskrete Modellierung mit AnyLogic

- Ablaufbeschreibung durch Flussdiagramme
  (Idee an GPSS-Flußdiagrammen orientiert)
- Im Gegensatz zur agentenorientierten Modellierung Einbindung von Ressourcen
- Modell ist Netz von Ressourcen
- Ergebnisse ressourcenorientiert
  (Auslastung, Warteschlangenlänge, ….)
  oder prozessorientiert
  (Durchlaufzeit, Wartezeit, …)
- Kombination mit Agenten, system dynamics Teilmodellen ist möglich

Beispiel *queue*



Parameters
Agent type
TCapacity: int capacity
Agent location // zurAnimation
Queueing discipline:
FIFO (default), *LIFO*, *Priority-based, Agent comparison*
enableTimeout: boolean, Timeout Value type: double
enablePreemption: boolean
forceStatisticsCollection: boolean



Ports
in
out
outTimeout
outPreempted

Weitere Eigenschaften:

- Modelle als eigenständige Java-Anwendungen
- Verschiedene Bibliotheken mit anwendungsspezifischen Standardkomponenten
  - Fußgängerbibliothek
  - Bahnbibliothek
  - Nutzung von GIS (Geographical Information System)
  - Definition eigener Bibliotheken
- Unterstützung bei der Experimentsteuerung
- Unterstützung bei der Optimierung
  - Nutzung von OptQuest
  - Export „optimierbarer“ Modelle

Modellierung von Fußgängerströmen

Animation

Modellsicht