UNIVERSITÄT SALZBURG

VO Einführung Simulation

SS 2020

Helge Hagenauer
FB Computerwissenschaften

Teil 1

1 Motivation - Was ist Simulation?

Einige Begriffsdefinitionen zu **Simulation**:

- Vortäuschung (von Krankheiten); Sachverhalte, Vorgänge (mit technischen, naturwissenschaftlichen Mitteln) modellhaft zu Übungs-, Erkenntniszwecken nachbilden, wirktlichkeitsgetreu nachahmen.

  (Duden)

- Technische oder naturwissenschaftliche Prozesse, Funktionen in ihren wesentlichen Grundzügen nachbilden.

  (Das Digitale Wörterbuch der deutschen Sprache; www.dwds.de)

- Simulation in der Informatik: Nachbildung von Vorgängen auf einer Rechenanlage auf der Basis von Modellen (= im Computer darstellbare Abbilder der realen Welt).

  (nach Informatik-Duden)

- Die Simulation oder Simulierung ist eine Vorgehensweise zur Analyse von Systemen, die für die theoretische oder formelmäßige Behandlung zu komplex sind. Dies ist überwiegend bei dynamischem Systemverhalten gegeben. Bei der Simulation werden Experimente an einem Modell durchgeführt, um Erkenntnisse über das reale System zu gewinnen. Im Zusammenhang mit Simulation spricht man von dem zu simulierenden System und von einem Simulator als Implementierung oder Realisierung eines Simulationsmodells. Letzteres stellt eine Abstraktion des zu simulierenden Systems dar (Struktur, Funktion, Verhalten). ...

  (de.wikipedia.org)

Beweggründe für Simulation

Allgemein zusammengefasst (nach B.Page, siehe Literatur [2]):

> Studying a system from multiple points of view and predicting how, as well as understanding why it will react to different inputs are main motivations for users of modelling tools.

⇨ Erlangung eines besseren Verständnisses für Systemverhalten und Wirkungsbeziehungen

Since simulation can explore models with any degree of complexity, it can further these goals where other techniques may fail. Whenever a problem's solution requires a model too complex for mathematical optimization, simulation should be the tool of choice.

⇨ Systeme beliebiger Komplexität behandelbar

Meist dient Simulation der Untersuchung von Abläufen, die in der Realität nicht durchführbar sind wegen:

- Zeitverbrauch (z.B. Klimamodelle, Wachstum von Organismen, ...)
- Kosten (z.B. Crashtest, Schulung an Simulatoren, ...)
- Gefahren (z.B. Airbag, Kernreaktoren, Schulung an Simulatoren, ...)
- ...

Anwendungsbereiche

Simulation kann praktisch in allen Bereichen angewendet werden, z.B.:

- Natur- und Ingenieurwissenschaften
- Wirtschafts- und Gesellschaftswissenschaften
- Informatik
- Psychologie und Medizin
- Ökologie, Umweltverhalten
- Geisteswissenschaften

Einige Anwendungsbeispiele

- Management:
  Betriebsabläufe, Personalplanung, Lagerverwaltung, Geschäftsprozesse

- Produktionssysteme:
  Planung/Kontrolle (automatischer) Fertigungssysteme

- Informatik, IT:
  Rechnerkonfiguration, Betriebssysteme, Netzwerkanalyse/-planung, Datenbankdesign, Sicherheitaspekte, Systemausfälle/Notfallstrategien

- Transportwesen, Verkehr, Logistik:
  Flughafen-Design, Planung von Verlade-/Containerterminalen, Verkehrssysteme planen, Straßennetz (Leitsysteme, Kapazitäten, Steuerung von Ampeln), Auslieferungssystem

2 Literaturhinweise

[1] Banks, J., J.S.Carson, B.L.Nelson, D.M.Nicol.
Discrete-Event System Simulation.
2010. 5th Edition. Pearson .

[2] Page, B., W.Kreutzer.
The Java Simulation Handbook – Simulating Discrete Event Systems with UML and Java.
2005. Shaker Verlag.

[3] Law, A.M.
Simulation Modeling & Analysis.
2015. McGraw-Hill Inc.

[4] Leemis, L.M., S.K.Park.
Discrete-Event Simulation: A First Course.
2006. Pearson Prentice Hall.

[5] Zeigler, B.P., H.Praehofer, T.G.Kim.
Theory of Modeling and Simulation.
2018. Academic Press.

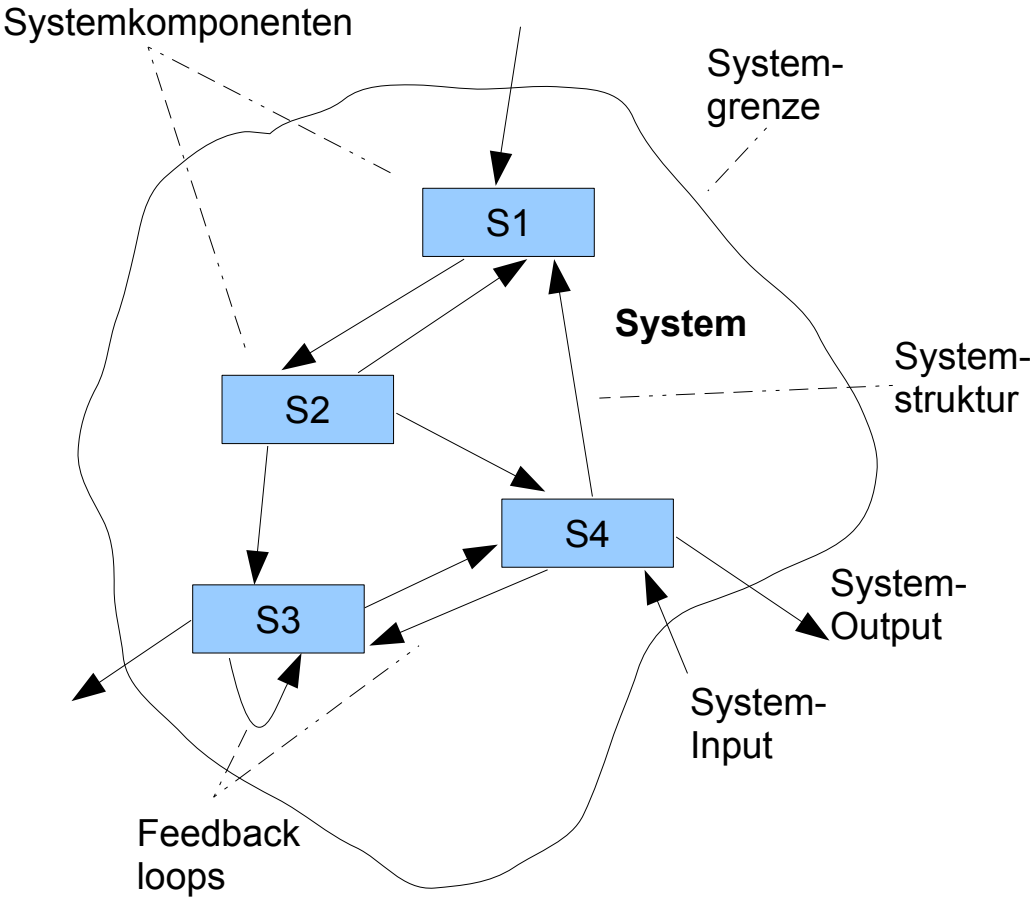
[6] ev. weitere Hinweise während der VO

3 Wichtige Begriffe

System

- Ausschnitt der Realität, welcher unter einer besonderen Fragestellung betrachtet wird

- kann materiell (real) oder immateriell (nicht stofflich, geistig, Ideengebilde) sein
- besitzt eine Schnittstelle zur Umwelt (von Fragestellung bestimmt)
- besteht aus identifizierbaren Komponenten, die zu einem bestimmten Zweck interagieren (innerhalb des Systems)



Modell

- materielle oder immaterielle Systeme zur Darstellung anderer Systeme
- experimentelle Manipulation der abgebildeten Strukturen und Zustände soll möglich sein

Ein Modell soll eine Bestimmung haben bzw. für die Bearbeitung bestimmter

Fragestellungen ausgerichtet werden.

Übergang Realsystem → Modell: *Abstraktion, Idealisierung* nötig, d.h. vereinfachte Darstellung des Originals unter Beachtung der Fragestellung.

Weiters ist bei Modellen zu bedenken:

- zu modellierende Eigenschaften sind abhängig von der Fragestellung (d.h. unterschiedliche Modelle zu einem Realsystem möglich)

- ausführliche Tests (statistisch, qualitativ) zur *Validierung* eines Modells notwendig
  → ausreichend Daten über Realsystem sollten vorliegen – Vergleichsmöglichkeit

- wenn Abweichung der gewonnenen Daten vom Original → Hinweis auf fehlerhafte Modellbildung:

  - große Differenz: mögliche Fehler im Modell – falsche Abbildung der Realität

  - kleine Differenz: Kalibrierung vornehmen (Änderung weniger Modellparameter)

Verschiedene Gesichtspunkte bei Modellklassifizierungen möglich, wie z.B. Zeitbezug, Verwendungszweck oder nach Art der Repräsentation:

- physikalisches Modell (z.B. maßstabgetreue Nachbildung eines Flugzeugs oder Schiffs)

- verbales Modell (z.B. Wegbeschreibung zu einer bestimmten Adresse)

- beschreibendes graphisches Modell (z.B. UML-Aktivitätsdiagramm)

- mathematisch-graphisches Modell (z.B. Petri-Netz)

- abstraktes mathematisches Modell (z.B. Menge von Differentialgleichungen)

- abstraktes algorithmisches Modell (z.B. diskrete Ereignissimulation)

Simulation

Jetzt zu Definitionen im Sinn dieser LV:

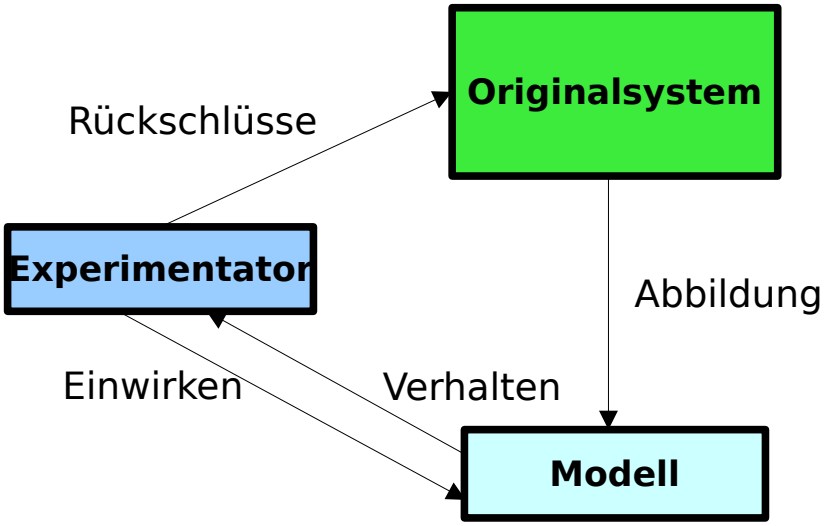
- Simulation is the process of describing a real system and using this model for experimentation, with the goal of understanding the system's behaviour or to explore alternative strategies for its operation.

  (Shannon 1975)

- Simulation is the modelling of dynamic processes in real systems, based on real data, and seeking predictions for a real system's behaviour by tracing a system's changes of state over time.

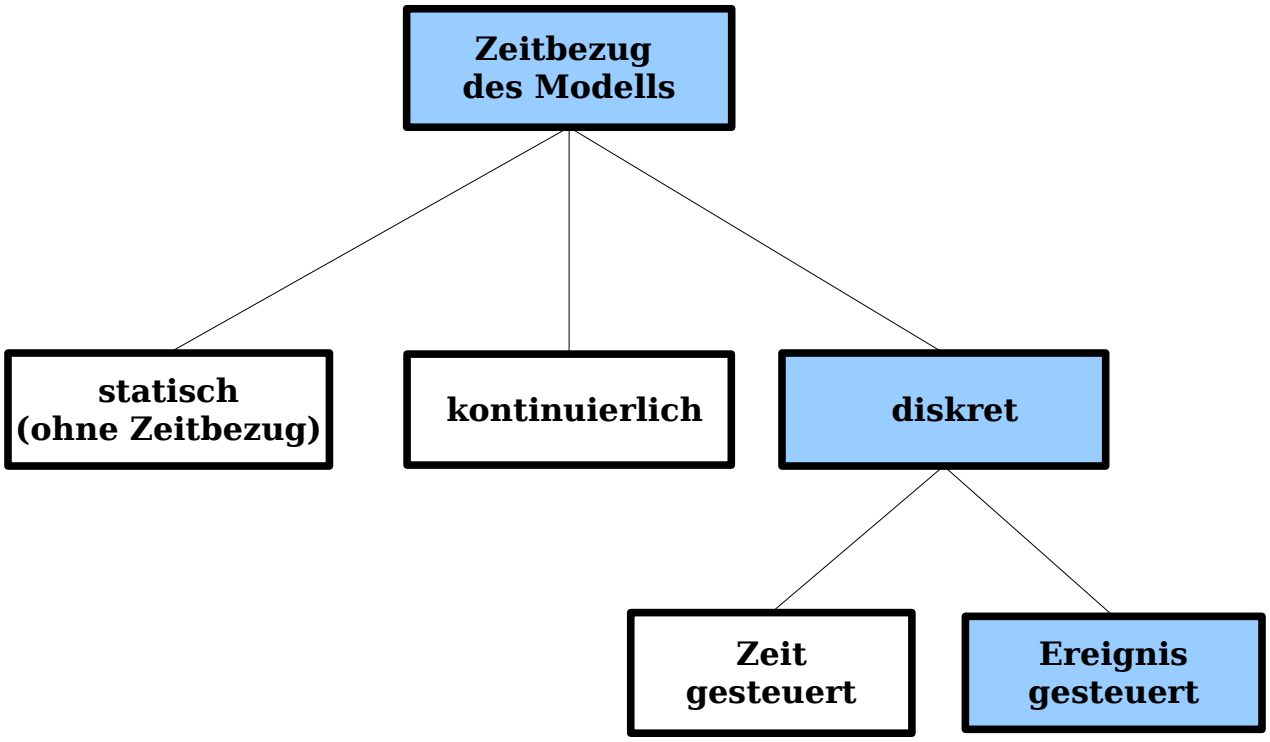
  (*The Java Simulation Handbook.* B.Page, W.Kreutzer [2])

Daraus ergeben sich folgende Beziehungen



⇨ beschäftigen uns mit: **Simulation als Teilgebiet der Informatik**

4 Modellklassifikation nach Zeitbezug



- **statische Modelle**: kein Bezug zur Zeit (z.B. Monte Carlo Simulation)

- **kontinuierliche Simulation**: stetige Zustandsänderung über der Zeit; Modell als System von Differentialgleichungen (mit freier Zeitvariablen);

  Simulation heißt hier: lösen der Modellgleichungen (meist numerisch) z.B. für komplexe biologische oder physikalische Prozesse (Wettervorhersage, Strömungslehre, Analyse von Ökosystemen, ...)

- **diskrete Ereignis Simulation (discrete event simulation)**: Änderung des Systemzustands nur zu bestimmten (Ereignis-) Zeitpunkten (keine Änderungen dazwischen!)

  Unterteilung in

  - **Ereignis gesteuert (event-driven)**: Zeitabstände zwischen aufeinanderfolgenden Ereignissen sind unterschiedlich

  - **Zeit gesteuert (time-driven)**: gleiche Zeitabstände zwischen

aufeinanderfolgenden Ereignissen

5 Komponenten und Grundkonzept diskreter Simulation

Grundlegende Komponenten von Simulationsmodellen sind:

Entitäten: Komponenten des Simulationsmodells, welche Teile des realen Systems modellieren und miteinander in Wechselbeziehung stehen.

Entitäten sind gekennzeichnet durch: Zustand und Transformationsregeln.

Entität ↔ Objekt (nach OOP): Entität entspricht einem Objekt, dessen Verhalten durch die Simulationszeit bestimmt wird.

Zustand (state): aktuelle Belegung der Attribute einer Entität.

Transformationsregeln (rules): Menge von Regeln (oder Methoden) welche den Zustand einer Entität im Laufe der Simulationszeit verändern (→ Zeitbezug nötig!).

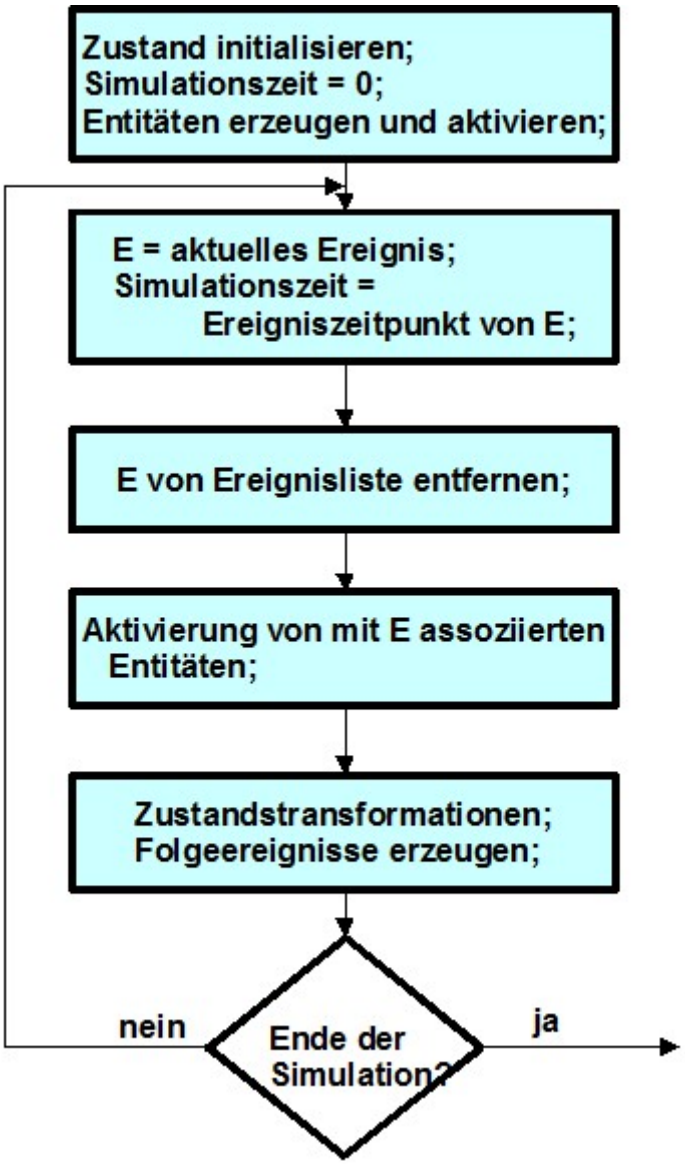
Simulationszeit: fiktive Modellzeit, unabhängig von der realen Zeit und der Ausführungsdauer einer Simulation

Diskrete Ereignis-Simulation bedeutet:

- Simulationszeit springt von Ereignis zu Ereignis.
  Ein Ereignis löst maßgebliche (relevante) Zustandsänderungen bei einer oder mehreren Entitäten aus.
- *Keine* Zustandsänderungen zwischen zwei aufeinanderfolgenden Ereignissen.
- Zustandsänderungen erzeugen i.A. Folgeereignisse.
- Jedes Ereignis

- besitzt einen *Ereignistyp*
- ist assoziiert mit einer Entität (oder mehreren)
- besitzt einen **Eintritts- oder Ereigniszeitpunkt** (= Zeitpunkt der Simulationszeit)

Daraus ergibt sich als Grundkonzept folgende prinzipielle Funktionsweise:



Dabei ist zu beachten:

- aktuelles Ereignis = Ereignis mit kleinstem Ereigniszeitpunkt

- **Ereignisliste**: verwaltet alle noch nicht verarbeiteten (zukünftigen) Ereignisse

- Ereignisse mit gleichem Eintrittszeitpunkt: Verarbeitung in beliebiger Reihenfolge (z.B. FCFS-Strategie, Prioritäten)

- Ende der Simulation: mehrere Möglichkeiten wie z.B. bestimmte Zeit verstrichen, spezielles Ende-Ereignis tritt ein, ... oder Ereignisliste ist leer

Beziehung Modellzustand – Simulationszeit

Es existieren folgende grundlegende Möglichkeiten um Zustandsänderungen und Simulationszeit zu synchronisieren:

Ereignis: beschreibt/bewirkt die Zustandsänderung von Entitäten zu einem bestimmten Zeitpunkt der Simulationszeit.

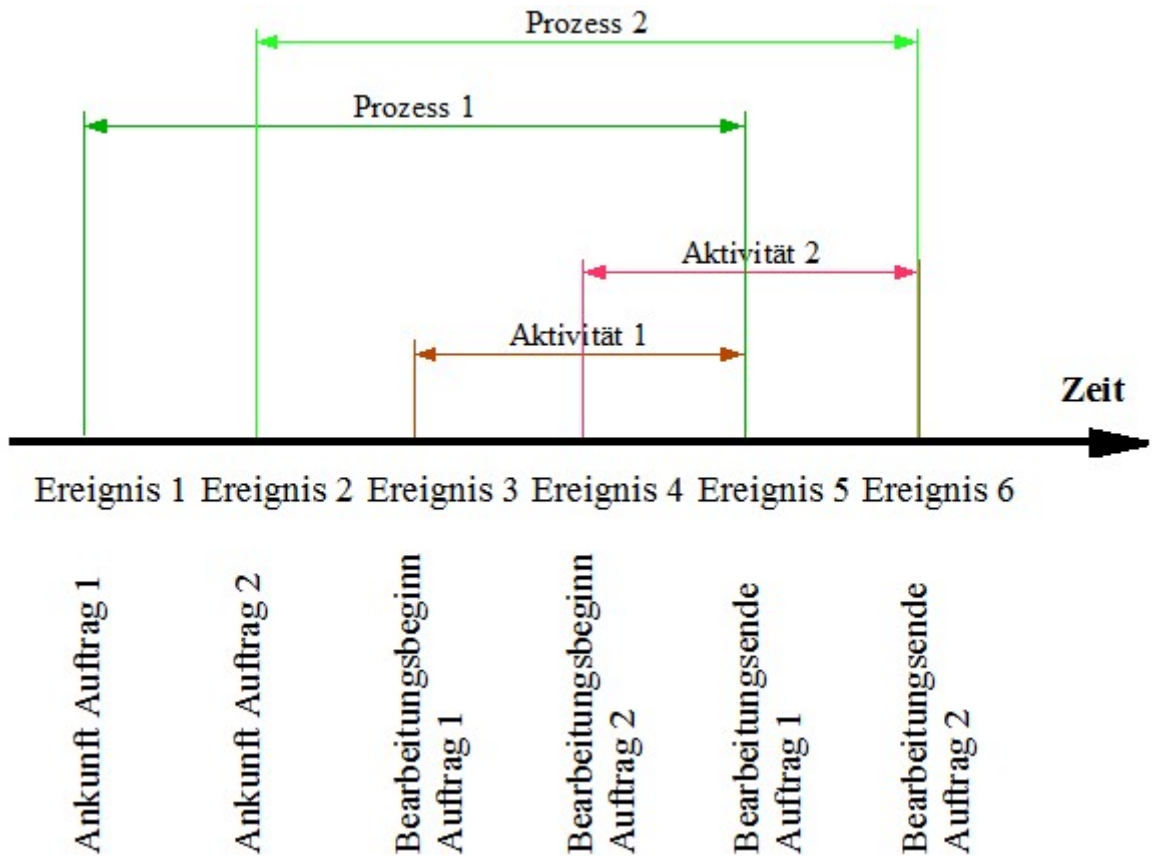
Externe Ereignisse: keine Abhängigkeit von anderen Ereignissen, werden von der Systemumgebung vorgegeben (z.B. eintreffen eines Auftrags).

Interne Ereignisse: entstehen durch Zustandsänderungen (z.B. Bearbetungsende eines Auftrags).

Aktivität: besteht aus einer Menge von Operationen, welche während eines Intervalls der Simulationszeit ausgeführt werden; Zustandsänderung (Wirkung) wird zu Beginn und am Ende der Aktivität ausgeführt (z.B. Bearbeitung eines Auftrags)

Prozess: besteht aus einer Folgen von Aktivitäten einer Klasse von Entitäten, die den Lebenszyklus dieser beschreiben
(z.B. gesamte Abwicklung vom Eintreffen bis zum Bearbeitungsende eines Auftrags)

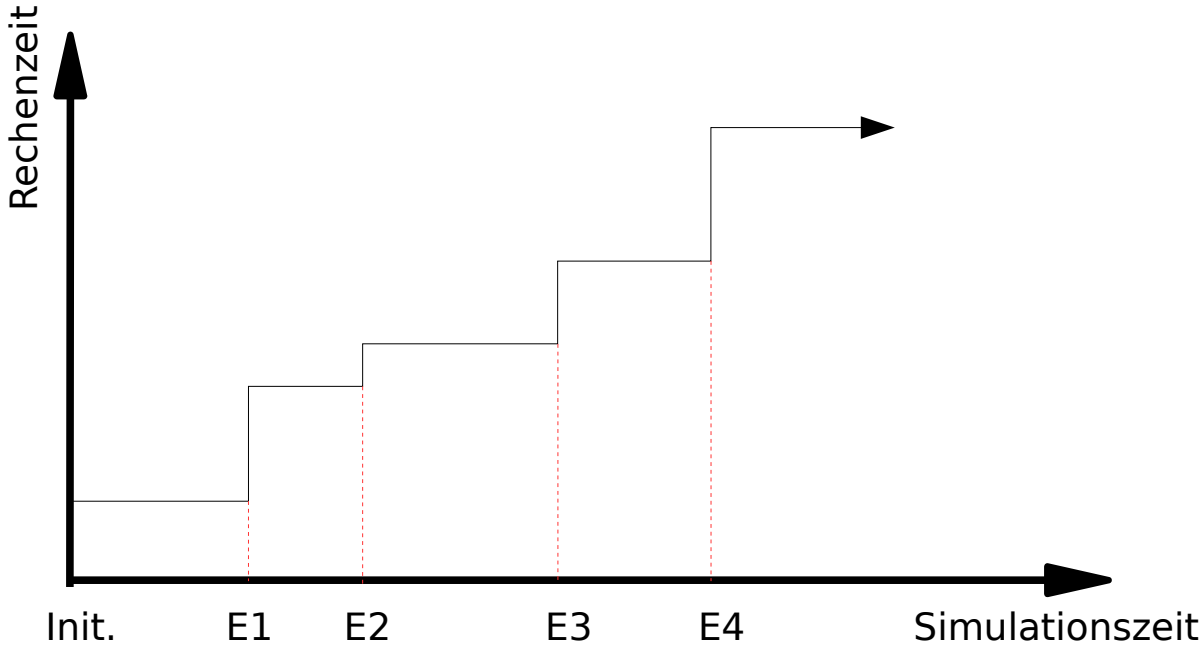
Zusammenhang dargestellt am Bsp. einer Auftragsbearbeitung (nach [2]):



Auch von Interesse ist die *orthogonale* Beziehung zwischen Simulationszeit und Rechenzeit.

- **Simulationszeit** schreitet sprunghaft voran (Abarbeitung von Ereignissen); ist *monoton steigend* (d.h. nicht fallend).
- **Rechenzeit** wird zur Ausführung am Rechner benötigt, wobei gilt:
  - Ereignisse verbrauchen Rechenzeit – aber keine Simulationszeit

- Intervalle zwischen Ereignissen verbrauchen Simulationszeit – aber keine Rechenzeit



Umsetzung in Software

Für die Implementierung wird benötigt:

- Darstellung des Systemzustands: Menge von Variablen die den Modellzustand zum Simulationszeitpunkt t beschreiben
- Simulationszeit: Variable, welche die aktuelle Simulationszeit darstellt
- Ereignisliste: Datenstruktur zur Verwaltung von Ereignissen, geordnet nach der Zeit;
  notwendige Operationen darauf: einfügen geordnet nach Ereigniszeitpunkten, Zugriff auf aktuelles Ereignis, Ereignis löschen, Ereignis verschieben
- Zähler für statistische Auswertung: Variablen zur Speicherung (Zählung) relevanter Daten

Beispiel: Schalter mit Warteschlange

Ein Schalter (Bank, Fahr- oder Eintrittskarten) wird als **Bedienstation mit Warteschlange (single server queueing system)** modelliert. Weitere Anwendungen davon sind z.B. Friseur, Arzt, Drucker oder Router in einem Netz, ...

Zu beachten ist:

- Kunden kommen zu zufälligen Zeitpunkten zum Schalter
- findet ein ankommender Kunde einen freien Schalter vor, wird er sofort bedient
- ist bei Ankunft des Kunden kein Schalter frei, reiht er sich am Ende der Warteschlange ein
- nach Ende des Bedienvorgangs wird jener Kunde aus der Warteschlange genommen und bedient, der am längsten wartet (falls vorhanden)
  → FIFO-Strategie
- Simulationsbeginn: Zeitpunkt 0 und kein Kunde im System (empty-and-idle)
- Simulationsende: wenn ein bestimmter Zeitpunkt in der Simulationszeit erreicht ist (z.B. Bankschalter schließt nach 4 Stunden)

6 Ereignisorientierter Modellierungsstil

Zur Erstellung eines Simulationsmodells können verschiedene **Modellierungsstile (world views)** verwendet werden. Die wichtigsten und meist verwendeten werden besprochen.

Zuerst der **ereignisorientierte Modellierungsstil**; auch *event scheduling* genannt; historisch gesehen älterer Stil.

Das Prinzip lautet:

- Betrachtung der Gesamtheit aller Zustandsänderungen aller relevanten Entitäten zu einem Ereigniszeitpunkt
- Aktivitäten dazwischen werden nicht direkt abgebildet – erst die entsprechenden Auswirkungen beim nächsten Ereigniszeitpunkt
- Während einer Ereignisabarbeitung vergeht *keine* Simulationszeit (es wird jedoch Rechenzeit benötigt!)

Zur Erstellung eines ereignisorientierten Modells ist zu beachten:

- Identifikation der relevanten Systemobjekte und ihrer Attribute
- Beschreibung *aller* Systemzustände, Ereignisse und Zustandsänderungen der Entitäten inklusive Interaktionen
  - *Vogelperspektive*
- Zusammenfassung aller Zustandsänderungen jener Entitäten, die durch Ereignisse zum selben Zeitpunkt stattfinden, zu *Ereignistypen*

Ereignisorientierter Modellierungsstil ermöglicht eine klare Trennung zwischen Systemstruktur (Entitäten) und Systemverhalten (Ereignisse).

Es ergeben sich 2 grundlegende Arten von Modellkomponenten:

- **Statische Komponenten**: Entitäten, die permanent existierende Objekte des Realsystems oder Klassen von Objekten darstellen; Attribute ermöglichen die Identifikation individueller Entitäten und legen die prinzipiell möglichen Zustandsübergänge fest.

- **Dynamische Komponenten**: alle Ereignisse (repräsentiert durch Ereignistypen) und temporäre Entitäten (Erzeugung und Vernichtung während der Simulation, ausgelöst durch Ereignisse).

Eine Software-Realisierung benötigt daher:

- **Ereignisroutinen/-methoden**: führen Zustandsänderungen durch und bestimmen somit das Modellverhalten durch
  - Änderung der Attributwerte von Objekten
  - Erzeugung und Löschen von temporären Objekten
  - neue, zukünftige Ereignisse der Ereignisliste hinzufügen oder existierende aus dieser löschen
- **Scheduler**: Steuerung der Ablaufkontrolle; sequenzielle Abarbeitung der nach Ereigniszeitpunkten geordneten Ereignisse (next event approach)
  → Simulationszeit springt von Ereigniszeitpunkt zu Ereigniszeitpunkt – nur dabei wird die Simualtionsuhr weiter gesetzt!

Einschub: Überblick DESMO-J

DESMO-J = Descrete-Event Simulation Modelling in Java
(siehe http://www.desmoj.de)

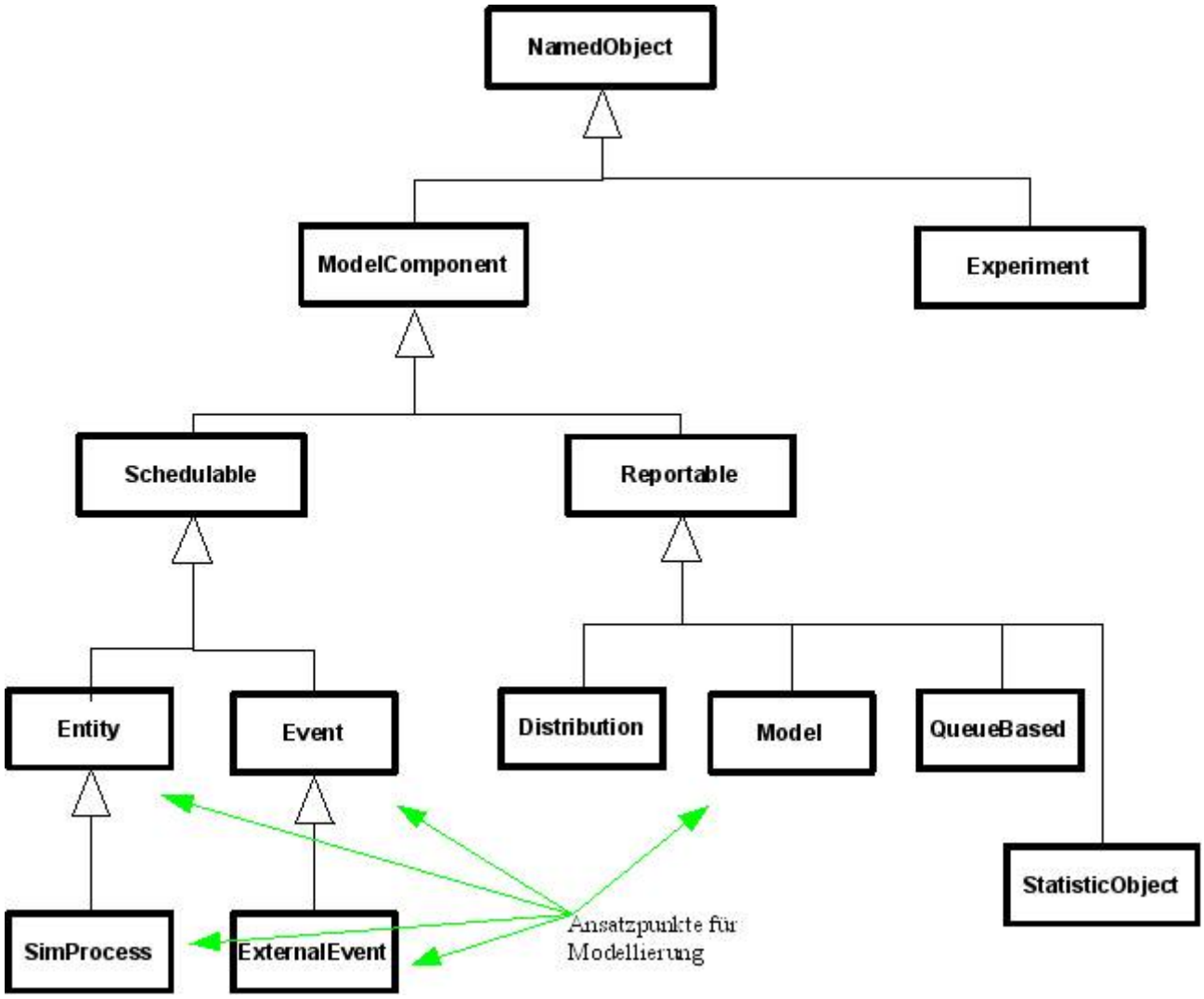
OO-Framework zur Erstellung von diskreten Ereignissimulationen (ereignis- und/oder prozessorientiert).

Wichtige Funktionen sind:

- Erzeugung und Verwaltung von Entitäten, Ereignissen und Prozessen – Modellerstellung
- Simulationszeit (-uhr)
- Bereitstellung einer Experimentierumgebung (Ereignislistenmechanismus, Scheduler, ...)
- Verwendung von Warteschlangen und weiterer Hilfsmittel

- Nutzung verschiedener Zufallszahlenvertreilungen
- Hilfsmittel zur statistischen Auswertung

Übersicht der Klassenhierarchie



Wichtige Klassen sind:

`Experiment`: Infrastrukutur für einen Simulationslauf (eines Modells)

`Model`: Verwaltung aller Objekte eines Modells - Darstellung des Modells

`Entity`: Darstellung modellspezifischer Entitäten

`SimProcess`: Darstellung von Entitäten mit eigenem „Lifecycle“ (Prozesse)

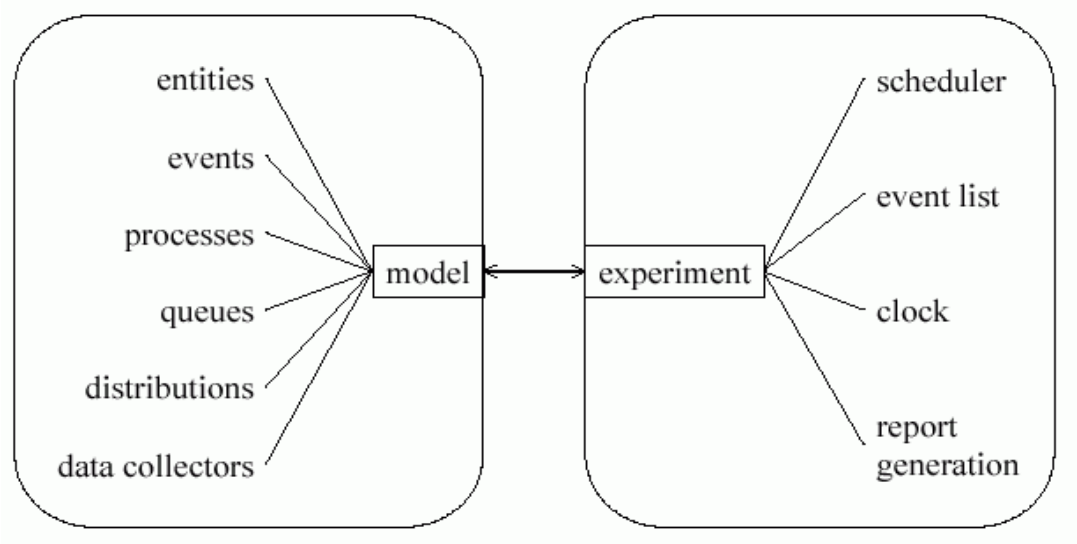
`Event`: Darstellung von Ereignissen

`Distribution`: verschiedenen Zufallszahlenverteilungen

`QueueBased`: Basisverhalten und Statistik für Warteschlangenkomponenten

`TimeSpan`: Zeitspanne der Simulationszeit
`TimeInstant`: Zeitpunkt der Simulationszeit

Ein Grundprinzip ist die Trennung von Modell und Experiment (Skizze aus DESMO-J Tutorial):



Leitfaden zur Modellerstellung

(nach DESMO-J Tutorial, allgemein anwendbar)

1. Pre-Implementierung
    - Modell definieren: was ist nötig, was kann weggelassen werden → Abstraktion

- aktive Entitäten (actors) bestimmen – werden meist zu Ereignissen oder Prozessen
- Aktionen der aktiven Entitäten festlegen

2. Modellklasse implementieren
   - Ableitung der zentralen Modellklasse von `Model`
   - Definition der notwendigen Infrastruktur: Warteschlangen, Zufallszahlengeneratoren, ...
   - `get...()`-Methoden für Infrastrukturelemente
   - erste Ereignisse oder Prozesse in die Warteliste eintragen – `doInitialSchedules()`
   - `main()`-Methode anpassen: z.B. Simulationsende festlegen
3. Entitäten implementieren
   - „Actor“-Klassen von `Event` oder `SimProcess` ableiten
   - Konstruktoren anpassen
   - Verhalten/Aktivitäten der „Actors“ in `eventRoutine()` oder `lifeCycle()` implementieren
4. Post-Implementierung
   - mit Modell experimentieren und Fehler korrigieren (Syntax, Semantik)
   - „Error“-Datei prüfen → eventuelle Inkonsistenzen
   - Modell validieren: schwierig (bis unmöglich), gute Simulationskenntnisse und Statistiktools nötig

7 Prozessorientierter Modellierungsstil

Später entstanden als ereignisorientierter Modellierungsstil, wurde jedoch zum meist angewendeten Stil.

Wichtige Merkmale:

- Darstellung des *vollständigen* **Lebenszyklus (lifecycle)** einer Entität als Prozess.

- Simulationszeit schreitet während *aktiver* Phasen der Entitäten voran – d.h. immer dann, wenn eine Zeitspanne vergeht.

Somit ergibt sich auf konzeptioneller Ebene:

> Prozesse laufen i.A. zeitlich parallel ab und übergeben die Kontrolle immer dann an den *Scheduler*, wenn die Simulationszeit weiter gesetzt wird.
> Danach kehrt die Kontrolle umittelbar oder nachdem andere (parallele) Prozesse ihre Zustandsänderungen durchgeführt haben zurück.

Dabei ist zu beachten:

- Zustandsänderungen geschehen während aktiver Prozessphasen.

- Im Modell dargestellte Aktivitäten oder Aktionen bewirken i.A. das Voranschreiten der Simulationszeit – jedoch Zustandsänderungen wirken augenblicklich, es verstreicht dabei *keine* Simulationszeit (benötigen aber Rechenzeit).

Diese Zustandsänderungen, ausgeführt von einem Prozess, sind z.B.

- Attribute von Entitäten modifizieren

- neue Prozesse (und deren Entitäten) generieren

- Aktivierung anderer Prozesse zu bestimmten Simulationszeitpunkten

- geplante Aktivierungen verschieben oder löschen

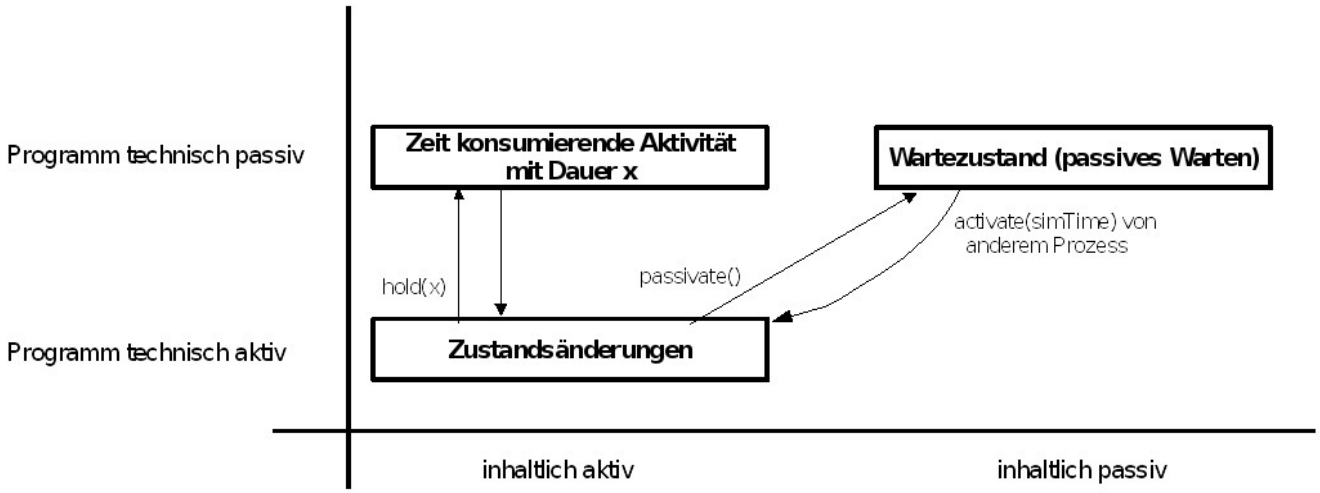
- selbst deaktivieren und Kontrolle an Scheduler oder anderen Prozess abgeben

- sich selbst oder andere Prozesse beenden

Somit ergibt sich, dass die Abbildung von zeitverbrauchenden Aktivitäten mittels *inaktiver* Prozessphasen geschieht. Es gibt 2 Arten davon:

- inaktiv für vorgegebenes fixes Intervall der Simulationszeit (mittels `hold()`): Abbildung einer *zeitverbrauchenden Aktivität* (Kontrolle inzwischen an Scheduler);

- inaktiv für unbestimmte Dauer der Simulationszeit (mittels `passivate()`): Abbildung eines *Wartezustands* (Kontrolle auf unbestimmte Zeit abgegeben, Reaktivierung meist durch andere Prozesse)



Der *Scheduler* steuert im Hintergrund die Abarbeitung beliebig vieler Prozesse:

- Ausführung der Prozesse in richtiger Reihenfolge und vorgesehenen Zeitabständen

- (Re-)Aktivierung eines Prozesses entspricht einem Ereignis

  - Verwaltung der zu aktivierenden Prozesse in einer *Ereignisliste*

  - Auswahl des nächsten Aktivierungs- (= Ereignis-)zeitpunkts und

entsprechenden Prozess reaktivieren

- Abarbeitung von Prozessen i.A. nicht von Anfang bis Ende ohne Unterbrechung (im Gegensatz zu Ereignissen)
  → Status bei Deaktivierung ist zu speichern, um spätere Fortsetzung zu ermöglichen

  daher: Implementierung dieses Ansatzes eher aufwändiger!

Zur Erstellung eines prozessorientierten Modells ist zu beachten:

- Idendifikation der relevanten Systemobjekte und ihrer Attribute
  → im Modell als Entitäten oder Prozesstypen dargestellt

- relevante Aktivitäten und Attribute dieser Objekte festlegen

- Beschreibung des *Lebenszyklus (lifecycle)* von jedem Prozesstypen

  d.h. aus der Sicht des zu modellierenden Objekts sind alle relevanten Aktivitäten inkl. deren Reihenfolge sowie Beziehungen zu anderen Entitäten zu beschreiben
  - *Froschperspektive*

Vergleich ereignis-/prozessorientierter Modellierungsstil

ereignisorientiert	prozessorientiert
ähnliche interne Sicht: sequentielle Abarbeitung der Ereignisliste und weitersetzen der Simulationsuhr an den Ereigniszeitpunkten	
Ereignisliste beinhaltet: Ereignisse mit Ereigniszeitpunkt (und Verweis auf betroffene Entitäten)	Ereignisliste beinhaltet: Prozesse mit ihren (Re-)Aktivierungzeitpunkten
typische Systemänderungen (meist keine Zeit verbrauchend) mittels Ereignissen natürlicher abbildbar	Abbildung aktiver und passiver Phasen von Komponenten näher an der Realität
schlechte Übersichtlichkeit bei komplexen Interaktionen von Komponenten	Übersicht besser durch Verwandschaft zu OO
Orientierung bei Modellbildung an internen Simulationsabläufen → größere semantische Lücke zum Realsystem	Orientierung bei Modellbildung eng am Realsystem

einfachere SW-technische Realisierung	höherer SW-technischer (Verwaltungs-) Aufwand (z.B. Synchronisation, Speicherung von Prozesszuständen)
→ Umsetzung praktisch in jeder gängigen Programmiersprache möglich	Umsetzung benötigt Konzepte wie Threads, Koroutinen, ...

Jedes prozessorientierte Modell kann in ein ereignisorientiertes Modell überführt werden:

> Prozessroutinen (z.B. lifeCyle) bestehen meist aus mehreren Ereignissen → Aufspaltung in einzelne Ereignisse führt zu ereignisorientierten Modell!

8 Weitere Modellierungsstile

Transaktionsorientierter Modellierungsstil

- abgeleitet aus der Blockdiagrammtechnik der Systemanalyse
- Blöcke: permanent vorhandene, statische Komponenten
- Transaktionen: temporäre, dynamische Elemente
- Transaktionen „durchwandern“ Blöcke, wobei ihr Zustand verändert wird
- Abbildung auf ereignisorientierten Stil leicht möglich

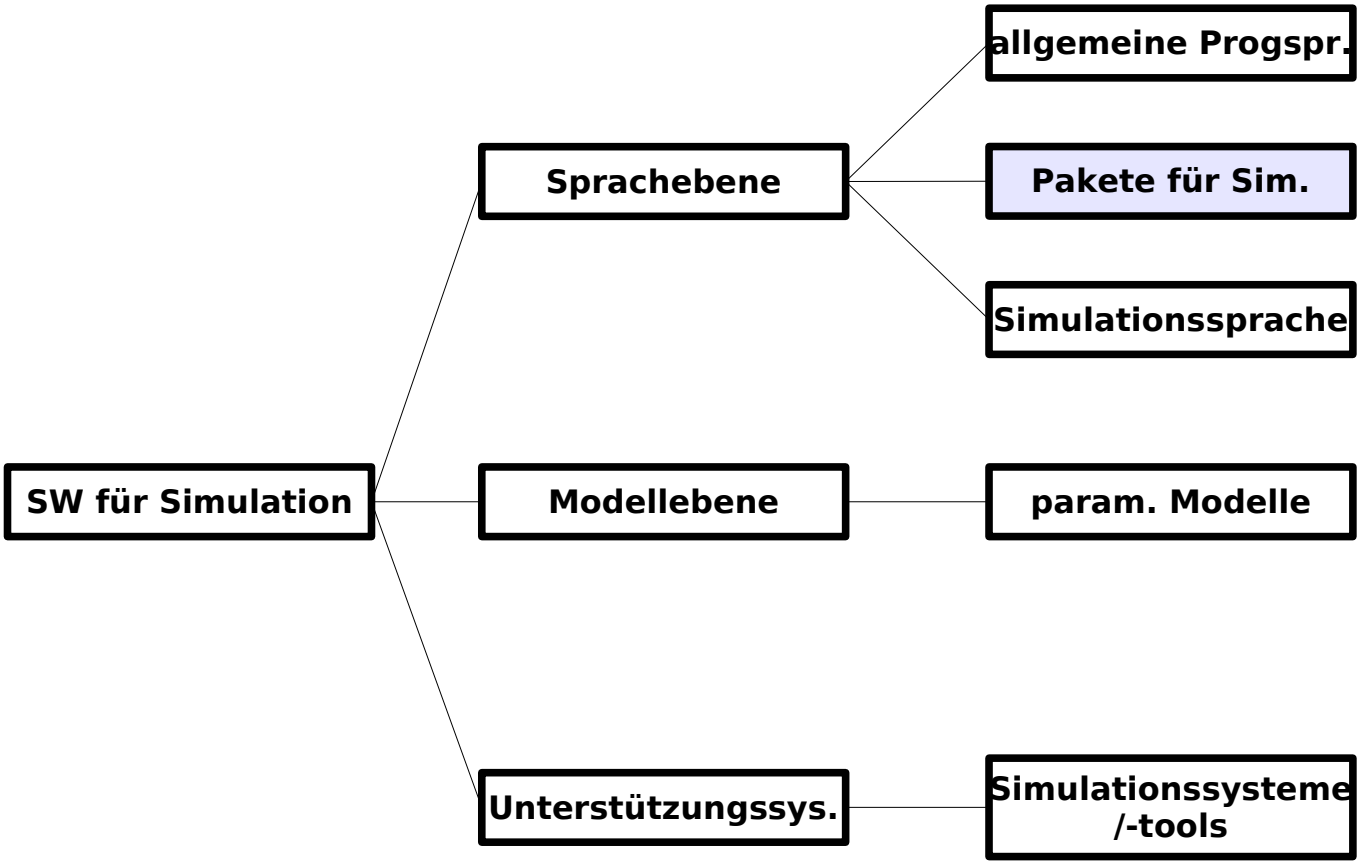
Aktivitätsorientierter Modellierungsstil

- Aktivitäten als Grundelemente, die bei gegebenen Vorbedingungen ausgeführt werden
- rechenintensiv: zu jedem Zeitpunkt sind alle Vorbedingungen aller möglichen Aktivitäten zu prüfen

- keine direkte Abbildung auf ereignisorientierten Stil möglich

9 Software für Simulation

Simulation ist eines der ältesten Anwendungsgebiete der Informatik, daher gab es auch schon früh spezielle Software (*Simulatoren*) dafür. Simulationssoftware kann nach folgenden Aspekten gegliedert werden (nach [2]):



Software auf Sprachebene:

- Simulator in einer (universellen oder speziellen) Programmiersprache realisiert
- Gestaltungsfreiheit wird kaum eingeschränkt
- keine Simulations spezifische Unterstützung (wie z.B. Ablaufsteuerung,

Statistik, ...)

- jedoch meist Konzepte für Pakete, Klassen, ... - damit wird entsprechende Funktionalität bereitgestellt;
  z.B. *Simulationspakete* mit praktisch beliebiger Erweiterungsmöglichkeit (z.B. DESMO-J)

- Alternative: spezielle *Simulationssprachen*: Konstrukte für Ablaufsteuerung und/oder Modellierung
  → raschere Modellbildung – jedoch geringere Flexibilität und Anwendungsbreite

Software auf Modellebene:

- Simulation für ein spezifisches Realsystem (z.B. Flugsimulator)

- keine Entwicklung in einer Programmiersprache nötig

- kaum Änderungen möglich – nur über Modellparameter

- bei Wiederverwendung ist genau zu prüfen, ob der Simulator das neue System korrekt abbildet

Unterstützungssysteme:

- einheitliche Umgebung für Modellierung und Experimente – *Simulationssysteme* oder *-tools*

- Zyklus Modellbildung – Experiment – Ergebnisanalyse wird unterstützt

- bieten insbesonders: Unterstützung bei der Modellbildung, Experimentierumgebung, Statistiktools

- eventuell auch Teilmodelle zur Wiederverwendung – Modellbanksysteme

Java und Simulation – wie werden Anforderungen erfüllt:

- Zustandsvariablen: primitive Typen, boolean, beliebige Datenstrukturen mittels Klassen

- Simulationsuhr: Kapselung der Simulationszeit, globale Variable
- Temporäre Entitäten/Objekte: mit new erzeugt, vom Garbage Collector entfernt
- Verwaltung zukünftiger Ereignisse: Containerklassen für Ereignisliste
- Warteschlangen: Containerklassen
- Zufallszahlengeneratoren: im Java-API vorhanden
- Statistische Auswertungen: entsprechende Objekte (z.B. mittels Observer Pattern)
- Ergebnisausgabe: Text- oder graphische Ausgabe, Dateien
- unterbrechbare Prozesse: Threads mit entsprechender Funktionalität