Einführung in SeSAm

Von
Daniel Beckmann, Philip Harborth, Jakob Acar, Thomas Kaiser,
Martin Pellengahr, Christoph Wünnemann, Philipp Thiemann

SeSAm Agenten in simulierten Umgebungen

Gliederung des Vortrags

- Was ist SeSAm?
- Entwicklungsumgebung
  - Agenten
  - Ressourcen
  - Umwelt
- Simulation
- Simulationselemente
- Fragen & Diskussion

Einführung in SeSAm

Was ist SeSAm?

- **SeSAm = Shell für Simulierte Agentensysteme**
- Simulationsumgebung zur Modellierung von **Multiagentensystemen**
- Entwicklung an der **Uni Würzburg**
- Implementierung in **Java**
- ursprünglich zur Simulation komplexer Agenten in der **Biologie**
- Hintergrund: **Künstliche Intelligenz**
- Speicherung der Simulation in einer **XML-Datei**
- bestehend aus Regeln mit Aktivitäten und Zustandsvariablen (→ UML-ähnlich)

Multiagentensysteme

- **Der Agent**
  - Definition
  - Agentenarchitekturen
- **Eigenschaften** von Multiagentensystemen
- der **SeSAm** – Simulator

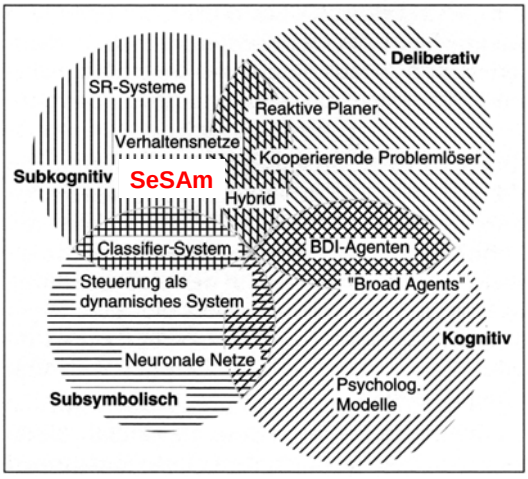
Multiagentensysteme – Der Agent (1/2)

- Definition :

  Ein **Agent** ist ein System, das sich in einer **Umwelt befindet**, und in der Lage ist, in dieser Umwelt **autonome Aktionen** durchzuführen, um seine Ziele zu erreichen. Der Agent hat **sensorischen Input**, kann seine Umgebung verändern.

Multiagentensysteme – Der Agent (2/2)

- Agentenarchitekturen



Multiagentensysteme – Eigenschaften

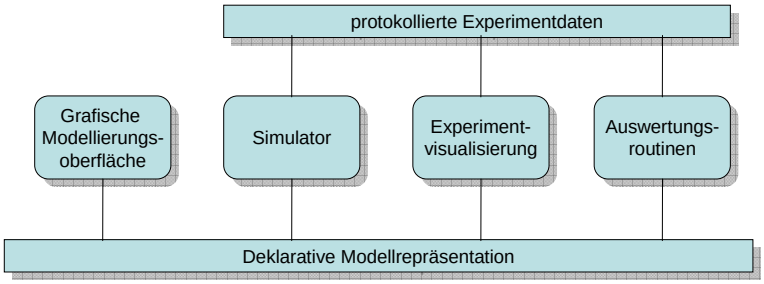
- **beschränkte Sicht** des einzelnen Agenten, unvollständige Informationen, beschränkte Problemlösefähigkeit
- **dezentrale Datenhaltung**, d.h. jeder Agent verwaltet seine Daten lokal
- Berechnungen und die Ausführung von **Aktionen** verschiedener Agenten geschehen **parallel** und müssen bei der Konstruktion beachtet werden
- im Idealfall verfügen MAS über **keine zentrale Kontrolle** (→ Folge der Kontrollautonomie der Agenten)
- Unterschied „verteilte Problemlöser“ und MAS:
  - bei „verteilten Problemlösern“ verfolgen alle Agenten ein globales Ziel
  - Bei Multiagentensystemen i.e.S. muss dies nicht der Fall sein („egoistischer Agent“)

SeSAm – Der Simulator (1/2)

- **grafische Modellier-, Experimentier- und Auswertungsumgebung**
- **Trennung von Modell und Simulator**
  - Vorteile: höhere Flexibilität und Abstraktionsniveau bei Modellentwicklung, Manipulierbarkeit und Zugänglichkeit des Modells
- **SeSAm ist Simulationsumgebung für viele (>500) Agenten,** nicht für wenige mächtige und kognitiv-planende Agenten
- trotz **Schlichtheit der Strukturen** kann Verhalten beträchtliche Komplexität aufweisen
- Modell kann anspruchsvoll sein, Mechanismen bei Umsetzung sollen einfach sein.

SeSAm – Der Simulator (2/2)

- Bestandteile des SeSAm-Systems



Eine **deklarative Repräsentation** hat Vorzüge, wenn die **Verständlichkeit der Darstellung** wichtig ist, während **prozedurale Repräsentation** häufig gewählt werden, wenn die **Effizienz** im Vordergrund steht.

SeSAm - Objektorientierter Rahmen

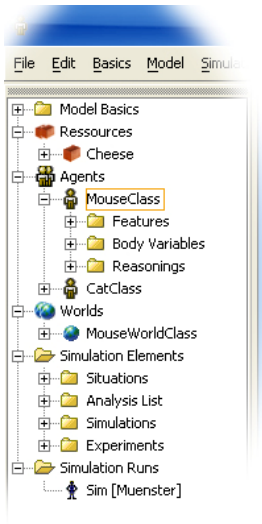
- Umsetzung im **objektorientierten Rahmen** (Klasse – Objekte)
- **UML - ähnlich**: Instrumentarium für die Analyse und den Entwurf objektorientierter System
- Darstellung der **Interaktionen** zwischen den einzelnen Einheiten steht im Mittelpunkt (z.B. Aktivitätsdiagramme)

SeSAm – Repräsentation und Entwicklung

- **Repräsentation**:
  - grundlegendes Schema: Regeln mit Aktivitäten und Zustandsvariablen
  - hoher Abstraktionsgrad der Beschreibung
  - Abstraktionsgrad der Schnittstellen: Regelaktionen, einfache Symbolische Nachrichten
  - deklarative Architektur
  - Strukturierung des Agentensystems nach Agenten, Ressourcen, Bezugssysteme
  - Strukturierungsverhalten: Aktivitäten usw.
- **Entwicklungsumgebung**:
  - vollständige grafische Verhaltensmodellierung
  - grafische Konfigurierung (nach Anpassung)
  - Animation
  - Auswertung um System und Export

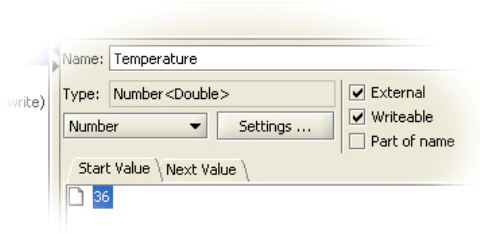
Aufbau der Entwicklungsumgebung

- Java Look & Feel Design
- Zugriff auf alle Bedienfunktionen und Modellelemente über Kontextmenüs in einem zentralen Menübaum
- Modellelemente: **Agenten**, **Ressourcen** und **Worlds**
- vier Simulationselemente zur Konfiguration, Auswertung und Automation der Simulationsläufe



Entwicklungsumgebung – Agentenkonzept

- **Agent = Body + Behavior**
- Agentenklassen lassen sich mit Hilfe von Status-**Variablen** und einem zugehörigen **Aktivitätsdiagramm** beschreiben
- die wichtigsten **Datentypen** der Variablen sind:
  - Number (Integer/Double)
  - Boolean
  - Position/Spatialinfo
  - Simobject
  - Iterator/List
  - String

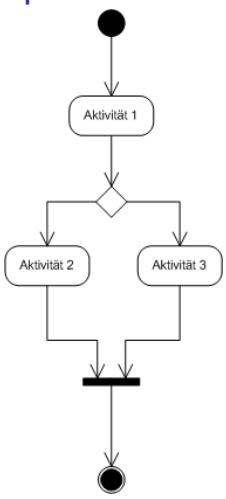


SeSAm Agenten in simulierten Umgebungen

Entwicklungsumgebung – Agentenkonzept

- UML-ähnliche Aktivitätsdiagramme (**Reasoning Engine**) beschreiben das Verhalten des Agenten und ermöglichen die Interaktion mit seiner Umwelt
- Komponenten:
  - **Start-/Endknoten**
  - **Activity nodes** (beinhalten einzelne Actions)
  - **Decision nodes**
  - Kontrollfluss durch stochastische oder deterministische **Regelpfeile**

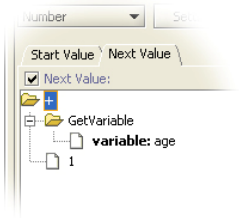
→ Mouse-Beispiel



Entwicklungsumgebung – Ressourcen

- im Gegensatz zu den Agenten haben Ressourcen **passiv**en Charakter
- sie können ihre **Umwelt nicht** eigenständig **verändern** und entstehen oder verschwinden durch ein Ereignis oder den Einfluss eines Agenten
- ihr interner Zustand wird **ausschließlich** durch ihre **Variablen** beschrieben
- Dynamik erfahren sie durch die ‚Next-Value'-Funktion

→Cheese-Beispiel 



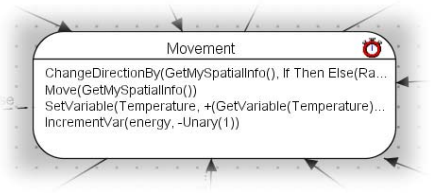
Entwicklungsumgebung – Umwelt

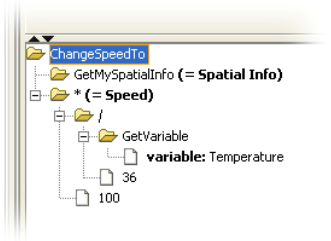
- die Rolle der **Umwelt** in der die Agenten eines Modells interagieren übernimmt in SeSAm die ‚**World**'
- eine World ist nichts anderes als ein **Agent**, mit dem sich die äußeren Einflüsse auf das Modell steuern lassen
- intern werden Klassen einer **World als Agentenklasse** verwaltet
- die Aktivitäten der World eignen sich, um **globale Parameter** zu setzen und um **Agenten** und Ressourcen bei Bedarf zu **erzeugen** oder zu **löschen**

→ MouseWorld-Beispiel

Entwicklungsumgebung – Interaktion der Agenten

- die Interaktion der Agentenklassen beginnt mit dem Start eines Simulationslaufes
- grundsätzlich führt jeder Agent in jedem Durchgang (Step) die Sequenz der Aktionen seiner jeweiligen Aktivität aus (vs. instantly)
- Reihenfolge: erst die World, dann die Agenten (zufällig)
- Aktionen lassen sich in einer **Präfix-Notation** in Baumdarstellung programmieren



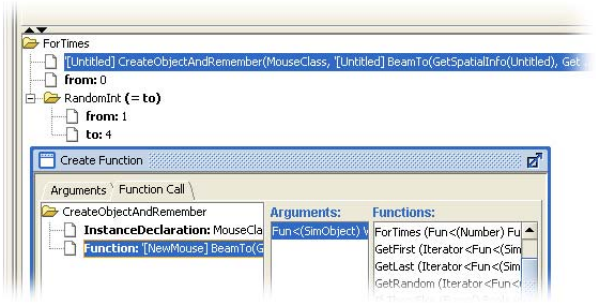


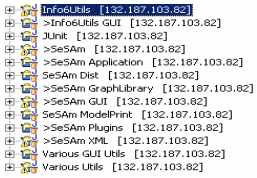
Entwicklungsumgebung – Interaktion der Agenten

- die Funktionsbibliothek liefert 12 verschiedene Funktionstypen
- gängige Typen sind:
  - Number Functions:
    - Number +(Number<Double>, ...)
    - Number<Double> GetSpeed(SpatialInfo)
  - Actions:
    - Void BeamTo(SpatialInfo, Position)
    - Void MoveTowardsPos(SpatialInfo, Position, Number<Double>)
  - Returning Iterators & Lists:
    - Iterator<T> Select(Iterator<SimObject>, Fun<(SimObject) Boolean)

Entwicklungsumgebung – Interaktion der Agenten

- bei Bedarf lassen sich benutzerdefinierte Funktionen aus bestehenden zusammensetzen (vgl. Makro)
- Implementierung von Plugins in Eclipse





Simulation

- Vorstellung der Simulation
  - 2 Agenten-Klassen (Maus & Katze)
  - 1 Ressource (Käse)
  - 1 Umwelt
- Maus-Agenten:
  - globale Temperatur beeinflusst Maus-Temperatur
  - Mäuse schlafen, laufen umher, fressen Käse oder paaren sich
  - Energie-Vorrat bestimmt das Verhalten

Simulation

- Katzen-Agent:
  - Katzen laufen umher, jagen Mäuse oder paaren sich
  - Energie-Vorrat bestimmt das Verhalten
- Käse-Ressource:
  - Einheiten werden von Mäusen aufgenommen
- Umwelt-Agent:
  - vernichtet veraltete Agenten bzw. Agenten ohne Energie
  - erzeugt zufällig neue Käse-Ressourcen
  - schreibt den Maus-Agenten die globale Temperatur

SeSAm Agenten in simulierten Umgebungen

Simulation

Praktische Vorführung in SeSAm …

Einführung in SeSAm

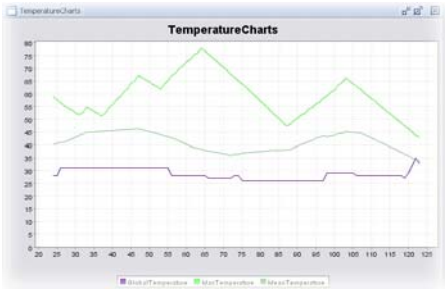
Simulationselemente

- **Situations**
  - grafische Erzeugung einer **Startkonfiguration**
  - Positionierung der Agenten auf dem Simulationsfeld
  - Konfigurationsmöglichkeit des Feldes (Größe, Torus, …)
  - auch: nachträgliche Manipulation der Variablen möglich

SeSAm Agenten in simulierten Umgebungen

Simulationselemente

- **Analysis Lists**
  - benutzerdefinierte **Datenselektion**
  - durch Kombination bestehender Funktionen kann der Benutzer einem **Analyse-Chart** verschiedene Werte zuordnen
  - Chart-Typen:
    - Datei-Ausgabe
    - Balkendiagramm
    - Zeitreihen-Diagramm
    - Tabelle



Einführung in SeSAm

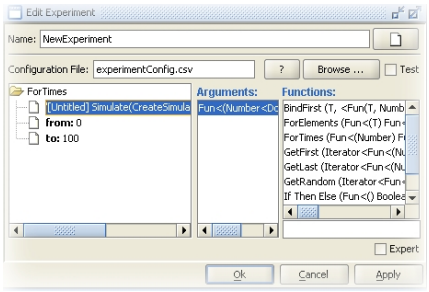
Simulationselemente

- **Simulations**
  - in einer Simulation können einer Situation mehrere Analyse-Charts zugeordnet werden
  - durch **Abbruchbedingungen** wird eine Simulation bei deren Erfüllung beendet
  - zusätzlich können während und nach der Simulation Tests durchgeführt werden

  →**komplette Deklaration** eines einzelnen Simulationslaufes

Simulationselemente

- **Experiments**
  - ermöglichen die **Automatisierung** von Simulationen
  - Start- und Endwerte der globalen Variablen zu jedem erzeugten Simulationslauf werden in eine CSV-Datei geschrieben



Fragen & Diskussion