AUFGABENBLATT 3

Modellierungsbeschreibung

abs16
30. Mai 2016

1 ALLGEMEINES

Dieses Dokument dient dazu, das Multiagentensystem des Projekts abs16 in seinen einzelnen Paketen darzustellen. Die Pakete werden hinsichtlich ihrer eigenen Funktionalität, ihrer Voraussetzungen an andere Pakete und ihren angebotenen Funktionalitäten beschrieben.

Das Endprodukt wird das Framework JADEX verwenden. Auf konkrete Beschreibung der Implementierung wird in diesem Dokument der Übersicht halber jedoch verzichtet.

2 PRODUKTÜBERSICHT

In diesem Abschnitt soll es darum gehen, die Produktfunktionen zu beschreiben. Dies geschah bereits ausführlich im Arbeitsplan. Daher werden die Funktionen nur nach Schlagwörtern aufgegriffen, geordnet und in verschiedene Gruppen eingeteilt, die dann von den Paketen realisiert werden.

Ziel dieses Multiagentensystems ist es, über proaktiv handelnde Agenten die Diffusion am Energiemarkt zu simulieren.

2.1 Produktfunktionen

Bevor die oben angesprochene Gruppierung der Funktionen vorgenommen wird, werden die Produktfunktionen rekapituliert:

Das Produkt umfasst Produkte mit Eigenschaften.

Das Produkt umfasst Agenten mit Eigenschaften, Präferenzen/Wunschprodukten und Verhalten. Agenten werden unter Kundengruppen zusammengefasst, die Wertebereiche für Eigenschaften und Wunschprodukte festlegen.

Agentenverhalten umfasst Interaktion zwischen Agenten in Form von Kommunikation und Interaktion mit der Welt in Form eingehender Kommunikationsevents (z. B. Werbung) und eingehender Events, die Agentenattribute verändern.

Die Konfiguration des Produkts lässt sich sowohl über eine GUI als auch über Konfigurationsdateien vornehmen. Konfigurationsdateien lassen sich aus der GUI heraus laden und speichern.

Am Ende der Simulation wird das gespeicherte Verhalten der Agenten in Form von Graphen ausgegeben.

2.2 Funktionsgruppen

Um die oben aufgezählten Produktfunktionen sinnvoll zu implementieren, werde sie in mehrere funktionale Gruppen eingeteilt, die dann von verschiedenen Komponenten und Paketen der Implementierung des Endprodukts realisiert werden.
Die Funktionen werden in folgende Gruppen unterteilt:

Simulationsfunktionen Funktionen der tatsächlichen Simulation innerhalb des Gesamtprodukts

Konfigurationsfunktionen Funktionen der Konfiguration beschreiben alle Ansprüche an Parametrisierung der Simulation

Ausgabefunktionen Funktionen der Ausgabe betreffen alle Ansprüche an Ausgabe von Ergebnissen der Simulation (Anm.: dies umfasst nicht die Darstellung der Ergebnisse)

Nutzer-Funktionen Funktionen für den Nutzer betreffen Ein- und Ausgabemöglichkeiten des Gesamtprodukts aus Nutzersicht

2.2.1 Simulationsfunktionen

Funktionen der Simulation greifen Teile der Absätze über Produkte, Agenten und Agentenverhalten auf.

Die Simulation muss...

- ... Agenten mit Eigenschaften und Wunschprodukten verarbeiten.
- ... Agentenverhalten verarbeiten.
- ... Produkte mit Eigenschaften und Kommunikations-Events verarbeiten.

2.2.2 Konfigurationsfunktionen

Funktionen der Konfiguration greifen Teile der Absätze über Produkte, Agenten und Konfigurationen auf.

Die Konfiguration der Simulation muss...

- ... Agentengruppen erstellen.
- ... Agenten aus Agentengruppen erstellen.
- ... Produkte erstellen.
- ... das Erstellen von Objekten über Methoden der Programmiersprache ermöglichen.
- ... das Erstellen von Objekten über Konfigurationsdateien ermöglichen.
- ... das Erstellen von Konfigurationsdateien ermöglichen.

2.2.3 Ausgabefunktionen

Funktionen der Ausgabe greifen den Absatz über das Ende der Simulation auf.

Die Ausgabe muss...

- ... das Speichern von Agentenverhalten ermöglichen.
- ... das Speichern von Agenten-Zuständen ermöglichen.
- ... die Ausgabe gespeicherter Daten über Methoden der Programmiersprache ermöglichen.

2.2.4 Nutzer-Funktionen

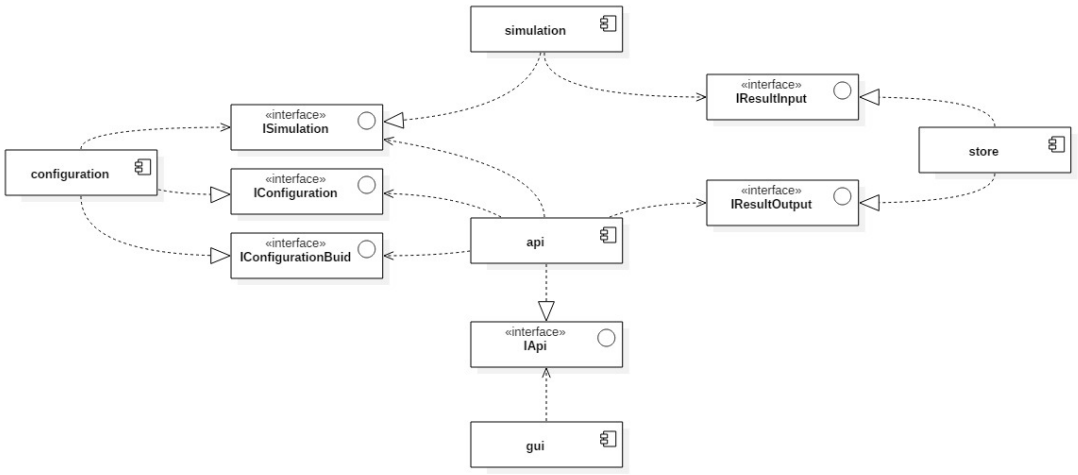
Funktionen für den Nutzer greifen Teile der Absätze über Konfiguration und das Ende der Simulation auf.

Der Nutzer kann...

- ... die Simulation über eine GUI konfigurieren.
- ... die Simulation über Konfigurationsdateien konfigurieren.
- ... Konfigurationen der Simulation in Konfigurationsdateien speichern.
- ... die Ergebnisse der Simulation über eine GUI einsehen.
- ... die Ergebnisse der Simulation in Dateien speichern lassen.

3 Grundsätzliche Struktur- und Entwurfsprinzipien

Das Endprodukt wird in seiner Implementierung in folgende Pakete eingeteilt:

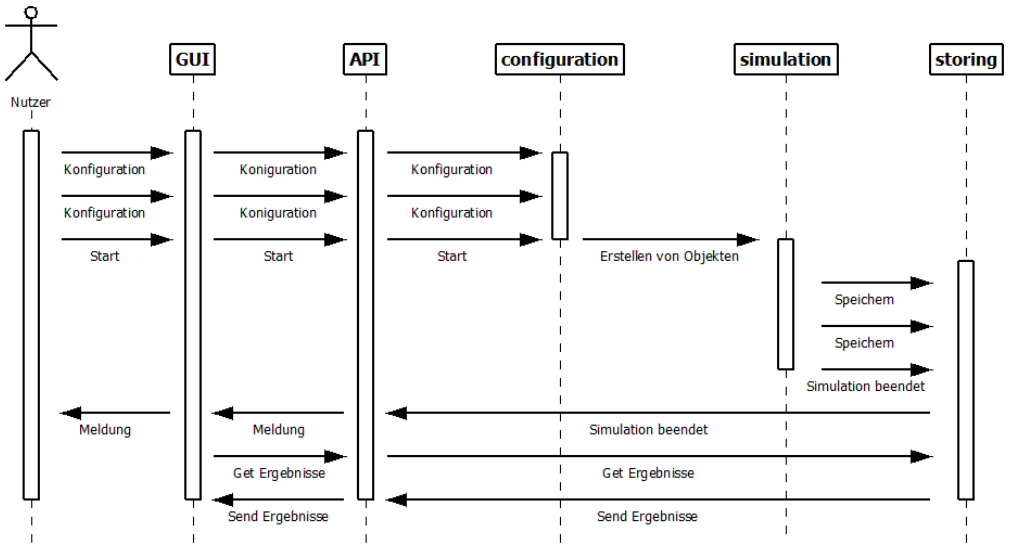


Dabei übernehmen Pakete jeweils die Implementierung von nicht unbedingt verschiedenen Funktionsgruppen.

Die vier Pakete `configuration`, `simulation`, `store` und `api` umfassen den Teil des Produkts, der sich insgesamt um die Simulation kümmert. Durch die API ist es möglich, diesen Teil des Produkts auch ohne eine GUI zu nutzen.

Die GUI dient dazu, dem technisch unerfahrenen Nutzer eine einfachere Steuerung der API zu ermöglichen.

Um die grobe Funktionalität der einzelnen Pakete zu umreißen, wird folgendes Sequenzdiagramm erläutert:



Nach dem Programmstart führt der Nutzer nacheinander durch mehrere Aktionen *Konfiguration* Einstellungen an der Simulation durch. Jede dieser Einstellungen wird von der API an das Paket configuration weitergereicht.

Nachdem die Konfiguration beendet wurde, startet der Nutzer durch die Aktion *Start* die Simulation. Die API reicht auch diese Aktion an das configuration-Paket weiter, das aus den Einstellungen die eigentliche Simulation startet.

Das simulation-Paket übernimmt nun die Berechnung des Multiagentensystems und speichert Ergebnisse der Berechnung über die Aktion *Speichern* in dem Paket storing. Nach Beenden der Simulation benachrichtigt das Paket simulation das Paket storing darüber. Dieses Paket leitet die Nachricht über die API bis zum Nutzer weiter.

Die GUI fragt über die Aktion *Get Ergebnisse* die Simulationsergebnisse selbstständig an, die am Ende dargestellt werden.

Die Lebenszyklen der Pakete werden sich in der Implementierung von der obigen Darstellung unterschieden. Dieses Sequenzdiagramm wurde für den Standardfall der Benutzung des Endprodukts entworfen und zeigt die Lebenszyklen der Pakete nur für ihre Hauptfunktionalität an.

4 Struktur- und Entwurfsprinzipien einzelner Pakete

4.1 Paket GUI

Durch dieses Paket wird eine grafischen Benutzeroberfläche für unser Produkt bereitgestellt. Dadurch wird es dem Benutzer erleichtert Daten und dazugehörige Konfigurationen einzugeben sowie die Simulation zu starten. Die sich dadurch ergebenen Ergebnisse werden außerdem zur einfachen und schnellen Auswertung grafisch dargestellt.

4.1.1 Angebotene Schnittstellen

Aus technischer Sicht bietet die GUI keine Schnittstellen an.

4.1.2 Paketstruktur

Die GUI ist in zwei Hauptansichten unterteilt. Die eine dient der Eingabe und Konfiguration von Daten durch den Benutzer (*mainAppView*), die andere der Ausgabe und Darstellung der Ergebnisdaten (*dataView)* . Den Aufbau der jeweiligen Fenster sowie ihrer festen Elemente werden jeweils durch eine `fmxl` Datei strukturiert. Diese werden von der `MainApp` Klasse zum erstellen der Fester während der Laufzeit verwendet.
Des weiteren wird durch sie auch die Hilfe dargestellt. Dafür wird jedoch keine *fxml* Datei verwendet sondern ein *WebView* erstellt und in diesem eine lokale *html* Datei angezeigt.
Das GUI Paket besteht aus zwei untergeordnete Pakete:

- **Controller**: Enthält Controller Klassen. Durch sie werden die dynamische Inhalt der Fenster erstellen und verwaltet sowie auf Interaktionen des Benutzers mit der Oberfläche reagieren.
- **Listener** : Enthält Listener Klassen welche von den Controllern genutzt werden um komplexere Datenstrukturen der Konfiguration anhand von Benutzereingaben zu erstellen bzw. zu entfernen.

Um den Paketaufbau übersichtlicher darzustellen werden die Klassen im folgenden **nicht** nach ihren (Unter-) Paketen getrennt behandelt .
Folgende Klassen werden genutzt um den Konfigurationsbaum des Hauptfensters darzustellen und zu verwalten:

- `TreeViewController`
  - Erstellt den Auswahlbaum für die GUI. Dieser ermöglicht es dem User auf alle Einstellungsmöglichkeiten bequem zugreifen zu können
  - Bei Veränderungen in der Konfiguration wird hier eine neue Ansicht erzeugt um den neusten Stand wiederzugeben.
- `TreeSelectionListener`

  - Empfängt alle Events die bei der Ausführung des TreeViewControllers erzeugt werden. Von hier wird entsprechend der Auswahl die zugehörige Klasse aufgerufen um im Bearbeitungsfenster Einstellungen vornehmen zu können.

- `IdentifiableTreeItem`
  - Ist eine Erweiterung der Klasse TreeItem aus JavaFx um einen Identifikator.
  - Ermöglicht dem TreeViewController und -Listener schnelle Vergleiche mithilfe des Identifikators, anstatt Strings nutzten zu müssen.

- `TreeNodeIdentifier`
  - Dient als Identifikator für IdentifiableTreeItem
  - Dies ist eine Enumeration mit einem Eintrag zu jedem Hauptstichpunkt in unserem Auswahlbaum.

Den Hauptobjekten der Simulation welche über die Benutzeroberfläche erstellt und konfiguriert werden können sind folgende Klassen zuzuordnen:

Hauptobjekte der Simulation:

- Agenten Attribute
  - `SelectedNewAgentAttributeController`
    - Erstellt die Eingabemaske für das Anlegen und Entfernen von Agenten-Attributen.
    - Die hiermit erzeugbare Liste an Agenten-Attributen ist relevant für andere Klassen. Diese umfassen Agenten-Gruppen, Agent-Attribut-Änderer und deren Unterpunkte.

- Agenten Gruppen
  - `SelectedNewAgentGroupController`
    - Erstellt die Eingabemaske für das Anlegen einer neuen Agenten-Gruppe
    - Übergibt die von Nutzer eingegebenen Daten an die Konfiguration. Diese erzeugt eine neues Agent Gruppen Objekt.
    - Beinhaltet eine Vorvalidierung der Nutzereingaben. Bei invaliden Eingaben wird eine Fehlermeldung ausgeben und entsprechende Textfeld markiert damit der Benutzer die fehlerhafte Eingabe leicht korrigieren kann.
  - `SelectedAgentGroupController`
    - Stellt die vorhandenen Eigenschaften einer Agenten Gruppe dar und bietet die Möglichkeit diese zu abzuändern.

- Attribute Changer
  - `SelectedNewAttributeChangerController`

        * Erstellt die Eingabemaske für das Anlegen eines neuen Agenten-Attribut-Änderers.
        * Übergibt die von Nutzer eingegebenen Daten an die Konfiguration. Diese erzeugt eine neues Attribut Changer Objekt.
        * Beinhaltet eine Vorvalidierung der Nutzereingaben. Bei invaliden Eingaben wird eine Fehlermeldung ausgeben und entsprechende Textfeld markiert damit der Benutzer die fehlerhafte Eingabe leicht korrigieren kann.
    - `SelectedAttributeChangerController`
        * Stellt die vorhandenen Regeln zur Änderung von Attribute vom ausgewählten Changer Objekt dar und erlaubt die Änderung dieser.

- Optimale Produkte
    - `SelectedNewOptimalProductController`
        * Erstellt die Eingabemaske für das Anlegen eines neuen optimalen Produkts.
        * Übergibt die von Nutzer eingegebenen Daten an die Konfiguration. Diese erzeugt eine neues Optimales Produkt Objekt.
        * Beinhaltet eine Vorvalidierung der Nutzereingaben. Bei invaliden Eingaben wird eine Fehlermeldung ausgeben und entsprechende Textfeld markiert damit der Benutzer die fehlerhafte Eingabe leicht korrigieren kann.
    - `SelectedOptimalProductController`
        * Stellt die vorhandenen Attribut Präferenzen eines optimalen Produktes dar.
        * Bietet die Möglichkeit neue Attribut-Präferenzen hinzufügen bzw. zu entfernen.
    - `ButtonAddOptimalProductPreferenceListener`
        * Stellt Funktionalitäten bereit welche genutzt werden wenn einem optimalen Produkt eine Attribut Präferenz hinzugefügt werden soll
        * Verarbeitet dabei die Inhalte der Eingabemaske und übergibt diese an die Konfiguration. Ist das Erstellen einer neuen Attribut Präferenz nicht möglich da die Benutzereingabe invalide ist so wird eine Fehlermeldung ausgeben und das entsprechende Text Feld markiert.
    - `ButtonRemoveOptimalProductPreferenceListener`
        * Stellt Funktionalitäten bereit welche genutzt werden wenn eine Attribut Präferenz eines Optimalen Produkts entfernt werden soll

- Produkt Attribute
    - `SelectedNewProductAttributeController`
        * Erstellt die Eingabemaske für das Anlegen und Entfernen von Produkt-Attributen.

        * Die hiermit erzeugbare Liste an Produkt-Attributen ist relevant für andere Klassen. Diese umfassen unter anderem Produkte, optimale Produkte und deren Unterpunkte.

- Produkte
  - `SelectedNewProductController`
    * Erstellt die Eingabemaske für das Anlegen eines neuen Produkts.
    * Übergibt die von Nutzer eingegebenen Daten an die Konfiguration. Diese erzeugt eine neues Produkt Objekt.
    * Beinhaltet eine Vorvalidierung der Nutzereingaben. Bei invaliden Eingaben wird eine Fehlermeldung ausgeben und entsprechende Textfeld markiert damit der Benutzer die fehlerhafte Eingabe leicht korrigieren kann.
  - `SelectedProductController`
    * Stellt die vorhandenen Eigenschaften eines Produktes dar und erlaubt diese zu ändern.

Funktionen der Controller Klassen der beiden Hauptansichten:

- `MainAppController`
  - Reagiert auf Interaktionen des Nutzers mit den festen Elementen der GUI
  - Stellt Funktionalität für das Laden und Speichern einer Konfigurationsdatei mit Hilfe des systemspezifischen Dateienmanager bereit
  - Leitet den Befehl für das Starten der Simulation weiter und gibt bei invalider Konfiguration eine Fehlermeldung aus
- `DataViewController`
  - Darstellung der Ergebnisse der Simulation als Graph
  - Reagiert auf Nutzereingaben durch die die Darstellungsweise des Ergebnisgraphen angepasst wird

Die verbleibende Klasse hat folgende Aufgabe:

- `ISelectController`
  - Interface Klasse
  - Beinhaltet nur die Funktion `selected()`
  - Wird von allen Controllern extended deren Name mit „Selected“beginnt

4.2 Paket API

Dieses Paket bietet die Schnittstelle zwischen dem Benutzer und unserem Produkt an. Jeder Anwender nutzt dieses Paket entweder direkt oder vermittelt durch die GUI.

In diesem Paket besteht nur wenig Programmlogik. Stattdessen werden alle wichtigen Funktionalitäten der Anwendung hierdurch nach außen angeboten. Dies beinhaltet insbesondere das Konfigurieren und Starten der Simulation, ebenso wie die Abfrage der Daten.

4.2.1 Angebotene Schnittstellen

Das Paket `API` bietet kein Interface an.

4.2.2 Paketstruktur

Das Paket Api biete also alle für den Nutzer relevanten Funktionen an. Dies umfasst die Konfiguration der Daten mit den Funktionen des Interfaces `IConfiguration` und andererseits die Abfrage von Ergebnisdaten über das Interface `IResultOutput`.

4.3 Paket Configuration

Dieses Paket ist für die Konfiguration der Simulation verantwortlich. Dafür sind Daten erforderlich, die der Simulation zum Start übergeben werden. Diese Daten konfigurieren somit die Simulation. Um dies zu erreichen, wurde eine Datenstruktur mit Integritätsbedingungen definiert. Diese kann mithilfe der Konfiguration erstellt, verändert, verifiziert, geladen und gespeichert werden. Die Datenstruktur besteht aus Folgenden Teilen:

Agentenattribut
: Ein Agentenattribut ist eine Eigenschaft eines Agenten. Es besteht aus einem einzigartigen Namen. Dieser wird durch eine Verteilungsfunktion zum Anfang der Simulation auf einen festen Zahlenwert gemappt, der aber zur Laufzeit der Simulation veränderlich ist. Beispiel: Alter : 12

Produktattribut
: Ein Produktattribut ist eine Eigenschaft eines Produktes. Es besteht aus einem einzigartigen Namen. Dieser wird durch ein Produkt ein fester Zahlenwert zugewiesen, der über die Simulation hinweg konstant bleibt. Beispiel: KW/h : 42

Präferenzattribut
: Ein Präferenzattribut beschreibt die Präferenz eines Agenten, bezogen auf ein Produktattribut. Es besteht aus einem einzigartigen Namen, einem Zahlenbereich angegeben mit Minimal-, Optimal- und Maximalwert. Darüber hinaus wird ein Gewicht angegeben, womit sich die Wichtigkeit dieser Präferenz gegen andere abwiegen lässt. Ein Präferenzattribut wird vor der Simulation auf ein Produktattribut zugewiesen, kaufbare Produkte müssen damit dieses Produktattribut innerhalb des angegebenen Zahlenbereiches aufweisen.

Optimales Produkt

Ein optimales Produkt ist eine Zuweisung von Produktattributen auf Präferenzattribute. Es hat einen einzigartigen Namen und ein Gewicht. Hier dient das Gewicht um ein optimales Produkt gegen andere abzuwägen. Optimale Produkte werden bei Agentengruppen eingetragen. Es ist möglich, Optimale Produkte miteinander zu kombinieren. Die enthaltenen Präferenzen nähern sich dabei aneinander an. Welches der Ausgangsprodukte einen höheren Einfluss auf das kombinierte Produkt hat, lässt sich über einen Prozentwert einstellen.

Produkt

Ein Produkt ist ein Zusammenschluss vieler Produktattribute. Identifiziert wird es über einen einzigartigen Namen. Unter diesem wird die Zuweisung der Produktattribute auf Zahlenwerte gespeichert.

Agentengruppen

Agenten werden in Gruppen zusammengefasst. Zu einer Agentengruppe gehören Optimale Produkte, Agentenattribute und deren Zuweisung auf eine Verteilungsfunktion und eine Angabe über die Anzahl der Agenten, die aus dieser Agentengruppe erstellt werden sollen. Agentengruppen haben einen einzigartigen Namen.

Attributveränderer

Ein Attributveränderer (AgentAttributeChanger) besteht aus drei Teilen: Eine Auslösebedingung, eine Zeitangebe und eine Liste an Regeln. Eine Regel besteht aus einem Agentenattribut und einem mathematischen Ausdruck. Die Auslösebedingung ist ein mathematischer Ausdruck, der sich zu wahr oder falsch evaluieren lässt. Die Zeitangabe gibt an, wie oft die Liste an Regeln evaluiert wird, wenn die Auslösebedingung zu wahr ausgewertet wird. Eine Regel weist dann ihrem Agentenattribut den Wert des zugewiesenen und evaluierten mathematishen Ausdruck zu. In diesem Ausdruck kann auch auf andere Agentenattribute referenziert werden. Domit ist es möglich, einen neuen Wert für ein Agentenattribut anhand von anderen Agentenattributen zu erstellen.

Folgende Integritätsbedingungen sind zu erfüllen:

- Jeder Entitätname ist innerhalb seiner Klasse einzigartig, von jeder Entität muss es mindestens eine Instanz geben.
- Für ein Präferenzattribut muss gelten: Minimalwert kleiner gleich Optimalwert kleiner gleich Maximalwert. Das Gewicht darf 1 nicht unterschreiten.
- Jedes Produkt muss für alle der Konfiguration hinzugefügten Produktattribute einen Wert hinterlegen.
- Jedes optimale Produkt muss für alle Produktattribute ein Präferenzattibut hinterlegen. Das Gewicht des optimalen Produktes darf 1 nicht unterschreiten.
- Jede Agentengruppe muss für alle Agentenattribute eine Verteilungsfunktion hinterlegen. Es muss mindestens ein optimales Produkt hinterlegt werden. Die Anzahl der zu generierenden Agenten muss mindestens 1 betragen.

4.3.1 SCHNITTSTELLEN

Das angebotene Interface, IConfiguration, gliedert sich folgendermaßen:

add... Methoden
Hiermit lassen sich die einzelnen Entitäten der Konfiguration hinzufügen. Ihre Einzigartigkeit wird überprüft.

remove... Methoden
Hiermit lassen sich die einzelnen Entitäten aus der Konfiguration entfernen. Jede Zuweisung von oder auf die Entität wird dabei mit gelöscht.

map... Methoden
Hiermit lassen sich die einzelnen Entitäten miteinander, mit Zahlenwerten oder Verteilungsfunktionen in Beziehung setzen.

demap... Methoden
Hiermit lässt sich eine bestimmte Beziehung entfernen.

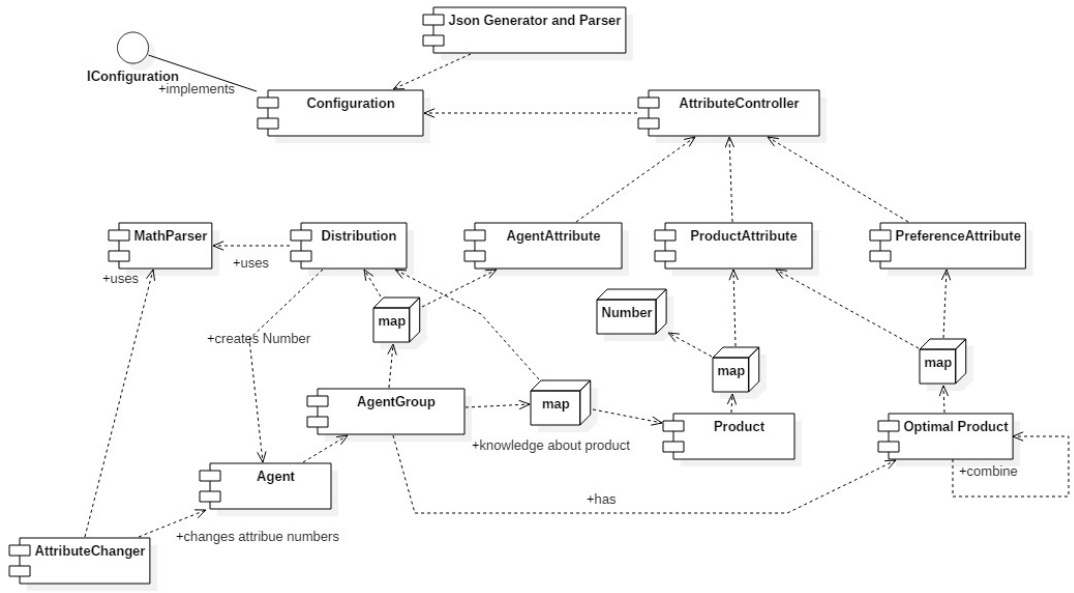
load... Methoden
Hiermit lässt sich eine Konfiguration als Teil oder als Ganzes laden, oder der bestehenden hinzufügen.

save... Methoden
Hiermit lässt sich eine Konfiguration als Teil oder als Ganzes speichern.

verify... Mathoden
Hiermit werden die Integritätsbedingungen überprüft und gegebenenfalls Agenten erzeugt, die der Simulation übergeben werden.

badConfigEntry
Hiermit wird überprüft, welche Teile der Konfiguration den Integritätsbedingungen noch nicht genügen. Zurückgegeben wird eine Liste von Strings. Jeder String repräsentiert eine Verletzung in Form einer menschenlesbaren Nachricht.

4.3.2 Paketstruktur



Json Parser

Die verschiedenen Entitäten (Attributtypen, Produkte, Optimalproducts, Agentengruppen, Attributveränderer) können einzeln gespeichert oder geladen werden. Das wird durch eigene Serializer/ Deserializer erreicht. Diesen wird eine entsprechende Attributliste übergeben. Es ist möglich, Konfigurationsteile oder eine ganze Konfiguration zu einer bestehenden hinzuzufügen. Kommen Entitäten sowohl in der zu ladenden als auch in der vorgehaltenen Konfiguration vor, wird die betreffende Entität nur geladen, wenn diese isomorph zu der vorgehaltenen ist. Aus dem Deserializer lässt sich nach dem Laden ein Status auslesen. Dieser besagt, wie viele Instanzen der jeweiligen Klassen geladen wurde.

Distribution

Jeder mathematische Ausdruck lässt sich einlesen. Verschachtelung ist möglich. Folgende Funktionen sind implementiert:

- Normalverteilung. : normal(median,standardDeviation)
- Gleichverteilung : uniformreal(min,max)
- Exponentialverteilung : exponential(median)
- Und-Verknüpfung : and(a,b)
- Oder-Verknüpfung : or(a,b)
- Negation : not(a)
- Mit $a, b \in \{-1, 1\}$

4.4 Paket simulation

Dieses Paket führt die Berechnung der Simulation und das Speichern von Ergebnissen dieser durch.

Damit implementiert es alle Funktionen der Gruppe Simulation.

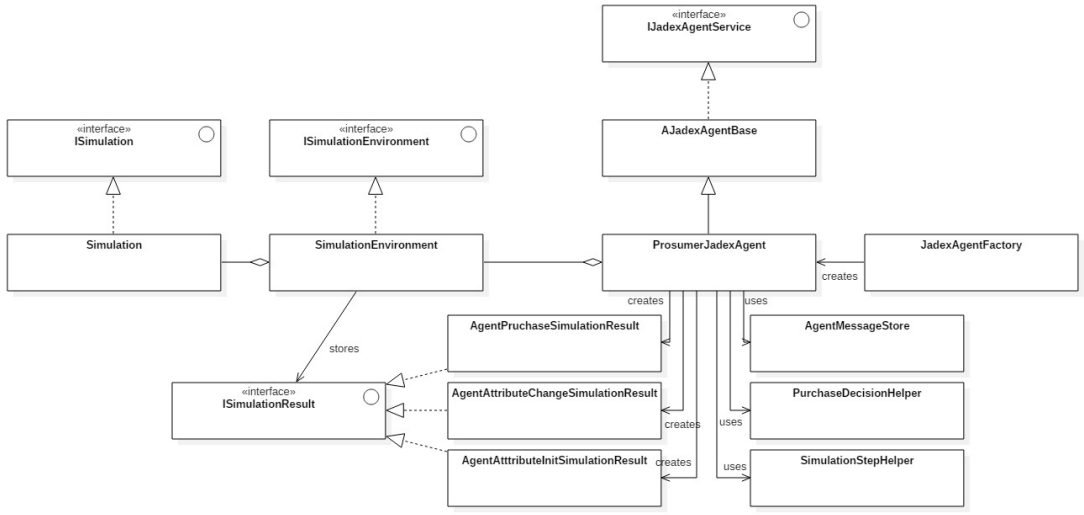
Dieses Paket verwendet als einziges das Framework Jadex, um Agenten programmlogisch proaktiv handeln zu lassen.

4.4.1 Angebotene Schnittstellen

Das Paket `simulation` bietet die Schnittstelle `ISimulation` nach außen an. Diese Schnittstelle erlaubt es dem Paket `configuration` erstellte Agenten, Produkte und AgentAttributeChanger zur Simulation hinzuzufügen, sowie die Simulationsdauer einzustellen.

Diese Schnittstelle erlaubt dem Paket `api`, die Simulation zu starten und abzufragen, ob die Simulation bereits beendet wurde.

4.4.2 Paketstruktur



Die Klasse `Simulation` mit dem Interface `ISimulation` implementiert alle wichtigen Funktionalitäten des Pakets nach außen.

Das Klasse `SimulationEnvironment` mit dem Interface `ISimulationEnvironment` implementiert eine Helferklasse zur Überwachung der Simulation. Sie überwacht Ergebnisse der Agenten und speichert sie über ein `IResultInput` Interface (s. Paket `store`). Dazu überwacht diese Klasse den Fortschritt der Agenten.

Die Klassen `ProsumerJadexAgent`, `JadexAgentBase` und das Interface `IJadexAgentService` implementieren proaktiv handelnde Jadex-Agenten. `JadexAgentBase` und `IJadexAgentService` werden dabei für die Einbindung des Frameworks Jadex benötigt und sollen hier nicht weiter beleuchtet werden. `ProsumerJadexAgent` führt die eigentliche Simulation von Prosumenten-Agenten durch.

Dazu führt er Schritte aus, die von ihm selbst verwaltet werden. In jedem dieser Schritte führt er folgende Aktivitäten durch:

- Er führt eine Kaufentscheidung durch (nichts zu kaufen, ist auch eine Kaufentscheidung)
- Er behandelt eingehende Nachrichten
- Er sendet Nachrichten
- Er verwaltet Simulationsergebnisse

Zur Verwaltung der Simulationsergebnisse erhält `SimulationEnvironment` bei Start des Agenten eine Instanz der Klasse `IntermediateFuture`. Diese Klasse wird von dem Framework Jadex angeboten und kann als Stream verstanden werden. Der Agent kann seine Simulationsergebnisse `AgentAttributeInitSimulationResult`, `AgentAttributeChangeSimulationResult` und `AgentPurchaseSimulationResult` in diesen *Stream* schreiben. Diese Ergebnisse implementieren alle das Interface `ISimulationResult` und können darüber von `SimulationEnvironment` in `store` gespeichert werden.

Um diese Schritte durchzuführen, bedient sich die Klasse `ProsumerJadexAgent` dreier *Helper-Klassen*. Diese Klassen sind:

- `AgentMessageHelper` - Verwaltet eingehende Nachrichten.
- `SimulationStepHelper` - Verwaltet Daten von Simulationsschritten und geht wenn notwendig in diesen zurück.
- `PurchaseDecisionHelper` - Verwaltet Bedürfnisse in Form der Klasse `OptimalProduct` und berechnet Kaufentscheidungen durch Objekte der Klasse `Product`.

Die Klasse `SimulationStepHelper` hält mehre Objekte der Klasse `SimulationStep`. Jedes dieser Objekte repräsentiert einen Schritt in der Simulation und hält Nachrichten, Bedürfnisse und Vorstellungen von Produkten zu diesem Simulationszeitpunkt. Dies ist notwendig, weil eventuell später eintreffende Nachrichten, die sich auf einen früheren Zeitpunkt beziehen, dazu führen, dass in der Simulation zurückgegangen werden muss.

Die Klasse `PurchaseDecisionHelper` verwaltet die eigentliche Kaufentscheidung. Zu einem Simulationsschritt werden die noch aktuellen Bedürfnisse des Agenten auf die Produkt-Vorstellung des Agenten abgebildet und das jeweilige Optimum - falls vorhanden - ausgewählt.

4.5 Paket Store

Dieses Paket ermöglicht die Speicherung von Kennzahlen während der Simulation und kann diese wiedergeben sowie speichern.

Diese Kennzahlen umfassen Kaufentscheidungen für Produkten durch Agenten bestimmter Gruppen sowie die Änderungen von Agenteneigenschaften.

Es implementiert damit die Funktionen der Ausgabefunktionsgruppe.

4.5.1 Angebotene Schnittstellen

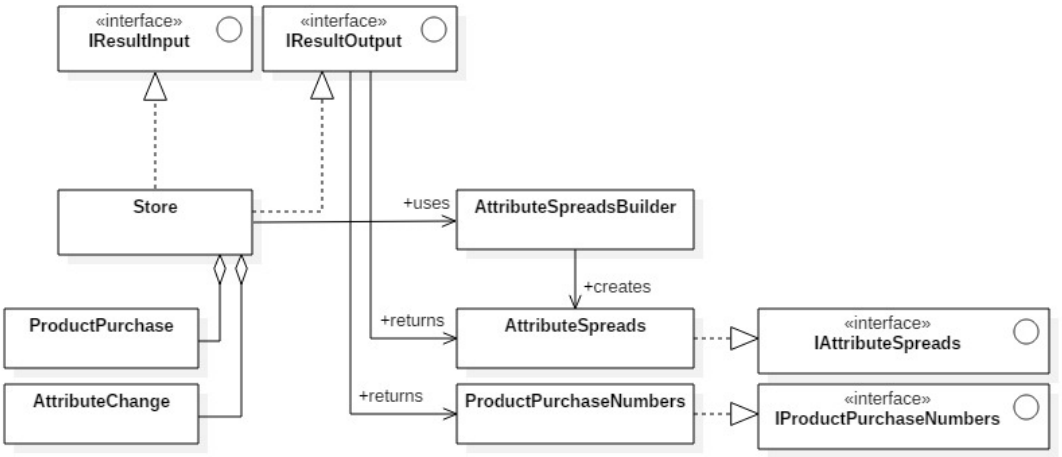
Das Paket `store` bietet die zwei Schnittstellen `IResultInput` und `IResultOutput` an.

`IResultInput` nimmt zwei Formen von Daten an. Einerseits mit Zeitstempel versehene Kaufentscheidungen und andererseits Agentenattribute. Für Agentenattribute können initiale Werte angelegt sowie Änderungen mit Zeitstempel eingetragen werden.

`IResultOutput` kann Daten (Kaufentscheidungen und Agentenattribute) in Zeitintervallen zurückgeben. Diese Zeitintervalle sind frei wählbar. Aus den eingegebenen Daten können dynamisch alle Werte zu beliebigen Zeitpunkten berechnet werden.

Zu allen Abfragemethoden gibt es eine äquivalente Methode zum Speichern einer JSON-Datei.

4.5.2 Paketstruktur



`Store` ist implementiert die Schnittstellen des Paketes. Es Speichert sowohl `ProductPurchase`- als auch `AttributeChange`-HashMaps. Diese sorgen für eine Komplexität von $O(1)$ zum Speichern und eine Komplexität von $O(n)$ zum Generieren von Ausgabedaten, wobei n die Anzahl der gespeicherten Werte ist.

Eingabe Kaufentscheidungen, Attributwerte und Attributveränderungen werden mit Referenzen auf Objekte der Klassen `AgentGroup`, `Product` und `AgentAttribute` gespeichert.

Da Jadex alle übergebenen Objekte serialisiert, werden der Speicherung nicht die ursprünglichen Objekte sondern Kopien übergeben. Der Store muss daher mithilfe der eindeutigen Namen die Originalobjekte ermitteln und anhand dieser die gewonnen Daten referenzieren.

Ausgabe Mit dem Interface `IAttributeSpreads` wird ein Objekt beschrieben, welches für eine Gruppe aus Agenten in einem Zeitintervall mit festgelegten Schritten jeweils Minimum, Maximum und Durchschnitt eines Attributes beschreibt. Um ein Objekt zu erzeugen, welches das Interface implementiert wird die Klasse `AttributeSpreadsBuilder` genutzt.

Das Interface `IProductPurchaseNumbers` beschreibt ein Objekt, welches die Anzahl getroffener Kaufentscheidungen einer Agentengruppe für ein bestimmtes Produkt während eines Zeitintervalls mit fester Schrittgröße widerspiegelt.

5 Datennmodell

Unser Multiagentensystem wird bei der Speicherung aller Daten auf Datenbanksysteme verzichten. Dafür gibt es drei Gründe:

- Ein Datenbanksystem würde eine eigene Anwendung zur Ausgabe der Daten erfordern
- Ein Datenbanksystem würde während der Laufzeit eine eigene Schnittstelle erfordern
- Daten in einem Datenbanksystem lassen sich nicht ohne weiteres durch den Endnutzer modifizieren und sind im hohen Grade von den Anwendungsprogrammen abhängig

Wir wählen - je nach Szenario - verschiedene *sprechende* Datentypen. Sie erlauben dem Nutzer auch ohne spezielle Anwendungsprogramme, Aufbau und Ergebnis der Simulation nachzuvollziehen oder zu bearbeiten.

5.1 Eingabedaten

Konfigurationsdateien werden über JSON-Dateien gespeichert. Dieses Format erlaubt Bearbeitung durch den Nutzer, z. B. in Texteditoren. Mit entsprechendem Vorwissen ist es somit möglich, manuell Konfigurationen zu erstellen oder zu modifizieren.

5.2 Ausgabedaten

Ausgabedaten werden in Form von Bilddateien oder CSV-Dateien ausgegeben. Sie sind im hohen Maße anwendungs- und systemunabhängig und erlauben es auch unerfahrenen Benutzern, Ergebnisse nachzuvollziehen und weiter zu verarbeiten.

6 Mathematisches Modell

6.1 Einleitung

Dieser Abschnitt soll einen Überblick über die Modellierung eines Agentensystem für Diffusionsprozesse geben. Die Modellierung wird in drei Abschnitte unterteilt:

- Agenten- & Produktmodellierung
- Entscheidungsprozessmodellierung
- Kommunikationsmodellierung

6.2 Agenten- & Produktmodellierung

6.2.1 Agentenmodellierung

Jedem Agenten sei ein eindeutiger Index $i \in \mathbb{N}$ zugewiesen. Sei A die Menge aller Indizes der Agenten.

Agenten-Eigenschaften (ab jetzt kurz: A-Eigenschaften) beschreiben Eigenschaften, die ein Agent besitzt. Jede dieser A-Eigenschaften wird durch einen numerischen Wert dargestellt, z.B. Alter. Jeder A-Eigenschaft sei ein eindeutiger Index $i_e \in \mathbb{N}$ zugewiesen. Sei E_A die Menge aller A-Eigenschaften.

Zu jedem Agent wird die Ausprägung der jeweiligen A-Eigenschaft in der Matrix

$$AE : A \times E_A \to \mathbb{R}, (i_A, i_{E_A}) \mapsto A - Eigenschaft \tag{6.1}$$

gespeichert.

6.2.2 Produktmodellierung

Jedem Produkt sei ein eindeutiger Index $i_P \in \mathbb{N}$ zugewiesen. Sei P die Menge aller Indizes der Produkte.

Produkt-Eigenschaften (ab jetzt kurz: P-Eigenschaften) beschreiben die Ausprägung einer Eigenschaft eines Produktes in einem Zahlenwert, z.B. Preis. Jeder P-Eigenschaft sei ein eindeutiger Index $i_{E_P} \in \mathbb{N}$ zugewiesen. Sei E_P die Menge aller P-Eigenschaften.

Zu jedem Produkt wird die Ausprägung der jeweiligen P-Eigenschaft in der Matrix

$$PE : P \times E_P \to \mathbb{R}, (i_P, i_{E_P}) \mapsto P - Eigenschaft \tag{6.2}$$

gespeichert.

Das Wissen, das ein Kunde über Produkte hat, wird in einer Matrix gespeichert, diese heißt Bekanntheitsmatrix. Der Bekanntheitswert stellt dar, wie akkurat das Wissen eines Agenten über ein Produkt ist. Die Matrix ist definiert durch:

$$APB : A \times P \to [-1, 1], (i_A, i_P) \mapsto Bekanntheit \tag{6.3}$$

Dabei hat der Wert des Wissens über ein Produkt folgende Bedeutung: Wissen ist immer relativ zu allen Bedürfnissen eines Agenten. Ist der Wert auf -1, dann bedeutet dies, dass der Agent das Produkt für das schlechteste hält. Er wird es niemals kaufen. Ist das Wissen auf 0 schätzt der Agent das Produkt genau so ein, wie es tatsächlich ist. Ist das Wissen auf 1 hält der Agent das Produkt in allen Belangen für optimal. Er wird das Produkt auf jeden Fall kaufen.

6.3 Entscheidungsprozessmodellierung

6.3.1 Wunschausprägung

Eine Wunschausprägung ist ein Tripel (min, opt, max) mit dem Minimum min, dem Optimum opt und dem Maximum max, wobei gilt $min \leq opt \leq max$.

Diese stellen dar, in welchem Rahmen sich die Ausprägungen der Eigenschaften eines Produktes sein sollten.

6.3.2 Bedürfnisse

Ein Bedürfnis ist ein Tripel (b, g, w).

Die Bedürfnisfunktion b weist P-Eigenschaften Wunschausprägungen zu.

$$b : E_P \to \mathbb{R} \times \mathbb{R} \times \mathbb{R}, i_{E_P} \mapsto (min, opt, max) \tag{6.4}$$

Die Gewichtungsfunktion g weist P-Eigenschaften die Gewichtung dieser zu.

$$g : E_P \to \mathbb{R}^+, i_{E_P} \mapsto P - Eigenschaft - Gewichtung \tag{6.5}$$

Die Wichtigkeit w stellt dar wie wichtig ein Bedürfnis ist. Es gilt $w \in [1, \infty]$. Je höher der Wert, desto wichtiger ist dem Agenten die Erfüllung dieses Bedürfnisses und desto eher ist er bereit, Kompromisse einzugehen.

6.3.3 Kaufentscheidung

Bevor Produkte evaluiert werden, muss aufgezeigt werden, wie Produktwissen jeweilige Produkt beeinflusst.

Sei $a \in A$ ein Agent, sei $p \in P$ ein Produkt. Definiere die Funktion bpe, die eine Produkt-Eigenschaft E_P über Wissen in Hinblick auf ein Bedürfnis ausgibt. Sei $pe = PE(p, E_P)$ der Wert der Produkt-Eigenschaft, sei $pb = APB(a, p)$.

$$bpe : \mathbb{R} \times \mathbb{R} \times (\mathbb{R}, \mathbb{R}, \mathbb{R}) \mapsto \mathbb{R}, (pe, pb, (min, opt, max)) \mapsto \begin{cases} pe + |opt - pe| pb, & pb \geq 0 \\ pe + |max - pe| pb, & pb < 0 \wedge pe >= opt \\ pe + |min - pe| pb, & pb < 0 \wedge pe < opt \end{cases} \tag{6.6}$$

Um den Nutzen eines Produktes zu berechnen ist zunächst die eine generische Erfüllungsfunktion e notwendig. Diese bestimmt wie sehr eine Produkteigenschaftausprägung eine Wunschausprägung erfüllt.

$$e : \mathbb{R} \times (\mathbb{R} \times \mathbb{R} \times \mathbb{R}) \to [0, 1], (x, (min, opt, max)) \mapsto \begin{cases} 0, & x < min \vee x > max \\ 1, & x = opt \\ \frac{x - min}{opt - min}, & x < opt \\ \frac{max - x}{max - opt}, & x > opt \end{cases} \tag{6.7}$$

Für in eine Richtung unbeschränkte Bedürfnisse werden hier zwei Sonderfälle eingeführt:

$$e : \mathbb{R} \times (\mathbb{R} \times \mathbb{R} \times \mathbb{R}) \to [0,1], (x,(min,opt,max)) \mapsto \begin{cases} 1 - \frac{1}{e^{\frac{ln(\frac{1}{0.05})}{opt-min}(x-min)}}, & min,opt \in \mathbb{R} \wedge max = \infty \\ 1 - \frac{1}{e^{\frac{ln(\frac{1}{0.05})}{max-opt}(max-x)}}, & max,opt \in \mathbb{R} \wedge min = -\infty \end{cases} \tag{6.8}$$

Der Wert v eines Produktes $p \in P$ hinsichtlich des Bedürfnisses $(b,g,w) \in B_{i_A}$ für einen Agenten $a \in A$ berechnet sich durch folgende Formel:

$$v_{a,p,(b,g,w)} = \frac{\sum_{i \in E_P} e(bpe(PE(p,i), APB(a,p), b(i)), b(i)) \cdot g(i)}{\sum_{i \in E_P} g(i)} \tag{6.9}$$

Anschaulich gesprochen ergibt sich der Wert v daraus, dass der Agent jedes Wunschprodukt-Attribut ((min,opt,max) Tripel) gegen das Attribut des aktuellen Produkts abschätzt. Dieses Produktattribut wurde von dem Wissen des Agenten beeinflusst. Aus diesen abgeschätzten Produktattributen, wird der gewichtete Mittelwert gebildet. Dies ist der Wert v des Produkts in Hinblick auf das Bedürfnis.

Um einen Kauf auszuführen muss entschieden werden ob ein Kauf eines Produktes $p \in P$ hinsichtlich des Bedürfnisses $(b,g,w) \in B_{i_A}$ für einen Agenten $a \in A$ sinnvoll ist. Diese Entscheidung berechnet sich folgendermaßen:

$$k_{a,p,(b,g,w)} = \begin{cases} 1, & \frac{1}{w} < v_{a,p,(b,g,w)} \\ 0, & sonst \end{cases} \tag{6.10}$$

Dieser berechnete Kauf sagt noch nichts darüber aus ob ein Kauf stattfindet, sondern lediglich ob dieser sinnvoll wäre.

Gekauft wird das Produkt, $p \in P$ hinsichtlich des Bedürfnisses $(b,g,w) \in B_{i_A}$ für einen Agenten $a \in A$, für das gilt $k_{a,p,(b,g,w)} = 1$ und $\forall p' \in P : k_{a,p',(b,g,w)} = 1 \implies v_{a,p',(b,g,w)} \leq v_{a,p,(b,g,w)}$.

6.4 Kommunkationsmodellierung

Agenten kommunizieren zwei Arten von Inhalten. Die Kommunikation dieser Inhalte geschieht analog. Zunächst werden wir betrachten, wie Agenten entscheiden und welche Inhalte sie senden.

Danach wird behandelt, wie Inhalte von Kommunikation Agenten beeinflussen.

Das Kommunizieren eines Inhalts nennen wir im folgenden das Senden einer Nachricht. Das Senden einer Nachricht ist dabei der atomare Handlungsschritt, den ein Agent beim Kommunizieren unternimmt.

In einem sog. *Kommunikationsschritt* unternimmt ein Agent mehrerer solcher atomaren Handlungsschritte.

Die Formalisierung der Kommunikation erfolgt ausschließlich exemplarisch für einen Agenten.

6.4.1 Nachrichteninhalte

Die Inhalte, die ein Agent kommuniziert, können folgender Natur sein:

- Ein Agent sendet eines seiner Bedürfnisse
- Ein Agent sendet Wissen über ein Produkt

Wieder sei $A \subset \mathbb{N}$ die Menge aller Agenten. Die Kommunikation, die ein Agent unternimmt, wird durch mehrere Kennziffern bestimmt:

- Sei $N_B \in \mathbb{N}$ die Anzahl aller Bedürfnis-Nachrichten, die ein Agent im Mittel versendet
- Sei $N_W \in \mathbb{N}$ die Anzahl aller Wissens-Nachrichten, die ein Agent im Mittel versendet
- Sei $K \in \mathbb{N}$ die Anzahl aller Agenten, die ein Agent im Mittel kennt

Ein Kommunikationsschritt läuft nun wiefolgt ab:

Bestimmung der Kennwerte Sei $P : \mathbb{R} \mapsto \mathbb{N}$ eine pseudo-Funktion. Diese gebe Zufallszahlen aus, die nach der Poisson-Verteilung verteilt sind. So gibt z. B. $P(10)$ einen Zufallswert einer mit $\lambda = 10$ verteilten Poisson-Verteilung aus.

Auf Basis der bereits beschriebenen Kennziffern werden nun für den Kommunikationsschritt charakteristische Kennwerte ermittelt.

- Sei $n_B = P(N_B)$ die Anzahl der Bedürfnis-Nachrichten, die ein Agent in diesem Kommunikationsschritt sendet.
- Sei $n_W = P(N_W)$ die Anzahl der Wissens-Nachrichten, die ein Agent in diesem Kommunikationsschritt sendet.
- Sei $k = P(K)$ die Anzahl der Agenten, die dieser Agent in diesem Kommunikationsschritt kennt.

Aus dem letzten Kommunikationsschritt sei eine Menge $AK' \subseteq A$ mit $|AK'| = min\{k', |A|\}$ gegeben, die die Menge aller Agenten enthalte, die der Agent im letzten Kommunikationsschritt kannte. k' sei dabei der k entsprechende Kennwert des letzten Kommunikationsschritts. Existiert kein solcher Schritt, dann sei $AK' = \emptyset$.

Für diesen Kommunikationsschritt sei nun $AK \subseteq A$ als Menge der Agenten, mit denen der kommunizierende Agent kommunizieren kann definiert als als eine zufällige Teilmenge aus AK', wenn $k \leq k'$ oder einer um zufällig aus A gewählten Teilmenge ak, sodass $AK = AK' \cup ak$ mit $|AK| = mink, |A|$.

Anschaulich gesprochen wächst oder schrumpft die Menge der Agenten, die ein Agent kennt (und mit denen er kommunizieren kann) auf Basis der Menge an Agenten, die er in der letzten Runde gekannt hat.

Weiterhin sei P die Menge aller Produkte, die dem Agent in diesem Kommunikationsschritt zur Verfügung stehen.

Sei B die Menge aller in diesem Kommunikationsschritt noch unbefriedigten Bedürfnisse des Agenten.

Ein Agent sendet in jeden Kommunikationsschritt $n_B + n_W$ Nachrichten. Deren Empfänger und Inhalte werden durch folgende Zufallsexperimente bestimmte.

BESTIMMUNG DER BEDÜRFNISNACHRICHTEN Bedürfnisnachrichten werden über folgendes Zufallsexperiment versendet:

for n_B *-Mal* **do**
  a := zieheZufälligAus(AK);
  b := zieheZufälligAus(B);
  sende Bedürfnis b and Agent a;
end

BESTIMMUNG DER WISSENSNACHRICHTEN Wissensnachichten werden über folgendes Zufallsexperiment versendet, dabei sei a' der sendende Agent:

for n_W *-Mal* **do**
  a := zieheZufälligAus(AK);
  p := zieheZufälligAus(P);
  sende Wissen $PI(a', p)$ and Agent a;
end

7 GLOSSAR

Modell "Modelle sind stets Modelle von etwas, nämlich Abbildungen, Repräsentationen natürlicher oder künstlicher Originale, die selbst wieder Modelle sein können.

[...]

Modelle erfassen im allgemeinen nicht alle Attribute des durch sie repräsentierten Originals, sondern nur solche, die den jeweiligen Modellerschaffern und/oder Modellbenutzern relevant scheinen."[3]

Agent "An intelligent agent is a computer system that is capable of flexible autonomous action in order to meet its design objectives. By flexible we mean that the system must be: responsive [,] proactive [and] social." [4]

Ein intelligenter Agent ist ein Computersystem, das zu flexiblen, autonomen Handlungen fähig ist, um seine angestrebten Entwurfsziele zu erreichen. Mit flexibel meinen wir, dass das System ansprechbar, proaktiv und sozial ist.

Multiagentensystem "By an agent-based system, we mean one in which the key abstraction used is that of an agent." [4]

Mit einem agentenbasierten System meinen wir eins, dessen Hauptabstraktion ein Agent ist.

Simulation System, das mit ähnlichen Eigenschaften wie ein Original erstellt wurde um Reaktionen und Veränderungen des Originals zu ermitteln. [5]

Innovationsdiffusion "Diffusion is the process by which an innovation is communicated through certain channels over time among the members of a social system." [6]

Diffusion ist der Prozess, durch den eine Innovation über bestimmte Kanäle an die Mitglieder einer Gesellschaft kommuniziert wird.

Paket Pakete sind Ansammlungen von Modellelementen beliebigen Typs, mit denen das Gesamtmodell in kleinere überschaubare Einheiten gegliedert wird. [7]

GUI Graphical User Interface - Graphische Benutzeroberfläche

Konfiguration Eine Konfiguration beschreibt einen abgeschlossenen Zustand des Endprodukts, aus dem eine Simulation generiert werden kann.

8 Quellen

[1] Goll J. Methoden und Architekturen der Softwaretechnik. 2011; 725.
[2] Goll J, Dausmann M. Architektur- und Entwurfsmuster der Softwaretechnik. Mit lauffähigen Beispielen in Java 2013; 30.
[3] Herbert Stachowiak: Allgemeine Modelltheorie, 1973; 131,132.
[4] Jennings N. R, Michael J. Wooldridge M. J. Agent Technology: Foundations, Applications, and Markets, 1998; 4,5.
[5] https://de.wiktionary.org/wiki/Simulation, 11.04.2016.
[6] Rogers E. M. Diffusion of Innovations, 1995; 5.
[7] http://www.oose.de/glossar/paket-2/, 11.04.2016