

# Agentenbasierte Modellierung und Simulation in Simio

## Masterarbeit

Abteilung Informatik  
Hochschule für Technik Rapperswil

Autor:	Thomas Kehl
Examinator:	Prof. Dr. Andreas Rinkel
Experte:	Knut Schmahl
Ausführung:	HS2017 und FS2018
Abgabe:	08. August 2018

## Selbstständigkeitserklärung

Hiermit bestätige ich, dass ich die vorliegende wissenschaftliche Arbeit mit dem Titel

**«Agentenbasierte Modellierung und Simulation in Simio»**

selbstständig verfasst und keine anderen als die angegebenen Quellen benutzt habe. Sämtliche Entlehnungen sind durch Quellenangaben kenntlich gemacht.

Es wurden keine durch Copyright geschützten Materialien (wie z.B. Bilder) in dieser Arbeit in unerlaubter Weise genutzt.

Berneck, 08. August 2018

---

Thomas Kehl

## Abstract

Simio LLC entwickelt und vertreibt die gleichnamige Software für Modellbildung und Simulation. Diese beinhaltet umfangreiche Funktionalität für diskrete Ereignissimulation sowie Systemdynamik. Für agentenbasierte Modellbildung und Simulation bestehen zwar Ansätze – diese sind in ihren Möglichkeiten bzw. Benutzerfreundlichkeit jedoch sehr beschränkt.

Diese Arbeit befasst sich mit der konkreten Implementierung eines Moduls für agentenbasierte Modellbildung und Simulation (kurz ABMS) in Simio. Dies umfasst zum einen eine Objektlibrary und zum anderen einen Designer. Die Objektlibrary ermöglicht die Integration der ABMS in die diskrete Ereignissimulation (kurz DES). Der Designer dient dazu, das Agentenverhalten einfach und ohne tiefgehende Programmierkenntnisse grafisch modellieren zu können.

Als Basis für die Modellierung des Agentenverhalten wurde das Modell der Extended Finite State Machine gewählt. Die benutzerfreundliche Umsetzung auf dieser Basis erfolgt durch Implementierung eines StateChart-Designers. Dieser ermöglicht die grafische Modellierung von StateCharts für die im aktuellen Modell eingefügten Agenten. Diese wiederum werden in Simio von der ebenfalls implementierten Objektlibrary zur Verfügung gestellt. Darüber hinaus bietet der neue StateChart-Designer die Möglichkeit, Actions und Guards der States bzw. Transitions in Form von C#-Expressions zu implementieren. Dieser Technologieansatz stellt für die Implementierung der Expressions die komplette Welt des .NET-Frameworks zur Verfügung.

Komplettiert wird die Implementierung durch ein «Getting Started – Tutorial», welches die Modellierung und die anschließende Simulation eines ConsumerMarket-Modells aufzeigt.

## Typographische Konvention

In dieser Arbeit werden folgende typographische Konventionen angewendet:

- Begriffe, die sich auf Sourcecode beziehen (z.B. Klassen, Methoden): `OnEnteredFreeSpace`
- Literaturverweise bzw. Quellenverweise sind in eckigen Klammern angegeben: [7]. Ggf. wird die Seitenzahl ergänzt: [2, S.11f]. Ein anschliessendes f bezieht die nachfolgende Seite mit ein. Zwei nachfolgende ff beziehen mehrere nachfolgenden Seiten mit ein.
- Anglizismen: Begriffe, die allgemein in Englisch bekannt sind, werden i. A. beibehalten.

## Inhalt

Selbstständigkeitserklärung .....	2
Abstract .....	3
Typographische Konvention .....	4
Inhalt .....	5
1 Ausgangslage .....	7
1.1 Einleitung .....	7
1.2 Aufgabenstellung .....	7
2 Umsetzkonzept der agentenbasierten Simulation .....	10
2.1 Agentenaspekte der Problemstellung .....	10
2.2 ComplexGateway: vom Network zum Freespace .....	11
2.3 Grafische Modellierung der EFSM .....	11
2.4 Ergebnis der Studie .....	13
3 Konzeptionelle Umsetzung der agentenbasierten Simulation in Simio .....	14
3.1 AddIn: SimioAgentLibrary .....	14
3.1.1 Modellbaum .....	16
3.1.2 Bearbeitungsbereich .....	16
3.1.3 Konfigurationsbereich .....	17
3.1.4 Hauptmenü .....	19
3.1.5 Meldungen .....	21
3.2 Technische Komponentenübersicht der SimioAgentLibrary .....	21
3.3 StateChart-Designer .....	23
3.3.1 Übersicht .....	23
3.3.2 Transitions .....	25
3.3.3 Branch .....	32
3.3.4 State und Composite State .....	33
3.3.5 History State .....	36
3.3.6 Entry Point .....	37
3.3.7 Final State .....	39
3.3.8 Agent Event Actions .....	39
3.3.9 Functions .....	40
3.3.10 Agent Movement .....	41
4 Runtime-Implementierung der AgentLibrary in Simio .....	44
4.1 Implementierung der StateMachine .....	44
4.1.1 Ablauf .....	46
4.1.2 CodeProvider .....	47
4.1.3 SteeringBehavior: Agent .....	48
4.2 Implementierung ComplexGateway .....	50
4.2.1 Eigenschaften .....	51
4.2.2 Prozesse .....	51
4.3 Implementierung AgentBase .....	53
5 Installationshandbuch der SimioAgentLibrary für Enduser .....	54

6	Anwendung der AgentLibrary .....	56
6.1	API-Dokumentation .....	56
6.2	Getting Started: Modellierung und Simulation eines Consumer-Markets .....	57
6.2.1	Schritt 1: Erstellung der Agent-Population .....	57
6.2.2	Schritt 2: Consumer-Verhalten definieren .....	59
6.2.3	Schritt 3: Diagramm hinzufügen, um das Resultat zu visualisieren .....	65
6.2.4	Schritt 4: Durchführung der Simulation .....	67
6.3	Verifikation Modellierungsaspekte Problemstellung «Produktion» .....	70
7	Schlussbemerkungen .....	76
7.1	Fazit .....	76
7.2	Potential zur Weiterentwicklung .....	76
7.2.1	IntelliSense und Syntax Highlighting für die Expressions sowie Functions .....	76
7.2.2	Weitere Perceptions .....	76
7.2.3	Erweiterungen in Bezug auf Agent-Movement .....	77
7.2.4	Optimierung bzgl. Performance und Memory .....	77
7.2.5	Erweiterung der API .....	77
7.2.6	Tiefere Integration in Simio .....	77
7.3	Danksagung .....	78
8	Management Summary .....	79
8.1	Ausgangslage .....	79
8.2	Vorgehen .....	79
8.3	Ergebnis .....	79
8.4	Ausblick .....	79
9	Verzeichnisse .....	80
9.1	Literaturverzeichnis .....	80
9.2	Abbildungsverzeichnis .....	81
9.3	Tabellenverzeichnis .....	83
9.4	Codelistings .....	83
9.5	Glossar .....	84
10	Anhänge .....	86
	Anhang A: API-Dokumentation .....	86
	Anhang B: Known Issues .....	93
	Anhang B: Planung .....	94
	Anhang C: Protokolle .....	95
	C.1 19. September 2017 .....	95
	C.2 08. November 2017 .....	95
	C.3 21. März 2018 .....	96
	Anhang E: Digitaler Anhang .....	96

## 1 Ausgangslage

### Wichtiger Hinweis:

Um den Ausführungen dieser Arbeit folgen zu können, ist es unerlässlich, [2] sowie [3] inhaltlich zu kennen. Diese Arbeit knüpft an [3] an. Zudem werden häufig dort definierte Begriffe sowie Sachverhalte verwendet.

### 1.1 Einleitung

In [1] wurde ein Ansatz entwickelt, wie die agentenbasierte Modellierung und Simulation im Rahmen von Simio umgesetzt werden kann. In der Folge wurde das Bedürfnis erkannt, Konzepte für (teil-)autonome Agenten zu entwickeln. Dieses autonome Agentenverhalten bildet den Kern für den praktischen Einsatz der agentenbasierten Modellierung und Simulation. Anschliessend wurden in [2] konzeptionelle Möglichkeiten aufgezeigt, wie autonome Agenten umgesetzt werden können. Auf dieser Basis wurde in [3] ein Prototyp in Simio umgesetzt. Diese Umsetzung wiederum basiert auf den Grundlagen von [1].

### 1.2 Aufgabenstellung

In dieser Arbeit geht es nun darum, den Prototyp aus [3] so weiterzuentwickeln, um einen breiten praktischen Einsatz zu gewährleisten. Das Hauptziel ist, die Erstellung von Modellen für die agentenbasierte Simulation in Simio mit minimalem Aufwand zu ermöglichen. Zum einen soll das UI so gestaltet werden, um agentenbasierte Modelle speditiv konstruieren zu können. Weiter sollen Bausteine o.ä., die die Modellierung unterstützen, zur Verfügung gestellt werden (ähnlich wie Simio das bereits in Form von Formeln und Funktionen macht).

Die Umsetzung geschieht auf Basis einer konkreten Problemstellung aus der Produktion. Diese enthält die wichtigsten Konzepte für agentenbasierte Simulation:

- Autonomes Handeln von Agenten
- Kommunikation zwischen Agenten

Die konkrete Problemstellung wird nachfolgend umschrieben und bildet die Basis für die Umsetzung in Simio. Der Schwerpunkt besteht darin, sicherzustellen, dass diese sowie ähnliche Problemstellungen von Benutzern von Simio ohne vertiefte Programmierkenntnisse modelliert werden können. Entsprechend ist der Prototyp aus [3] dahingehend weiterzuentwickeln, damit die Agenten bzw. ihr Verhalten im Zuge der Modellierung auf einfache Art und Weise beschrieben werden können.

In Abbildung 1 ist der Prozess für die Modellierung auf einer hohen Abstraktionsebene abgebildet. Für diese Arbeit wird in einem ersten Schritt das Problem spezifiziert und weiter der Schritt «Umsetzung» fokussiert. Da der Lösungsansatz anhand des StateMachine-Konzepts bereits in [3, S.58ff] validiert und verifiziert wurde, werden diesen Punkten hier keine besondere Beachtung geschenkt. Dies insbesondere auch deswegen, dass der Fokus auf der benutzerfreundlichen Implementierung der in [3] entwickelten Konzepte in Simio liegt.



Abbildung 1: Modellierungsprozess

Folgende Hauptaspekte werden auf Basis der Problemstellung auf den Prototyp aus [3] angewendet:

- Bereiche identifizieren, die ausgebaut und/oder optimiert werden müssen (GUI, Logik usw.)
- Konzeption möglicher Lösungen
- Implementierung

## Problemstellung: Produktion

In Produktionsprozessen ist es vielfach erforderlich, dass zu bestimmten Zeitpunkten in Bezug auf die produzierten Komponenten mehr oder weniger komplexe Entscheidungen getroffen werden. Dies könnte so modelliert werden, indem die Komponenten eine gewisse Logik erhalten. Somit werden die Komponenten als Agenten modelliert. Diese können nun mit Hilfe ihrer Logik die entsprechenden Entscheidungen treffen und so den Produktionsprozess beeinflussen.

In Abbildung 2 ist ein solcher Produktionsprozess grafisch abgebildet. Der Ablauf wird in der Folge kurz beschrieben.

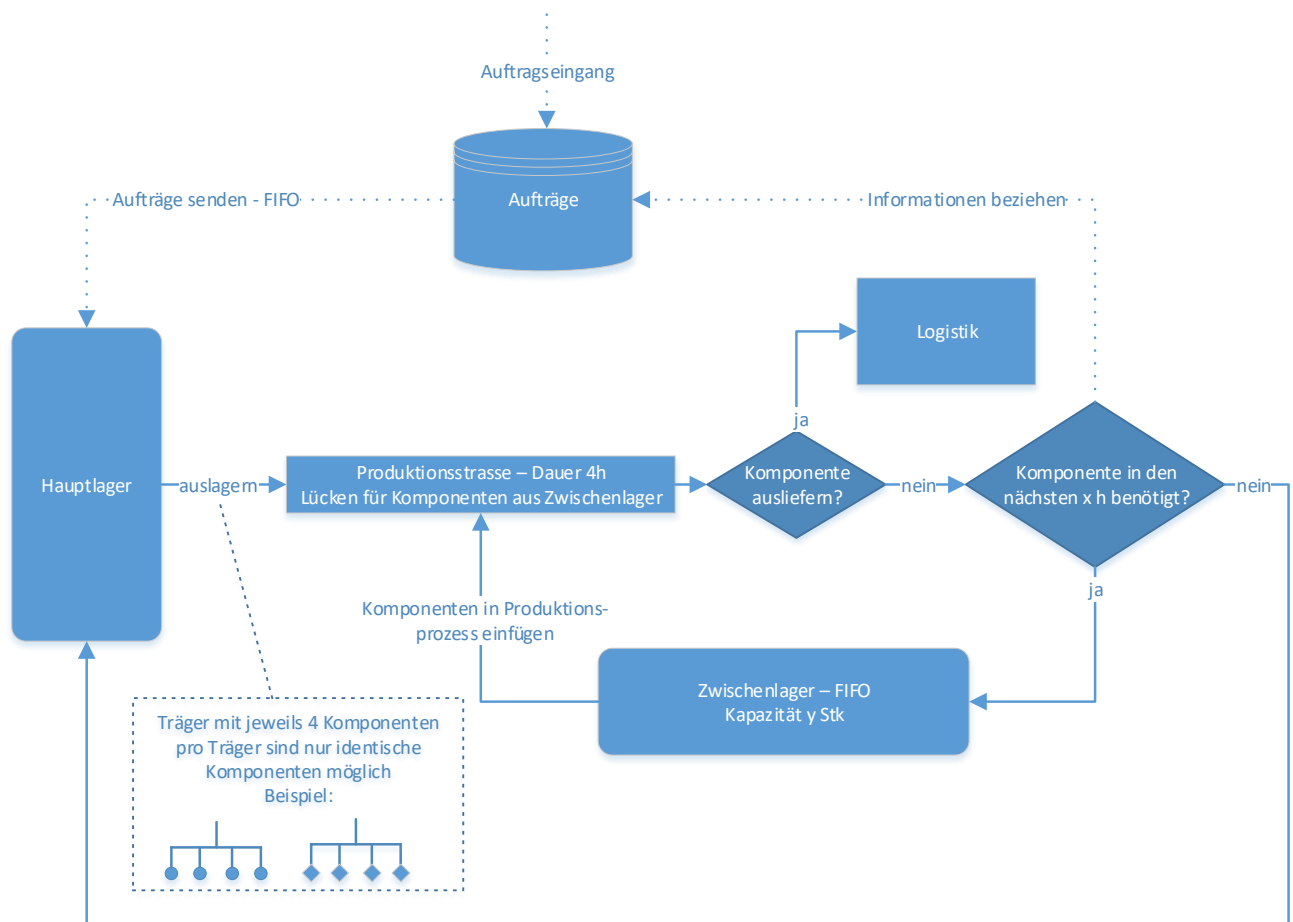


Abbildung 2: Problemstellung Produktion

Es treffen Aufträge ein, die in einer Datenbank gespeichert werden. Diese Aufträge werden mit einer FIFO-Strategie ans Hauptlager weitergegeben. Dort werden, in Bezug auf die eingetroffenen Aufträge, Komponententräger konfektioniert. Ein Träger kann nur gleiche Komponenten beinhalten. Die Träger werden immer mit 4 Komponenten bestückt. Dies auch dann, wenn der Auftrag nur ein, zwei oder drei Komponenten erfordert. Anschliessend wird der bestückte Träger an die Produktionsstrasse für die Bearbeitung weitergegeben. Nach



der Bearbeitung werden die vom entsprechenden Auftrag benötigten Komponenten ausgeliefert. Die überzähligen Komponenten werden zwischengelagert, sofern sie in einer definierten Zeitspanne benötigt werden. Ansonsten werden sie ins Hauptlager zurückgeschoben. Vom Zwischenlager werden sie entsprechend ihrem Bedarf direkt in die Produktionsstrasse eingefügt.

In diesem Modell werden die Komponenten als Agenten modelliert. Diesen fällt die Aufgabe zu, am Ende der Produktionsstrasse zu entscheiden, ob die Komponente ins Zwischenlager übergeben werden oder zurück ins Hauptlager soll. Zusätzlich ist ein Controller-Agent erforderlich, der im Hauptlager die Aufträge koordiniert.

## 2 Umsetzkonzzept der agentenbasierten Simulation

Das Konzept der Extended Finite State Machine (EFSM) gemäss [3, S.45ff] hat sich bewährt. Aus diesem Grund wird als Basis daran festgehalten. Es stellen sich nun jedoch folgende Fragen:

- Wie kann ein Benutzer in Simio möglichst einfach und komfortabel Agenten sowie deren jeweilige StateMachine modellieren?
- Wann und wie wird die StateMachine bei einer Simulation ausgeführt?

Nachfolgend werden für die Umsetzung anhand der Problemstellung die Aspekte identifiziert, die berücksichtigt werden müssen.

### 2.1 Agentenaspekte der Problemstellung

Folgende Objekte sollen als Agenten modelliert werden können:

- Kontroller
- Komponente

Der Kontroller kennt die Auftragsliste (Excel bzw. davon abgeleitete CSV-Datei). Jeder Auftrag besitzt einen Komponententyp und einen Zeitpunkt. Zum definierten Zeitpunkt generiert der Kontroller jeweils 4 Komponenten. Dabei wird zuerst gefragt, ob es solche im Zwischenlager hat. Wenn ja, werden diese verwendet. Es werden max. 4 Stück aus dem Zwischenlager verwendet. Wenn es weniger hat als 4, wird vom Kontroller mit «neuen» Komponenten auf 4 aufgefüllt. Der Träger durchläuft anschliessend die Produktionsstrasse. Kommt der Träger mit den 4 Komponenten beim «DecisionGateway» an, werden die entsprechenden Aufträge geliefert. Mit den übrigen Teilen wird nun beim Kontroller gefragt, ob es auf der Auftragsliste in die Zukunft (jetzt +2h) Verwendung hat. Wenn ja, kommen diese Komponenten ins Zwischenlager. Wenn nein, kommen sie zurück ins Hauptlager - genauer werden sie in «SnkMainStorage» terminiert und später bei Bedarf wieder neu erstellt.

Im nächsten Schritt werden die Kern-Aspekte identifiziert, die für die Umsetzung berücksichtigt werden müssen. Diese sind in Tabelle 1 aufgelistet. Zusätzlich wurde(n) die entsprechende(n) Anforderung(en) an die Umsetzung abgeleitet.

Nr.	Aspekt	Anforderung
1	Kontroller kennt Auftragsliste (Excel bzw. CSV-Datei)	Möglichkeit, Exceldateien bzw. allgemein Dateien zu lesen
2	Kontroller generiert 4 Komponenten	Agent soll andere Agenten generieren können
3	... wird gefragt, ob im Zwischenlager ...	Agent soll mit anderen Agenten kommunizieren können
4	Wenn ja, werden diese verwendet	Agent soll anderen Agenten Befehle erteilen können
5	Der Träger durchläuft die Produktionsstrasse	Agenten sollen sich auch als Entities in der diskreten, prozessorientierten Ereignissimulation bewegen können
6	... beim DecisionGateway werden ...	An bestimmten Stellen im System soll die Agentenlogik ausgeführt werden
7	Einfache und komfortable Modellierung	Grafischer StateChart-Designer

**Tabelle 1: Aspekte und deren Umsetzungsanforderung(en) aus Problemstellung**

Im Abschnitt 2.4 wird referenziert, welche Anforderungen mit welchen Elementen implementiert werden.

## 2.2 ComplexGateway: vom Network zum Freespace

In [1-3] wurden die Agenten von der Sink erstellt und direkt in den Freespace transferiert. In der Folge handelte es sich um autonome Objekte, die gemäss ihrer Logik beliebig interagieren konnten. Die Verwendung eines speziell implementierten SteeringBehavior (welches die Basis für die Ausführung der Agentenlogik bildet – siehe [3, S.26ff]) ist gemäss Vorgabe von Simio nur für Entities möglich, die sich im Freespace befinden.

Das gesamte Konzept der agentenbasierten Simulation für Simio basiert auf der Implementierung von SteeringBehaviors.

Nun besteht jedoch die Anforderung, dass Agenten sowohl autonom (d.h. im Freespace) interagieren können, als auch als Entities im Zuge einer diskreten, prozessorientierten Ereignissimulation (DES) eingesetzt werden können. So funktionieren die Agenten nicht nur als autonome Agenten, sondern auch als normale Entities mit der von Simio standardmässig zur Verfügung gestellten Funktionalität. Z.B. können sie sich auf Paths im Network bewegen, in einem Server verarbeitet werden usw. Diese kombinierte Verwendung sowohl als Entities als auch als Agenten soll in der gleichen Simulation möglich sein. D.h. die Sink erstellt Objekte vom Basistyp «AgentBase», die vorerst als normale Entities funktionieren. Bei Bedarf sollen diese jedoch als autonome Agenten interagieren können. Andererseits soll es möglich sein, autonom agierende Agenten wieder zu normal funktionierende Entities zu transformieren. Um dieses Ziel zu erreichen, wird das neue Objekte «ComplexGateway» implementiert (siehe Kapitel 4.2). Dieses Objekt hat die Aufgabe, Objekte vom Typ «AgentBase» aus der bekannten DES in den Freespace zu transferieren, damit diese als Agenten gemäss ihrer Logik autonom interagieren können (sowie vice versa). Die DES gibt jedoch weiterhin die Zyklen vor, in der die Agentenlogik ausgeführt wird. Diese Zykluslänge kann mit der Eigenschaft «TimeInterval» des SteeringBehaviors vorgegeben werden (siehe Abschnitt 4.1.3). Es ist jedoch möglich, diese Einstellung während der Simulation zu ändern (siehe Kapitel 6.1 bzw. Tabelle 26).

Nachdem der Agent vom ComplexGateway in den Freespace (Continuous Space) transferiert wurde, kann er sich frei bewegen. Detaillierte Informationen bzgl. dem Movement von Agenten im Continuous Space, sind im Abschnitt 3.3.10 zu finden. Darüber hinaus werden dort Überlegungen angestellt, wie sich Modellansätze «Discrete Space» bzw. «GIS Space» umsetzen liessen. Wird der Agent wieder ins Netzwerk transferiert, kann sein Ziel-Node angegeben werden (siehe Funktion MoveToNode in Tabelle 26).

Objekte vom Typ «ComplexGateway» sollen als normale Modellierungsobjekte in Simio zur Verfügung gestellt werden. Damit sie sich einfach in ein bestehendes Simulationsmodell bzw. in dessen Network integrieren lassen, erhalten sie einen Input- sowie einen Output-Node. Weitere Informationen zur konkreten Implementierung finden sich im Kapitel 4.2.

## 2.3 Grafische Modellierung der EFSM

Die Modellierung der EFSM stellt den Kern dar. Es muss ein Tool implementiert werden, welches diese Modellierung ermöglicht. Der zentrale Anspruch bzgl. Benutzerfreundlichkeit steht und fällt mit dieser Implementierung. Dieser Anspruch impliziert im Grunde die Umsetzung in Form einer grafischen Variante. Bevor jedoch eine komplette Bibliothek für grafisches Design von StateCharts entwickelt wird, lohnt sich eine Evaluation von möglichen bestehenden Komponenten, die evtl. zur Verwendung als Basis geeignet wären.

Da die komplette Implementierung mit C# auf der Basis von .NET erfolgt, liegt der Fokus auf Komponenten, die ebenfalls auf dieser Basis implementiert sind. In die Evaluation werden folgende Komponenten, welche die Grundvoraussetzungen erfüllen, miteinbezogen:

- 1) Diagramming for WinForms, V6.5.2, MindFusion  
<https://mindfusion.eu/flowchart-net.html>
- 2) AddFlow for WinForms, V2015, Lasalle Technologies  
<http://www.lasalle.com/products.htm#winforms>
- 3) WinForms Diagram Control, V17.2.4, DevExpress  
<https://www.devexpress.com/Products/NET/Controls/WinForms/Diagrams/>
- 4) GoDiagram, V5.2.0, Northwoods Software  
<https://www.nwoods.com/products/godiagram/index.html>

Hinweis: Die Nummerierung stellt keine Klassifizierung o.ä. dar, sondern dient lediglich als Referenz für die Evaluationsmatrix (Tabelle 2). Folgende Kriterien werden bewertet:

- **Integration C#** – die komplette Entwicklung basiert auf C#. Deswegen ist eine gute Integration unerlässlich.  
Gewichtung: 2
- **API-Dokumentation** – wie gut ist das Interface der Komponente dokumentiert?  
Gewichtung: 4
- **Support** – Wie gut ist der Support erreichbar? Wie schnell erfolgt eine Reaktion? Dies wurde mit einer «Test-Supportanfrage» geprüft.  
Gewichtung: 3
- **Usability** – Wie benutzerfreundlich gestaltet sich die Oberfläche für den Enduser? Wie komfortabel ist der Designer und was stellt dieser bereits für Standardfunktionalität zur Verfügung?  
Gewichtung: 4
- **Preis** – Wie teuer ist die Komponente?  
Gewichtung: 1
- **Update-Zyklen** – in welchen Zyklen werden Updates mit allfälligen Fehlerkorrekturen usw. zur Verfügung gestellt? Dieser Punkt beinhaltet auch die Bewertung bzgl. Aktualität der Komponente.  
Gewichtung: 2

In der Matrix werden pro Kategorie Punkte zwischen 1 (schlecht) und 10 (gut) verteilt. Die Evaluation wurde im November 2017 durchgeführt.

	Integration in C#	API-Doku	Support	Usability	Preis	Update-Zyklen Letzter Release / letztes Update
1	10	5	6	8	8 (\$600)	9 (2017/2017)
2	8	4	4	5	10 (\$399)	8 (2015/2017)
3	10	10	9	9	5 (\$999)	10 (2017/2017, jedoch mehrere)
4	6	5	8	4	3 (\$2995)	3 (2015/2015)

Tabelle 2: Evaluationsmatrix Komponenten für StateChart-Designer

1) Diagramming for WinForms, MindFusion	116
2) Addflow for WinForms, Lasalle	90
3) WinForms Diagram Control, DevExpress	148
4) GoDiagram, Northwoods	81

Tabelle 3: Gewichtetes Resultat der Evaluation

Als Sieger der Evaluation geht die Komponente «WinForms Diagram Control» von DevExpress ([4]) hervor. Dieses Tool bringt den zusätzlichen Vorteil, dass Simio auch auf den Komponenten von DevExpress basiert. So kann für den StateChart-Designer ein ähnliches Look&Feel wie in Simio erreicht werden, was die Benutzer-Thomas Kehl

freundlichkeit unterstützt. Weiter ist die DevExpress Suite im Zuge von weiteren Projektentwicklungen bereits verfügbar. Da die Lizenz pro Entwickler gilt und die Anzahl Projekte nicht einschränkt, fallen keine weiteren Kosten an.

## 2.4 Ergebnis der Studie

Zusammenfassend sind folgende Elemente zu implementieren. Dabei ist jeweils angegeben, welche Aspekte aus Tabelle 1 adressiert werden:

- **StateChart-Designer** – ermöglicht pro Agent die Modellierung seiner EFSM.  
Dies adressiert den Punkt 7.
  - Die Actions, Conditions und Guards der States bzw. Transitions sollen vom Benutzer direkt in der grafischen Oberfläche als C#-Expressions implementiert werden können. Dazu soll der komplette standardmässige Funktionsumfang des .NET Framework zur Verfügung stehen. Dies adressiert den Punkt 1.
  - Es soll eine API zur Verfügung gestellt werden, welche die Funktionalität von Simio exponiert. Dies adressiert die Punkte 2-4.
- **StateMachine** – Komponente, welche die grafische Modellierung sowie Konfiguration der EFSM während der Simulation schlussendlich ausführt. Dies adressiert die Punkte 1-5.
- **ComplexGateway** – ermöglicht die Transformation von Entities zu autonomen Agenten und anschließender Ausführung der als EFSM vorliegender Agentenlogik und umgekehrt. Dies adressiert den Punkt 6.

Die weiteren Objekte wie «AgentBase» sowie dessen Spezifizierungen wurden bereits in [1] bzw. [3] entwickelt und können von dort übernommen werden.

## 3 Konzeptionelle Umsetzung der agentenbasierten Simulation in Simio

Das Softwarepaket «AnyLogic» inkludiert die Möglichkeit von agentenbasierter Simulation. Unter anderem wird die Modellierung auch in Form von StateCharts angeboten. Die nachfolgenden Konzepte sind tw. von dort inspiriert ([5] und [6]).

### 3.1 AddIn: SimioAgentLibrary

Das AddIn «SimioAgentLibraryAddIn» bildet den Einsprungspunkt in die agentenbasierte Modellierung und Simulation. Das AddIn wird von der Klasse `AgentLibraryAddIn` implementiert. Diese wiederum implementiert die von der SimioAPI zur Verfügung gestellten Interfaces `IDesignAddIn` und `IDesignAddInGuiDetails` (siehe Abbildung 3).

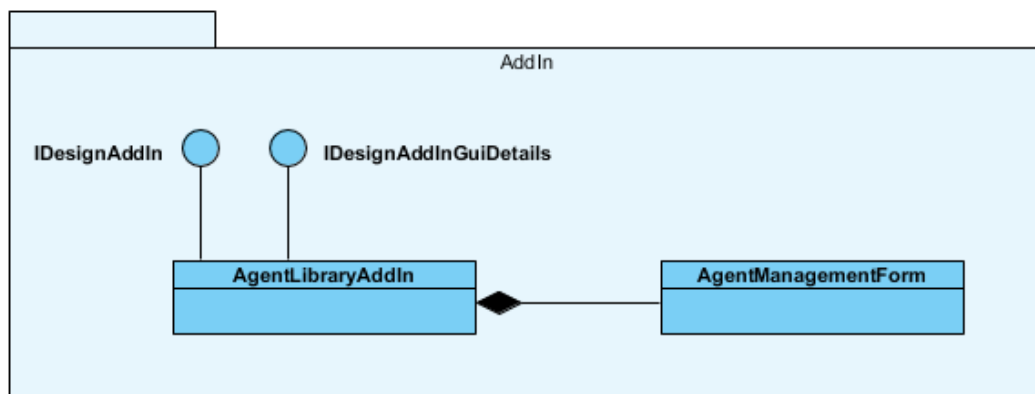


Abbildung 3: Klassenübersicht `AgentLibraryAddIn`

Anhand dieser Implementierung ist Simio in der Lage, das AddIn zu laden und in einem eigenen Karteireiter im Ribbon-Menü zur Verfügung zu stellen (in Abbildung 4 zu sehen). In [1, S.60f.] sowie [3, S.24f] finden sich weiterführende Informationen zur Entwicklung von AddIn's für Simio.

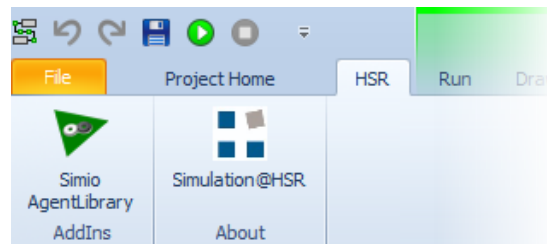


Abbildung 4: AddIn «SimioAgentLibrary»

Die Klasse `AgentManagementForm` implementiert die Oberfläche für die Modellierung der StateCharts für die einzelnen Agenten. Diese ist in Abbildung 5 in einer Übersicht dargestellt. In den nachfolgenden Abschnitten werden die folgenden Bereiche im Detail beschrieben:

- 1) Modellbaum
- 2) Bearbeitungsbereich
- 3) Konfigurationsbereich
- 4) Hauptmenü
- 5) Meldungen

Als Referenz sind in Abbildung 5 die einzelnen Bereiche mit zu obiger Auflistung korrespondierenden Nummern versehen.

Die Details zum Kernbereich, d.h. zum StateChart-Designer sind ausführlich in Kapitel 3.2 zu finden.

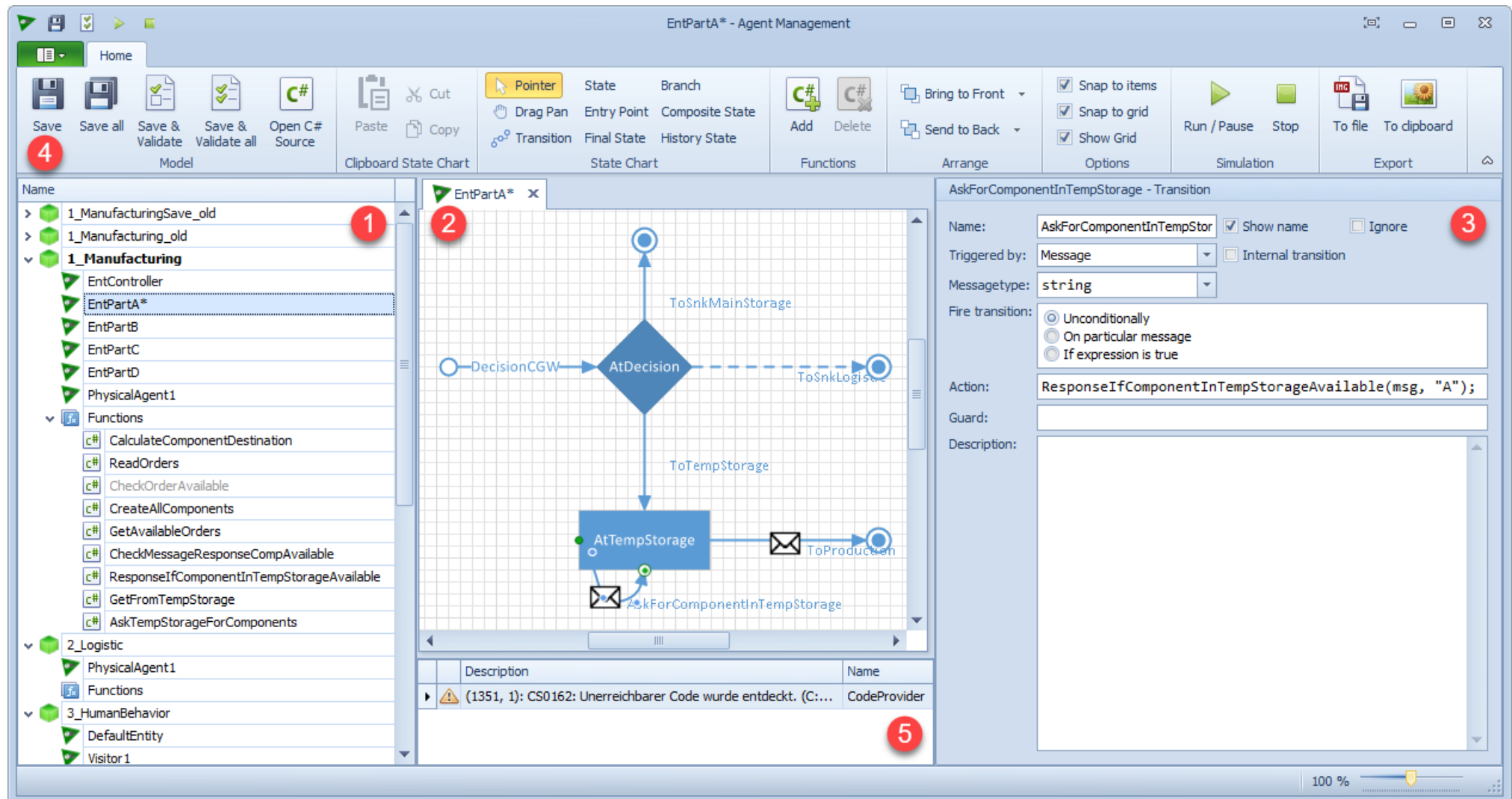


Abbildung 5: Oberfläche «AgentManagementForm»

## 3.1.1 Modellbaum

Der Modellbaum listet alle Modelle auf, welche sich im aktuell in Simio geöffneten Projekt befinden (siehe Abbildung 6). Für jedes Modell werden die Agenten (d.h. alle Objekte vom Basistyp «AgentBase») aufgeführt. Zusätzlich existiert für jedes Modell ein Bereich «Functions». In diesem Bereich können mit C#-Funktionen (genauer Methoden) implementiert werden. Diese Funktionen stehen anschliessend für die Verwendung als C#-Expressions in den Actions, Conditions, Guards usw. der States und Transitions im StateChart-Designer zur Verfügung. Grau dargestellte Elemente sind auf inaktiv gestellt. Informationen für das Hinzufügen bzw. Entfernen von Functions finden sich in Abschnitt 3.1.4.6. Informationen zu deren Implementierung sind im Abschnitt 3.3.9 zu finden.

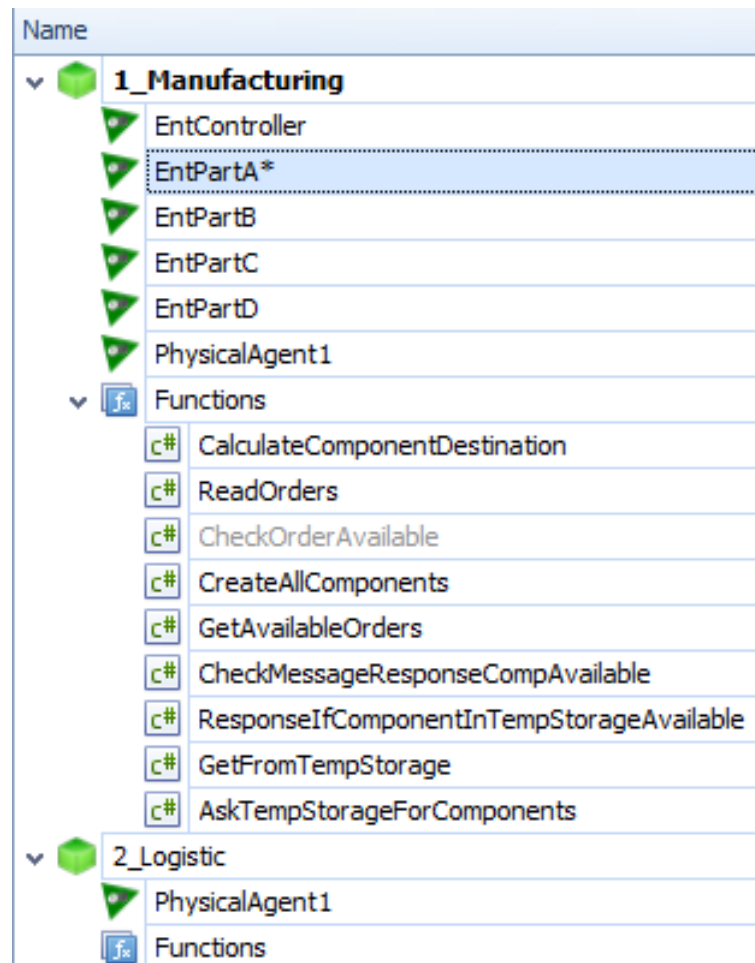


Abbildung 6: Modellbaum

Der Modellbaum wird bei entsprechenden Änderungen im aktuellen Projekt in Simio automatisch aktualisiert. D.h. sobald Modelle eingefügt, gelöscht oder umbenannt werden, wird die Änderung entsprechend umgehend im Modellbaum repräsentiert. Dies gilt auch für Agentenobjekte eines Modells. Das in Simio aktive Modell wird im Modellbaum mit fetter Schrift dargestellt (in Abbildung 6 entspricht dies «1\_Manufacturing»).

## 3.1.2 Bearbeitungsbereich

Der Bearbeitungsbereich dient dazu, die StateMachine in Form eines StateChart grafisch zu modellieren (siehe Abbildung 7). Ebenso können im Bearbeitungsbereich die C#-Funktionen (Abschnitt «Functions» in Abbildung 6) programmiert werden. Eine zur Bearbeitung geöffnete C#-Funktion ist in Abbildung 10 ersichtlich.



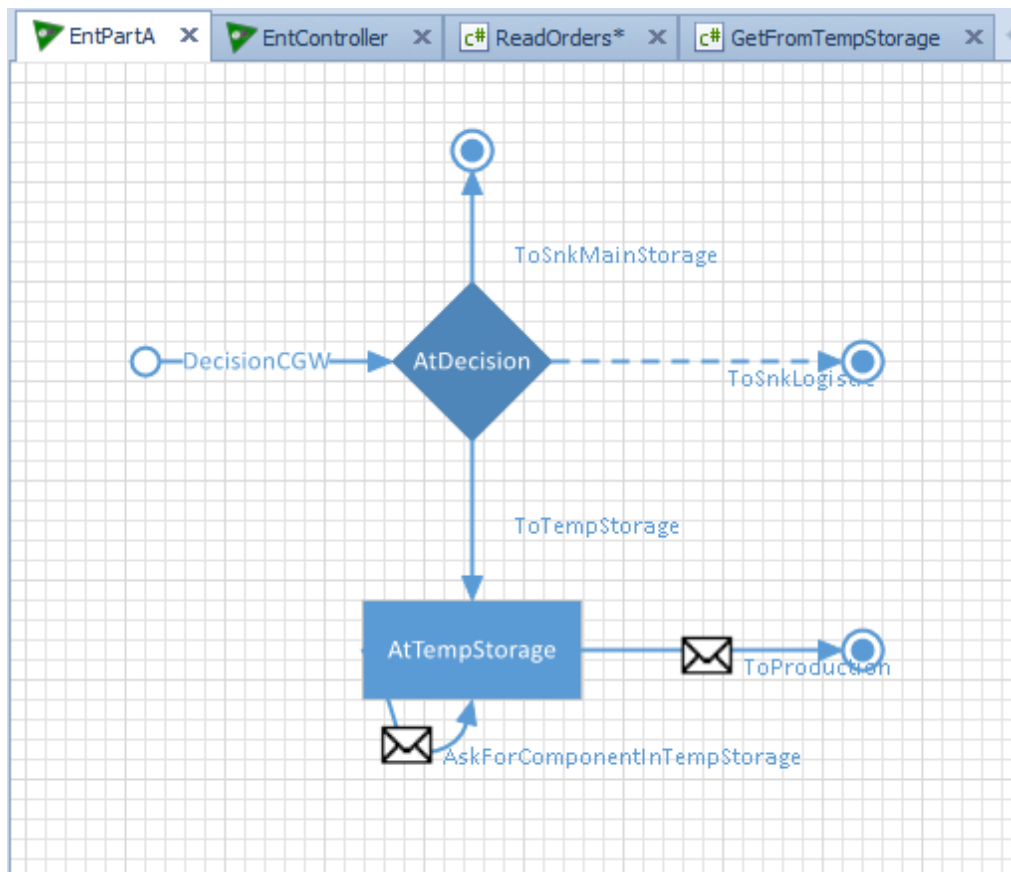


Abbildung 7: Bearbeitungsbereich mit StateChart

Für die Modellierung des StateCharts der StateMachine, stellt das Hauptmenü die Tools gemäss Abbildung 8 zur Verfügung. Diese können vom Anwender durch Drag&Drop in das aktuelle StateChart im Bearbeitungsbe-  
reich eingefügt werden. Ausführliche Informationen zu den einzelnen Tools sind in Kapitel 3.2 zu finden.

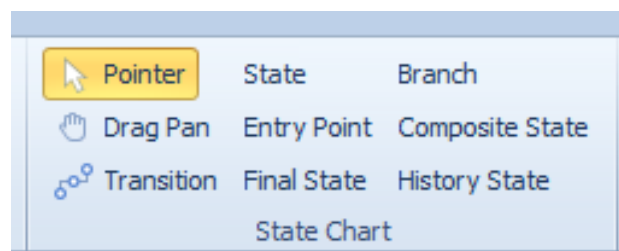


Abbildung 8: Tools für die StateChart-Modellierung

### 3.1.3 Konfigurationsbereich

In diesem Bereich kann das im Bearbeitungsbereich bzw. im Modellbaum selektierte Element konfiguriert werden. Bspw. können für States die Expressions für Entry und Exit implementiert werden (siehe Abbildung 9). Wie zuvor bereits erwähnt, erfolgt die Implementierung mit C#. Reicht eine Zeile aus, kann die Logik direkt als C#-Expression in die entsprechende Action eingefügt werden. Alternativ besteht die Möglichkeit, eine Function zu erstellen und diese in der gewünschten Action zu verwenden. Es kann jedoch auch ein beliebiger Code aus beliebig referenzierten Assemblies verwendet werden (siehe auch Abschnitt 3.3.9).

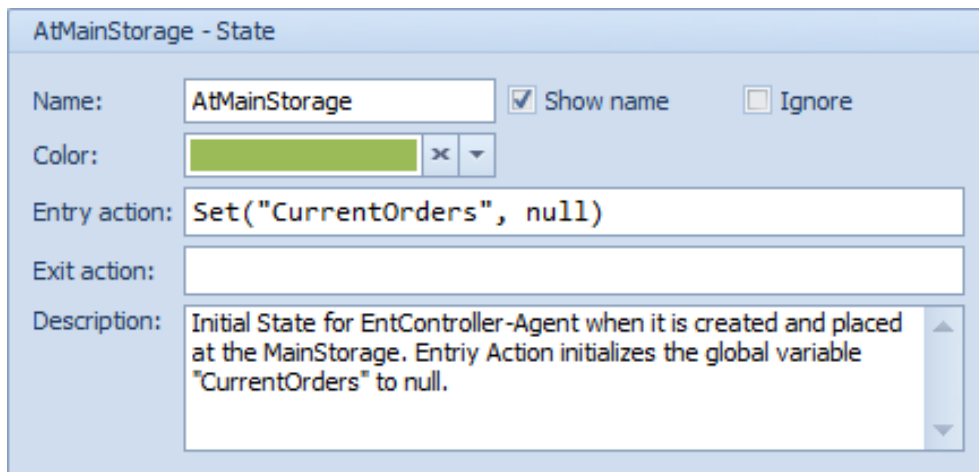


Abbildung 9: Konfiguration eines States

Wenn im Bearbeitungsbereich eine Function aktiv ist, besteht die Möglichkeit, die gewünschten Argumente sowie den Rückgabetyp der Function zu konfigurieren. Dies ist in Abbildung 10 dargestellt:

- links ist der C#-Sourcecode der Function ersichtlich und editierbar
- rechts kann die Function konfiguriert werden

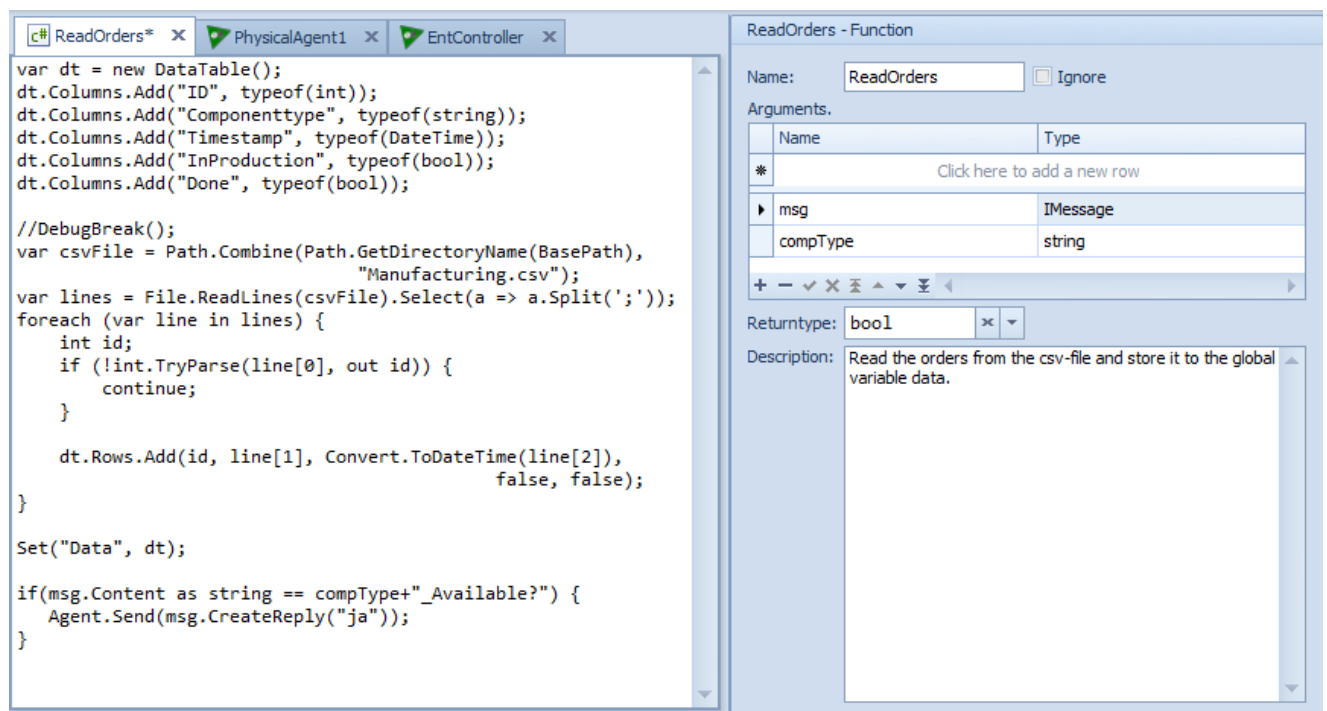


Abbildung 10: Implementierung und Konfiguration einer C# Funktion

## 3.1.4 Hauptmenü

Abbildung 11 zeigt das Hauptmenü. Es stellt diverse Möglichkeiten zur Verfügung, auf die in den nachfolgenden Abschnitten im Detail eingegangen wird.

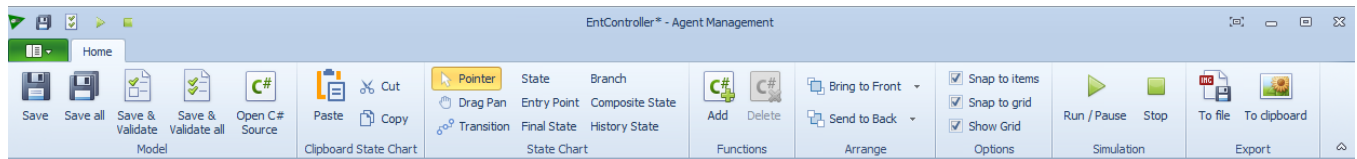


Abbildung 11: Hauptmenü

### 3.1.4.1 Save

Über «Save» sowie «Save all» werden die Elemente gespeichert. «Save» speichert nur das aktuelle Modell währenddem «Save all» alle Modelle speichert. Die Daten (StateCharts, Functions sowie alle Konfigurationen) werden im aktiven Simio-Projekt abgelegt. D.h. es werden keine zusätzlichen Dateien o.ä. generiert. Zu diesem Zweck generiert die SimioAgentLibrary gemäß Tabelle 4 zusätzliche Properties.

Property	Objektyp	Zweck
StateChart	AgentBase	Speichert das zum Agenttyp zugehörige StateChart als serialisiertes XML-Dokument.
AgentTypeStateChartProperties	AgentBase	In diesem Property werden die Eigenschaften des Agenttyp gespeichert. Das Objekt <code>AgentType</code> wird in ein XML-Dokument serialisiert und als <code>string</code> in diesem Property gespeichert.
StateMachineFunctions	Model	Die Functions werden in einer Liste (Objekt <code>FunctionList</code> ) zusammengefasst und ebenso in Form eines serialisierten XML-Dokuments als <code>string</code> gespeichert.

Tabelle 4: Speicherorte der Elemente der SimioAgentLibrary

Da der Anwender die Properties nicht bearbeiten können muss, werden diese mit dem Status `Visible = false` erstellt. Sie sind aber in den jeweiligen Propertydefinitions zum Modell bzw. dem entsprechenden AgentBase-Objekt ersichtlich und können dort, falls gewünscht, auch auf `Visible = true` umgestellt werden.

### 3.1.4.2 Validierung

Der Validierungsmechanismus führt folgende Aktionen durch:

- Die anhand des StateCharts modellierte StateMachine wird auf einen konsistenten Zustand hin überprüft. Bei diesem Vorgang werden folgende Eigenschaften kontrolliert:
  - Es muss zwingend genau ein «Entry Point» vorhanden sein.
  - Ein «Entry Point» muss zwingend auf einen State zeigen.
  - Ein «Initial State»-Pointer muss sich innerhalb eines «Composite State» befinden.
  - Ein «Composite State» muss genau einen «Initial State»-Pointer aufweisen.
  - Eine «Transition» muss einen «Start-State» sowie einen «Ziel-State» aufweisen.
  - Ein «History State» muss sich innerhalb eines «Composite State» befinden.
  - Ein «History State» darf keine ausgehenden «Transitions» aufweisen.
  - Ein «Branch» darf nur eine «Default-Transition» aufweisen.
- Die C# Sourcecodes müssen kompilierbar sein.

Wird vom Validierungsmechanismus ein Problem (Fehler bzw. Warnung) festgestellt, wird dieses im Meldungsfenster (siehe Abschnitt 0) ausgegeben.

### 3.1.4.3 *Open C# Source*

Sobald die Simulation gestartet wird, bzw. im Zuge der Validierung, werden alle C# Sourcecodefragmente des aktiven Modells (Functions und Actions der Modellelemente im StateChart-Designer) für deren Kompilierung in einer zentralen Datei zusammengefasst (siehe auch Kapitel 4.1). Dieser Menüpunkt erlaubt das Öffnen und Anzeigen dieser Datei. Dies kann die Suche nach Problemen erheblich vereinfachen. Im Bereich Meldungen (siehe Abschnitt 0) wird bspw. bei Syntaxfehlern im C#-Code die entsprechende Zeilennummer angezeigt. Diese Zeilennummer bezieht sich auf diese Datei. Eine entsprechende Beispieldatei befindet sich auf der Projektdaten-CD (siehe Anhang E).

### 3.1.4.4 *Clipboard Operations*

Über diese Tools lassen sich Elemente des StateChart kopieren bzw. ausschneiden und einfügen.

### 3.1.4.5 *StateChart*

Diese Gruppe (siehe Abbildung 8 für eine grössere Darstellung) stellt die Elemente für den StateChart-Designer zur Verfügung. Diese können vom Anwender durch Drag&Drop in das aktuelle StateChart im Bearbeitungsreich eingefügt werden. Ausführliche Informationen zu den einzelnen Tools finden sich in Kapitel 3.2.

### 3.1.4.6 *Functions*

Mit «Add» bzw. «Delete» kann eine neue Function hinzugefügt bzw. eine bestehende Function gelöscht werden. Eine neue Function wird Im Modellbaum zum Bereich «Functions» hinzugefügt (siehe Abschnitt 3.1.1). Informationen zu deren Implementierung sind im Abschnitt 3.3.9 zu finden.

### 3.1.4.7 *Options*

Diese Gruppe stellt folgende Einstellungen zur Verfügung:

- «Snap to items»: Richtet die Elemente des StateChart untereinander aus.
- «Snap to grid»: Positioniert die Elemente des StateChart in Bezug auf das Raster.
- «Show Grid»: Zeigt wahlweise das Raster an.

### 3.1.4.8 *Simulation*

«Run / Pause» ermöglicht den Start bzw. das Pausieren der Simulation des aktiven Modells. Mit «Stop» kann die Simulation beendet werden. Diese Tools entsprechen exakt denjenigen auf dem Simio-Hauptfenster um eine Simulation zu starten, zu pausieren bzw. zu beenden – jedoch mit folgender Ergänzung: Der komplette C#-Code (Functions und Expressions) wird kompiliert, damit dieser für die Simulation in der aktuellen Fassung zur Verfügung steht.

### 3.1.4.9 *Export*

Hier kann das aktuelle StateChart wahlweise in eine Datei («To file») oder in die Zwischenablage («To clipboard») exportiert werden.

## 3.1.5 Meldungen

Diese Liste enthält Meldungen, die von der SimioAgentLibrary an diversen Stellen generiert werden können. Bspw. werden hier Validierungsmeldungen ausgegeben (siehe Abschnitt 3.1.4.2 sowie Abbildung 12).

Description	Name
⚠ Hanging transition.	Transition 'ToTempStorage'
❌ (315, 19): CS1061: 'SimioAgentLibrary.IntelligentObjects.Agent.SteeringBehavior.Agent.StateMachine.API.IAgent' enthält keine Definition für 'MoveNode', ...	CodeProvider
⚠ (1306, 1): CS0162: Unerreichbarer Code wurde entdeckt. (C:\ProgramData\Simio\AgentLibrary\1_ManufacturingModel.cs)	CodeProvider
❌ (1341, 21): CS0103: Der Name 'orderss' ist im aktuellen Kontext nicht vorhanden. (C:\ProgramData\Simio\AgentLibrary\1_ManufacturingModel.cs)	CodeProvider
❌ (1352, 32): CS1002: ; erwartet. (C:\ProgramData\Simio\AgentLibrary\1_ManufacturingModel.cs)	CodeProvider
❌ (1353, 5): CS1002: ; erwartet. (C:\ProgramData\Simio\AgentLibrary\1_ManufacturingModel.cs)	CodeProvider
❌ (1353, 11): CS1003: Syntaxfehler. '(' erwartet. (C:\ProgramData\Simio\AgentLibrary\1_ManufacturingModel.cs)	CodeProvider
❌ (1353, 11): CS1525: Ungültiger Ausdruck '='. (C:\ProgramData\Simio\AgentLibrary\1_ManufacturingModel.cs)	CodeProvider
❌ (1353, 13): CS1026: ) erwartet. (C:\ProgramData\Simio\AgentLibrary\1_ManufacturingModel.cs)	CodeProvider

Abbildung 12: Meldungsausgabe

## 3.2 Technische Komponentenübersicht der SimioAgentLibrary

In Abbildung 14 ist die Komponentenübersicht dargestellt. Das AgentLibraryAddIn stellt den Einsprungspunkt zur Verfügung (siehe Kapitel 3.1). Vom AddIn wird die eigentliche AgentManagementForm (Abbildung 5) erstellt und angezeigt. Die Klassenstruktur der AgentManagementForm ist in Abbildung 13 dargestellt.

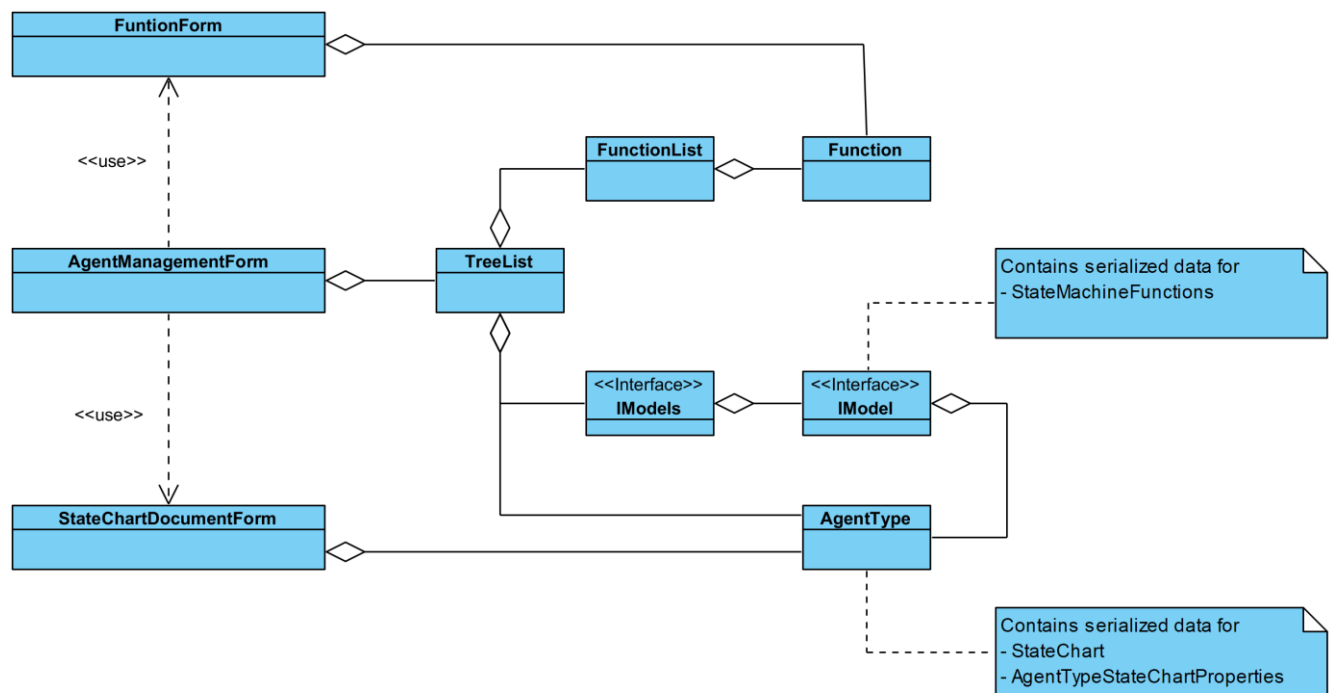


Abbildung 13: Klassendiagramm «AgentManagementForm»

Die Klasse TreeList entspricht dem Modellbaum. Die Klasse FunctonForm stellt die Bearbeitungsmöglichkeit der C#-Funktionen zur Verfügung (siehe Abbildung 10). Die Klasse StateChartDocumentForm stellt den StateChart-Designer dar (siehe Abbildung 7). Dieser ermöglicht die Modellierung der StateCharts für die Agenten. Die Details des StateChart-Designers finden sich im nachfolgenden Kapitel 3.3.

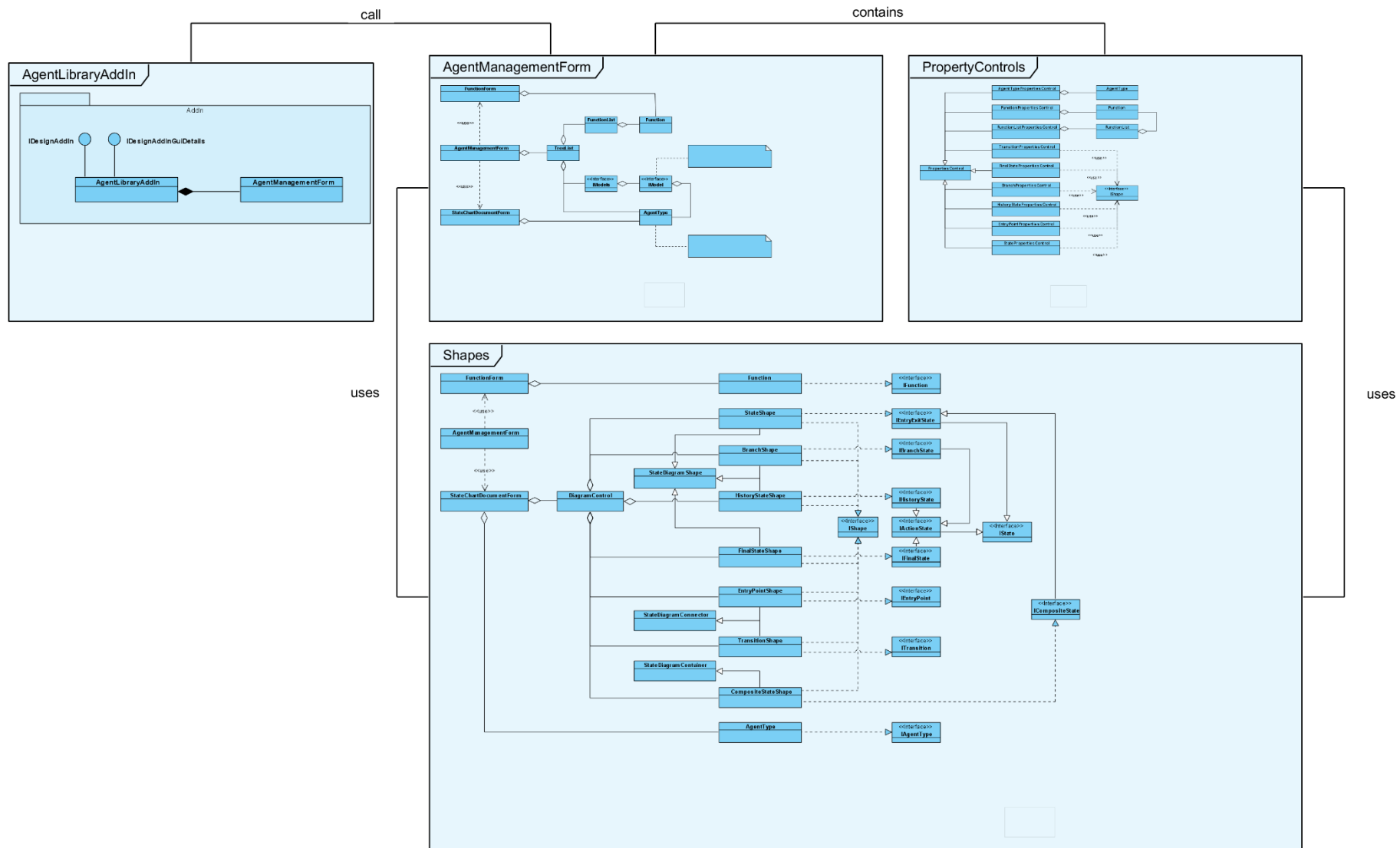


Abbildung 14: Technische Komponentenübersicht der SimioAgentLibrary

### 3.3 StateChart-Designer

Der StateChart-Designer stellt den Kern der Oberfläche für die SimioAgentLibrary dar. Er ermöglicht die Modellierung von StateCharts für die Agenten. Ein StateChart kann aus den folgenden Elementen bestehen:

- Transitions
- States
- Entry Point
- Final State
- Branch
- Composite State
- History State

Im nachfolgenden Abschnitt ist eine technische Übersicht in Form eines UML Klassendiagramms zu finden. In diesem Diagramm sind die möglichen Elemente eines StateCharts wiedergegeben. Dies ermöglicht es, die Brücke zwischen technischer Implementierung und Funktionsweise des StateChart-Designers zu schlagen. Die Funktionsweise für die einzelnen Elemente ist in den weiteren Abschnitten ausführlich beschrieben.

#### 3.3.1 Übersicht

Der StateChart-Designer wird von der Klasse `StateChartDocumentForm` repräsentiert. Diese enthält ein Objekt vom Typ `DiagramControl`, welches von der DevExpress Suite [4] zur Verfügung gestellt wird (siehe Kapitel 2.3). Die Elemente, welche das `DiagramControl` für die Komposition der StateCharts verwendet, werden durch die `Shape`-Klassen repräsentiert. Dies entspricht den Klassen, welche das Interface `IShape` implementieren. Abbildung 15 zeigt eine vollständige Klassenübersicht des StateChart-Designers.

Die `Shape`-Klassen implementieren zusätzlich jeweils spezifische Interfaces und stellen die Schnittstelle zu der `StateMachine` dar. Diese muss für die Ausführung der Simulation in der Lage sein, das StateChart zu laden. Durch die Verwendung dieser Interfaces kann die `StateMachine` die gleiche Objektstruktur verwenden, wie der StateChart-Designer. Dies hat den grossen Vorteil, dass allfällige zukünftige Erweiterungen für die Konfiguration der `StateMachine` zentral an einem Ort durchgeführt werden können. Dies umfasst sowohl für die Runtime der `StateMachine` als auch für deren grafische Repräsentation. In Kapitel 4.1 finden sich die technischen Details zur Implementierung der `StateMachine`. Dort ist auch zu sehen, dass diese auf den spezifischen Interfaces der `Shape`-Klassen aufbaut und keine neuen Objekte generiert werden.

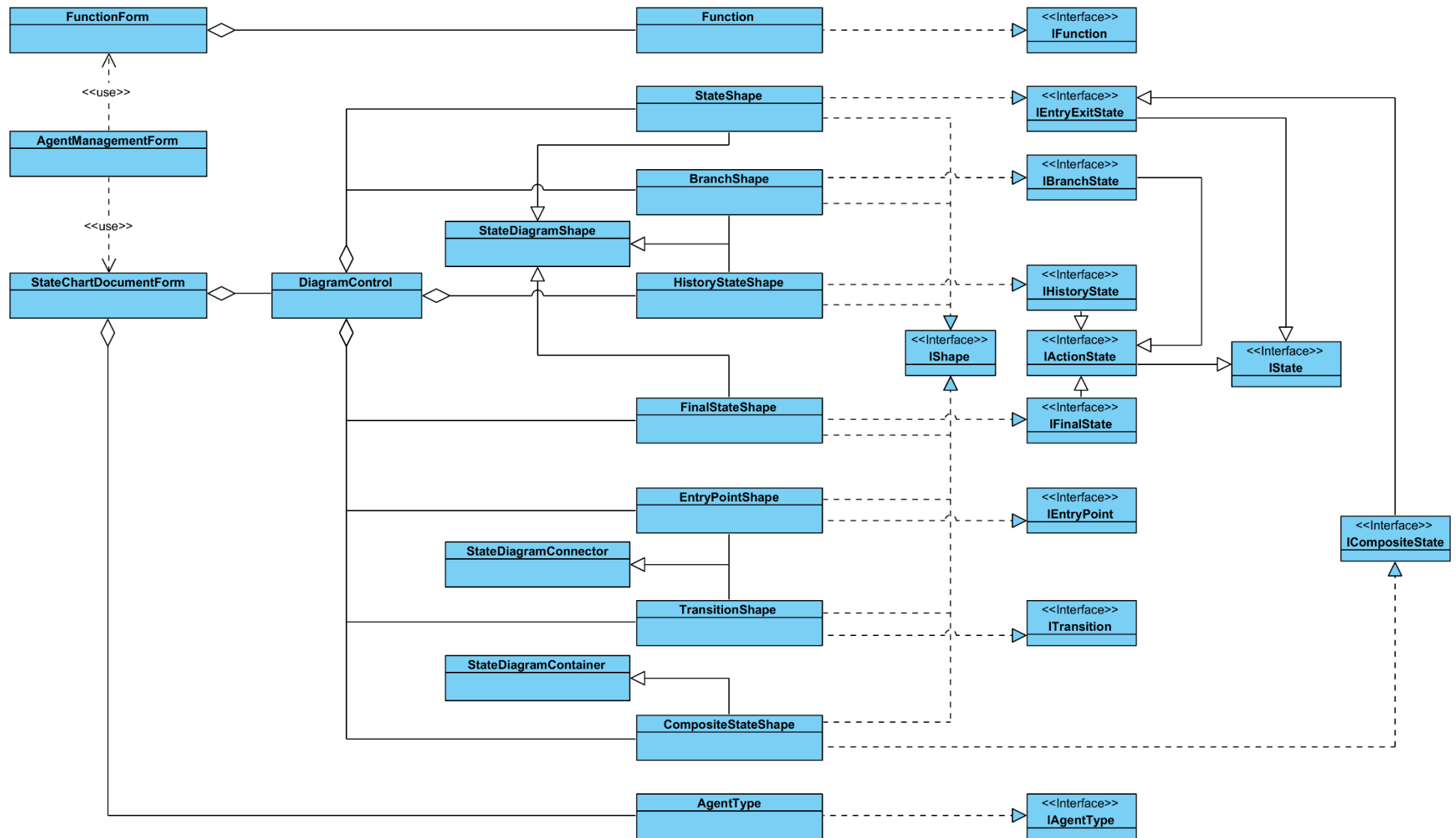


Abbildung 15: Klassenübersicht StateChart-Designer



Für die Konfiguration der Elemente im StateChart-Designer werden die `PropertiesControl`-Klassen verwendet. Eine entsprechende Übersicht zeigt Abbildung 16. Die konkreten `XYZPropertiesControl`-Klassen erben von der zentralen Klasse `PropertiesControl` und werden jeweils abhängig vom im StateChart-Designer selektierten Element im Konfigurationsbereich angezeigt (siehe Abschnitt 3.1.3).

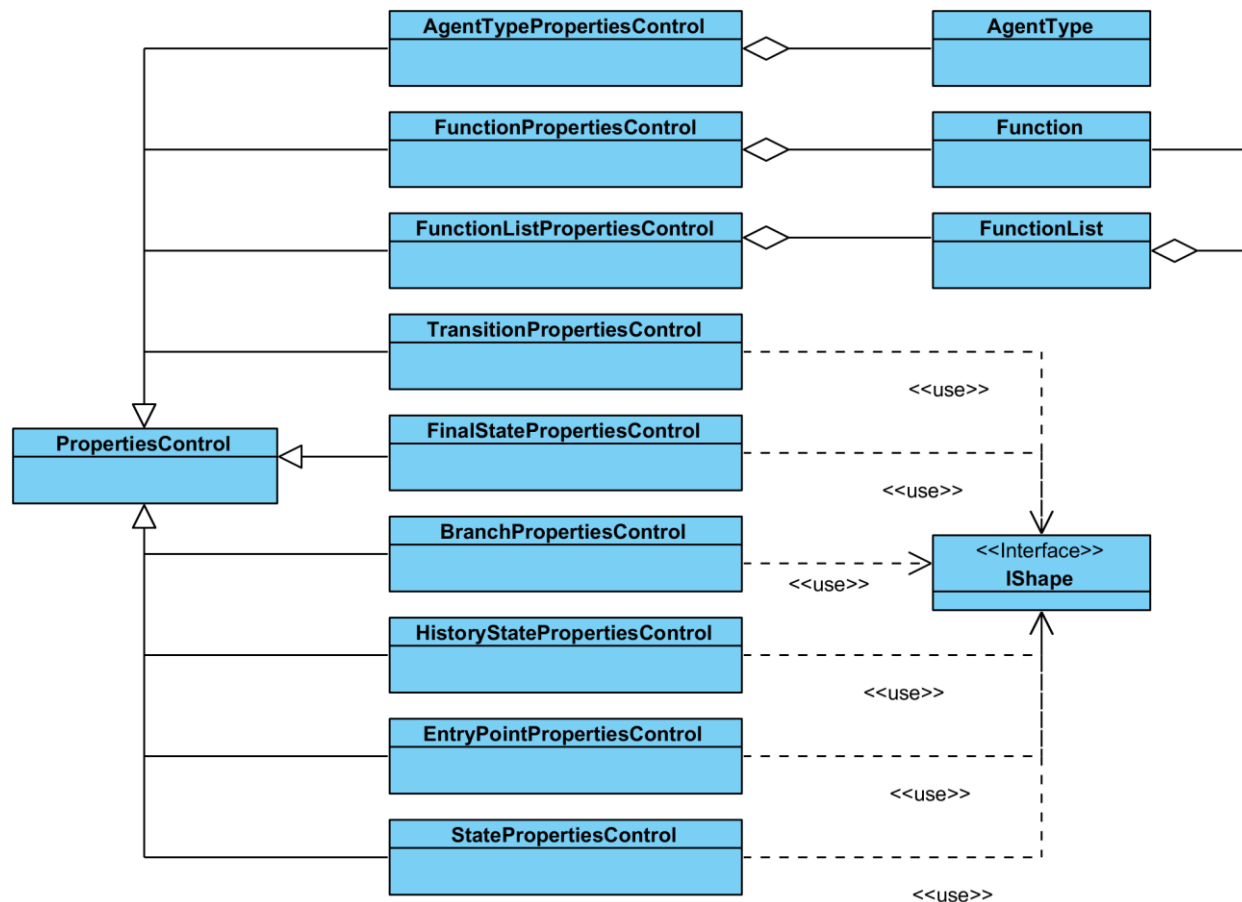


Abbildung 16: PropertyControls für den StateChart-Designer

In den nachfolgenden Abschnitten wird auf die Details der einzelnen Elemente des StateChart-Designers eingegangen. U.a. sind für jedes Element die möglichen Einstellungen aufgelistet. Tabelle 5 zeigt die allgemeinen Einstellungen, die jedes Element aufweist. Diese werden anschliessend nicht mehr explizit erwähnt.

Einstellung	Beschreibung
<b>Name</b>	Name des Elements.
<b>Show name</b>	Legt fest, ob der Elementname im StateChart-Designer angezeigt werden soll oder nicht.
<b>Ignore</b>	Wird diese Einstellung auf «ja» gesetzt, wird das Element für die Ausführung ignoriert. Zu ignorierende Elemente werden im StateChart-Designer grau dargestellt.
<b>Color</b>	Definiert die Farbe, mit welcher das Element im StateChart-Designer angezeigt wird (sofern es nicht ignoriert werden soll).
<b>Description</b>	In dieser Eigenschaft kann eine Beschreibung des Elements hinterlegt werden.

Tabelle 5: Allgemeine Einstellungen für Elemente im StateChart-Designer

### 3.3.2 Transitions

Eine Transition bezeichnet einen Wechsel von einem State zu einem anderen. Sobald das angegebene Triggerereignis auftritt und die Guard erfüllt ist, wird die angegebene Action ausgeführt und es erfolgt ein Übergang.

Der Startpunkt einer Transition wird mit dem Quell-State und der Endpunkt mit dem Ziel-State verbunden. Wenn der Startpunkt einer Transition entweder auf oder innerhalb eines States liegt und das Ziel ausserhalb

des States, wird dieser State durch die Transition verlassen. Wird die Transition ausgeführt, wird somit für diesen State die Exit-Action ausgeführt. Liegt der Startpunkt einer Transition ausserhalb eines States und das Ziel entweder auf oder im State, wird dieser State durch die Transition aktiviert. Wird die Transition ausgeführt, wird folglich für diesen State die Entry-Action ausgeführt.

Zusätzlich gibt es einen speziellen Typ von Transition: die interne Transition. Eine interne Transition verbindet mit ihrem Start- sowie Endpunkt den gleichen State. Folglich wird bei der Ausführung einer internen Transition weder die Exit- noch die Entry-Action ausgeführt.

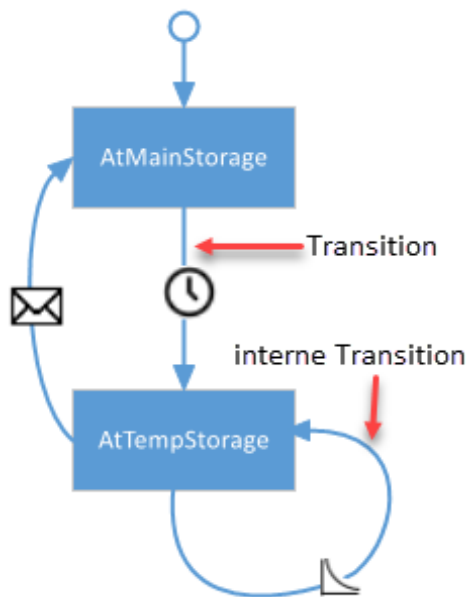


Abbildung 17: Transitionen

Die zur Verfügung stehenden Transitionsarten sind in Tabelle 6 aufgeführt. In den nachfolgenden Abschnitten werden die einzelnen Arten zusätzlich im Detail erläutert.






Art der Transition	Symbol	Beschreibung
<b>Timeout</b>		Die Transition wird ausgelöst, nachdem eine bestimmte vorgegebene Zeit abgelaufen ist.
<b>Rate</b>		Die Transition wird abhängig von einer definierten Rate ausgelöst. Die Rate entspricht einer exponentiellen Verteilung.
<b>Condition</b>		Die Transition wird ausgelöst, sobald eine logische Bedingung zutrifft.
<b>Message</b>		Die Transition wird ausgelöst, sobald der Agent, dessen Verhalten durch das aktuelle StateChart konfiguriert wird, eine bestimmte Nachricht empfängt.
<b>AgentArrival</b>		Die Transition wird ausgelöst, wenn der Agent ein definiertes Ziel erreicht.

Tabelle 6: Mögliche Transitionsarten

Eine Transition, die eine Branch verlässt (siehe Abschnitt 3.3.3) stellt einen Spezialfall dar. Diese bietet ausschliesslich die Möglichkeit, zwischen «Condition» und «Default» zu wählen. Wird «Condition» gewählt, kann eine Condition angegeben werden, die bestimmt, ob die Transition für das Verlassen der Verzweigung verwendet werden soll. Wird «Default» gewählt, wird die betreffende Transition ausgeführt, sofern für alle anderen die Condition nicht erfüllt ist (Fallback).

Wie erwähnt werden in den nachfolgenden Abschnitten die verschiedenen Transitionsarten inkl. deren Einstellungen beschrieben. Wenn bei der Einstellung «unterstützt C#-Expressions» angegeben ist, wird der Wert der

Einstellung zur Laufzeit mit Hilfe des C#-Compilers evaluiert. D.h. es sind komplexe Ausdrücke wie Berechnungen oder Funktionsaufrufe möglich. Bspw. wären für die Einstellung «Timeout» der Timeout-Transition die Werte gemäss Tabelle 7 zugelassen.

30	Konstante
$4 + 6.5 / 2$	Berechnung
<code>GetTimeOutForScreenOff()</code>	Aufruf einer Funktion und Verwendung deren Rückgabewertes (siehe auch Abschnitt 3.3.9).

Tabelle 7: Mögliche Werte für eine Einstellung die C# Expression unterstützt

### 3.3.2.1 Timeout

Die Transition wird aktiviert, nachdem die angegebene Zeitspanne (Timeout) verstrichen ist. Diese Art kann zum Modellieren von Verzögerungen und in Kombination mit alternativen Transitions von Zeitüberschreitungen verwendet werden.

Es stehen folgende Einstellungen zur Verfügung:

Einstellung	Beschreibung
<b>Timeout</b>	Zeitspanne, nach der die Transition aktiviert wird (unterstützt C#-Expressions).
<b>TimeoutUnit</b>	Einheit der Zeitspanne (Millisekunden, Sekunden, Minuten, Stunden, Tage, Wochen, Monate).
<b>Action</b>	Aktion, die bei Aktivierung der Transition ausgeführt werden soll (unterstützt C#-Expressions).
<b>Guard</b>	Schutzbedingung, die erfüllt sein muss, damit die Transition aktiviert wird (unterstützt C#-Expressions).

Tabelle 8: Einstellungen für Timeout-Transition

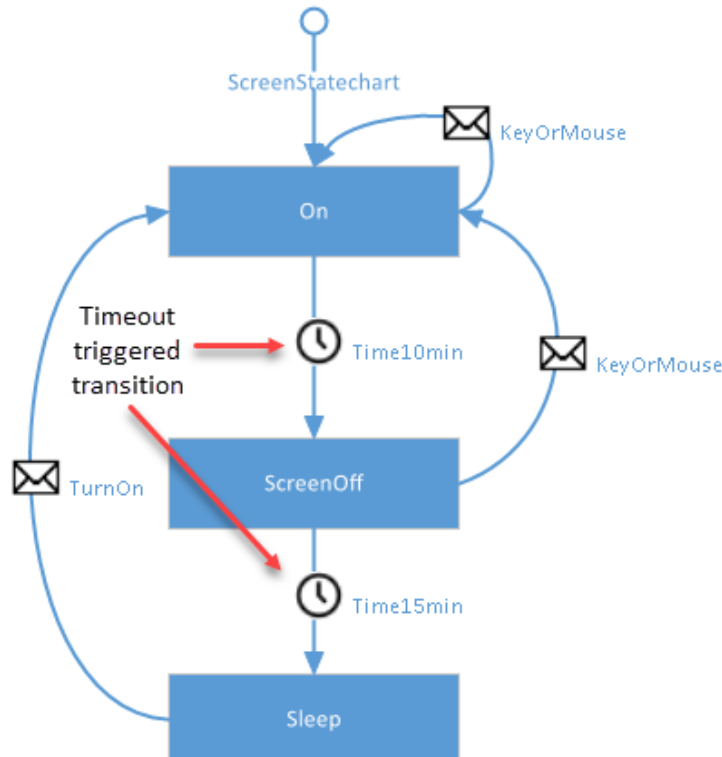


Abbildung 18: Timeout-Transition

Abhängig von der Einheit (TimeoutUnit) für das Timeout muss ggf. das TimeInterval für das SteeringBehavior angepasst werden (siehe Abschnitt 4.1.3). Das TimeInterval gibt die Taktlänge für die diskrete Ereignis-

simulation (DES) vor. Nach Ablauf des Intervalls werden jeweils die States der Agenten überprüft und ggf. Transitions ausgelöst. Wenn nun ein Timeout z.B. mit der Einheit Millisekunden definiert wird, muss das Timeintervall der DES kleiner oder gleich lang sein, damit die Transition zum korrekten Zeitpunkt ausgelöst werden kann. Das Timeintervall kann auch dynamisch zur Laufzeit der Simulation angepasst werden (siehe Kapitel 6.1).

### 3.3.2.2 Rate

Die Rate-Transition entspricht im Grundsatz der Timeout-Transition. Es wird jedoch keine explizite Zeitspanne vorgegeben. Stattdessen wird diese anhand einer exponentiell verteilten Zufallszahl berechnet, wobei die angegebene Rate als Mittelwert verwendet wird:

$$timeout = \frac{\log_e(U)}{rate}, \text{ wobei } U = \text{uniform verteilte Zufallszahl zwischen 0 und 1}$$

Konkret bedeutet dies z.B.:

wenn die Rate = 5, so wird die Transition im Durchschnitt 5 Mal pro Zeiteinheit (RateUnit) ausgeführt.

Es stehen folgende Einstellungen zur Verfügung:

Einstellung	Beschreibung
<b>Rate</b>	Rate, mit der die Transition aktiviert wird (unterstützt C#-Expressions).
<b>RateUnit</b>	Einheit der Rate (Millisekunden, Sekunden, Minuten, Stunden, Tage, Wochen, Monate).
<b>Action</b>	Siehe Tabelle 8.
<b>Guard</b>	Siehe Tabelle 8.

Tabelle 9: Einstellungen für Timeout-Transition

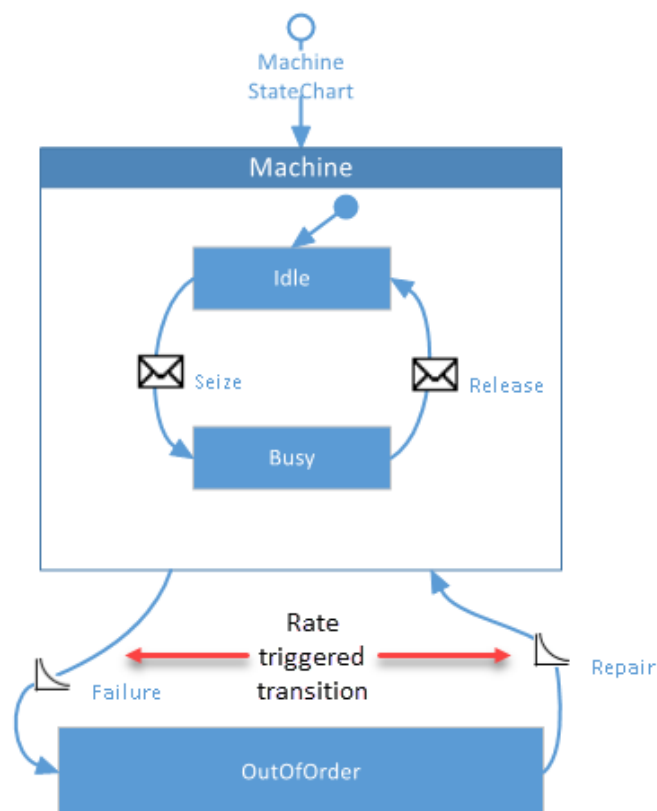


Abbildung 19: Rate-Transition

Für die Rate-Transition gilt es in Bezug auf die Einheit (RateUnit) das gleiche zu beachten wie bei der Timeout-Transition (siehe Abschnitt 3.3.2.1 letzter Absatz).

### 3.3.2.3 Condition

Die Condition-Transition wird aktiviert, wenn die konfigurierte Bedingung zu «true» wechselt. Gelangt der Agent in einen Quell-State einer Condition-Transition deren Bedingung bereits «true» ist, wird die Transition sofort aktiviert.

Es stehen folgende Einstellungen zur Verfügung:

Einstellung	Beschreibung
<b>Condition</b>	Bedingung, die für die Aktivierung der Transition erfüllt sein muss (unterstützt C#-Expressions).
<b>Action</b>	Siehe Tabelle 8.
<b>Guard</b>	Siehe Tabelle 8.

Tabelle 10: Einstellungen für Condition-Transition

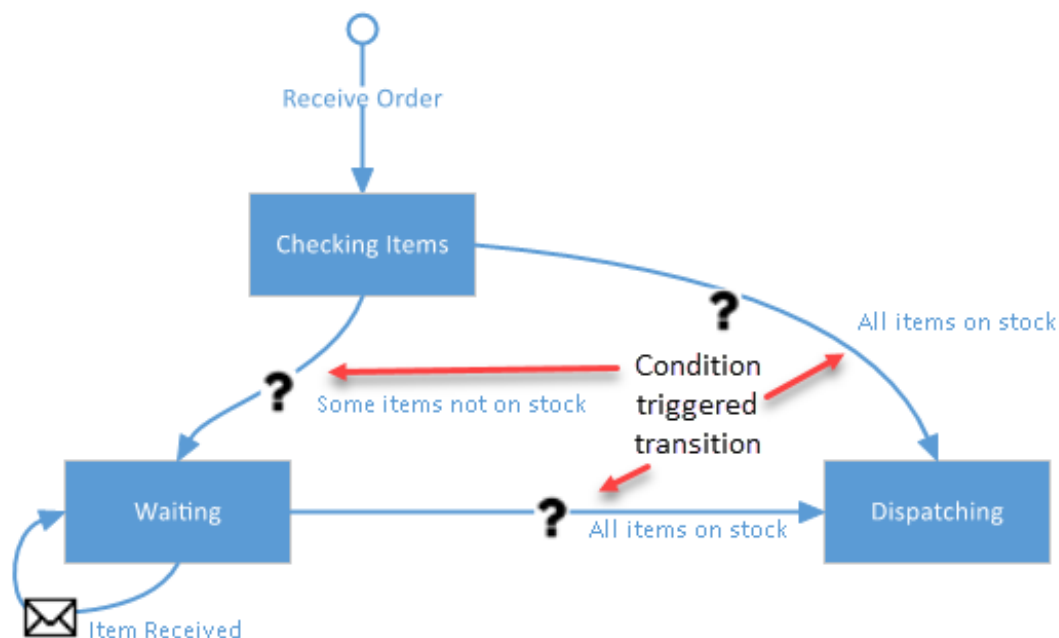


Abbildung 20: Condition-Transition

Für die Condition-Transition muss das sogenannte «Sensitivity Problem» beachtet werden. Wenn z.B. die Bedingung den Ausdruck  $x \geq 3$  aufweist und sich  $x$  kontinuierlich über die Zeit ändert, kann es vorkommen, dass  $x$  zwischen den Schritten der Simulation den Wert 3 überschreitet und sich wieder verringert, bevor der nächste Simulationsschritt ausgeführt wird (siehe Abbildung 21). Wenn Systeme modelliert werden, bei denen ein solches Verhalten kritisch ist, sollte der TimeInterval entsprechend verringert werden (siehe Kapitel 6.1).

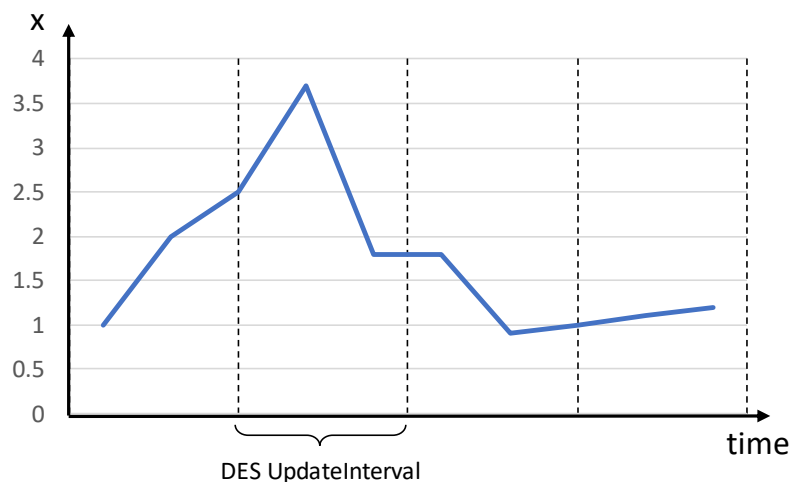


Abbildung 21: Sensitivity Problem

### 3.3.2.4 Message

Eine solche Transition wird ausgeführt, wenn der Agent eine Nachricht empfängt, die dem definierten Message-Deskriptor entspricht. Als Message-Deskriptor stehen folgende Möglichkeiten zur Verfügung:

- Ein beliebiges Objekt eines bestimmten Typs. Hier kann jeder verfügbare Typ verwendet werden – dies beinhaltet auch selbst definierte Typen.
- Filtern von Nachrichten nach Inhalt. Es wird gemäss vorhergehendem Punkt ein bestimmter Typ als Message-Deskriptor definiert. Anschliessend werden nur Nachrichten akzeptiert, die einer bestimmten Instanz dieses Deskriptors entsprechen.
- Eine beliebige Expression, die auf `true` ausgewertet werden soll.

Eine Nachricht kann mit der API-Funktion `Send` versendet werden (siehe Kapitel 6.1).

Es stehen folgende Einstellungen zur Verfügung:

Einstellung	Beschreibung
<b>MessageType</b>	Typ des Message-Deskriptors
<b>Fire Transition</b>	Hier stehen folgende Möglichkeiten zur Auswahl: <ul style="list-style-type: none"> <li>• «Unconditionally»: Es wird ausschliesslich der Typ des Message-Deskriptors überprüft.</li> <li>• «On particular message»: Zusätzlich zum Typ kann die Nachricht bezüglich einer bestimmten Instanz gefiltert werden. Die entsprechende Instanz kann mit der Eigenschaft «Message» spezifiziert werden.</li> <li>• «If expression is true»: Mit der Eigenschaft «Expression» kann eine komplexe Bedingung angegeben werden, welche auf <code>true</code> ausgewertet wird.</li> </ul>
<b>Message</b>	Nachrichteninstanz, auf die reagiert werden soll (unterstützt C#-Expressions).
<b>Expression</b>	Beliebige Expression, die auf <code>true</code> ausgewertet werden soll (unterstützt C#-Expressions).
<b>Action</b>	Siehe Tabelle 8.
<b>Guard</b>	Siehe Tabelle 8.

Tabelle 11: Einstellungen für Message-Transition

Nachfolgende Schritte erläutern die korrekte Definition einer Message-Transition:

- 1) Wenn eine Typ-Prüfung durchgeführt werden soll, muss der gewünschte Typ mit der Einstellung «MessageType» definiert werden.
- 2) Soll keine Typ-Prüfung erfolgen, kann `object` als Typ definiert werden. Dies entspricht dem Basistyp aller möglichen Typen (C# Standard).
- 3) Wenn ein Inhalt spezifiziert werden soll, kann dies mit der Einstellung «Message» vorgenommen werden. Dazu muss bei «Fire Transition» «On particular message» gewählt werden. Die Prüfung erfolgt durch Verwendung der Basismethode `Equals(messageContent, descriptor)`. Mögliche Nachrichten sind z.B. "STOP!" für Nachrichten vom Typ `string`, 3.2 für Nachrichten vom Typ `double` usw. Es kann jedoch auch eine C#-Expression angegeben werden, deren Return-Wert für die Vergleichsprüfung verwendet wird.
- 4) Soll kein spezifischer Inhalt vorgegeben werden, kann «Fire Transition» auf «Unconditionally» belassen werden.
- 5) Wenn eine komplexe Nachricht überprüft werden soll, kann eine C#-Expression angegeben werden. Dazu muss für «Fire Transition» «If expression is true» gewählt werden.

Für die Einstellung «Message», «Expression» sowie «Action» steht die Variable `msg` (Typ `IMessage` – siehe Kapitel 6.1) zur Verfügung, welche die effektive Message enthält. Dies erlaubt es, mit Hilfe einer C#-Expression eine Function aufzurufen und dieser die Nachricht zu übergeben.

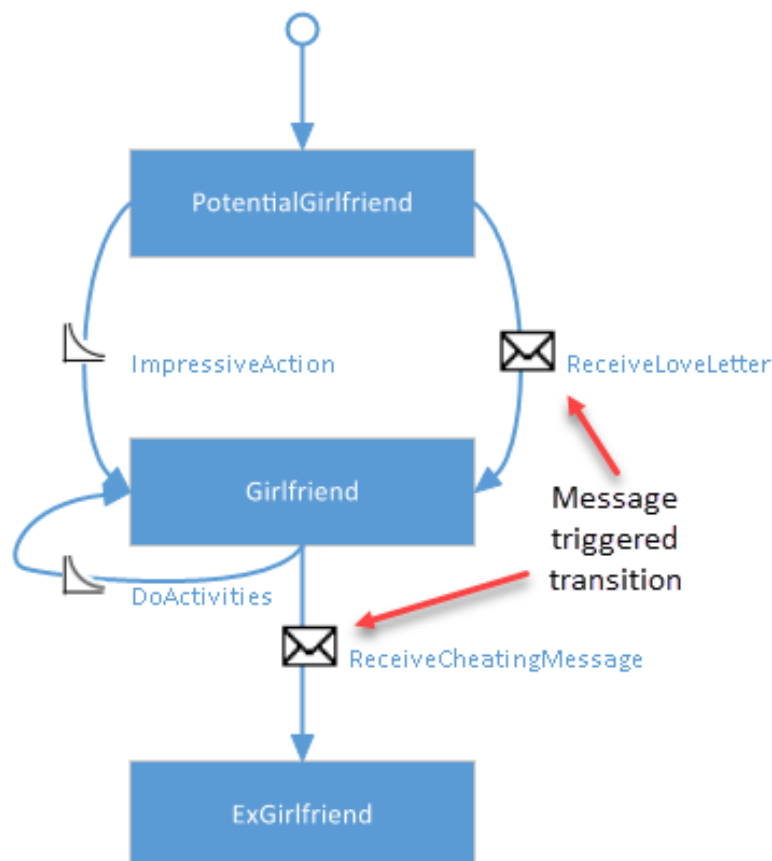


Abbildung 22: Message-Transition

Gesendete Nachrichten werden in einer Queue gespeichert. Pro Zyklus der DES wird jeweils eine Nachricht verarbeitet.

## 3.3.2.5 AgentArrival

Eine AgentArrival-Transition wird aktiviert, wenn der Agent sein Ziel erreicht. Voraussetzung ist eine vorgängige Auslösung einer Bewegung mit `MoveTo`, `MoveToNode` oder `JumpTo` (siehe Kapitel 6.1).

Es stehen folgende Einstellungen zur Verfügung:

Einstellung	Beschreibung
Action	Siehe Tabelle 8.
Guard	Siehe Tabelle 8.

Tabelle 12: Einstellungen für AgentArrival-Transition

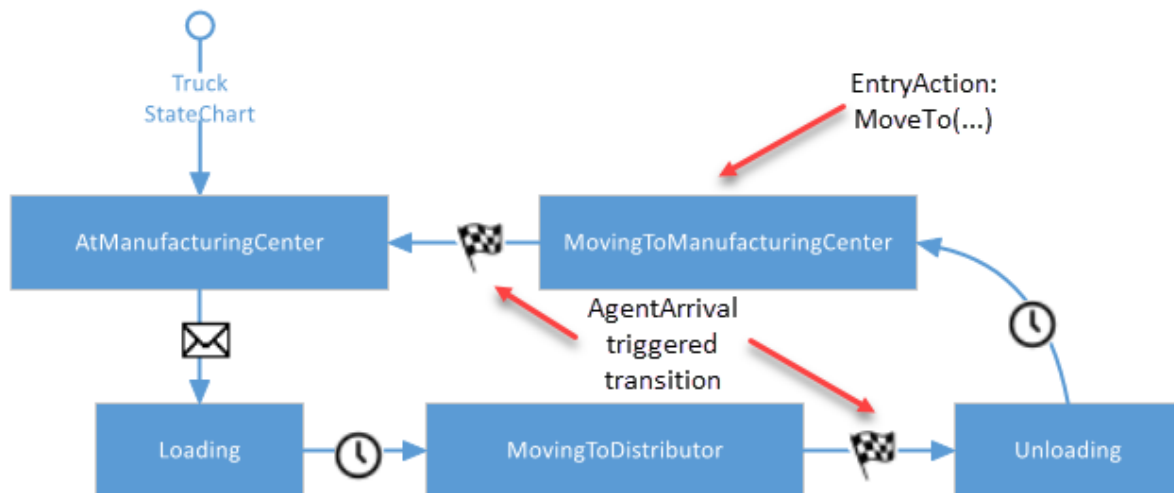


Abbildung 23: AgentArrival-Transition

## 3.3.3 Branch

Ein Branch repräsentiert eine Verzweigung oder eine Kombination. Mit einem Branch kann eine Transition, die mehr als einen Zielzustand aufweist, erstellt werden. Ebenso können mehrere Transitions zu einer gemeinsamen Action zusammengeführt werden.

Für Transitions, die von einem Branch wegführen, kann zwischen «Condition» und «Default» gewählt werden. Wird «Condition» gewählt, kann für die Transition eine Condition definiert werden. Wenn ein Agent auf einen Branch trifft, wird die Action des Branch ausgeführt. Anschliessend werden für alle Transitions, für die eine «Condition» konfiguriert wurde, diese entsprechend ausgewertet. Die erste Transition, deren Condition `true` ergibt, wird anschliessend ausgeführt. Wird keine Condition auf `true` ausgewertet, wird die «Default-Transition» ausgeführt. Demzufolge kann nur eine Transition als «Default-Transition» deklariert werden. Die Reihenfolge, in der die Conditions der Transitions ausgewertet werden, ist nicht deterministisch. Folglich sollte vermieden werden, dass gleichzeitig die Conditions von mehreren Transitions `true` ergeben, da in diesem Fall das Verhalten des Agenten nicht deterministisch ist.

Der Agent verbleibt niemals in einem Branch.



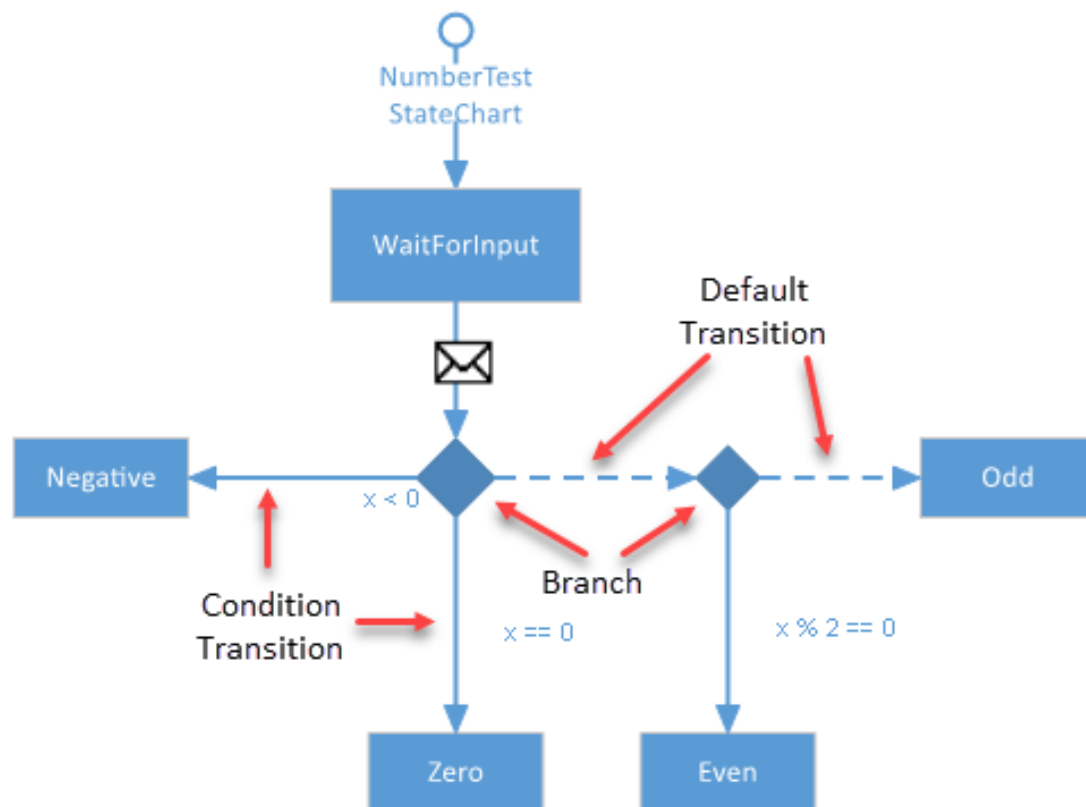


Abbildung 24: Branch

Es stehen folgende Einstellungen zur Verfügung:

Einstellung	Beschreibung
<b>Action</b>	Diese Aktion wird ausgeführt, wenn der Branch aktiviert wird (unterstützt C#-Expressions).

Tabelle 13: Einstellungen für Branch

### 3.3.4 State und Composite State

Das State-Objekt repräsentiert einen bestimmten Zustand eines Agenten. Gelangt ein Agent in einen bestimmten Zustand, wird das entsprechende State-Objekt aktiviert und dessen Entry-Action ausgeführt. Verlässt ein Agent den Zustand wieder, wird die Exit-Action ausgeführt. Ein State kann entweder einfach (Simple state) oder, wenn er weitere Zustände beinhaltet bzw. zusammenfasst, zusammengesetzt (Composite state) sein. Der Agent kann sich jedoch nur in einem einfachen State befinden. Es gilt jedoch die Ausführungsreihenfolge der Entry- bzw. Exit-Actions zu beachten (siehe Abschnitt 3.3.4.1).

Es stehen folgende Einstellungen zur Verfügung:

Einstellung	Beschreibung
<b>Entry action</b>	Diese Aktion wird ausgeführt, wenn ein Agent den betreffenden Zustand annimmt (unterstützt C#-Expressions).
<b>Exit action</b>	Diese Aktion wird ausgeführt, wenn ein Agent den betreffenden Zustand verlässt (unterstützt C#-Expressions).

Tabelle 14: Einstellungen für State-Objekt

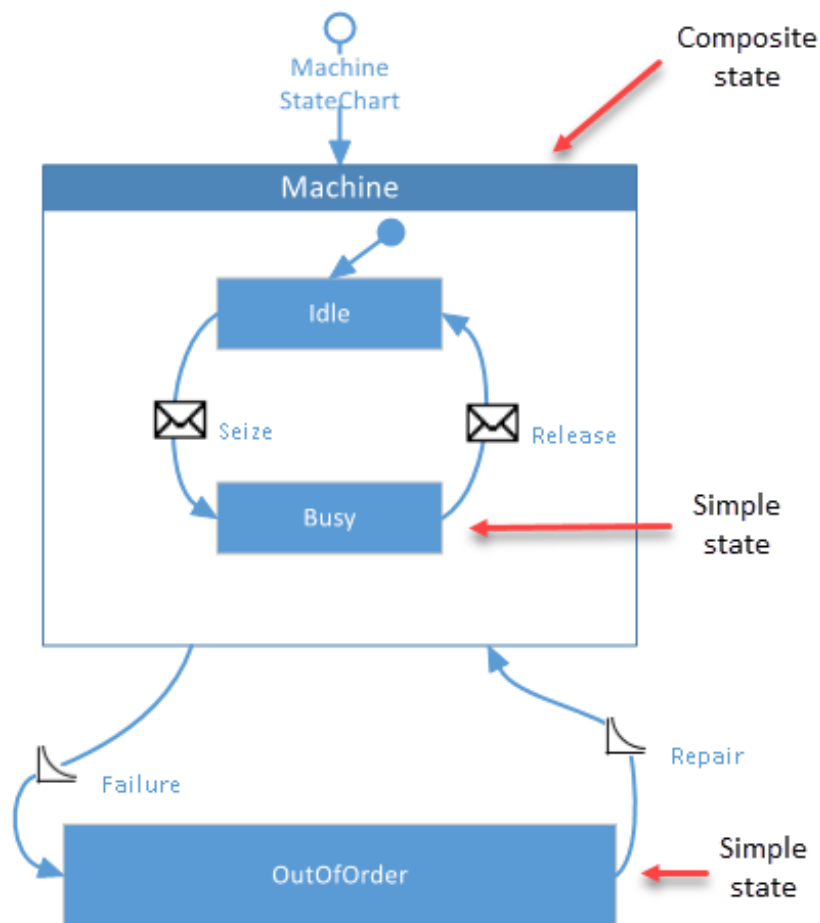


Abbildung 25: Simple bzw. Composite state

### 3.3.4.1 Ausführungsreihenfolge

Nachfolgender Algorithmus kommt für die Bestimmung der Ausführungsreihenfolge der einzelnen Aktionen zur Anwendung, wenn der Zustand gewechselt wird:

- 1) State Exit-Actions startend mit dem aktuellen «SimpleState» bis zum äussersten «CompositeState».
- 2) Action der Transition.
- 3) State Entry-Actions startend mit dem äussersten «CompositeState» bis zum neuen «SimpleState».

Für die Ausführung der mit den States bzw. Transitions assoziierten Actions, wird keine Modelzeit benötigt.

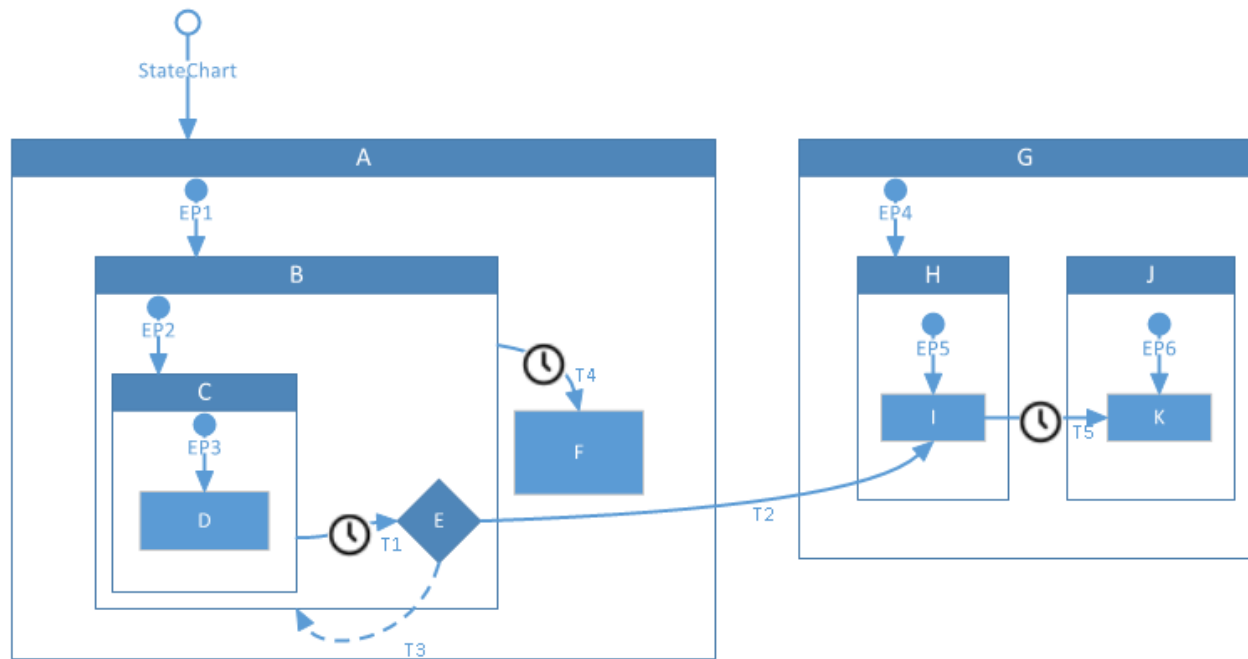


Abbildung 26: Szenario für Ausführungsreihenfolge

Bezogen auf das Beispiel in Abbildung 26 ergibt sich unter Annahme von D als aktuellen Zustand und T1 als nächste Transition, folgende Ausführungsreihenfolge:

- 1) Exit-Action von State D
- 2) Exit-Action von State C
- 3) Action von Transition T1
- 4) Action von Branch E

Anschließend wird, abhängig von der Condition der Transition T2, entweder die Transition T2 oder T3 gewählt. Angenommen, es wird T3 gewählt, ergibt sich folgende Aktivierungsreihenfolge:

- 5) Action von Transition T3
- 6) Action von EntryPoint EP2 (Exit- bzw. Entry-Action von B werden nicht ausgeführt, da der State nicht verlassen wird)
- 7) Entry-Action von State C
- 8) Action von EntryPoint EP3
- 9) Entry-Action von State D

Angenommen es wird bei Punkt 3) nicht T3 sondern T2 gewählt, ergibt sich folgende Aktivierungsreihenfolge:

- 5) Exit-Action von State B
- 6) Exit-Action von State A (Actions von F werden nicht ausgeführt)
- 7) Action von Transition T2
- 8) Entry-Action von State G
- 9) Entry-Action von State H
- 10) Action von EntryPoint EP5
- 11) Entry-Action von State I

Bei Aktivierung der Transition  $T_5$  lautet die weitere Aktivierungsreihenfolge:

- 12) Exit-Action von State  $I$
- 13) Exit-Action von State  $H$
- 14) Action von Transition  $T_5$
- 15) Entry-Action von State  $J$
- 16) Action von EntryPoint  $EP_6$
- 17) Entry-Action von State  $K$

### 3.3.5 History State

Ein CompositeState kann «Shallow History States» sowie «Deep History States» beinhalten.

- Ein «Shallow History State» ist eine Referenz auf den zuletzt besuchten Zustand auf derselben Hierarchieebene in einem Composite State.
- Ein «Deep History State» ist eine Referenz auf den zuletzt besuchten Simple State in einem Composite State. Dieser kann auf der gleichen oder einer tieferen Hierarchieebene liegen als der History State.

Ein History State kann keine ausgehenden Transitionen aufweisen. Gelangt der Agent in einen History State, wird die Ausführung unmittelbar im referenzierten «echten» State fortgesetzt.

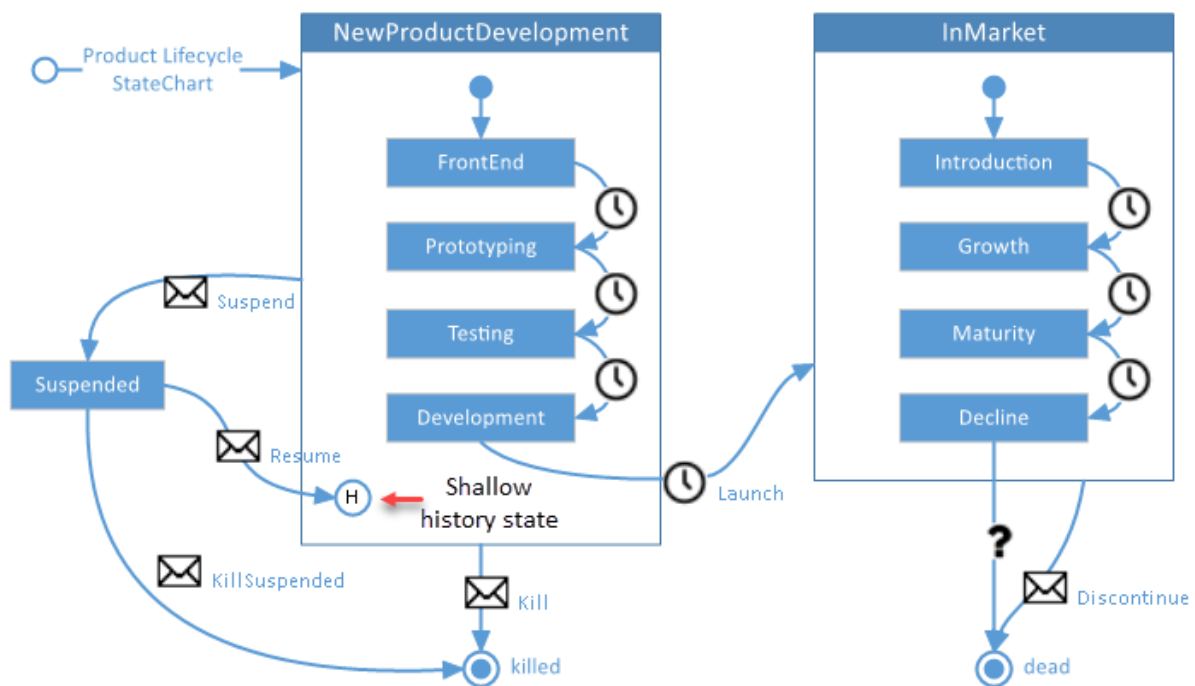


Abbildung 27: History State

Es stehen folgende Einstellungen zur Verfügung:

Einstellung	Beschreibung
<b>History type</b>	Hier kann definiert werden, ob es sich um einen Shallow oder einen Deep History State handelt.
<b>Action</b>	Diese Aktion wird ausgeführt, wenn der History State aktiviert wird (unterstützt C#-Expressions).

Tabelle 15: Einstellungen für History State

Abbildung 28 zeigt den Unterschied zwischen «Shallow History State» und «Deep History State» auf. Angenommen der zuletzt besuchte Zustand im Composite State  $C_1$  ist  $B$ . Sobald der Agent den «Shallow History

State» erreicht (linker Teil der Abbildung), wird die Ausführung beim Zustand C2 fortgesetzt. Dies entspricht dem zuletzt besuchten Zustand auf derselben Hierarchieebene wie der History State. Wenn der Agent jedoch den «Deep History State» erreicht (rechter Teil der Abbildung), wird die Ausführung beim Zustand B fortgesetzt. Dies entspricht dem zuletzt besuchten Simple State auf der gleichen oder einer tieferen Hierarchieebene wie der History State.

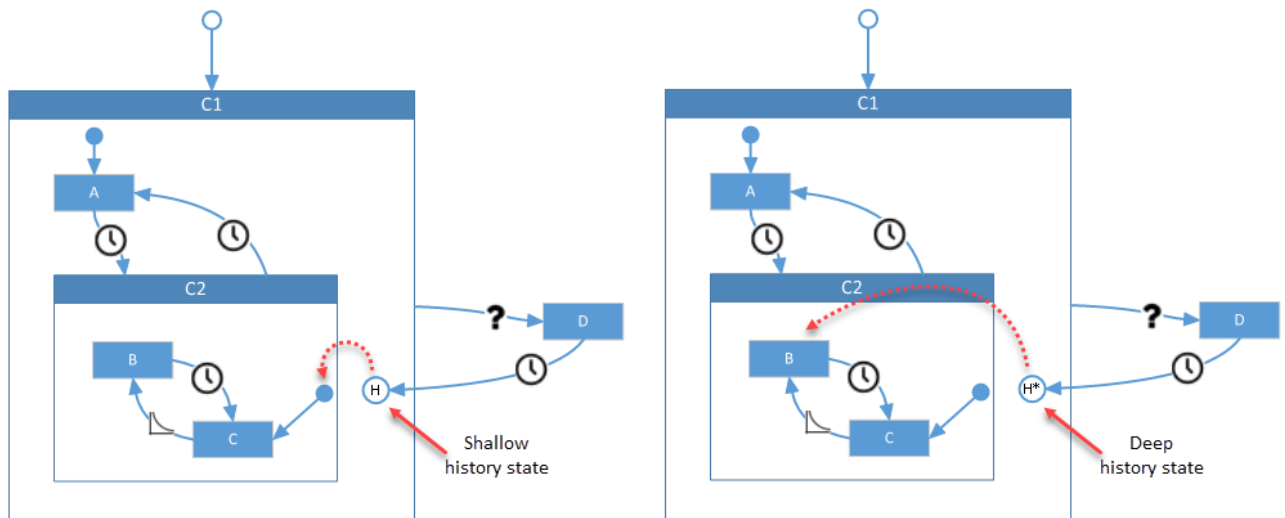


Abbildung 28: Shallow vs. Deep History State

### 3.3.6 Entry Point

Ein Entry Point wird verwendet, um den initialen State eines StateCharts festzulegen. In jedem StateChart muss genau ein EntryPoint existieren. Wird bei einem Entry Point die Eigenschaft «Main Entry Point» auf «Nein» gesetzt, kann das Element als Initial State Pointer innerhalb eines Composite State eingesetzt werden. Dies definiert bei einem Übergang in einen Composite State den initialen Zustand im Composite State. Auf jeder Ebene in einem Composite State sollte genau ein Initial State Pointer existieren.

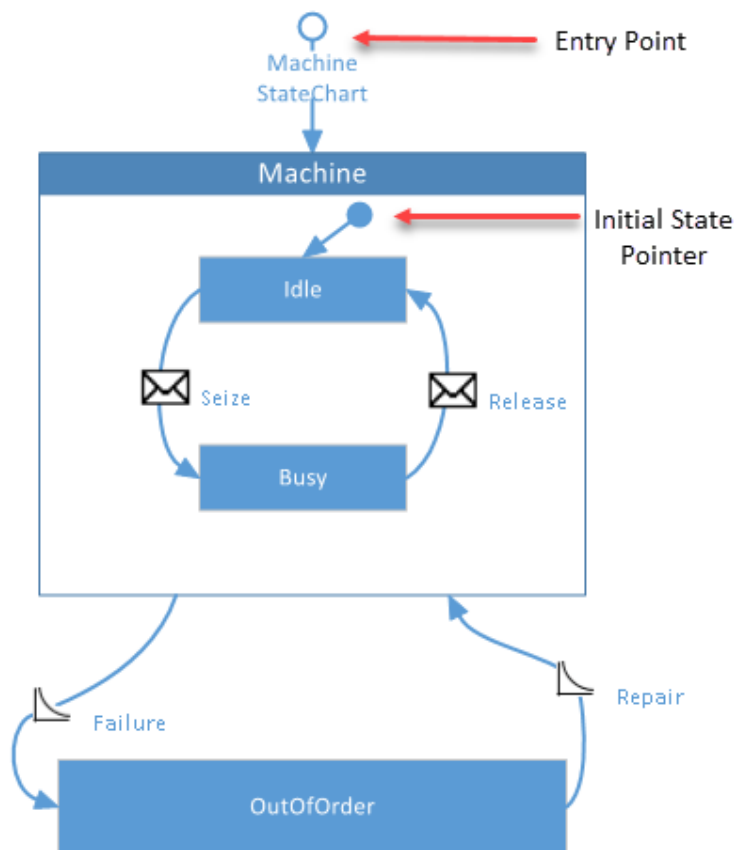


Abbildung 29: Entry Point und Initial State Pointer

Es stehen folgende Einstellungen zur Verfügung:

Einstellung	Beschreibung
<b>Main entry point</b>	Hier kann definiert werden, ob es sich um einen Entry Point (=ja) oder um einen Initial State Pointer (=nein) handelt.
<b>Action</b>	Diese Aktion wird ausgeführt, wenn der Entry Point bzw. der Initial State Pointer aktiviert wird (unterstützt C#-Expressions).

Tabelle 16: Einstellungen für Entry Point-Objekt

## 3.3.7 Final State

Der Final State repräsentiert den Endpunkt in einem StateChart. Gelangt der Agent in einen Final State, wird als letztes die Action des Final States sowie der Event `OnDestroy` (siehe Tabelle 18) des Agenten ausgeführt. Danach geschieht nichts mehr. Es ist nicht möglich, einen Final State mit einer weiteren Transition wieder zu verlassen.

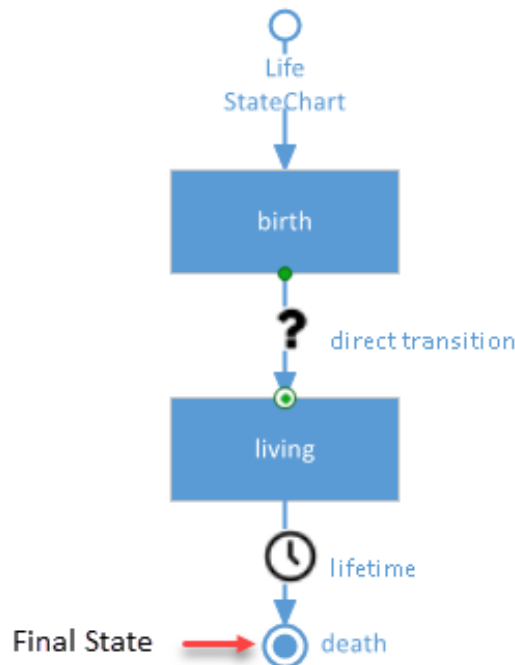


Abbildung 30: Final State

Es stehen folgende Einstellungen zur Verfügung:

Einstellung	Beschreibung
Action	Diese Aktion wird ausgeführt, wenn der Entry Point bzw. der Initial State Pointer aktiviert wird (unterstützt C#-Expressions).

Tabelle 17: Einstellungen für Final State

## 3.3.8 Agent Event Actions

Ein Agent definiert die Standardevents gemäss Tabelle 18:

Event	Beschreibung
OnStartup	Dieser Event wird ausgeführt, sobald für einen Agenten die StateMachine geladen und gestartet wurde. Dies passiert genau dann, wenn er zum ersten Mal in einem ComplexGateway eintrifft (siehe auch Kapitel 4.2). Dieser Event eignet sich für zusätzliche Initialisierungsschritte des Agenten.
OnDestroy	Dieser Event wird ausgeführt, wenn der Agent in einen FinalState wechselt.
OnArrival	Dieser Event wird ausgeführt, wenn der Agent an seinem Ziel ankommt. Mit Hilfe einer Transition kann ebenfalls eine Action definiert werden, die bei Erreichung des Ziels ausgeführt wird (siehe Abschnitt 0).
OnBeforeStep	Dieser Event wird bei jedem Zyklus der DES aufgerufen, in dem die Agentenlogik ausgeführt wird (genauer: vor der Zyklusausführung). Weitere Informationen bzgl. der DES und des Zyklus finden sich in Kapitel 0.
OnStep	Analog OnBeforeStep – jedoch wird dieser Event nach dem Zyklus aufgerufen.

Tabelle 18: Agent Standardevents

## 3.3.9 Functions

Es besteht die Möglichkeit, Functions zu definieren, die wiederum in C#-Expressions verwendet werden können. Dies bietet das Potential um einerseits komplexe Logik zu programmieren und andererseits Logik zu zentralisieren.

Informationen zum Hinzufügen und Entfernen von Functions sind im Abschnitt 3.1.4.6 zu finden.

In Abbildung 31 ist eine Übersicht über die globale Konfiguration von Functions ersichtlich. In Tabelle 19 sind die einzelnen Konfigurationseinstellungen anschliessend beschrieben.

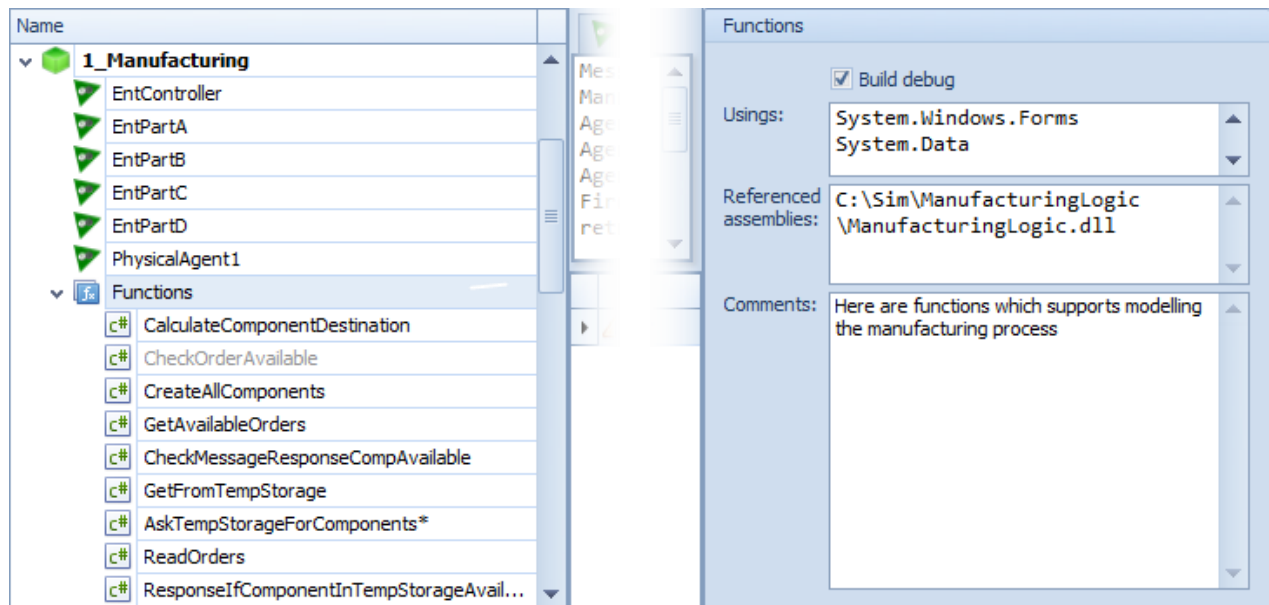


Abbildung 31: Globale Konfiguration für Functions

Einstellung	Beschreibung
<b>Build debug</b>	Mit diesem Flag kann gewählt werden, ob der kompilierte Code Debug-Symbole enthalten soll oder nicht.
<b>Usings</b>	<p>Hier können zusätzliche Usings angegeben werden, die bei der Generierung der Codedatei eingefügt werden sollen.</p> <p>Standardmässig werden die folgenden Usings hinzugefügt:</p> <pre>using System; using System.Linq; using SimioAgentLibrary.Types; using System.Collections.Generic; using SimioAgentLibrary.IntelligentObjects.Agent; using SteeringBehavior.Agent.StateMachine.API;</pre>
<b>Referenced assemblies</b>	Erlauben es, zusätzliche Assemblies zu referenzieren. Dies ist dann sinnvoll, wenn die Agentenlogik z.B. mit Visual Studio oder einer anderen Entwicklungsumgebung implementiert werden soll. Diese Logik kann dann mit Visual Studio o.ä. in eine Assembly kompiliert werden, welche hier als Referenz angegeben werden kann. In der Folge kann diese Logik beliebig verwendet werden.
<b>Comments</b>	Kommentare können hier eingetragen werden.

Tabelle 19: Globale Konfigurationseinstellungen

Abbildung 32 zeigt die Implementierung und die Konfiguration einer einzelnen Function. Im linken Bereich wird die Function implementiert. Dies ist mit Standard C#-Code möglich. Dabei wird C# 5.0 (.NET Framework 4.5) unterstützt. Der Grund, weshalb C# 6.0 bzw. C# 7.0 nicht unterstützt wird, hängt damit zusammen, dass Micro-



soft mit der Einführung von C# 6.0 die Architektur für «hosted compilation» geändert hat. Diese neue Architektur ist jedoch instabil und langsam<sup>1</sup>. Deswegen wird bewusst C# 5.0 verwendet. Für die «hosted compilation» wird CS-Script verwendet. Für weitere Informationen siehe Kapitel 4.1.

Im rechten Bereich kann die Function konfiguriert werden. Es muss ein Name vergeben werden. Zusätzlich kann gewählt werden, ob die Function für den Buildprozess sowie die Simulation ignoriert werden soll. Anschliessend können die Argumente der Function in einer Liste definiert werden. Jedes Argument benötigt einen Namen und einen Typ, wobei der Typ bekannt sein muss. Evtl. ist es nötig, referenzierte Assemblies oder Usings hinzuzufügen (siehe Tabelle 19). Falls es sich nicht um eine void-Funktion handelt, kann bei «Returntype» der entsprechende Rückgabotyp definiert werden. Abschliessend besteht die Möglichkeit, eine Beschreibung einzufügen.

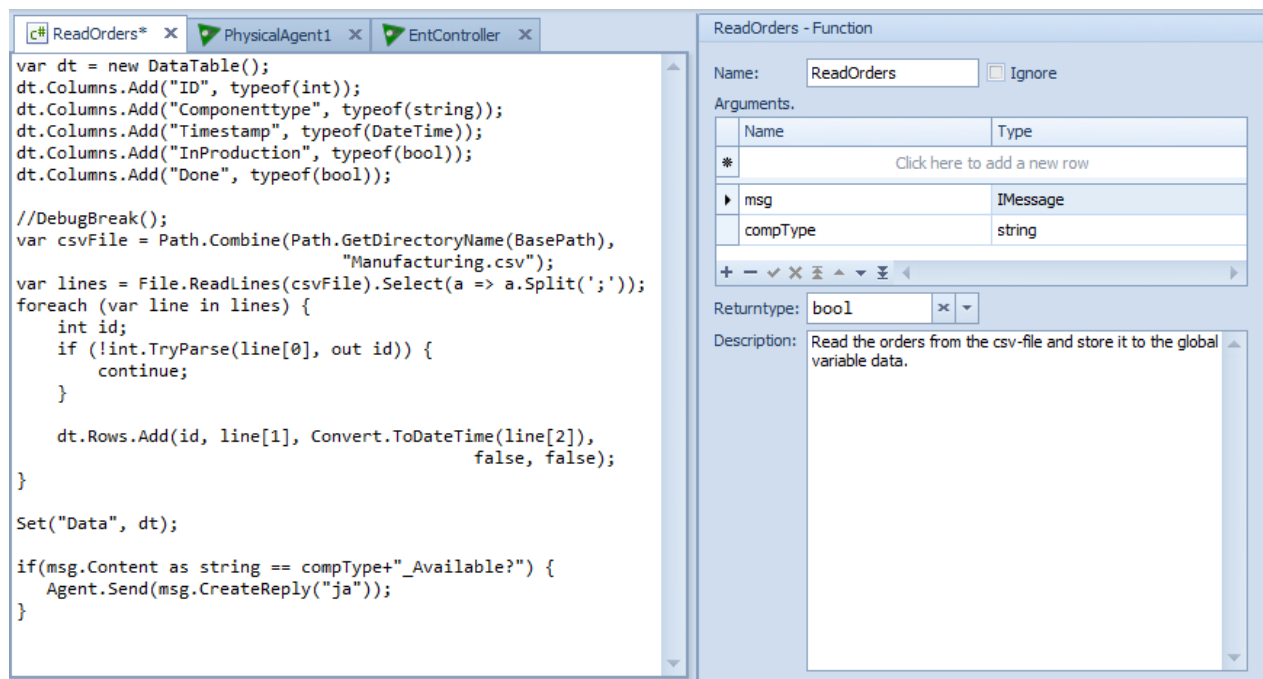


Abbildung 32: Implementierung und Konfiguration einer einzelnen Funktion

Listing 1 zeigt die Signatur der Function `ReadOrders`, welche auf Basis der Konfiguration in Abbildung 32 erstellt wird.

```
Public bool ReadOrders(IMessage msg, string compType) {
    /* ... */
}
```

Listing 1: Signatur der Funktion `ReadOrders()`

### 3.3.10 Agent Movement

Für die Bewegung der Agenten im Raum kann dieser auf drei möglichen Arten erfasst werden:

- Continuous Space
- GIS Space (siehe Abschnitt 3.3.10.1)
- Discrete Space (siehe Abschnitt 3.3.10.2)

In der aktuellen Version der `SimioAgentLibrary` ist ausschliesslich die Erfassung des Raums als Continuous Space implementiert.

<sup>1</sup> CS-Script: Support for C# 6 syntax - <https://github.com/oleg-shilo/cs-script/wiki/C%23-6-support>

Wie in Kapitel 2.2 aufgezeigt, ist das Objekt «ComplexGateway» dafür verantwortlich, Agenten vom Netzwerk der DES in den Freespace zu transferieren. Sobald sich ein Agent im Freespace befindet, kann er sich kontinuierlich bewegen, d.h. der Raum wird wie bereits erwähnt als Continuous Space interpretiert. Die SimioAgentLibrary stellt hierfür umfassende Funktionalität zur Verfügung:

- Es kann der aktuelle Standort festgelegt bzgl. abgerufen werden.
- Der Agent kann kontinuierlich mit einer bestimmten Geschwindigkeit von einem Ort zum anderen bewegt werden.
- Für die kontinuierliche Bewegung an ein bestimmtes Ziel kann eine (beliebig komplexe) PathFinding-Logik mitgegeben werden (siehe Listing 2).
- Bei der Ankunft kann eine gewünschte Aktion ausgeführt werden.

Die entsprechende Funktionalität wird durch die API zur Verfügung gestellt. Für weitere Informationen siehe Kapitel 6.1.

Wenn eine komplexe Bewegungslogik implementiert werden soll, kann die API-Methode gemäss Listing 2 verwendet werden.

```
public interface IAgent {  
    void Move(IMovingLogic movingLogic);  
}
```

**Listing 2: API für komplexe Bewegungslogik**

Die Methode `Move` von Listing 2 ermöglicht die Übergabe der komplexen Bewegungslogik in Form eines Objekts, welches das Interface `IMovingLogic` (siehe Listing 3) implementiert.

```
public interface IMovingLogic {  
    bool TryGetNewTimeInterval(out double newTimeInterval);  
    void DoSteer(ISTeering steering, IAgent agent);  
    MovingAction CurrentAction { get; }  
}
```

**Listing 3: Interface IMovingLogic**

Die SimioAgentLibrary ruft die Methode `DoSteer` auf. Hier kann die gewünschte Action (siehe Abschnitt 4.1.3.2) und falls erforderlich ein neues TimeInterval berechnet und gesetzt werden. Für die Implementierung von Klassen bietet der StateChart-Designer keine implizite Möglichkeit. Stattdessen kann ein beliebiger C#-Code Editor (z.B. Visual Studio) verwendet werden. Die Klasse wird dann in eine ClassLibrary kompiliert, welche einfach referenziert werden kann (siehe Tabelle 19 – «Referenced assemblies»). Anschliessend kann das Objekt in einer Function instanziiert, sowie mit der Methode `IAgent.Move` übergeben und verwendet werden. Auf diese einfache Art und Weise kann eine beliebig komplexe Bewegungslogik für Agenten implementiert und verwendet werden.

Die SimioAgentLibrary enthält das Wander-Sample von Simio als MovingLogic. Zu finden ist diese Implementierung in der Klasse `...API.MovingLogic.Wander`.

### 3.3.10.1 GIS Space

Für die Bewegung im GIS Space ist eine Karte erforderlich. Zusätzlich erfordert dies die Implementierung der Bewegung auf vorgegebenen Routen (Strassen, Wege usw.). Die Verwendung bestimmter Routen hängt wiederum von der Art des Agenten ab. Z.B. kann ein LKW keine Fahrradwege benutzen usw. Wie bereits erwähnt, wurde diese Bewegungsart in der aktuellen Fassung nicht implementiert.

## 3.3.10.2 *Discrete Space*

Bei dieser Art der Bewegung wird der Raum in Zellen eingeteilt. In der Folge können sich Agenten von Zelle zu Zelle bewegen, vorausgesetzt die Zielzelle ist noch nicht besetzt. Diese Bewegungsart wurde ebenfalls nicht implementiert.

## 4 Runtime-Implementierung der AgentLibrary in Simio

Für die Implementierung wurden folgende Versionen verwendet:

Software	Version
Simio	Version 10.165.15383 Student Edition
Visual Studio	Enterprise 2017, Version 15.7.3
WiX Toolset <sup>2</sup> (wird für die Erstellung des Installers benötigt)	Version 3.11

### 4.1 Implementierung der StateMachine

Für die technische Implementierung der StateMachine auf Basis des StateChart, welches im StateChart-Designer erstellt wird, kommt die Library «Stateless»<sup>3</sup> zum Einsatz. «Stateless» ist eine einfache Library, welche die Erstellung und den Betrieb von State Machines mit C# unterstützt. Die grosse Stärke der Library liegt in ihrem generischen Ansatz. Dies ermöglicht es, States sowie Transitions durch beliebige Objekte zu repräsentieren. So können die Objekte aus dem StateChart-Designer (siehe Abbildung 15) direkt als States bzw. Transitions verwendet werden. Um trotzdem eine Abstraktion bzw. Entkopplung zum StateChart-Designer zu erreichen, implementieren die Objekte jeweils konkrete Interfaces. Diese finden nun in der Implementierung der StateMachine wieder Verwendung (siehe Abschnitt 4.1.1 bzw. Abbildung 33 – `List<IState>` bzw. `List<ITransition>`).

In Listing 4 ist beispielhaft die Instanziierung einer StateMachine mit anschliessender Konfiguration eines States und einer Transition mit «Stateless» abgebildet.

```

1. IState state = ...
2. IState destination = ...
3. var stateMachine = new StateMachine<IState, ITransition>();
4. var configuration = stateMachine.Configure(state);
5. var entryAction = this.codeProvider.GetAction(obj, state,
    nameof(IEntryExitState.EntryAction));
6. configuration.OnEntry(() => { entryAction(); });
7. var exitAction = this.codeProvider.GetAction(obj, state,
    nameof(IEntryExitState.ExitAction));
8. configuration.OnExit(() => { exitAction(); });
9. var guard = this.codeProvider.GetAction<bool>(obj, transition.Id,
    nameof(ITransition.Guard));
10. configuration.PermmitIf(transition, destinationState, () => guard());

```

Listing 4: Instanziierung und Konfiguration der StateMachine

In Zeile 3 wird die StateMachine erzeugt. Für die Deklaration der Typen für States bzw. Transitions wird `IState` bzw. `ITransition` verwendet. Anschliessend wird der konkrete State `state` konfiguriert. Der State besitzt eine Entry-Action sowie eine Exit-Action. Diese werden vom CodeProvider als Delegates (`entryAction` bzw. `exitAction`) zur Verfügung gestellt und können nun für den State entsprechend mit `OnEntry()` bzw. `OnExit()` konfiguriert werden. Die Actions werden in der Folge im Zuge der Simulation ausgeführt, wenn ein Agent zum entsprechenden State wechselt.

Ab Zeile 9 wird eine Transition konfiguriert. Der CodeProvider liefert wiederum einen Delegate für die Guard-Action (`guard`). Anschliessend wird die Transition zum State `destinationState` konfiguriert. Diese wird ausgeführt, sobald die Guard-Action `true` zurückliefert.

Weitere Informationen zum CodeProvider finden sich im Abschnitt 4.1.2.

<sup>2</sup> WiX Toolset – Windows Installer XML (Microsoft Reciprocal License)

<http://wixtoolset.org/>

<sup>3</sup> Stateless – A simple library for creating state machines in C# code (Apache License, Version 2.0)

<https://github.com/dotnet-state-machine/stateless>

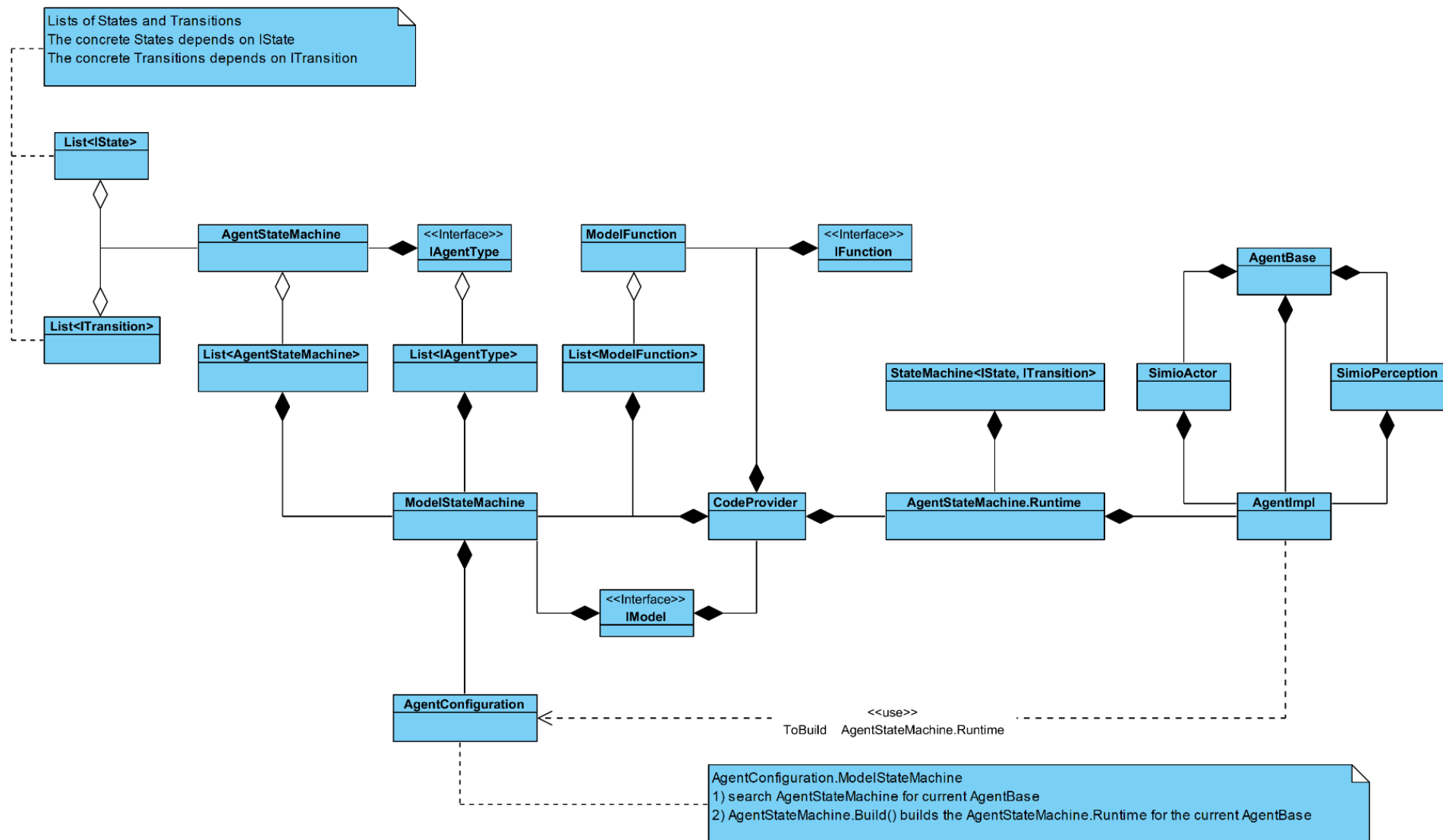


Abbildung 33: Klassendiagramm für Implementierung der StateMachine

Für die Actions der States sowie Transitions (Entry-Action, Exit-Action, Transition-Action usw.) kommen die C#-Functions bzw. C#-Expressions, die im StateChart-Designer definiert wurden zum Einsatz. Diese werden On-the-fly validiert und kompiliert. Zu diesem Zweck wird die Library CS-Script<sup>4</sup> als «hosted compilation»-Library verwendet (siehe Abschnitt 4.1.2).

## 4.1.1 Ablauf

Wird die Simulation gestartet, werden die Elemente der StateMachine geladen. Anschliessend werden die C#-Functions bzw. C#-Expressions kompiliert. Wird im Laufe der Simulation ein Agent instanziiert (genauer sein SteeringBehavior nachdem der Agent zu einem ComplexGateway gelangt), wird seine StateMachine erstellt und gestartet. Diese steuert anschliessend das Verhalten des Agenten. Der Ablauf vom Start der Simulation bis zum Start der StateMachine eines Agenten ist in Abbildung 34 dargestellt.

Der Beschrieb der Verhaltenssteuerung durch die StateMachine bei einem Zyklus ist in Abschnitt 4.1.3 zu finden.

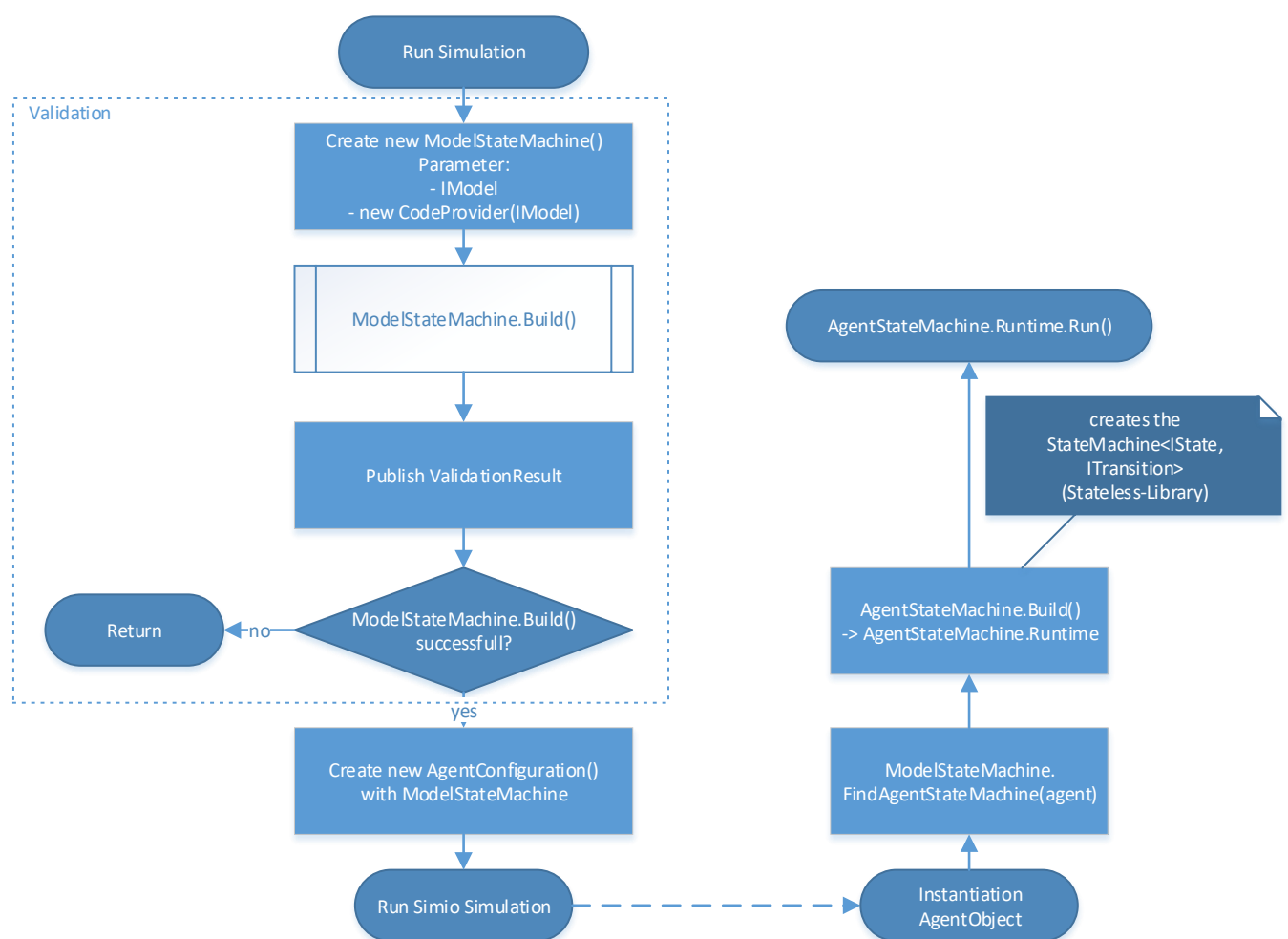


Abbildung 34: Start Simulation und Erstellung StateMachine

Der Subprozess «ModelStateMachine.Build()» ist in Abbildung 35 im Detail ersichtlich.

Wenn nur das Modell validiert werden soll (d.h. ohne Start der Simulation), so wird im Ablaufdiagramm von Abbildung 34 ausschliesslich der mit «Validation» markierte Bereich ausgeführt.

<sup>4</sup> CS-Script – The C# Script Engine (Lizenz: MIT)

<https://github.com/oleg-shilo/cs-script> bzw. <http://www.csscript.net/>

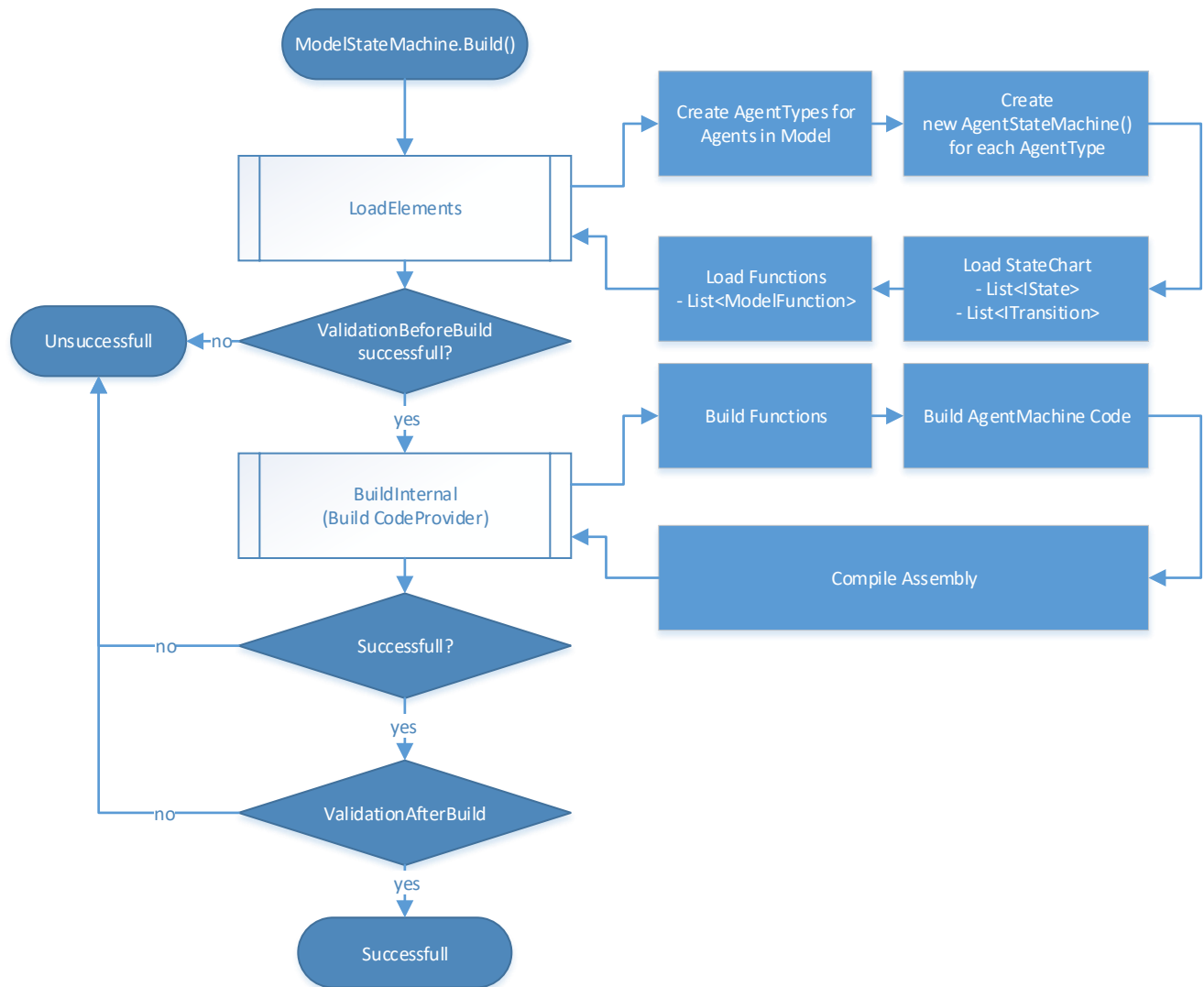


Abbildung 35: Detaillierung des Subprozess ModelStateMachine.Build()

## 4.1.2 CodeProvider

Die Komponente «CodeProvider» ist zur Laufzeit für die Bereitstellung der kompilierten C#-Functions sowie C#-Expressions verantwortlich. Für die Kompilierung werden alle Sourcecodes gesammelt und in eine zentrale C#-Datei geschrieben. Es wird pro AgentType (d.h. pro StateChart) eine Klasse erstellt. Diese beinhaltet alle Actions, der im StateChart enthaltenen States sowie Transitions. Abschliessend wird zusätzlich eine Klasse erstellt, welche alle im Modell definierten C#-Functions enthält.

Im nächsten Schritt wird diese Klasse mit Hilfe von CS-Script kompiliert und als Assembly ins Memory geladen. Diese kann nun in der Folge wie ein «normales» Assembly verwendet werden. D.h. es können dynamisch Klassen und Methoden geladen werden. Für ein einfacheres Handling werden vom «CodeProvider» entsprechende Methoden zur Verfügung gestellt (siehe auch Listing 4).

Die zentrale C#-Datei kann, falls erwünscht, einfach geöffnet und inspiziert werden (siehe Abschnitt 3.1.4.3). Eine entsprechende Beispieldatei befindet sich auf der Projektdaten-CD (siehe Anhang E).

Weitere Informationen in Bezug auf die Konfiguration des Compilers (Debug, References usw.) sind in Abschnitt 3.3.9 zu finden.

## 4.1.3 SteeringBehavior: Agent

Um die Agentenlogik in den Simulationsprozess von Simio integrieren zu können, wurde ein SteeringBehavior implementiert. Dieses wurde als «Agent» benannt. Detaillierte Informationen wie SteeringBehaviors in Simio implementiert werden können, finden sich in [1, S.32ff] sowie in [3, S.23ff].

Die Komponenten des Agent-SteeringBehaviors sind in Abbildung 36 dargestellt. Im Zentrum steht die Klasse `AgentImpl`, welche die Methode `Steer` beinhaltet. Diese Methode wird bei jedem Simulationszyklus aufgerufen.

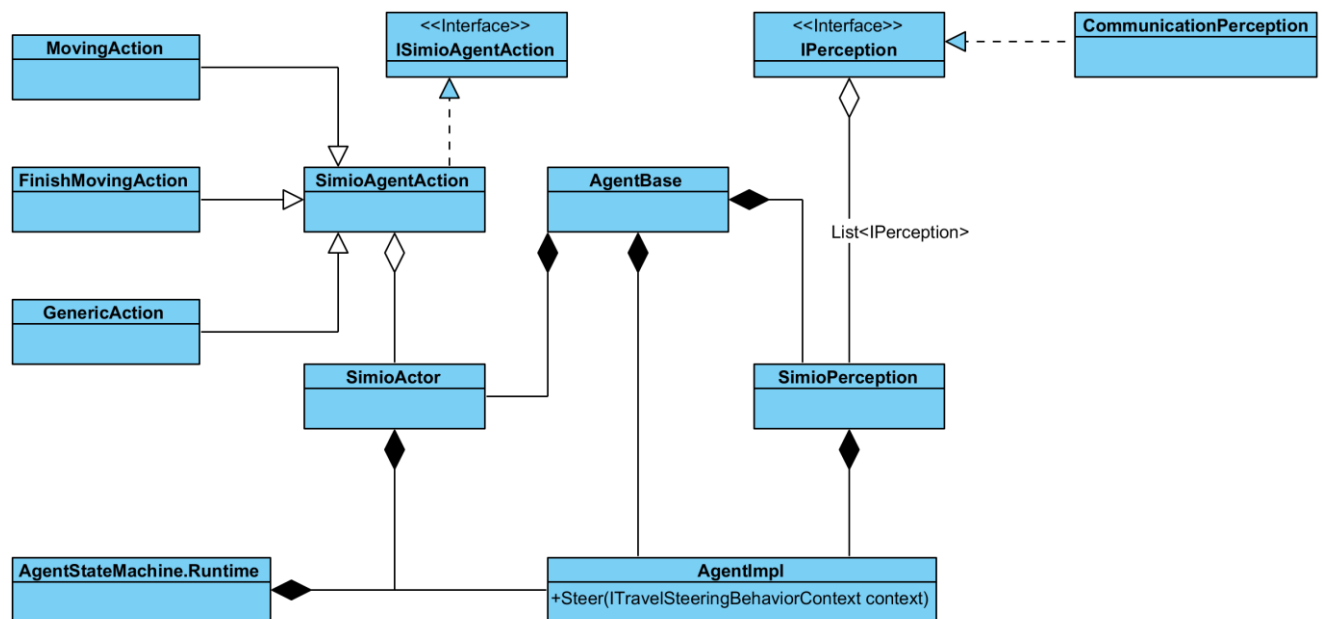


Abbildung 36: Komponenten für Agent-SteeringBehavior

Abbildung 37 zeigt die Konfiguration für das SteeringBehavior. Für das Entity «EntController» wurde das Agent-SteeringBehavior gewählt. Anschliessend kann mit der Einstellung «Update Time Interval» das TimeInterval für den Simulationszyklus konfiguriert werden. Diese Einstellung kann bei Bedarf auch zur Laufzeit geändert werden (siehe Kapitel 6.1 bzw. Anhang A).

Properties: EntController (PhysicalAgent)	
<input checked="" type="checkbox"/> Show Commonly Used Properties Only	
<input type="checkbox"/> Process Logic	
Initial Health	100
Initial Place	null
Agent Sink	null
FireImpact	20
<input type="checkbox"/> Steering Behavior	<b>Agent</b>
<input type="checkbox"/> Update Time Interval	0.5
Units	<b>Minutes</b>

Abbildung 37: Konfiguration für Agent-SteeringBehavior



In Abbildung 38 ist der Ablauf des SteeringBehaviors eines Simulationszyklus abgebildet. Sobald der Agent bei einem ComplexGateway eintrifft, wird er von diesem in den Freespace transferiert. Befindet sich ein Agent im Freespace, kommt sein SteeringBehavior zum Zug. Simio führt nun bei jedem Simulationszyklus (abhängig vom TimeInterval) die Methode `Steer` des SteeringBehaviors aus.

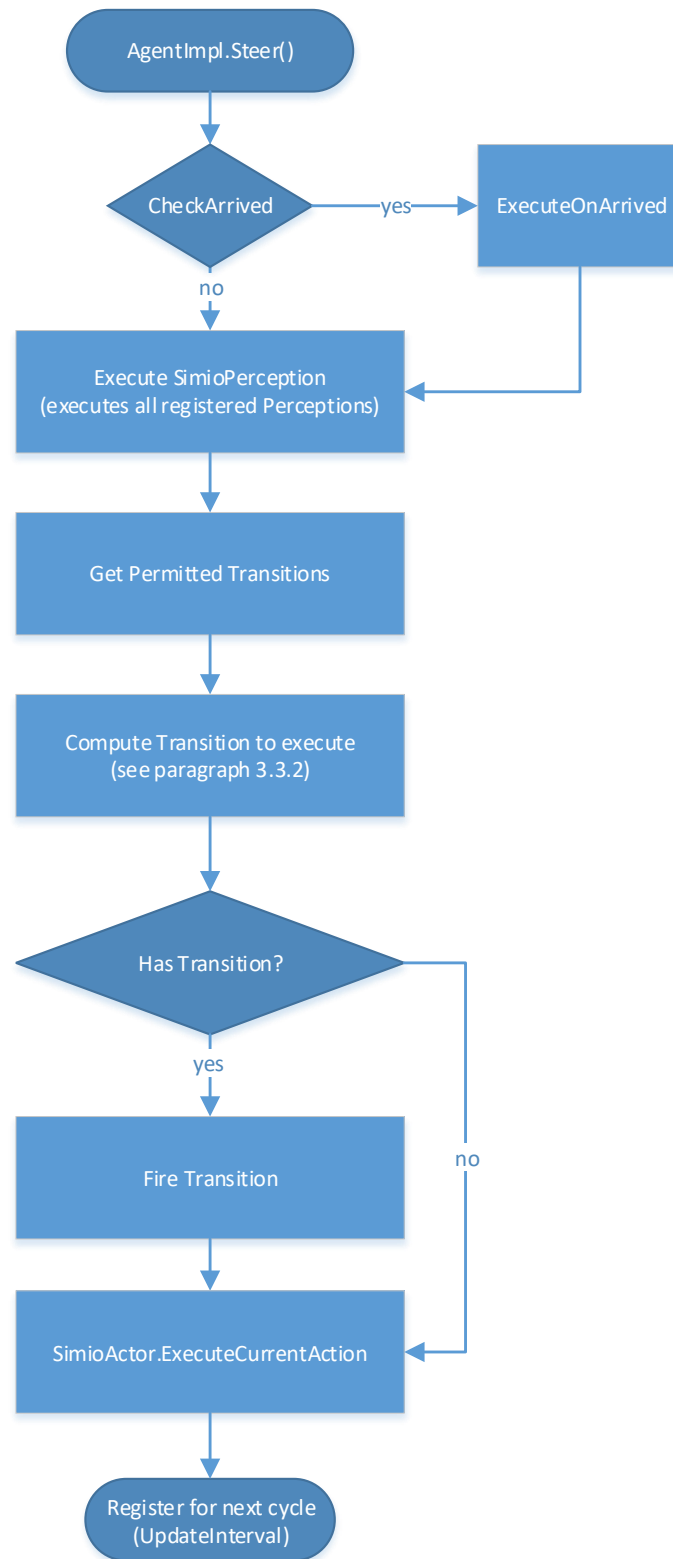


Abbildung 38: Ablauf des SteeringBehaviors bei einem Simulationszyklus

## 4.1.3.1 Perceptions

Zu Beginn des Simulationszyklus werden alle registrierten Perceptions ausgeführt. Diese generieren ggf. Eingabedaten, die im weiteren Verlauf (z.B. in den Actions der States sowie Transitions) verwendet bzw. ausgewertet werden können. Bspw. kann die konkrete Perception `CommunicationPerception` Nachrichten empfangen und für die Verarbeitung zur Verfügung stellen. Die Struktur der Perceptions wurde bewusst offen gestaltet, damit bei Bedarf auf einfache Weise beliebige konkrete Perceptions erweitert werden können.

## 4.1.3.2 Actions

Die Actions der States sowie Transitions können via API-Funktionen (siehe Kapitel 6.1 bzw. Anhang A) Actions aktivieren. Z.B. wird über die Funktion `MoveTo` die Action `MovingAction` aktiviert. Diese initiiert und steuert die Bewegung eines Agenten. Erreicht der Agent sein Ziel, wird die Action `FinishMovingAction` aktiviert, welche die Bewegung beendet. Diese wird am Schluss des Zyklus ausgeführt. Das aktuelle Action-Konzept ermöglicht die Kapselung der (tw. komplexen) Logik für die Ausführung der Action. Es ermöglicht analog zum Konzept der Perceptions ebenfalls bei Bedarf eine einfache und unkomplizierte Erweiterung.

## 4.2 Implementierung ComplexGateway

Damit das Agent-SteeringBehavior (siehe Abschnitt 4.1.3) die Agentenlogik ausführen kann, muss der Agent in den Freespace transferiert werden. Diese Aufgabe übernimmt das ComplexGateway-Objekt. Wie in Kapitel 0 bereits erwähnt, kann sich ein Agent auf einem Simio-Network bewegen und so als Entity Teil einer herkömmlichen DES sein. Soll der Agent seine eigene Logik ausführen können, kann er mit Hilfe eines ComplexGateway in den Freespace transferiert werden. Wird ein Agent das erste Mal in den Freespace transferiert, wird seine StateMachine initialisiert und gestartet (siehe auch Abschnitt 4.1.1). Gleichzeitig wird der globale Event `Agent.OnStartup` ausgeführt (siehe Abschnitt 3.3.8).

In Abbildung 39 ist ein ComplexGateway-Objekt dargestellt. Zusätzlich wurden die einzelnen Bestandteile beschriftet.

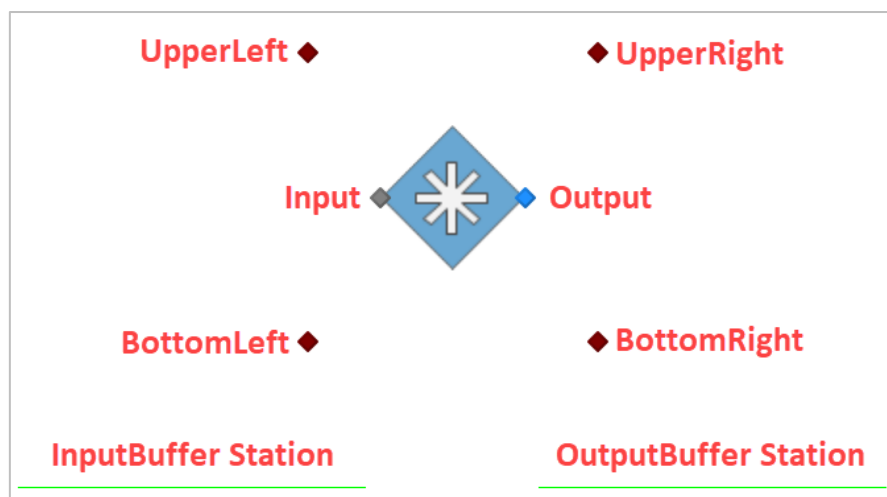


Abbildung 39: ComplexGateway

Ein ComplexGateway besteht aus einem InputBuffer und einem OutputBuffer sowie zugehörigen Warteschlangen. Der ComplexGateway selber weist ebenfalls eine konfigurierbare Kapazität auf, welche bei Bedarf die Anzahl der Agenten, die vom ComplexGateway in den Freespace transferiert werden können, beschränkt. Über die beiden Nodes «Input» sowie «Output» kann der ComplexGateway mit einem Netzwerk verbunden werden. Die Nodes «UpperLeft», «UpperRight», «BottomLeft» sowie «BottomRight» dienen dazu, den ComplexGateway räumlich zu begrenzen. So können z.B. physische «Places» modelliert werden [1, S.26ff – wobei dort das Objekt «Place» dem Objekt «ComplexGateway» hier entspricht].

## 4.2.1 Eigenschaften

In Tabelle 20 sind die Einstellungsmöglichkeiten für das ComplexGateway-Objekt aufgelistet und beschrieben.

Einstellung	Beschreibung
<b>Capacity Type</b>	Methode, mit der die Kapazität des ComplexGateway angegeben wird. Mögliche Werte: Fixed, Work Schedule
<b>Initial Capacity</b>	Wenn Capacity Type = Fixed Die anfängliche feste Kapazität des ComplexGateway. Mögliche Werte: Simio-Expression
<b>Work Schedule</b>	Wenn Capacity Type = Work Schedule Kapazitätsarbeitszeitplan, dem dieser ComplexGateway folgt. Mögliche Werte: Work Schedule Name
<b>Ranking Rule</b>	Diese statische Rangfolge-Regel wird verwendet, um Agenten anzuordnen, die auf die zugewiesene Kapazität dieses ComplexGateway-Objekts warten. Mögliche Werte: First In First Out, Last In First Out, Smallest Value First, Largest Value First
<b>Ranking Expression</b>	Der Ausdruck, der mit der Rangfolge «Smallest Value First» oder «Largest Value First» verwendet wird. Mögliche Werte: Simio-Expression
<b>TimeDistribution</b>	Verteilung der Agenten über die Zeit. Diese Eigenschaft wird nur verwendet, wenn der ComplexGateway als «Place»-Objekt verwendet wird [1, S.26ff]. Mögliche Werte: Repeat Group
<b>ParentComplexGateway</b>	Zusammenfassung mehrerer ComplexGateway-Objekte. Diese Eigenschaft wird verwendet, wenn der ComplexGateway als «Place»-Objekt verwendet wird [1, S.26]. Mögliche Werte: ComplexGateway-Objekt
<b>FacilityName</b>	Das Objekt in dessen Freespace der Agent transferiert werden soll. Wenn diese Eigenschaft nicht angegeben wird, wird standardmässig das unmittelbar dem ComplexGateway übergeordnete Objekt verwendet, welches eine Facility unterstützt. Mögliche Werte: Objekt
<b>Output Buffer – Capacity</b>	Die Anzahl Agenten, die sich gleichzeitig im OutputBuffer des ComplexGateways befinden können. Mögliche Werte: Integer > 0
<b>Input Buffer – Capacity</b>	Die Anzahl Agenten, die sich gleichzeitig im InputBuffer des ComplexGateways befinden können. Mögliche Werte: Integer > 0
<b>Name</b>	Der Name des ComplexGateways
<b>Description</b>	Die Beschreibung des ComplexGateways

Tabelle 20: Einstellungen für ComplexGateway

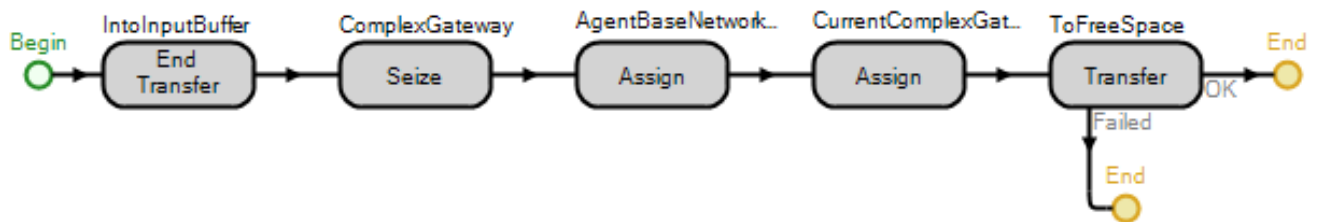
Die erweiterten Eigenschaften (sichtbar, wenn «Show Commonly Used Properties Only» nicht angewählt) entsprechen den Standardeigenschaften der Simio-Objekte und sind entsprechend in [7] ausführlich beschrieben.

## 4.2.2 Prozesse

Die Nodes «Input» und «Output» sind jeweils mit der Station «InputBuffer» bzw. «OutputBuffer» verbunden. D.h. wenn ein Agent im ComplexGateway beim Node «Input» eintrifft, wird er in die Station «InputBuffer» transferiert. Dies löst den Event «InputBuffer.Entered» aus, dem der Prozess «OnEnteredInputBuffer» zugewiesen ist (siehe Abbildung 40). Wird der Agent vom Freespace zurück ins Netzwerk transferiert (Agent.MoveToNode – siehe Tabelle 26) gelangt er zum Node «Output», welcher den Agenten in die Station «OutputBuffer» transferiert, was wiederum analog den Event «OutputBuffer.Entered» auslöst. Die einzelnen Prozessschritte sind in Tabelle 21 bzw. Tabelle 22 beschrieben.

## OnEnteredInputBuffer

⚡ InputBuffer.Entered



## OnEnteredOutputBuffer

⚡ OutputBuffer.Entered

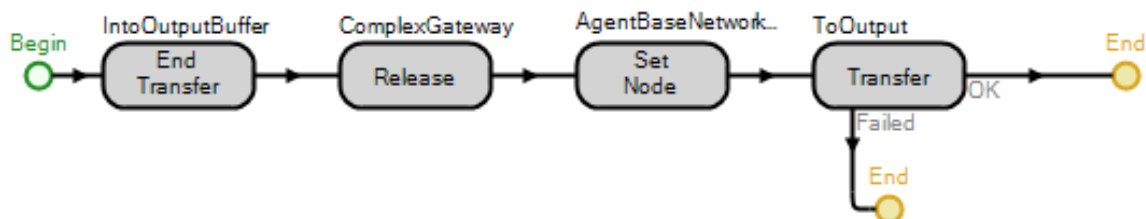


Abbildung 40: Prozesse des ComplexGateway-Objekts

Prozessschritt	Beschreibung
<b>End Transfer – IntoInputBuffer</b>	Zeigt an, dass der Transfer des Agenten in den InputBuffer des ComplexGateway beendet ist.
<b>Seize – ComplexGateway</b>	Für den Agenten wird versucht, eine Kapazität von 1 im ComplexGateway zu reservieren (siehe auch Tabelle 20).
<b>Assign – AgentBaseNetworkPlaceOutputStation</b>	Hier wird die OutputStation in die Simio-State-Variable «NetworkComplexGatewayOutputStation» des AgentBase gespeichert. Diese Information wird benötigt, damit der Agent wieder ins Netzwerk (konkret in die korrekte Station) transferiert werden kann (siehe Kapitel 4.3 bzw. Abbildung 41).
<b>Assign – CurrentComplexGateway</b>	Für die Verwendung in der API wird der aktuelle ComplexGateway in die Simio-State-Variable «CurrentComplexGateway» des AgentBase gespeichert.
<b>Transfer – ToFreeSpace</b>	Der Agent wird in den Freespace transferiert. Zu diesem Zeitpunkt wird sein SteeringBehavior instanziiert und seine Agentenlogik ausgeführt (siehe Abschnitt 4.1.3). Damit sich der Agent in der korrekten Facility befindet (dies beeinflusst die Anzeige sowie sein Koordinatensystem), kann diese in der Eigenschaft «FacilityName» des ComplexGateway-Objekts definiert werden. Diese Eigenschaft wird nun in diesem Prozessschritt berücksichtigt (siehe Tabelle 20).

Tabelle 21: Prozessschritte für Prozess «OnEnteredInputBuffer»

Prozessschritt	Beschreibung
<b>End Transfer – IntoOutputBuffer</b>	Zeigt an, dass der Transfer des Agenten in den OutputBuffer des ComplexGateway beendet ist.
<b>Release – ComplexGateway</b>	Die für den Agenten reservierte Kapazität wird wieder freigegeben.
<b>Set Node – AgentBaseNetworkDestinationNode</b>	Mit der API-Funktion <code>MoveToNode</code> (siehe Kapitel 6.1) kann für den Agenten ein Zielnode angegeben werden. Die Funktion speichert den gewünschten Node in die Simio-State-Variable «NetworkDestinationNode» des AgentBase. Diese Variable wird in diesem Prozessschritt verwendet, um für den Agenten den entsprechenden Ziel-Node zu setzen.
<b>Transfer – ToOutput</b>	Der Agent wird in den Output-Node des ComplexGateway transferiert.

Tabelle 22: Prozessschritte für Prozess «OnEnteredOutputBuffer»

## 4.3 Implementierung AgentBase

Das AgentBase-Objekt wurde in [1, S.30ff] entwickelt. Für die Wiederverwendung wurde der Prozess «OnEnteredFreeSpace» (siehe [1, S.32]) leicht angepasst. Konkret wurde der Prozessschritt «Transfer – TransferToSink» verallgemeinert, damit der Agent nach Abschluss der Ausführung seiner Agentenlogik nicht standardmässig in einer Sink terminiert wird. Stattdessen soll er in die OutputStation des ComplexGateways transferiert werden. Der neue Prozessschritt «Transfer – ToDestination» (Abbildung 41) transferiert den Agenten in die Destination, welche im Prozess «OnEnteredInputBuffer» des ComplexGateway beim Schritt «Assign – AgentBaseNetworkPlaceOutputStation» (Tabelle 21) zwischengespeichert wurde.

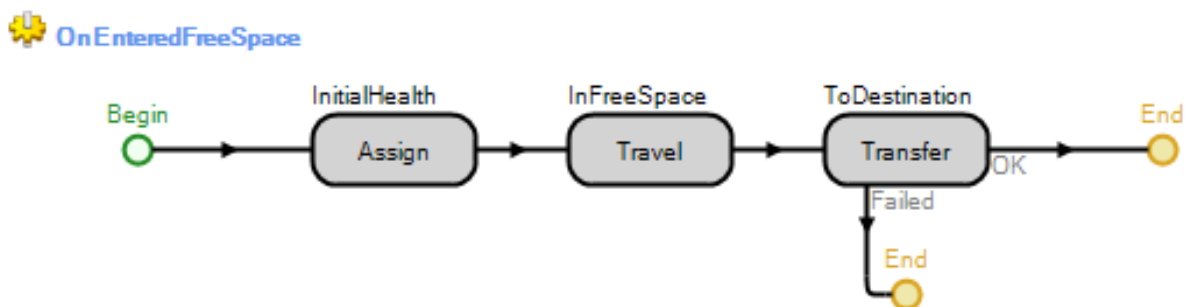


Abbildung 41: Prozess um einen Agenten in den Freespace zu transferieren

## 5 Installationshandbuch der SimioAgentLibrary für Enduser

Für die bequeme Grundinstallation der AgentLibrary wird der Installer «SimioAgentLibraryInstaller.msi» zur Verfügung gestellt (siehe Anhang E). Dieser installiert alle nötigen Komponenten in den folgenden Ordner:

```
%programfiles(x86)%\Simio\UserExtensions\SimioAgentLibrary
```

Nach der Installation steht das AddIn gemäss Kapitel 3.1 zur Verfügung<sup>5</sup>. Anschliessend müssen jedoch noch die Modellierungskomponenten in Simio eingebunden werden. Um dies zu erreichen, stehen zwei Möglichkeiten zur Verfügung:

- 1) Die AgentLibrary mit den Modellierungskomponenten wird global in Simio eingebunden (gilt für alle Projekte).
- 2) Die AgentLibrary mit den Modellierungskomponenten wird für jedes Projekt manuell eingebunden.

Vorgehen für die globale Einbindung:

1. Start von Simio
2. Im Hauptmenü kann über «File» - «Settings» der Dialog für die Einstellungen geöffnet werden. Bei der Einstellung «Additional Libraries To Load» wird nun die AgentLibrary eingetragen. Dazu muss der Pfad

```
%programfiles(x86)%\Simio\UserExtensions\SimioAgentLibrary\Data\AgentLibrary.spfx
```

durch den effektiven Pfad (d.h. ohne Platzhalter %programfiles(x86)%) ersetzt werden. Dieser lautet z.B.

```
C:\Program Files (x86)\Simio\UserExtensions\SimioAgentLibrary\Data\AgentLibrary.spfx
```

siehe auch Abbildung 42.

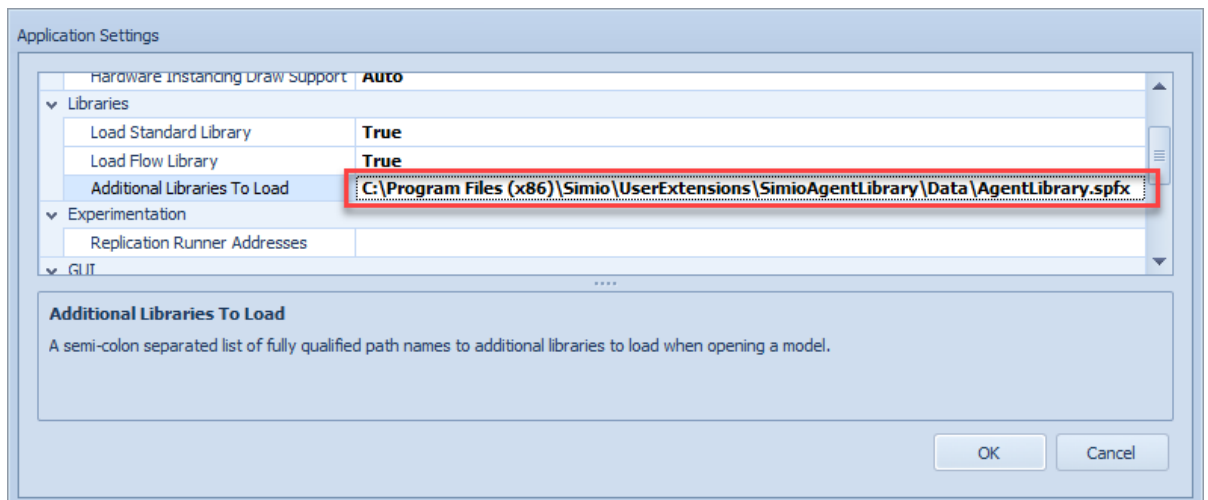


Abbildung 42: SimioAgentLibrary einbinden

<sup>5</sup> Hinweis: Für den Betrieb der SimioAgentLibrary ist eine lizenzierte Version von Simio erforderlich (keine Simio Personal Edition).

3. Simio beenden und neu starten. Nun sollte die AgentLibrary automatisch bei jedem Start von Simio geladen werden (siehe Abbildung 43).

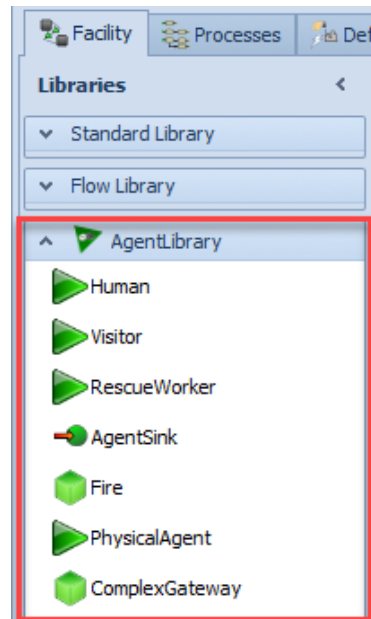


Abbildung 43: AgentLibrary in Simio

Alternativ kann die Library jeweils auch manuell eingebunden werden:

1. Start von Simio.
2. Laden der Library über das Hauptmenü von Simio (Abbildung 44).

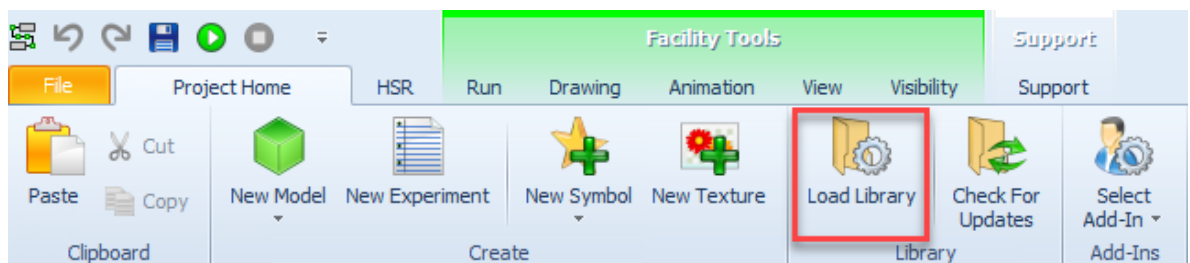


Abbildung 44: Manuelles Laden der SimioAgentLibrary

Im anschließenden Dialog für die Dateiauswahl muss folgende Datei bzw. Library ausgewählt werden:

```
%programfiles(x86)%\Simio\UserExtensions\SimioAgentLibrary\Data\AgentLibrary.spfx
```

3. Nun ist die Library gemäß Abbildung 43 verfügbar. Es ist zu beachten, dass diese für jedes neue Projekt manuell gemäß Anleitung eingebunden werden muss.

Vom Installer werden ebenfalls die Modelle, die in Kapitel 6.2 und 6.3 Verwendung finden, installiert. Diese befinden sich nach der Installation in folgenden Pfad:

```
%programfiles(x86)%\Simio\UserExtensions\SimioAgentLibrary\Data\
```

Nun können die Elemente der AgentLibrary für die Modellierung verwendet werden. Ebenso können für die Agenten eines Modells gemäß Kapitel 3.1 die jeweiligen StateCharts modelliert werden.

## 6 Anwendung der AgentLibrary

### 6.1 API-Dokumentation

Neben der Funktionalität des kompletten .NET Frameworks sowie 3rd-Party-Libraries (siehe u.a. Abschnitt 3.3.9) stellt die AgentLibrary weitere spezifische Funktionalität in Form einer API zur Verfügung.

Nachfolgende Übersicht (Abbildung 45) zeigt die Struktur der API. Die einzelnen Members in Form von Objekten, Properties und Methoden sind ausführlich im Anhang A beschrieben. Eine Übersicht ist in Tabelle 23 wiedergegeben.

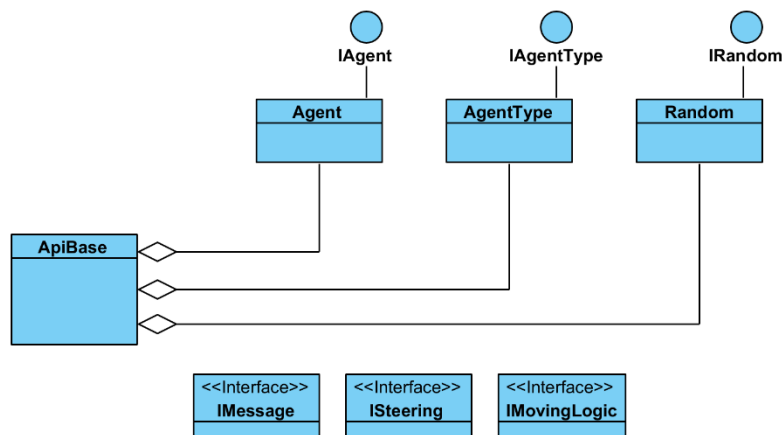


Abbildung 45: Struktur der API

Als Einstiegspunkt dient das Property `API`, welches in jeder C#-Expression sowie in den C#-Functions im State-Chart-Designer zur Verfügung steht. Über dieses können die weiteren Objekte wie `AgentType`, `Agent` oder `Random` abgerufen werden. Die Interfaces `IMessage`, `ISteering` sowie `IMovingLogic` werden als Typen für Argumente bzw. Rückgabewerte verwendet.

Objekt	Beschreibung
<b>API:ApiBase</b>	Dieses Objekt definiert diverse Properties und Methoden, die für das komplette Modell gelten.
<b>Agent:IAgent</b>	Dieses Objekt stellt Funktionalität bezogen auf einen konkreten Agenten zur Verfügung.
<b>AgentType:IAgentType</b>	Dieses Objekt stellt Funktionalität bezogen auf den aktuellen Agenttyp zur Verfügung.
<b>Random:IRandom</b>	Random definiert diverse Methoden, um Zufallszahlen zu erzeugen. Als Basis wurde die Bibliothek aus [8] verwendet und mit einigen weiteren Methoden (z.B. für die Triangular-Verteilung u.v.a.) ergänzt. Die Beschreibungen der Verteilungsfunktionen im Anhang A sind angelehnt an [9].
<b>IMessage</b>	Dieses Interface dient als Typ für die Nachrichten zwischen Agenten.
<b>ISteering</b>	Über dieses Interface kann die Steuerung für die Agenten angesprochen werden (wird in Kombination mit <code>IMovingLogic</code> verwendet).
<b>IMovingLogic</b>	Die Methode <code>IAgent.Move(IMovingLogic movingLogic)</code> ermöglicht die Definition einer komplexen Bewegungslogik. Weitere Informationen für die Verwendung von <code>IMovingLogic</code> und <code>ISteering</code> sind in Abschnitt 3.3.10 zu finden.

Tabelle 23: Objekte der API



## 6.2 Getting Started: Modellierung und Simulation eines Consumer-Markets

Dieses Kapitel beschreibt Schritt -für- Schritt die Erstellung eines einfachen, agentenbasierten Modells unter Verwendung der SimioAgentLibrary. Das Beispiel implementiert ein Consumer-Market-Model. Jeder Consumer wird als Agent modelliert. Die Simulation hat zum Ziel, den Markteintritt eines Produktes zu untersuchen. Da menschliche Entscheidungen (=Consumer-Entscheidungen) immer Stochastik beinhalten, eignet sich die agentenbasierte Modellierung und Simulation hervorragend für Marktsimulationen.

Folgende Annahmen werden getroffen:

- Das Modell beinhaltet 500 Personen, die das Produkt nicht verwenden.
- Eine Kombination von Mund-zu-Mund-Propaganda sowie Werbung soll die Personen ermuntern, das Produkt zu kaufen.
  - Die Werbung bewirkt, dass pro Tag 1% der potentiellen Consumer das Produkt kaufen möchten.
  - In Bezug auf Mund-zu-Mund-Propaganda verhält es sich so, dass ein bestehender Consumer durchschnittlich mit einem potentiellen Consumer pro Tag spricht. Durchschnittlich werden 1% der potentiellen Consumer das Produkt auf Basis der Mund-zu-Mund-Propaganda kaufen.
- Für die Auslieferung des Produkts an einen Consumer wird eine bestimmte Zeitdauer benötigt. Diese kann zwischen 1 und 25 Tagen betragen – durchschnittlich jedoch 2 Tage.
- Ein Consumer wartet durchschnittlich 7 Tage mit einer Abweichung von +/- 15% auf das Produkt, bis er sich entscheidet, vom Kauf zurückzutreten.
- Die Lebensdauer des Produkts beträgt 6 Monate.

Das komplette Modell befindet sich als Simio Projektdatei auf der Projektdaten-CD (siehe Anhang E – ConsumerMarket.spfx). Zusätzlich wird das Modell vom Installer installiert (siehe Kapitel 5). Die Problemstellung ist angelehnt an [10].

### 6.2.1 Schritt 1: Erstellung der Agent-Population

Hinweis: Es wird davon ausgegangen, dass die AgentLibrary gemäss Kapitel 5 erfolgreich installiert und global eingebunden wurde.

1. Starten Sie Simio. Es wird ein leeres Projekt erstellt. Speichern Sie dieses unter dem Namen «ConsumerMarket» ab.
2. Wechseln Sie zum Facility-Window des Modells.
3. Nun erstellen Sie eine neue Source (Element der StandardLibrary) und benennen diese «ConsumerSource».
4. Als nächstes erstellen Sie ein Agent-Objekt vom Typ «Human» sowie einen ComplexGateway und beschriften beide Elemente gemäss Abbildung 46. Der Output-Node der ConsumerSource wird unter Verwendung eines Path-Objekts (StandardLibrary) mit dem Input-Node des StartGateway verbunden.

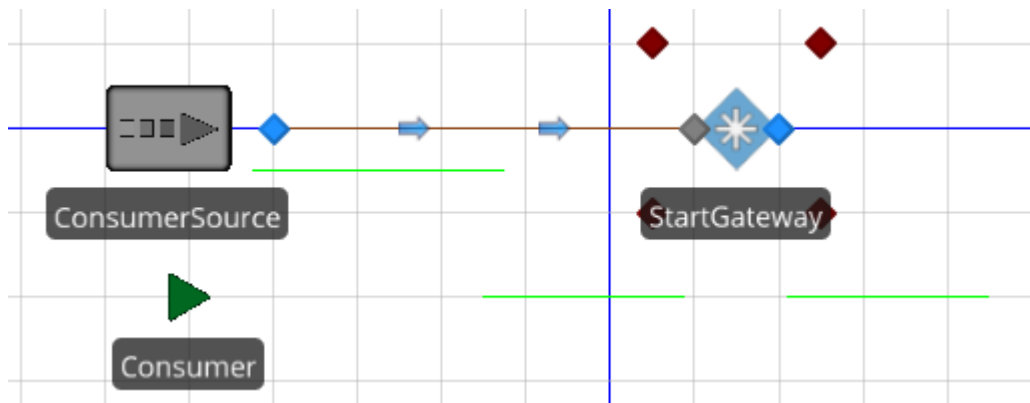


Abbildung 46: Grundmodell für Konsumentenmarkt

- Dem Consumer-Objekt werden nun drei Symbole zugewiesen, um den Zustand des Agenten während der Simulation zu visualisieren. Dazu wählen Sie im Facility-Window das Consumer-Objekt aus und fügen mit «Add Additional Symbol» (Hauptmenü «Symbols») zwei weitere Symbole hinzu. Abschliessend werden den einzelnen Symbolen verschiedene Farben zugewiesen (siehe Abbildung 47).

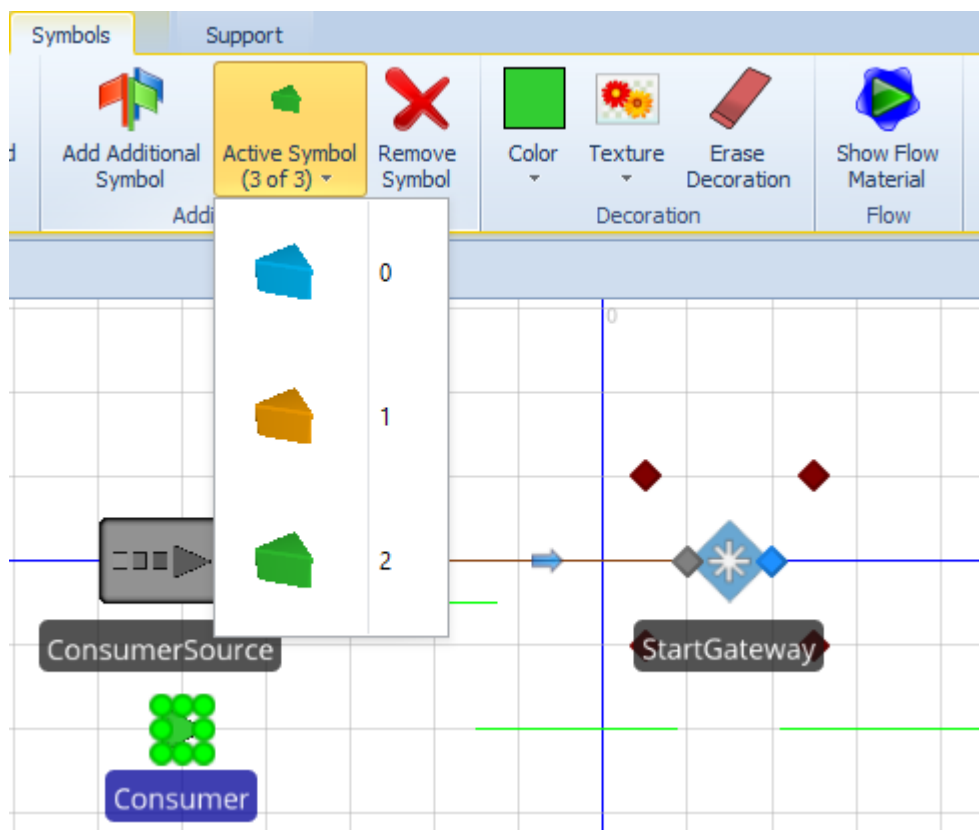


Abbildung 47: Consumer-Objekt mit verschiedenen Symbolen und Farben

Für die Zuweisung des aktuellen Symbols soll eine Simio-State-Variable erstellt werden. Wechseln Sie zu «Definitions» und wählen Sie dort den Bereich «States». Nun erstellen Sie eine neue State-Variable vom Typ «String» und nennen diese «ConsumerSymbol». Die Berechnung des Symbols erfolgt in der Eigenschaft «Current Symbol Index» des Consumer-Objekts mit folgender Simio-Expression:

```
Math.If(String.Length(ConsumerSymbol) <= Human.ID, 0,
String.ToReal(String.Substring(ConsumerSymbol, Human.ID, 1)))
```

Fügen Sie diese (einzellig) in die Eigenschaft «Current Symbol Index» ein (siehe Abbildung 48).

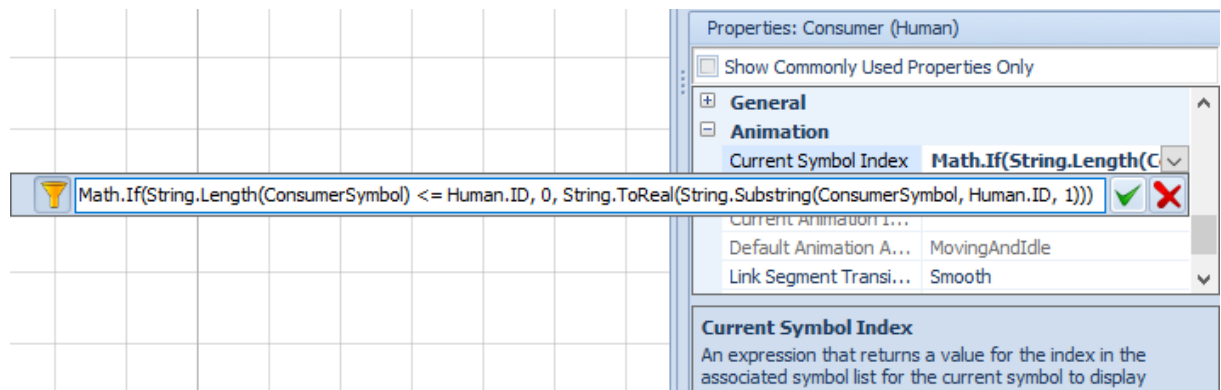


Abbildung 48: Berechnung «Current Symbol Index» für Consumer-Objekt

6. Weiter muss die Anzahl der Consumer, die Simio beim Start der Simulation erstellen soll, konfiguriert werden. Gemäss Annahme soll mit einer Population von 500 Consumer simuliert werden. Dazu wählen Sie das Objekt ConsumerSource und konfigurieren dieses wie folgt:

- Entity Type = Consumer
- Entities Per Arrival = 500
- Maximum Arrivals = 500
- Interarrival Time = Infinity

Mit dieser Einstellung werden direkt beim Start der Simulation 500 Entities vom Typ Consumer erzeugt.

## 6.2.2 Schritt 2: Consumer-Verhalten definieren

Im nächsten Schritt modellieren wir das Verhalten der Agenten. Zu diesem Zweck wird für das Consumer-Objekt ein StateChart erstellt.

1. Starten Sie die Agent Management Oberfläche im Hauptmenü von Simio über «HSR» - «Simio AgentLibrary» (Abbildung 4). In der Folge wird in der geöffneten Agent Management Oberfläche das vorhandene Model mit dem erstellten Agenten «Consumer» angezeigt (Abbildung 49).

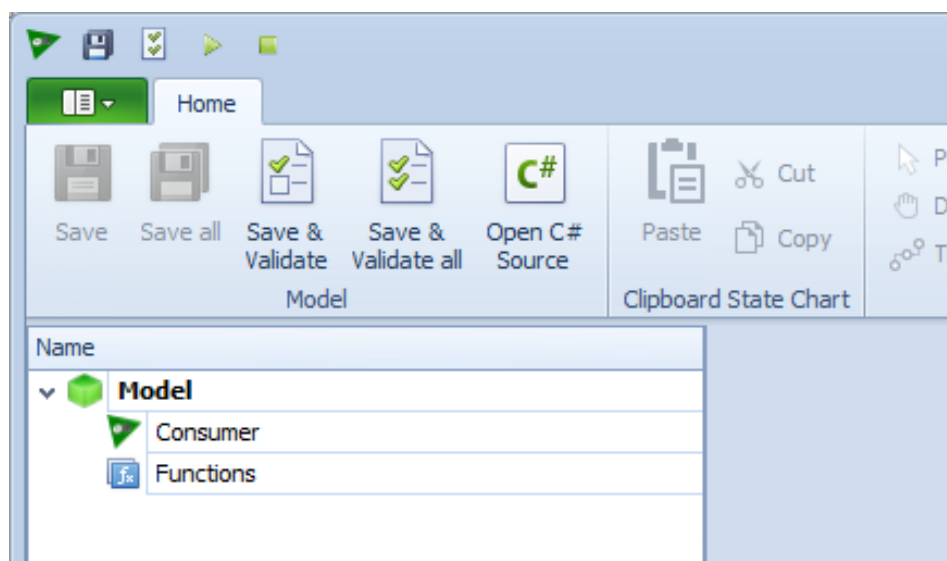


Abbildung 49: Agent Management Oberfläche mit Model und Agent «Consumer»

Nach Doppelclick auf «Consumer» öffnet sich der StateChart-Designer, welcher die Modellierung des StateChart für den Agenten ermöglicht.

2. Ein Consumer kann drei verschiedene Zustände annehmen:
  - a. Potentieller Consumer
  - b. Zum Kauf entschlossener Consumer, der auf das Produkt wartet
  - c. Consumer

Somit modellieren wir die drei Zustände, beschriften diese und weisen ihnen je eine Farbe zu (gemäß Abbildung 50). Die Farben sollen mit den jeweiligen Farben des Consumer-Objekt in Abbildung 47 korrespondieren.



Abbildung 50: Zustände für einen Consumer

3. Wird ein Consumer erstellt, soll er initial den Zustand «Potential Consumer» erhalten. Zusätzlich soll er im Facility-Window zufällig in einem definierten Bereich positioniert werden. Für diesen Bereich bestimmen wir das Rechteck  $\begin{pmatrix} 10 \\ 0 \end{pmatrix}$  bis  $\begin{pmatrix} 40 \\ 30 \end{pmatrix}$  im Facility-Window von Simio (siehe Abbildung 52). Um den Consumer in diesem Bereich zu positionieren, erstellen wir im StateChart-Designer einen Entry Point, der auf den State «Potential Consumer» zeigt und beschriften diesen mit «Set position» (Abbildung 60). Um die initiale Position zuzuweisen, erstellen wir eine neue Function «SetInitialPosition» (Menu Gruppe «Functions» - «Add») und fügen diese für den Entry Point als Action ein (Abbildung 51).

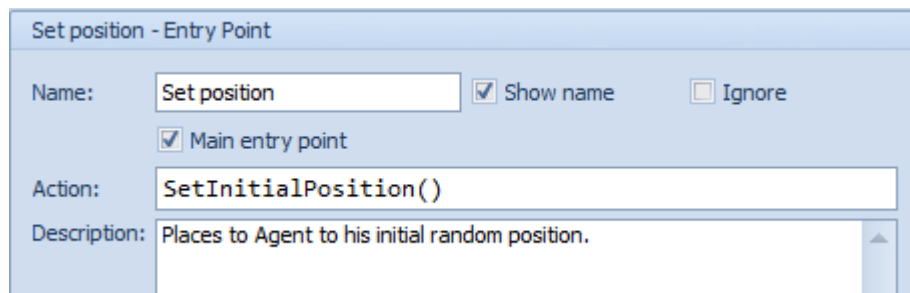


Abbildung 51: Konfiguration des Entry Points

Der Code für die Function «SetInitialPosition» kann aus Listing 5 entnommen werden.

```
var x = Random.GetUniform(10.0,40.0);  
var y = Random.GetUniform(0.0,30.0);  
Agent.JumpTo(new Vector2(x,y));
```

Listing 5: Function SetInitialPosition

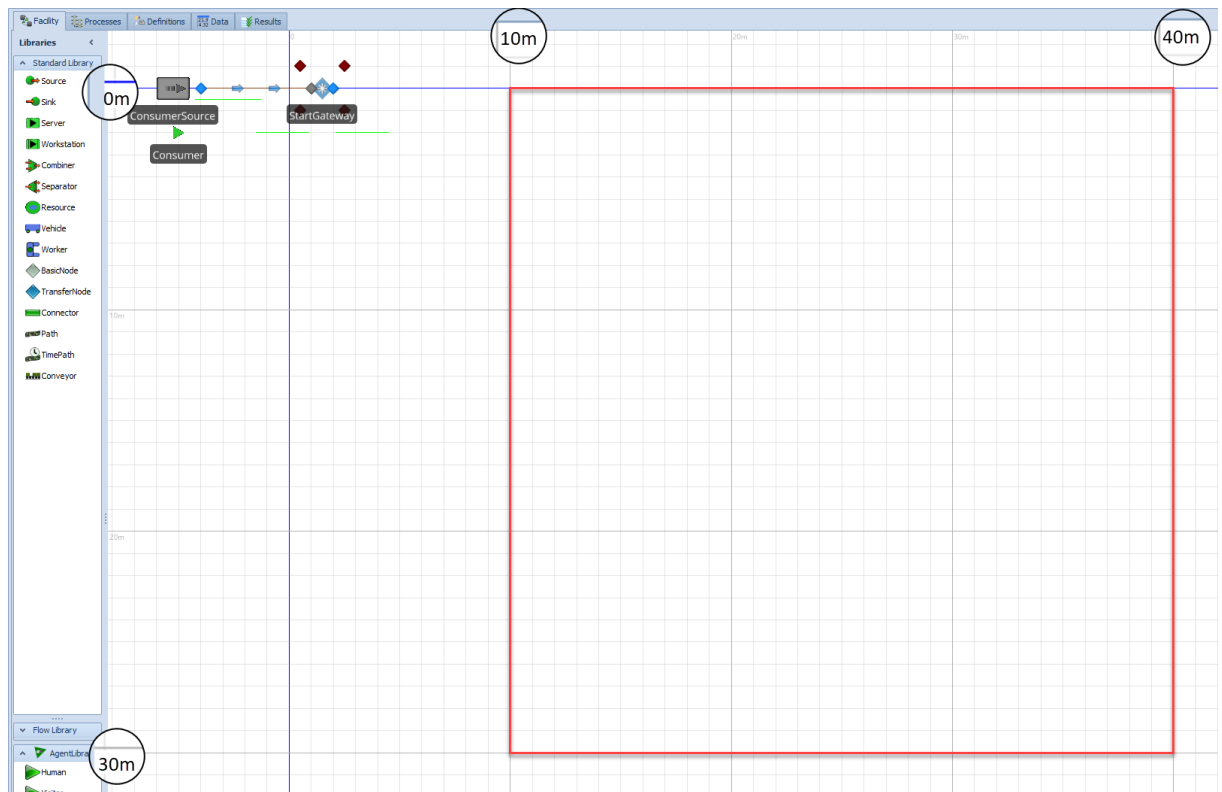


Abbildung 52: Region der Consumer-Agenten im Facility-Window

Damit für die Positionierung der Agenten das Koordinatensystem des Facility-Window verwendet wird, muss dieses beim Object «StartGateway» konfiguriert werden. Dazu setzen Sie die Eigenschaft «FacilityName» des «StartGateway» auf die Simio-Expression «Model» (siehe Abbildung 53).

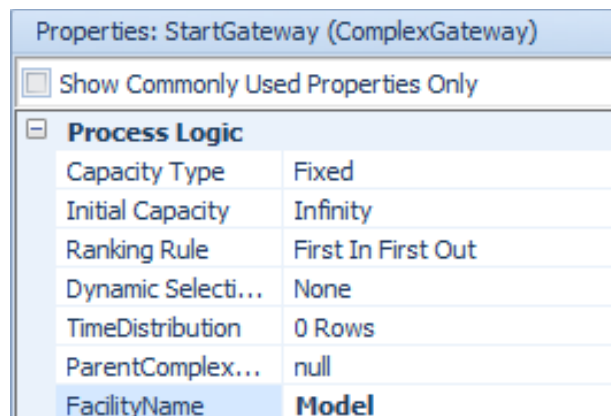


Abbildung 53: FacilityName für StartGateway

- Im nächsten Schritt soll die Annahme modelliert werden, dass täglich 1% der potentiellen Consumer das Produkt kaufen. Zu diesem Zweck fügen wir eine Transition zwischen «Potential Consumer» und «WantsToBuy» ein. Diese benennen wir «Ad» (Abbildung 60). Die Transition ist vom Typ «Rate» und erhält eine Rate von 0.01 per Day, was einer Verteilung von 1% täglich entspricht (Abbildung 54). Der Typ Rate entspricht einer exponentiellen Verteilung mit dem Mittelwert «rate». Die Modellierung der Beeinflussung durch «Mund-zu-Mund-Propaganda» erfolgt in einem späteren Schritt.

Abbildung 54: Konfiguration der Transition «Ad»

- Ein Consumer wartet durchschnittlich 7 Tage (mit einer Abweichung von +/- 15%) auf den Erhalt des Produkts. Ansonsten tritt er vom Kauf zurück. Um dies umzusetzen, fügen wir zwischen «WantsToBuy» und «Potential Consumer» eine Transition vom Typ «Timeout» ein und bezeichnen diese mit «CantWait». Um das Verhalten zu modellieren, verwenden wir eine Triangular-Verteilung mit Durchschnittswert 7 und Abweichung 0.15 (Abbildung 55).

Abbildung 55: Konfiguration der Transition «CantWait»

- Als nächstes soll die Lieferzeit des Produkts modelliert werden. Diese beläuft sich zwischen 1 und 25 Tagen mit einem Durchschnittswert von 2 Tagen. Für diesen Aspekt können wir wiederum eine Transition vom Typ «Timeout» verwenden. Wir fügen diese zwischen «WantsToBuy» und «Consumer» ein und benennen sie «Purchase». Da wir das Minimum, Maximum sowie den Durchschnittswert kennen, können wir die normale Triangular-Verteilung verwenden (Abbildung 56).

Purchase - Transition	
Name:	Purchase <input checked="" type="checkbox"/> Show name <input type="checkbox"/> Ignore
Triggered by:	Timeout
Timeout:	Random.GetTriangular(1, 25, 2) days
Action:	
Guard:	
Description:	Let's assume it typically takes a user two days to get the product. This means once the consumer's statechart enters the state WantsToBuy, it will proceed to the state User with a two-day delay.

Abbildung 56: Konfiguration der Transition «Purchase»

- Das Produkt hat eine Lebensdauer von 6 Monaten. Für diesen Aspekt kommt ebenfalls eine Transition vom Typ «Timeout» zwischen «Consumer» und «WantsToBuy» zum Einsatz. Diese benennen wir «Discard». Als Timeoutwert konfigurieren wir 6 Monate (Abbildung 57).

Discard - Transition	
Name:	Discard <input checked="" type="checkbox"/> Show name <input type="checkbox"/> Ignore
Triggered by:	Timeout
Timeout:	6 months
Action:	
Guard:	
Description:	DiscardTime = 6 Define the Products lifespan

Abbildung 57: Konfiguration der Transition «Discard»

- Nun fehlt noch die Anforderung bzgl. «Mund-zu-Mund-Propaganda». Ein Consumer soll täglich mit einem anderen Agenten in Kontakt treten. Dies erreichen wir mit einer internen Transition vom Typ «Rate» beim State «Consumer». Die Transition benennen wir «Contact». Als Action soll an einen beliebigen Agenten die Nachricht «Buy» gesendet werden (Abbildung 58).

Contact - Transition	
Name:	Contact <input checked="" type="checkbox"/> Show name <input type="checkbox"/> Ignore
Triggered by:	Rate <input type="checkbox"/> Internal transition
Rate:	1 per day
Action:	Agent.SendToRandom("Buy")
Guard:	
Description:	Contact one Agent per Day and send him the Message "Buy" to initiate Word-of-Mouth-Propaganda.

Abbildung 58: Konfiguration der Transition «Contact»

1% der potentiellen Consumers reagieren auf die Mund-zu-Mund-Propaganda. Dies erreichen wir mit einer Transition vom Typ «Message» zwischen «PotentialConsumer» und «WantsToBuy». Diese benennen wir «WoM». Die Transition reagiert auf den Erhalt der Message «Buy» - jedoch nur in 1% der auftretenden Fälle (siehe Abbildung 59).

WoM - Transition

Name:

WoM

☒ Show name

☐ Ignore

Triggered by:

Message

Message type:

string

Fire transition:

☐ Unconditionally  
☒ On particular message  
☐ If expression is true

Message:

"Buy"

Action:

Guard:

Random.RandomTrue(0.01)

Description:

AdoptionFraction = 0.01  
  
 To define a person's influence on others, a number that we'll express as the percentage of people who will use the product after they come into contact with the consumer.

Abbildung 59: Konfiguration der Transition «WoM»

Das komplette Zustandsdiagramm ist in Abbildung 60 ersichtlich.



Abbildung 60: Komplettes Zustandsdiagramm für Consumer Agent



## 6.2.3 Schritt 3: Diagramm hinzufügen, um das Resultat zu visualisieren

Um das Resultat zu visualisieren, werden zwei Aspekte implementiert:

- 1) Einfärben der Agenten im Facility-Window in der jeweiligen Farbe des Zustandes
- 2) Anzeige der Anzahl Agenten pro Zustand in einem Status-Plot

Damit dies erreicht werden kann, muss in Simio pro Zustand eine Simio-State-Variable erstellt werden, welche die Anzahl der jeweiligen Agenten beinhaltet:

1. Wechseln Sie zu Simio. Dort wählen Sie den Bereich «Definitions» und anschliessend «States». Nun erstellen Sie drei State-Variablen vom Typ «Integer»:
  - NumberPotential
  - NumberWantsToBuy
  - NumberConsumers
2. Nun müssen die Simio-State-Variablen addiert bzw. subtrahiert werden, sobald ein Agent im State-Chart den entsprechenden State betritt bzw. verlässt. Dazu wechseln Sie zurück zum StateChart-Designer. Für jeden State soll nun in der Entry-Action die entsprechende State-Variable um 1 addiert sowie in der Exit-Action um 1 subtrahiert werden. Zusätzlich möchten wir in der Entry-Action dem Agenten die entsprechende Farbe zuweisen. Da die Entry-Action somit aus zwei Expressions besteht, erstellen wir für jeden Zustand eine Function, die wir in der Entry-Action aufrufen können. Somit werden folgende Functions erstellt:
  - EntryPotentialConsumerState
  - EntryWantsToBuyState
  - EntryConsumerState

Die Functions erhalten jeweils den Code gemäss Listing 6 bis Listing 8.

```
SetStateValue("NumberPotential", GetStateValue<int>("NumberPotential")+1);  
SetConsumerSymbol(0);
```

**Listing 6: Function EntryPotentialConsumerState**

```
SetStateValue("NumberWantsToBuy", GetStateValue<int>("NumberWantsToBuy")+1);  
SetConsumerSymbol(1);
```

**Listing 7: Function EntryWantsToBuyState**

```
SetStateValue("NumberConsumers", GetStateValue<int>("NumberConsumers")+1);  
SetConsumerSymbol (2);
```

**Listing 8: Function EntryConsumerState**

Nun können die Functions «EntryXXXState» im StateChart-Designer in den jeweiligen Entry-Actions der betreffenden States eingebunden werden. Abbildung 61 zeigt dies exemplarisch für den State «Potential Consumer».

3. Damit die Simio-State-Variablen beim Verlassen des States durch einen Agenten dekrementiert werden, muss die entsprechende Logik in die Exit-Action eingefügt werden (siehe Listing 9 sowie Abbildung 61).

```
SetStateValue("NumberPotential", GetStateValue<int>("NumberPotential")-1)
```

**Listing 9: Function EntryConsumerState**

Abbildung 61: Entry-Action und Exit-Action für Zustand "Potential Consumer"

4. Zusätzlich muss die Funktion «SetConsumerSymbol» erstellt werden. Sie manipuliert die Simio-State-Variable «ConsumerSymbol» welche wiederum als Basis für die Berechnung der Symbole für die Agenten dient. Die Funktion besitzt einen Parameter vom Typ «int» (siehe Abbildung 62).

Name	Type
symbolID	int

Abbildung 62: Parameter für Funktion «SetConsumerSymbol»

Der Code für die Funktion «SetConsumerSymbol» zeigt Listing 10.

```
var symb = GetStateValue<string>("ConsumerSymbol");
symb = (symb?? "").PadRight(Agent.ID, '0');
var arr = symb.ToCharArray();
arr[Agent.ID-1] = symbolID.ToString()[0];

SetStateValue("ConsumerSymbol", new string(arr));
```

Listing 10: Function SetConsumerSymbol

Der Algorithmus funktioniert so, dass ein String (Simio-State-Variable «ConsumerSymbol») erstellt wird. Für jeden Agenten wird an seiner Position im String (anhand `Agent.ID`) die jeweilige Symbolnummer gespeichert. Diese Information wird beim Consumer-Agenten anschliessend verwendet, um das aktuelle Symbol zu berechnen (siehe Abbildung 48).

Nun können die State-Variablen «NumberPotential», «NumberWantsToBuy» und «NumberConsumers» als Datenquelle für den Status-Plot verwendet werden.

1. Wechseln Sie zu Simio. Fügen Sie im Facility-Window einen Status-Plot hinzu (Simio Hauptmenü «Animation»). Als «Time Range» wählen Sie 4 Wochen. Ändern Sie die Einstellung «Additional Expression» wie in Abbildung 63 gezeigt.

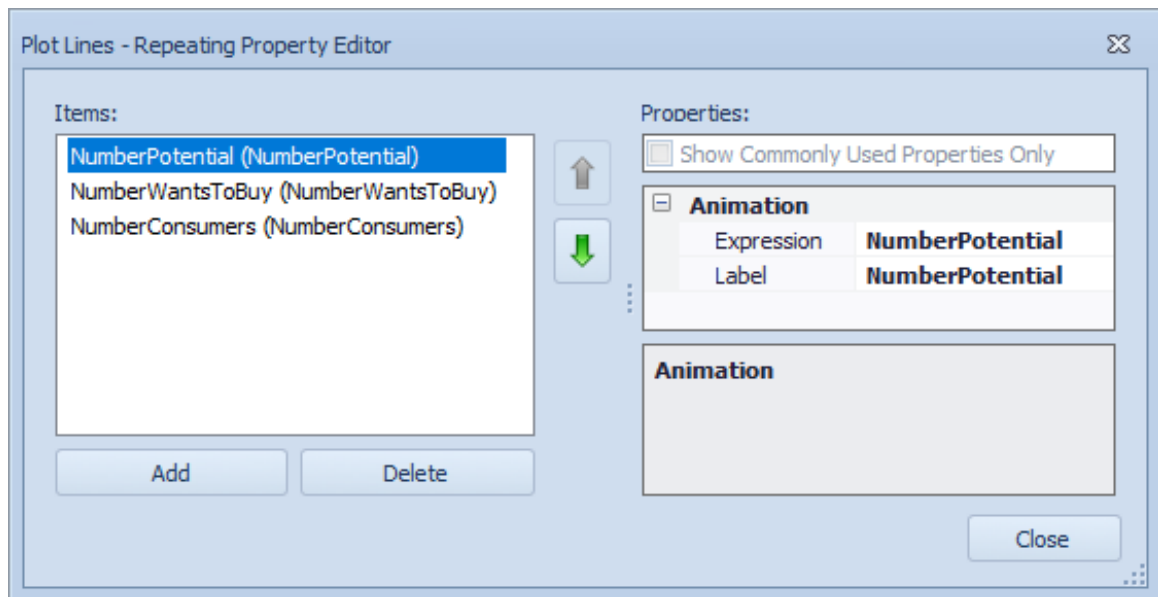


Abbildung 63: Additional Expressions für Status-Plot

## 6.2.4 Schritt 4: Durchführung der Simulation

Nun kann die Simulation ausgeführt werden. Diese soll einen Zeitraum von 20 Wochen umfassen.

1. Konfigurieren Sie im Hauptmenü unter «Run» einen beliebigen Startpunkt sowie das Ende nach 20 Wochen (siehe Abbildung 64).

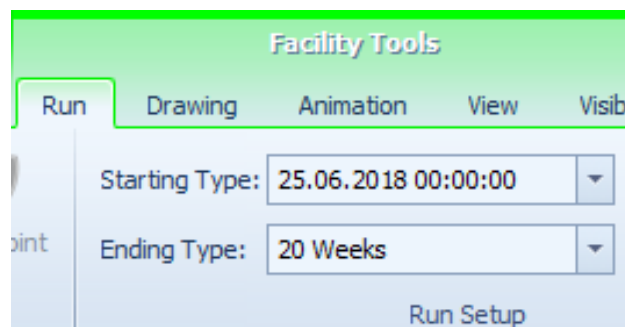


Abbildung 64: Dauer der Simulation

2. Konfigurieren Sie den «Speed Factor» unter «Run» - «Animation Speed» auf 10000.000.
3. Damit die Performance hoch bleibt, konfigurieren Sie das «Update Time Interval» des Consumer-Agenten auf 4 Stunden (siehe Abbildung 65).

Properties: Consumer (Human)	
<input type="checkbox"/> Show Commonly Used Properties Only	
<input checked="" type="checkbox"/> <b>Process Logic</b>	
Initial Health	100
Initial Place	null
Agent Sink	null
FireImpact	20
<input checked="" type="checkbox"/> Steering Behavior	Agent
<input checked="" type="checkbox"/> Update Time Interval	4
Units	Hours

Abbildung 65: «Update Time Interval» für den Consumer-Agenten

- Starten Sie die Simulation in der Agent Management-Oberfläche im Hauptmenü unter «Simulation» - «Run / Pause». Dies stellt sicher, dass alle Komponenten mit dem aktuellen Code kompiliert und konfiguriert werden.

Ein mögliches Resultat nach einer Simulationsdauer von 20 Wochen ist in Abbildung 66 und Abbildung 67 zu sehen.

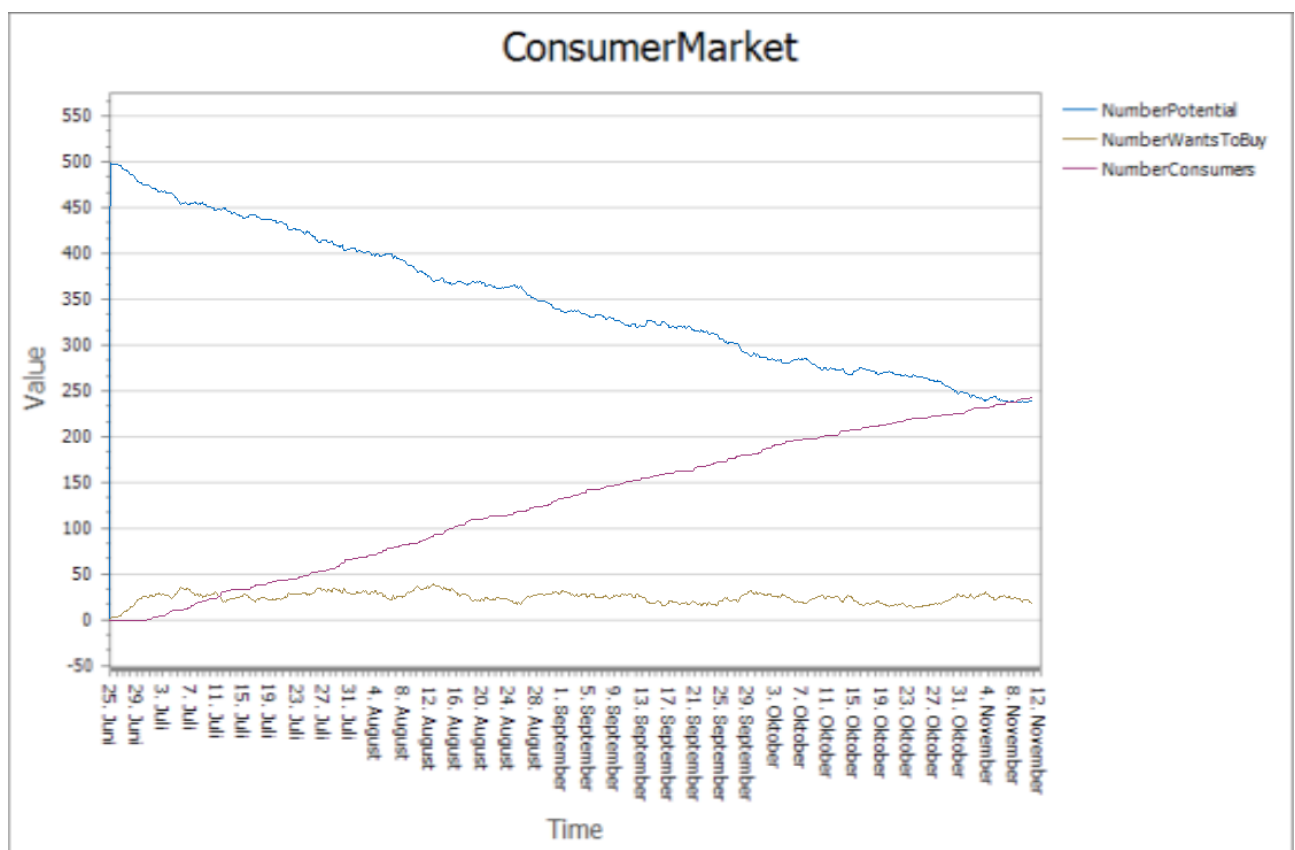


Abbildung 66: Resultat ConsumerMarket

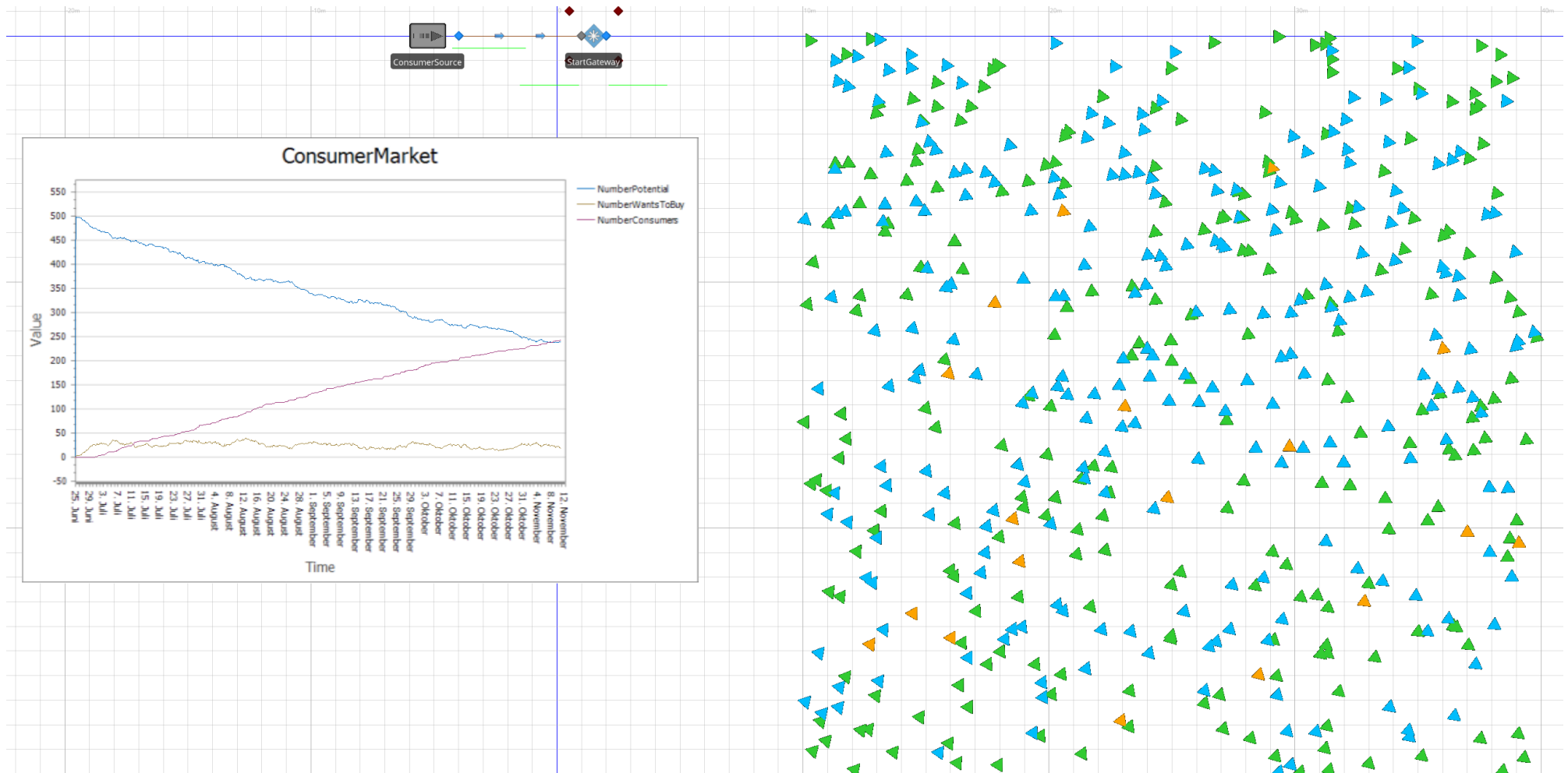


Abbildung 67: Resultat ConsumerMarket - Übersicht

## 6.3 Verifikation Modellierungsaspekte Problemstellung «Produktion»

In diesem Kapitel werden einzelne Aspekte in Bezug auf die Modellierung der Problemstellung aus Kapitel 1.2 betrachtet. Das Modell wurde nicht komplett umgesetzt. Vielmehr wurden einzelne Schlüsselaspekte gemäss Beschreibung in Kapitel 2.1 bzw. Tabelle 1 als Basis für die Implementierung verwendet. Ein vollständig dokumentiertes Modell befindet sich in Kapitel 6.2 (ConsumerMarket). Das Modell «Produktion» mit den umgesetzten Aspekten befindet sich als Simio Projektdatei auf der Projektdaten-CD (siehe Anhang E – Manufacturing.spfx). Zusätzlich wird das Modell vom Installer installiert (siehe Kapitel 5). Das Simio-Modell ist in Abbildung 71 ersichtlich.

Das Modell beinhaltet folgende Agententypen:

- Controller
- EntPartA
- EntPartB
- EntPartC
- EntPartD

wobei EntPartA bis EntPartD jeweils vom Verhalten identisch sind. Die Aufteilung dient ausschliesslich dazu, die vier verschiedenen Komponentenarten zu unterscheiden.

Für die Koordination wird von der Source «SrcController» ein Controller-Agent erstellt. Abbildung 68 zeigt das StateChart dieses Agenten. Er befindet sich während der Simulation immer beim ComplexGateway «MainStorage» und ist für die Entscheidung verantwortlich, ob Komponenten aus dem MainStorage oder aus dem TempStorage verwendet werden sollen. Wie in Abbildung 68 ersichtlich, prüft der Controller sobald Aufträge verfügbar sind, ob sich entsprechende Komponenten beim TempStorage befinden. Dazu sendet der Controller eine Nachricht an alle Agenten. Erhält er eine positive Antwort, weist er den entsprechenden Absender-Agenten (=Komponente), sich vom TempStorage zum TNodeStart zu bewegen (d.h. die entsprechende Komponente wird aus dem TempStorage ausgelagert und verwendet). Erhält der Controller hingegen keine Antwort, erstellt er neue Komponenten (d.h. diese werden aus dem MainStorage ausgelagert und verwendet).

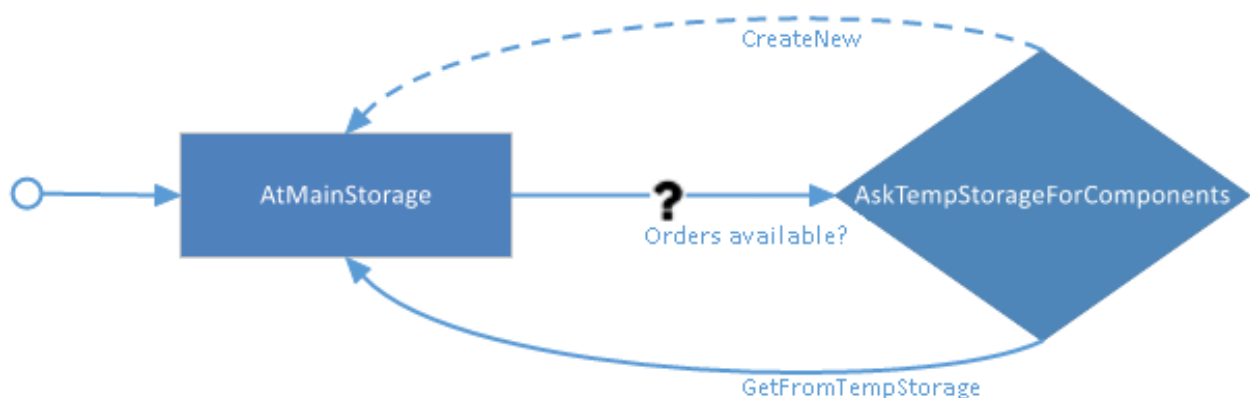


Abbildung 68: StateChart Controller-Agent

Die Komponenten werden gemäss den eintreffenden Aufträgen vom Controller erzeugt oder (wie vorgängig beschrieben) beim TempStorage angefordert. Anschliessend passieren diese die Produktion und gelangen zum ComplexGateway «Decision». Dort wird die Agentenlogik für die Komponenten gemäss StateChart in Abbildung 69 angewendet. Zuerst wird entschieden, ob die Komponente zurück geht zum Hauptlager, zum Zwischenlager oder ausgeliefert werden soll. Im Fall «zurück zum Hauptlager» wird die Komponente in der Sink «SnkMainStorage» terminiert (eingelagert) und ggf. vom Controller im Hauptlager neu erstellt (ausgelagert). Im Fall «zum Zwischenlager» erfolgt der Transfer zum ComplexGateway «TempStorage» - d.h. der Agent wechselt

in den State «AtTempStorage». Ansonsten wird die Komponente ausgeliefert – d.h. in der Sink «SnkLogistic» terminiert. Befindet sich der Agent im State «AtTempStorage» (d.h. im Zwischenlager), wartet er auf eine der folgenden Nachrichten (diese werden vom Controller gesendet):

- «X\_Available» - Der Controller prüft, ob sich eine bestimmte Komponente im Zwischenlager befindet. Wenn X seinem Komponententyp (d.h. A, B, C oder D) entspricht, erstellt der Agent eine Antwortnachricht mit dem Inhalt «yes». Anschliessend prüft der Controller, wie viele «yes»-Antworten er erhalten hat. Abhängig von der Anzahl (0-4) erstellt er ggf. zusätzliche Komponenten neu (wenn <4), um einen Produktionsträger mit 4 Komponenten zu füllen.
- «ToProduction» - Der Controller fordert eine bestimmte Komponente vom Zwischenlager an. Die Komponenten, die der Controller vom Zwischenlager verwenden möchte, kann er mit der Nachricht «ToProduction» anfordern.

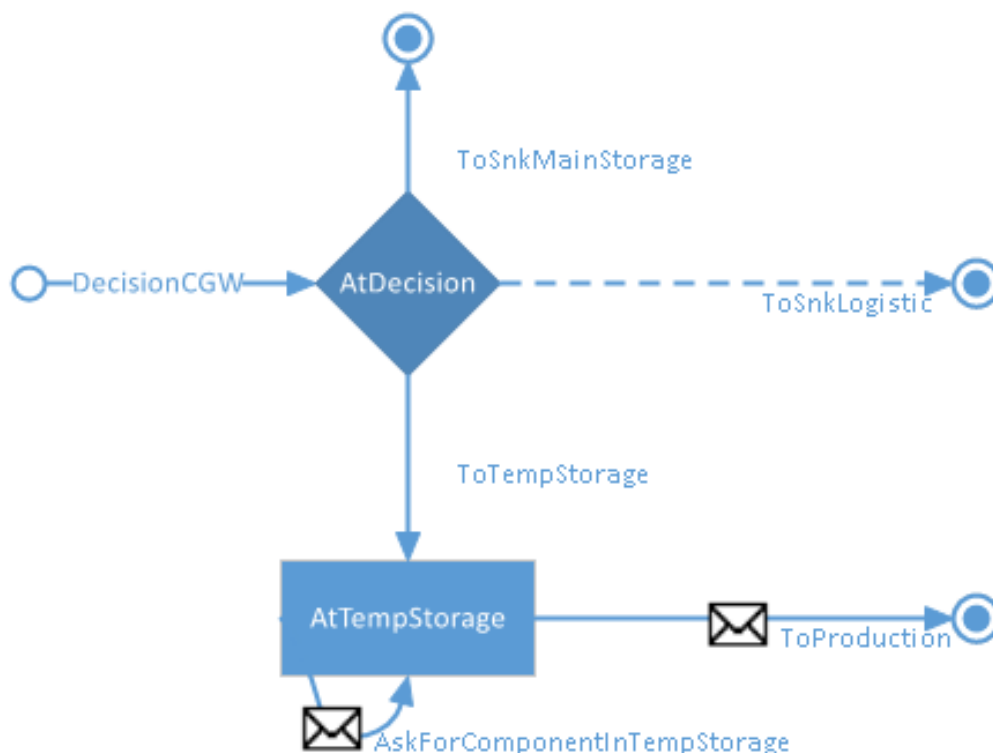


Abbildung 69: StateChart EntPartX-Agent

Mit einem Status-Plot kann die Anzahl der Komponenten im TempStorage über die Zeit visualisiert werden.

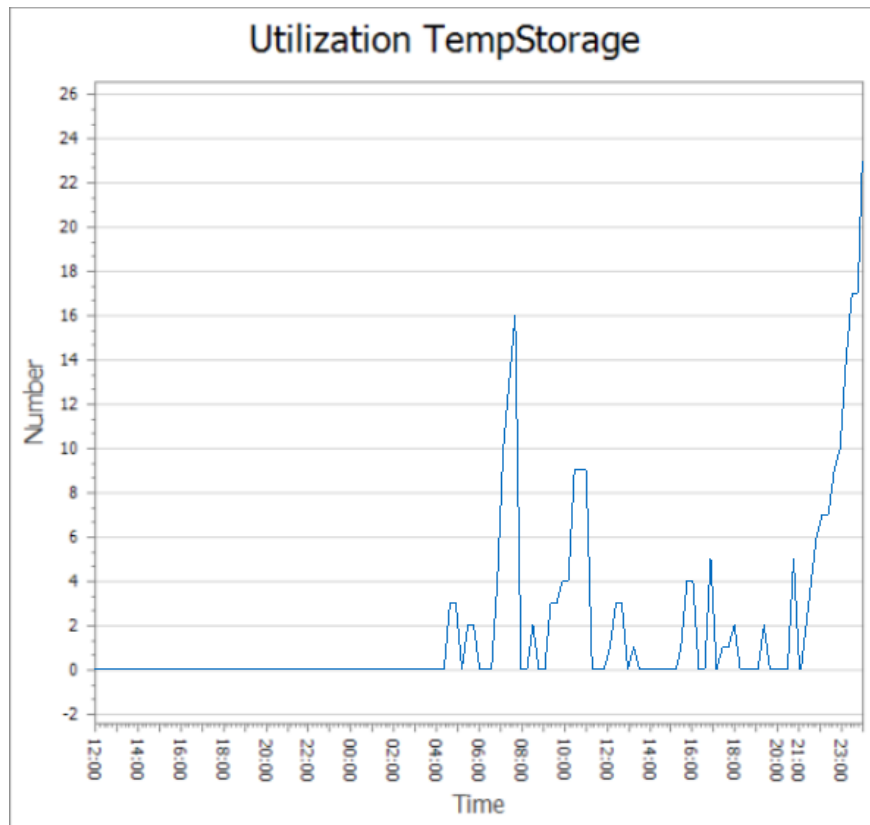


Abbildung 70: Auslastung TempStorage



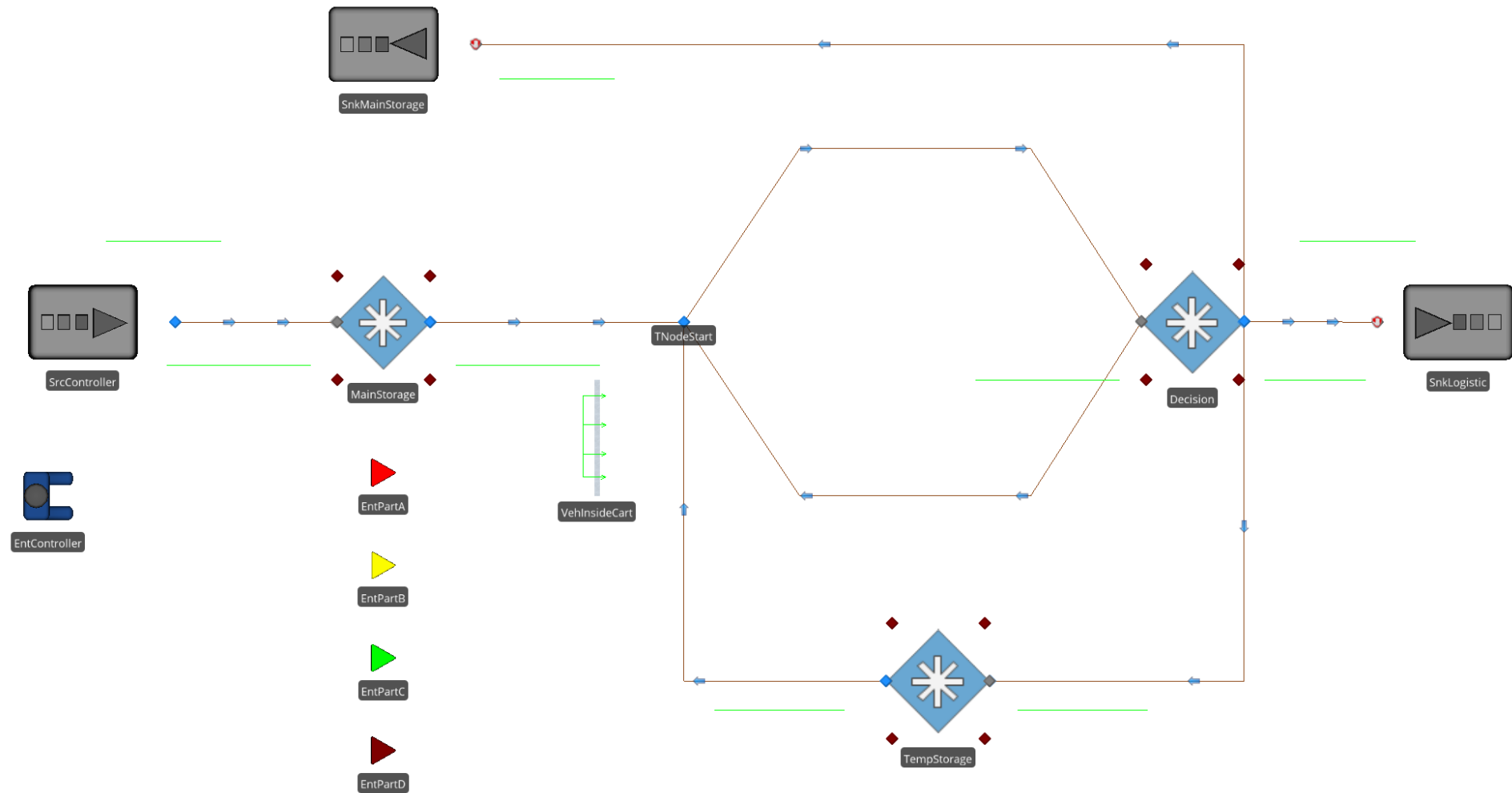


Abbildung 71: Modell für Problemstellung «Produktion»

Die in Kapitel 2.1 formulierten Aspekte werden abschliessend bezogen auf ihre Umsetzung erläutert und verifiziert. Um die Übersichtlichkeit zu verbessern, wird die Tabelle 1 hier wiederholt:

Nr.	Aspekt	Anforderung
1	Kontroller kennt Auftragsliste (Excel bzw. CSV-Datei)	Möglichkeit, Exceldateien bzw. allgemein Dateien zu lesen
2	Kontroller generiert 4 Komponenten	Agent soll andere Agenten generieren können
3	... wird gefragt, ob im Zwischenlager ...	Agent soll mit anderen Agenten kommunizieren können
4	Wenn ja, werden diese verwendet	Agent soll anderen Agenten Befehle erteilen können
5	Der Träger durchläuft die Produktionsstrasse	Agenten sollen sich auch als Entities in der diskreten, prozessorientierten Ereignissimulation bewegen können
6	... beim DecisionGateway werden ...	An bestimmten Stellen im System soll die Agentenlogik ausgeführt werden
7	Einfache und komfortable Modellierung	Grafischer StateChart-Designer

**Tabelle 24: Aspekte und deren Umsetzungsanforderung(en) aus Problemstellung**

## 1) Dateien einlesen

Dies ist mit Standard-.NET Funktionalität gewährleistet. Im Model wird die Exceldatei (bzw. CSV-Datei) Manufacturing.csv mit dem Code gemäss Listing 11 eingelesen. Dies geschieht in der Action «OnStartup» (siehe Abschnitt 3.3.8) des Kontrollers.

```
var dt = new DataTable();
dt.Columns.Add("ID", typeof(int));
dt.Columns.Add("Componenttype", typeof(string));
dt.Columns.Add("Timestamp", typeof(DateTime));
dt.Columns.Add("InProduction", typeof(bool));
dt.Columns.Add("Done", typeof(bool));

var dir = Path.GetDirectoryName(BasePath);
var csvFile = Path.Combine(dir, "Manufacturing.csv");
var lines = File.ReadLines(csvFile).Select(a => a.Split(';'));
foreach (var line in lines) {
    int id;
    if (!int.TryParse(line[0], out id)) {
        continue;
    }

    dt.Rows.Add(id, line[1], Convert.ToDateTime(line[2]), false, false);
}

Set("Data", dt); // Saves the Orderslist in global variable Data
```

**Listing 11: CSV-Datei einlesen**

## 2) Agenten generieren

Für diese Anforderung wurde die Möglichkeit implementiert, dass ein Agent beliebige Events in Simio aufrufen kann (siehe Listing 12). In Simio kann einem Event wiederum ein Prozess zugewiesen werden. Dieser kann durch Verwendung des Prozesssteps «Create» beliebige Entities bzw. Agenten generieren (siehe Abbildung 72).

```
FireEvent("CreatePart_4A");
```

**Listing 12: Simio-Event aufrufen**

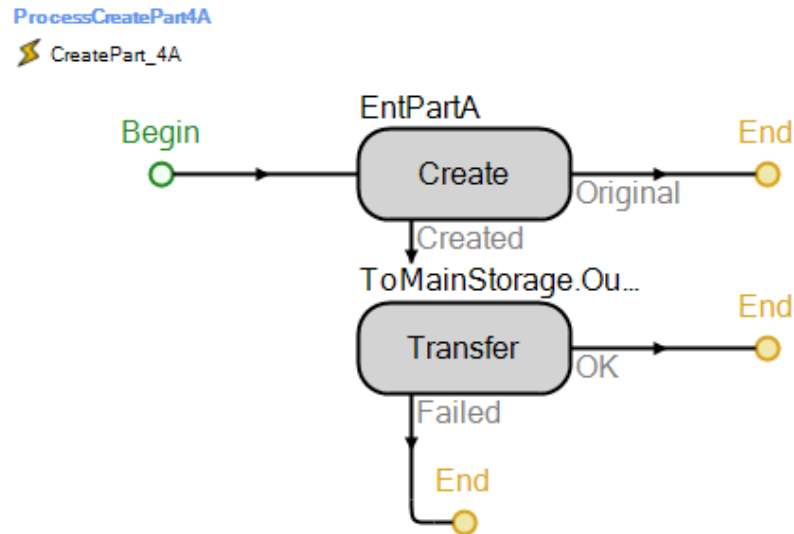


Abbildung 72: Prozess für die Erstellung von EntPartA-Objekte

### 3) Mit Agent(en) kommunizieren

Agenten können beliebige Nachrichten versenden. Der Controller versendet z.B. die Nachricht «A\_Available» um zu fragen, ob EntPartA-Komponenten im Zwischenlager verfügbar sind (siehe Listing 13).

```

var orders = GetAvailableOrders();

foreach (var row in orders) {
    var compType = (string)row["Componenttype"];
    Agent.Send(compType + "_Available?", compType); // Agent has to reply with
                                                    // Message compType
}

```

Listing 13: Versand Nachricht für Prüfung, ob Komponenten im Zwischenlager verfügbar

### 4) Befehle erteilen

Für diese Anforderung kann auch der Nachrichtenmechanismus verwendet werden. Alternativ kann ein bestimmtes Agenten-Objekt ermittelt und direkt auf diesem Objekt Actions ausgeführt werden.

### 5) Agent als Entity

Entity bildet das Basisobjekt für Agenten. Dies ermöglicht es, Agenten als normale Entities verwenden zu können.

### 6) Ausführung von Agentenlogik

Diese Anforderung wird durch das Objekt «ComplexGateway» erfüllt (siehe Kapitel 4.2).

### 7) Grafischer StateChart-Designer

Siehe Kapitel 3.3.

## 7 Schlussbemerkungen

### 7.1 Fazit

Mit dem Ergebnis ist es möglich, in Simio agentenbasierte Modelle zu erstellen, sowie deren Simulation durchzuführen. Darüber hinaus kann die agentenbasierte Simulation mit der in Simio bekannten prozessorientierten diskreten Ereignissimulation kombiniert werden.

Wie in [3, S.45ff] bereits als Grundlage erarbeitet, wurde die agentenbasierte Modellierung auf Basis von Finite State Machines umgesetzt. Konkret können für Agenten bzw. für deren Verhalten StateCharts modelliert werden. Zu diesem Zweck wurde ein grafischer StateChart-Designer entwickelt und integriert, der die Komplexität, die mit der agentenbasierten Modellierung und Simulation einhergeht, weitgehend vor dem Benutzer verbirgt. Gleichwohl ist es trotzdem möglich, komplexe Modelle zu entwickeln, da der Designer bei Bedarf die komplette und umfangreiche Welt der .NET Softwareentwicklung zur Verfügung stellt. Weiter konnten die Möglichkeiten für Erweiterungen, die Simio zur Verfügung stellt, so genutzt werden, dass aus Benutzersicht eine nahezu nahtlose Integration der Library in Simio erreicht werden konnte.

Insofern, dass das Hauptziel

*«Es soll eine Bibliothek erstellt werden, damit Problemstellungen für agentenbasierte Simulation in Simio von Benutzern ohne vertiefte Programmierkenntnisse modelliert werden können.»*

vollumfänglich erreicht und umgesetzt wurde, kann die Arbeit als Erfolg gewertet werden.

### 7.2 Potential zur Weiterentwicklung

Im Rahmen dieser Arbeit wurden einige Aspekte ausgeklammert oder nicht weiter vertieft. Dieser Abschnitt greift diese nochmals auf und zeigt, inwiefern sie für die Weiterverfolgung interessant sind. Wo vorhanden, werden zudem Lösungsansätze skizziert.

#### 7.2.1 IntelliSense und Syntax Highlighting für die Expressions sowie Functions

Der StateChart-Designer bietet verschiedene Möglichkeiten, C#-Expressions sowie C#-Functions zu editieren. Um die Entwicklung zu vereinfachen, könnten diese Editoren IntelliSense sowie Syntax Highlighting unterstützen. Da insbesondere für den StateChart-Designer Controls der DevExpress Suite eingesetzt werden (siehe Kapitel 2.3), bietet es sich an, hier entsprechend die Funktionalität der DevExpress-Controls zu nutzen. Insbesondere bieten die Komponenten bereits einen Mechanismus für Syntax Highlighting an. Weiter wird die Implementierung von IntelliSense durch das AutoComplete-Feature<sup>6</sup> ermöglicht.

#### 7.2.2 Weitere Perceptions

Für den Empfang von Nachrichten von anderen Agenten wurde die CommunicationPerception implementiert. Das Konzept der Perception ist jedoch allgemein gehalten, damit weitere Perceptions implementiert werden können (siehe Abschnitt 0 für weitere Informationen). In [1] wurde ein PathFinding-Behavior implementiert. Dieses erfordert z.B. die Erkennung von Hindernissen. Es wäre jedoch denkbar, weitere komplexe Wahrnehmungsmechanismen für Agenten zu implementieren. Das aktuelle Konzept ermöglicht dies auf einfache Art und Weise, damit der Fokus auf die Komplexität der konkreten Perception gelegt werden kann.

---

<sup>6</sup> IntelliSense und Syntax Highlighting mit DevExpress-Komponenten:

<https://documentation.devexpress.com/WindowsForms/12107/Controls-and-Libraries/Rich-Text-Editor/Syntax-Highlighting>

<https://www.devexpress.com/Support/Center/Question/Details/Q297129/showing-intellisense-on-screen-position>

<https://www.devexpress.com/Support/Center/Question/Details/Q245070/please-tell-me-how-to-display-list-box-in-xtrarichedit-at-the-caretposition-for>

## 7.2.3 Erweiterungen in Bezug auf Agent-Movement

Bzgl. Agent Movement wurde in der aktuellen Fassung nur die Bewegung im Continuous Space implementiert. Insbesondere in Abschnitt 3.3.10 wurden zwei weitere Möglichkeiten für Movement angeführt:

- GIS Space
- Discrete Space

Analog zum Perception-Modell wurde ein Action-Modell implementiert, welches die Erweiterung von zusätzlichen Actions ebenso einfach gestaltet (siehe Abschnitt 4.1.3.2 für weitere Informationen). Dies ermöglicht die Implementierung von zusätzlichen Bewegungsarten wie oben aufgeführt.

Weiter wären zusätzliche API-Funktionen für den Bereich Movement sinnvoll und nützlich. Z.B. könnten dies sein (nicht abschliessend – siehe auch Abschnitt 7.2.5):

- `MoveInTime()`
- `Stop()`
- `GetTimeToArrival`
- `GetIsMoving`
- ...

## 7.2.4 Optimierung bzgl. Performance und Memory

Der Fokus dieser Umsetzung lag in erster Linie bei der Funktionalität und dem benutzerfreundlichen Handling und nur sekundär auf Performance bzw. Optimierung. Diese wurden jedoch bei jedem Schritt berücksichtigt aber nicht explizit untersucht. Es haben sich bei der Verifizierung einige Ansätze bzgl. Optimierung im Bereich Performance gezeigt. Dies v.a. dann, wenn viele Agenten zum Einsatz kommen (z.B. 1000 oder mehr). Wie bereits [1, S.58] aufzeigt, könnte das Konzept der Bulk-Agenten implementiert werden. Mit Hilfe eines Bulk-Agenten werden durch einen einzelnen Agenten eine Masse von Agenten repräsentiert. Bei Bedarf bzw. beim Auftreten von bestimmten Aktionen/Ereignissen, könnte die Möglichkeit bestehen, die Auflösung dieses Bulk-Agenten zu erhöhen und diesen durch mehrere einzelne Agenten zu ersetzen.

## 7.2.5 Erweiterung der API

Die API stellt aktuell die unbedingt nötigen Funktionen für die Verwendung im StateChart-Designer zur Verfügung (siehe Kapitel 6.1). Im Zuge der Umsetzung von weiteren Modellen entsteht mit Sicherheit das Bedürfnis von erweiterter Funktionalität der API. Durch Kapselung kann diese bei entsprechendem Bedarf auf einfache Art und Weise erweitert werden.

## 7.2.6 Tiefere Integration in Simio

Für die Integration der SimioAgentLibrary – insbesondere der Agent Management Form sowie der Implementierung der StateMachine zur Laufzeit der Simulation wurde der AddIn- sowie der SteeringBehavior-Mechanismus von Simio verwendet. Dies ermöglicht die Integration für eine benutzerfreundliche Anwendung der SimioAgentLibrary. Es gibt jedoch einige Bedürfnisse, die eine tiefere Integration in Simio erfordern. Z.B. wäre es tw. wünschenswert, wenn bestehende Simio-Expressions verwendet werden könnten usw. Diese Integration müsste von Simio vorgenommen werden bzw. bedarf einer aktiven Unterstützung von Simio.

## 7.3 Danksagung

An dieser Stelle möchte ich mich bei all denjenigen bedanken, die mich während der Anfertigung dieser Masterarbeit unterstützt und motiviert haben.

Zuerst gebührt mein Dank Herrn Prof. Dr. Andreas Rinkel, der meine Masterarbeit betreut und begutachtet hat. Für die hilfreichen Anregungen und die konstruktive Kritik bei der Erstellung dieser Arbeit möchte ich mich herzlich bedanken.

Ein besonderer Dank gilt der Firma Simio LLC für die unkomplizierte Bereitstellung von Lizenzen mit vollem Funktionsumfang sowie für die hervorragende Supportunterstützung bei spezifischen Fragen und Problemen.

Abschliessend möchte ich mich bei meiner Frau Sybille bedanken, ohne deren Unterstützung mein Studium nicht möglich gewesen wäre.

Thomas Kehl, Berneck im August 2018

## 8 Management Summary

### 8.1 Ausgangslage

Agentenbasierte Modellierung und Simulation ist ein relativ moderner Ansatz zur Modellierung von Systemen. Er ermöglicht die Modellierung der Dynamik von komplexen sowie kybernetischen Systemen. Es handelt sich dabei oft um sich selbst organisierende Systeme, welche emergente Effekte erzeugen, wie z.B. das Fluchtverhalten von Menschen.

Mit der Software Simio ist es möglich, diskrete Ereignissimulationen durchzuführen. Jedoch bietet die Software von Haus aus keine (bzw. nur sehr spartanische) Möglichkeiten für agentenbasierte Simulationen. Die Stärke von Simio besteht in der objektorientierten Umsetzung der diskreten Ereignissimulation. Auf dieser Basis sind im Vorfeld dieser Arbeit die Anforderungen an eine agentenbasierte Simulation in Simio entwickelt worden. Im Zuge dessen hat sich folgender Hauptaspekt manifestiert:

Im Grundsatz soll ein Agent-Objekt entwickelt werden. Dieses Objekt weist folgende zentrale Eigenschaften auf:

- 1) Es soll eine eigene Logik beinhalten können.
- 2) Es soll die Umgebung wahrnehmen können.
- 3) Agent-Objekte sollen untereinander kommunizieren können.

Das zentrale Ziel der Arbeit kann folgendermassen umrissen werden:

*«Es soll eine Bibliothek erstellt werden, damit Problemstellungen für agentenbasierte Simulation in Simio von Benutzern ohne vertiefte Programmierkenntnisse modelliert werden können.»*

### 8.2 Vorgehen

Zu Beginn wurde eine Problemstellung formuliert, die alle drei zentralen Eigenschaften vereinigt. Auf dieser Basis wurde ein Umsetzungskonzept in Form einer Studie entwickelt. Weiter beschreibt die Arbeit die konkrete konzeptionelle Umsetzung sowie anschliessend die Runtime-Implementierung. Im Fokus steht dabei das Konzept und die Implementierung des StateChart-Designers. Dieser stellt den Kern dieser Arbeit dar. Mit ihm steht und fällt das oben definierte zentrale Ziel der Benutzerfreundlichkeit.

### 8.3 Ergebnis

Es wurde eine umfangreiche, praxistaugliche sowie v.a. benutzerfreundliche Library für agentenbasierte Modellierung und Simulation in Simio erstellt. Diese besteht aus der Objektlibrary um Agentenobjekte in Simio-Modelle einfügen zu können sowie einer Agent Management Form, welche einen grafischen Designer für die Modellierung des Agentenverhalten in Form von StateCharts zur Verfügung stellt. Weiter können für die States und Transitions des StateCharts C#-Expressions verwendet werden. Dies eröffnet vor allem für die Umsetzung von komplexen Problemstellungen die komplette Welt der .NET Softwareentwicklung.

### 8.4 Ausblick

Das Ergebnis bietet verschiedene Ansätze in Bezug auf die Weiterentwicklung. Zum einen kann die Funktionalität ausgebaut werden. Zum anderen kann der Komfort in Bezug auf die Benutzerfreundlichkeit für die Modellierung erhöht werden. Durch die Verwendung von entsprechend wartungsfreundlichen und erweiterbaren Softwarekonzepten ist beides problemlos und einfach möglich.

## 9 Verzeichnisse

### 9.1 Literaturverzeichnis

- [1] KEHL, THOMAS: Agentenbasierte Simulation mit Simio, Projektarbeit I, Winter 2016/17.
- [2] KEHL, THOMAS: Autonome Agenten, Seminararbeit II, Frühling 2017.
- [3] KEHL, THOMAS: Autonome Agenten in Simio, Projektarbeit II, Sommer 2017.
- [4] DEVEXPRESS: Diagrams, API/Examples/Instructions/Screenshots/Source Code;  
<https://documentation.devexpress.com/WindowsForms/114833/Controls-and-Libraries/Diagrams>. [Online, letzter Zugriff 03.05.2018].
- [5] ANYLOGIC: agent-based modeling (and downloads); <https://www.anylogic.com/use-of-simulation/agent-based-modeling/>. [Online, letzter Zugriff 04.07.2018].
- [6] WIKIBOOKS: Simulation with AnyLogic/Agent-Based Modeling;  
[https://en.wikibooks.org/wiki/Simulation\\_with\\_AnyLogic/Agent-Based\\_Modeling](https://en.wikibooks.org/wiki/Simulation_with_AnyLogic/Agent-Based_Modeling)  
[https://en.wikibooks.org/wiki/Simulation with AnyLogic/Agent-Based Modeling](https://en.wikibooks.org/wiki/Simulation_with_AnyLogic/Agent-Based_Modeling). [Online, letzter Zugriff 04.07.2018].
- [7] SIMIO LLC.: Simio Reference Guide; <http://cdn.simio.com/software/Simio%20Reference%20Guide.pdf>. [Online, letzter Zugriff 07.06.2018].
- [8] JOHN D. COOK: Simple Random Number Generation;  
<https://www.codeproject.com/Articles/25172/Simple-Random-Number-Generation>. [Online, letzter Zugriff 27.06.2018].
- [9] JOHNSON ET. AL.: Continuous Univariate Distributions, Wiley 1994.
- [10] ILY GRIGORYEV: AnyLogic 7 in Three Days: A Quick Course in Simulation Modeling, 3. Auflage, 8. Dez. 2014.
- [11] KEHL, THOMAS: Autonome Agenten, Seminararbeit I, Frühling 2016.



## 9.2 Abbildungsverzeichnis

Abbildung 1: Modellierungsprozess .....	7
Abbildung 2: Problemstellung Produktion .....	8
Abbildung 3: Klassenübersicht AgentLibraryAddIn .....	14
Abbildung 4: AddIn «SimioAgentLibrary» .....	14
Abbildung 5: Oberfläche «AgentManagementForm» .....	15
Abbildung 6: Modellbaum .....	16
Abbildung 7: Bearbeitungsbereich mit StateChart .....	17
Abbildung 8: Tools für die StateChart-Modellierung .....	17
Abbildung 9: Konfiguration eines States .....	18
Abbildung 10: Implementierung und Konfiguration einer C# Funktion .....	18
Abbildung 11: Hauptmenü .....	19
Abbildung 12: Meldungsausgabe .....	21
Abbildung 13: Klassendiagramm «AgentManagementForm» .....	21
Abbildung 14: Technische Komponentenübersicht der SimioAgentLibrary .....	22
Abbildung 15: Klassenübersicht StateChart-Designer .....	24
Abbildung 16: PropertyControls für den StateChart-Designer .....	25
Abbildung 17: Transitionen .....	26
Abbildung 18: Timeout-Transition .....	27
Abbildung 19: Rate-Transition .....	28
Abbildung 20: Condition-Transition .....	29
Abbildung 21: Sensitivity Problem .....	30
Abbildung 22: Message-Transition .....	31
Abbildung 23: AgentArrival-Transition .....	32
Abbildung 24: Branch .....	33
Abbildung 25: Simple bzw. Composite state .....	34
Abbildung 26: Szenario für Ausführungsreihenfolge .....	35
Abbildung 27: History State .....	36
Abbildung 28: Shallow vs. Deep History State .....	37
Abbildung 29: Entry Point und Initial State Pointer .....	38
Abbildung 30: Final State .....	39
Abbildung 31: Globale Konfiguration für Functions .....	40
Abbildung 32: Implementierung und Konfiguration einer einzelnen Funktion .....	41
Abbildung 33: Klassendiagramm für Implementierung der StateMachine .....	45
Abbildung 34: Start Simulation und Erstellung StateMachine .....	46
Abbildung 35: Detaillierung des Subprozess ModelStateMachine.Build() .....	47
Abbildung 36: Komponenten für Agent-SteeringBehavior .....	48
Abbildung 37: Konfiguration für Agent-SteeringBehavior .....	48
Abbildung 38: Ablauf des SteeringBehaviors bei einem Simulationszyklus .....	49
Abbildung 39: ComplexGateway .....	50
Abbildung 40: Prozesse des ComplexGateway-Objekts .....	52
Abbildung 41: Prozess um einen Agenten in den Freespace zu transferieren .....	53
Abbildung 42: SimioAgentLibrary einbinden .....	54
Abbildung 43: AgentLibrary in Simio .....	55
Abbildung 44: Manuelles Laden der SimioAgentLibrary .....	55
Abbildung 45: Struktur der API .....	56
Abbildung 46: Grundmodell für Konsumentenmarkt .....	58
Abbildung 47: Consumer-Objekt mit verschiedenen Symbolen und Farben .....	58
Abbildung 48: Berechnung «Current Symbol Index» für Consumer-Objekt .....	59

Abbildung 49: Agent Management Oberfläche mit Model und Agent «Consumer» .....	59
Abbildung 50: Zustände für einen Consumer .....	60
Abbildung 51: Konfiguration des Entry Points .....	60
Abbildung 52: Region der Consumer-Agenten im Facility-Window .....	61
Abbildung 53: FacilityName für StartGateway .....	61
Abbildung 54: Konfiguration der Transition «Ad» .....	62
Abbildung 55: Konfiguration der Transition «CantWait» .....	62
Abbildung 56: Konfiguration der Transition «Purchase» .....	63
Abbildung 57: Konfiguration der Transition «Discard» .....	63
Abbildung 58: Konfiguration der Transition «Contact» .....	63
Abbildung 59: Konfiguration der Transition «WoM» .....	64
Abbildung 60: Komplettes Zustandsdiagramm für Consumer Agent .....	64
Abbildung 61: Entry-Action und Exit-Action für Zustand "Potential Consumer" .....	66
Abbildung 62: Parameter für Funktion «SetConsumerSymbol» .....	66
Abbildung 63: Additional Expressions für Status-Plot .....	67
Abbildung 64: Dauer der Simulation .....	67
Abbildung 65: «Update Time Interval» für den Consumer-Agenten .....	68
Abbildung 66: Resultat ConsumerMarket .....	68
Abbildung 67: Resultat ConsumerMarket - Übersicht .....	69
Abbildung 68: StateChart Controller-Agent .....	70
Abbildung 69: StateChart EntPartX-Agent .....	71
Abbildung 70: Auslastung TempStorage .....	72
Abbildung 71: Modell für Problemstellung «Produktion» .....	73
Abbildung 72: Prozess für die Erstellung von EntPartA-Objekte .....	75

## 9.3 Tabellenverzeichnis

Tabelle 1: Aspekte und deren Umsetzungsanforderung(en) aus Problemstellung .....	10
Tabelle 2: Evaluationsmatrix Komponenten für StateChart-Designer .....	12
Tabelle 3: Gewichtetes Resultat der Evaluation .....	12
Tabelle 4: Speicherorte der Elemente der SimioAgentLibrary .....	19
Tabelle 5: Allgemeine Einstellungen für Elemente im StateChart-Designer .....	25
Tabelle 6: Mögliche Transitionsarten .....	26
Tabelle 7: Mögliche Werte für eine Einstellung die C# Expression unterstützt .....	27
Tabelle 8: Einstellungen für Timeout-Transition .....	27
Tabelle 9: Einstellungen für Timeout-Transition .....	28
Tabelle 10: Einstellungen für Condition-Transition .....	29
Tabelle 11: Einstellungen für Message-Transition .....	30
Tabelle 12: Einstellungen für AgentArrival-Transition .....	32
Tabelle 13: Einstellungen für Branch .....	33
Tabelle 14: Einstellungen für State-Objekt .....	33
Tabelle 15: Einstellungen für History State .....	36
Tabelle 16: Einstellungen für Entry Point-Objekt .....	38
Tabelle 17: Einstellungen für Final State .....	39
Tabelle 18: Agent Standardevents .....	39
Tabelle 19: Globale Konfigurationseinstellungen .....	40
Tabelle 20: Einstellungen für ComplexGateway .....	51
Tabelle 21: Prozessschritte für Prozess «OnEnteredInputBuffer» .....	52
Tabelle 22: Prozessschritte für Prozess «OnEnteredOutputBuffer» .....	53
Tabelle 23: Objekte der API .....	56
Tabelle 24: Aspekte und deren Umsetzungsanforderung(en) aus Problemstellung .....	74
Tabelle 25: Properties und Methoden der API .....	88
Tabelle 26: Properties und Methoden für Agent:IAgent .....	89
Tabelle 27: Properties und Methoden für AgentType:IAgentType .....	90
Tabelle 28: Properties und Methoden für IRandom .....	91
Tabelle 29: Properties und Methoden für IMessage .....	92
Tabelle 30: Properties und Methoden für IMovingLogic .....	93
Tabelle 31: Properties und Methoden für ISteering .....	93
Tabelle 32: Struktur der Projektdaten-CD .....	96

## 9.4 Codelistings

Listing 1: Signatur der Funktion ReadOrders() .....	41
Listing 2: API für komplexe Bewegungslogik .....	42
Listing 3: Interface IMovingLogic .....	42
Listing 4: Instanziierung und Konfiguration der StateMachine .....	44
Listing 5: Function SetInitialPosition .....	60
Listing 6: Function EntryPotentialConsumerState .....	65
Listing 7: Function EntryWantsToBuyState .....	65
Listing 8: Function EntryConsumerState .....	65
Listing 9: Function EntryConsumerState .....	65
Listing 10: Function SetConsumerSymbol .....	66
Listing 11: CSV-Datei einlesen .....	74
Listing 12: Simio-Event aufrufen .....	74
Listing 13: Versand Nachricht für Prüfung, ob Komponenten im Zwischenlager verfügbar .....	75

## 9.5 Glossar

Hinweis: Als Quelle für die Beschreibung der Begriffe im Glossar diente mehrheitlich Wikipedia. Da es sich um Begriffsdefinitionen handelt, ist Wikipedia als Quelle hinreichend geeignet.

Begriff	Beschreibung
<b>.NET Framework</b>	.NET Framework (sprich dot net) ist ein von Microsoft entwickeltes Software-Framework, das hauptsächlich auf Microsoft Windows läuft. Es enthält eine grosse Klassenbibliothek namens Framework Class Library (FCL) und bietet Sprachkompatibilität (jede Sprache kann in anderen Sprachen geschriebenen Code verwenden) über mehrere Programmiersprachen hinweg.
<b>3rd-Party-Libraries</b>	Eine 3rd-Party-Library ist eine Softwarekomponente eines Drittanbieters, die so entwickelt wurde, dass sie frei vertrieben und in beliebigen Applikationen verwendet werden kann.
<b>Action</b>	Bei einer Action handelt es sich um eine Aktion in Form von Softwarecode, die zu einem definierten Zeitpunkt ausgeführt wird.
<b>API</b>	Eine Programmierschnittstelle, häufig nur kurz API genannt (englisch application programming interface), ist ein Programmteil, der von einem Softwaresystem anderen Programmen zur Anbindung an das System zur Verfügung gestellt wird. Im Gegensatz zu einer Binärschnittstelle (ABI) definiert eine Programmierschnittstelle nur die Programmanbindung auf Quelltext-Ebene. Zur Bereitstellung solch einer Schnittstelle gehört meist die detaillierte Dokumentation der Schnittstellen-Funktionen mit ihren Parametern auf Papier oder als elektronisches Dokument.
<b>Assembly</b>	Eine Assembly ist eine Laufzeiteinheit, die aus Typen und anderen Ressourcen besteht. Alle Typen in einer Assembly haben dieselbe Versionsnummer. Oft hat eine Assembly nur einen Namespace und wird von einem Programm verwendet. Es kann jedoch mehrere Namespaces umfassen. Ein Namespace kann sich auch auf mehrere Assemblies verteilen. In großen Designs kann eine Assembly aus mehreren Dateien bestehen, die durch ein Manifest (d. h. ein Inhaltsverzeichnis) zusammengehalten werden. In C# ist eine Assembly die kleinste verwendete Bereitstellungseinheit und eine Komponente in .NET – vergleichbar mit einer JAR-Datei in Java.
<b>C#-Expression</b>	Eine C#-Expression ist eine in C# ausgedrückte Kombination aus einer oder mehreren Konstanten, Variablen, Operatoren und Funktionen. C# interpretiert die Expression um einen neuen Wert zu berechnen. Dieser Vorgang wird auch als Evaluation bezeichnet. Action, Conditions usw. bestehen aus C#-Expressions.
<b>Condition</b>	Eine Condition ist eine Bedingung, die erfüllt sein muss, damit eine bestimmte Action ausgeführt wird.
<b>CSV-Datei</b>	Das Dateiformat CSV steht für Englisch Comma-separated values (seltener Character-separated values) und beschreibt den Aufbau einer Textdatei zur Speicherung oder zum Austausch einfach strukturierter Daten. Die Dateinamenserweiterung lautet .csv. Ein allgemeiner Standard für das Dateiformat CSV existiert nicht, jedoch wird es im RFC 4180 grundlegend beschrieben. In CSV-Dateien können Tabellen oder Liste unterschiedlicher Länge abgebildet werden.
<b>DES</b>	Eine diskrete Ereignissimulation (DES) modelliert den Betrieb eines Systems als eine diskrete Folge von Ereignissen in der Zeit. Jedes Ereignis tritt zu einem bestimmten Zeitpunkt auf und markiert eine Zustandsänderung im System. Zwischen aufeinanderfolgenden Ereignissen wird keine Änderung im System angenommen; Somit kann die Simulation direkt von einem Ereignis zum nächsten springen. Siehe [11] für weitere Informationen.
<b>Drag&amp;Drop</b>	Drag and Drop (deutsch «Ziehen und Ablegen»), ist eine Methode zur Bedienung grafischer Benutzeroberflächen von Rechnern durch das Bewegen grafischer Elemente mittels eines Zeigegerätes. Ein Element kann damit gezogen und über einem möglichen Ziel losgelassen werden. Dieses kann zum Beispiel markierter Text oder das Symbol einer Datei sein. Im Allgemeinen kann Drag and Drop genutzt werden, um Aktionen auszuführen oder Beziehungen zwischen zwei abstrakten Objekten herzustellen.

Begriff	Beschreibung
<b>EFSM</b>	Eine Extended Finite State Machine (EFSM - deutsch: Erweiterter endlicher Zustandsautomat) ist eine Finite State Machine (kurz FSM), welche ausser ihrem Zustand noch weitere Variablen verwaltet, die den jeweiligen Zustandsübergang beeinflussen und/oder in Aktionen geändert werden können.
<b>Facility-Window</b>	In diesem Fenster stellt Simio das aktuelle Modell graphisch dar.
<b>Fallback</b>	Mit Fallback wird ein Wert bezeichnet, der verwendet wird, wenn alle Bedingungen die zu anderen Werten führen, negativ ausfallen.
<b>Freespace</b>	Mit Freespace bezeichnet Simio den kompletten Raum, in dem sich Agenten, die sich nicht auf einem definierten Netzwerk befinden, bewegen können.
<b>GIS</b>	Geoinformationssysteme, Geographische Informationssysteme (GIS) oder Räumliche Informationssysteme (RIS) sind Informationssysteme zur Erfassung, Bearbeitung, Organisation, Analyse und Präsentation räumlicher Daten. Geoinformationssysteme umfassen die dazu benötigte Hardware, Software, Daten und Anwendungen.
<b>Guard</b>	Schutzbedingungen (englisch Guard conditions) sind Boolesche Ausdrücke, die basierend auf dem Wert von erweiterten Zustandsvariablen und Ereignisparametern dynamisch ausgewertet werden. Guard-Bedingungen wirken sich auf das Verhalten einer Zustandsmaschine aus, indem Aktionen oder Transitionen nur dann aktiviert werden, wenn sie auf TRUE ausgewertet werden und sie deaktiviert werden, wenn sie auf FALSE ausgewertet werden.
<b>Hosted Compilation</b>	Unter Hosted Compilation wird die Verwendung eines Computerprogramms als Teil eines Programms verstanden, das zur Laufzeit weitere Teile desselben Programms erzeugt - zum Beispiel einen Compiler, der seinen eigenen Quellcode kompilieren kann. Typischerweise handelt es sich um PlugIn-Systeme, welche so erweitert werden können.
<b>IntelliSense</b>	IntelliSense ist ein ursprünglich von Microsoft angebotenes Hilfsmittel zur automatischen Vervollständigung bei der Bearbeitung von Quellcode durch einen Programmierer. Dabei erhält der Programmierer während seiner Arbeit zusätzliche Informationen und Auswahlmöglichkeiten, die ihm die Arbeit erleichtern und die insbesondere die Menge des manuell einzugebenden Quellcodes reduzieren soll.
<b>Serialisierung</b>	Die Serialisierung ist eine Abbildung von strukturierten Daten auf eine sequenzielle Darstellungsform. Serialisierung wird hauptsächlich für die Persistierung von Objekten z.B. in Dateien, Zeichenketten usw. verwendet. Nach der Serialisierung liegt ein Objekt mehrfach vor: Als externe Darstellung (z.B. als Datei) und im Arbeitsspeicher. Wird nach der Serialisierung eine Änderung am Objekt im Arbeitsspeicher vorgenommen, hat dieses keine Auswirkung auf das serialisierte Objekt in der externen Darstellung. Die Umkehrung der Serialisierung, also die Umwandlung eines Datenstroms in Objekte, wird als Deserialisierung bezeichnet.
<b>Simio-State</b>	Bei einem Simio-State handelt es sich um eine State-Variable, die in Simio erstellt wurde. Der Wert eines Simio-States kann zur Laufzeit der Simulation ausgelesen und insbesondere verändert werden.
<b>Sink</b>	Englisch für Senke. Stelle, an der Entitäten während einer Simulation terminiert werden.
<b>State</b>	Mit State wird der Zustand der EFSM bezeichnet. Verschiedene States können mittels Transitions verbunden werden. So ist es möglich zwischen States zu wechseln. Ein State kann eine Entry- sowie eine Exit-Action beinhalten. Diese werden entsprechend bei der Aktivierung bzw. der Deaktivierung des Zustands ausgeführt.
<b>SteeringBehavior</b>	Ein SteeringBehavior bezeichnet in Simio eine implementierte Logik für das Verhalten von Entities. Bei der Konfiguration zu einem Entity kann ein vorhandenes SteeringBehavior ausgewählt werden.
<b>Transition</b>	Eine Transition verbindet States in einer EFSM. Eine Transition kann eine Guard beinhalten, die bestimmt, ob die Transition, d.h. der Wechsel zu einem State ausgeführt werden soll.
<b>Triangular-Verteilung</b>	Siehe Tabelle 28.
<b>UI</b>	Die Benutzeroberfläche (User Interface, UI) umfasst die interaktiven Aspekte von Computerbetriebssystemen. Insbesondere wird die grafische Computeranzeige auf einem Bildschirm als UI bezeichnet.

## 10 Anhänge

Anhang A: API-Dokumentation

Anhang B: Known Issues

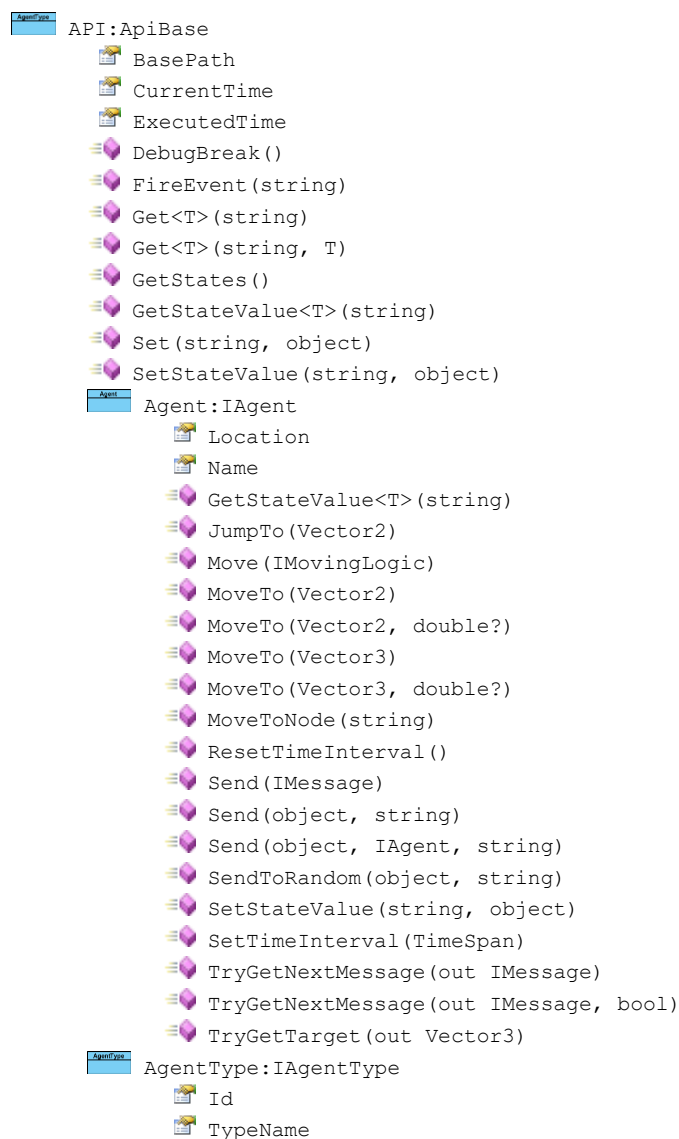
Anhang C: Planung

Anhang D: Protokolle

Anhang E: Digitaler Anhang

### Anhang A: API-Dokumentation

Nachfolgende Übersicht zeigt die Struktur der API. Die einzelnen Members in Form von Objekten, Properties und Methoden sind anschliessend in den nachfolgenden Tabellen beschrieben.



**Random:IRandom**

- GetBeta(double, double)
- GetCauchy(double, double)
- GetChiSquare(double)
- GetExponential()
- GetExponential(double)
- GetGamma(double, double)
- GetInverseGamma(double, double)
- GetLaplace(double, double)
- GetLogNormal(double, double)
- GetNormal()
- GetNormal(double, double)
- GetStudentT(double)
- GetTriangular(double, double)
- GetTriangular(double, double, double)
- GetTriangularAverage(double, double)
- GetUniform()
- GetUniform(double, double)
- GetUniform(int, int)
- GetWeibull(double, double)
- RandomTrue(double)

**IMessage**

- AllReceiver
- AllReplyTo
- ConversationId
- Content
- InReplyTo
- IsBroadcast
- IsRepsonse
- Language
- Ontology
- Performative
- ReplyWith
- Sender
- AddReceiver(IAgent)
- AddReplyTo(IAgent)
- ClearAllReceiver()
- ClearAllReplyTo(IAgent)
- CreateReply(object)
- CreateReply(Performative)
- CreateReply(Performative, Language)
- RemoveReceiver(IAgent)
- RemoveReplyTo(IAgent)

**IMovingLogic**

- CurrentAction
- DoSteer(ISTeering, IAgent)
- TryGetNewTimeInterval(IAgent)

**ISTeering**

- TimeInterval
- Continue()
- DoNotMove()
- FinishMove()
- MoveTo(Vector3, double?)
- SetLocation(Vector3)

API – Members	Typ
<b>IAgent</b> Agent { <b>get</b> ; }	Property
Gibt den aktuellen Agenten zurück (siehe Tabelle 26).	
<b>IAgentType</b> AgentType { <b>get</b> ; }	Property
Gibt in Bezug auf den aktuellen Agenten den AgentType zurück (siehe Tabelle 27).	
<b>string</b> BasePath { <b>get</b> ; }	Property
Ermittelt den Pfad der aktuellen Simio-Projektdatei.	
<b>DateTime</b> CurrentTime { <b>get</b> ; }	Property
Aktuelle Simulationszeit.	
<b>TimeSpan</b> ExecutedTime { <b>get</b> ; }	Property
Ausgeführte Zeitspanne der Simulation.	
<b>IRandom</b> Random { <b>get</b> ; }	Property
Gibt das Objekt für die Erzeugung von Zufallswerten gemäss bestimmten Verteilungen zurück (siehe Tabelle 28).	
<b>void</b> DebugBreak()	Methode
Wird die Simulation mit angefügtem Debugger (z.B. von Visual Studio) ausgeführt, bewirkt der Aufruf dieser Methoden einen Haltepunkt im Debugger.	
<b>void</b> FireEvent( <b>string</b> eventname)	Methode
Ausführung eines Events in Simio. Der Event muss im aktuellen Modell existieren, ansonsten tritt ein Fehler auf.	
<b>T</b> Get< <b>T</b> >( <b>string</b> key)	Methode
Lesen einer globalen Variable.	
<b>T</b> Get< <b>T</b> >( <b>string</b> key, <b>T</b> def)	Methode
Lesen einer globalen Variable. Ist die Variable nicht vorhanden oder entspricht nicht dem Typ <b>T</b> , wird def zurückgegeben.	
<b>Dictionary&lt;string, object&gt;</b> GetSates()	Methode
Gibt eine Liste mit allen Simio-States und deren aktuellen Werte zurück.	
<b>T</b> GetStateValue< <b>T</b> >( <b>string</b> statename)	Methode
Liest eine Simio-Statevariable des Modells und gibt deren aktuellen Wert zurück.	
<b>void</b> Set( <b>string</b> key, <b>object</b> value)	Methode
Speichern einer globalen Variable.	
<b>void</b> SetStateValue( <b>string</b> statename, <b>object</b> value)	Methode
Schreibt eine Simio-Statevariable des Modells.	
<b>IAgent</b> Agent { <b>get</b> ; }	Objekt
Aktuelles Agenten-Objekt (siehe Tabelle 26)	
<b>IAgentType</b> AgentType { <b>get</b> ; }	Objekt
Aktuelles AgentType-Objekt (siehe Tabelle 27)	

Tabelle 25: Properties und Methoden der API



Agent:IAgent – Members	Typ
<b>Vector3</b> Location { <b>get</b> ; }	Property
Aktuelle Position des Agenten.	
<b>string</b> Name { <b>get</b> ; }	Property
Name des Agenten.	
<b>T</b> GetStateValue< <b>T</b> >( <b>string</b> statename)	Methode
Liest eine Simio-Statevariable des Agenten und gibt deren aktuellen Wert zurück.	
<b>void</b> JumpTo( <b>Vector2</b> target)	Methode
Der Agent wird direkt an die Position <b>target</b> gesetzt.	
<b>void</b> Move( <b>IMovingLogic</b> movingLogic)	Methode
Weisst dem Agenten eine <b>MovingLogic</b> zu. Dieses Objekt beinhaltet die Logik, wie sich der Agent bewegen soll (siehe 3.3.10 sowie Tabelle 30).	
<b>void</b> MoveTo( <b>Vector2</b> target)	Methode
Der Agent bewegt sich mit seiner Standardgeschwindigkeit zum 2D-Punkt <b>target</b> .	
<b>void</b> MoveTo( <b>Vector2</b> target, <b>double?</b> velocity)	Methode
Der Agent bewegt sich mit der Geschwindigkeit <b>velocity</b> zum 2D-Punkt <b>target</b> .	
<b>void</b> MoveTo( <b>Vector3</b> target)	Methode
Der Agent bewegt sich mit seiner Standardgeschwindigkeit zum 3D-Punkt <b>target</b> .	
<b>void</b> MoveTo( <b>Vector3</b> target, <b>double?</b> velocity)	Methode
Der Agent bewegt sich mit der Geschwindigkeit <b>velocity</b> zum 3D-Punkt <b>target</b> .	
<b>void</b> MoveToNode( <b>string</b> nodename)	Methode
Der Agent bewegt sich im Netzwerk zum Node <b>nodename</b> (siehe Kapitel 0)	
<b>void</b> ResetTimeInterval()	Methode
Verwendet das für das <b>SteeringBehavior</b> konfigurierte <b>TimeInterval</b> .	
<b>void</b> Send( <b>IMessage</b> message)	Methode
Versendet eine Nachricht. Dabei handelt es sich um eine Broadcast-Nachricht, die an alle Agenten versendet wird ( <b>IMessage</b> siehe Tabelle 29).	
<b>void</b> Send( <b>object</b> msg, <b>string</b> replyWith = <b>null</b> )	Methode
Versendet eine Nachricht mit <b>replyWith</b> als Antwort auf eine andere Nachricht. Dabei handelt es sich um eine Broadcast-Nachricht, die an alle Agenten versendet wird.	
<b>void</b> Send( <b>object</b> msg, <b>IAgent</b> dest, <b>string</b> replyWith = <b>null</b> )	Methode
Versendet eine Nachricht mit <b>replyWith</b> als Antwort auf eine andere Nachricht. Die Nachricht wird nur an den Agenten <b>dest</b> gesendet ( <b>IAgent</b> siehe Tabelle 26).	
<b>void</b> SendToRandom( <b>object</b> msg, <b>string</b> replyWith = <b>null</b> )	Methode
Versendet eine Nachricht mit <b>replyWith</b> als Antwort auf eine andere Nachricht. Die Nachricht wird zufällig an einen Agenten (sofern vorhanden) gesendet.	
<b>void</b> SetStateValue( <b>string</b> statename, <b>object</b> value)	Methode
Schreibt eine Simio-Statevariable des Agenten.	
<b>void</b> SetTimeInterval( <b>TimeSpan</b> interval)	Methode
Ändert das <b>TimeInterval</b> der Simulation auf <b>interval</b> .	
<b>bool</b> TryGetNextMessage(out <b>IMessage</b> message)	Methode
Ermittelt die nächste empfangene Nachricht und entfernt diese aus der Schlange. Ist keine Nachricht vorhanden, gibt die Methode <b>false</b> zurück ( <b>IMessage</b> siehe Tabelle 29).	
<b>bool</b> TryGetNextMessage(out <b>IMessage</b> message, <b>bool</b> remove)	Methode
Ermittelt die nächste empfangene Nachricht und entfernt diese auf Wunsch aus der Schlange. Ist keine Nachricht vorhanden, gibt die Methode <b>false</b> zurück ( <b>IMessage</b> siehe Tabelle 29).	
<b>bool</b> TryGetTarget(out <b>Vector3</b> target)	Methode
Wenn sich der Agent bewegt, kann das Ziel ( <b>target</b> ) gelesen werden. Die Methode gibt <b>false</b> zurück, wenn sich der Agent nicht bewegt und somit kein definiertes Ziel hat.	

Tabelle 26: Properties und Methoden für Agent:IAgent

AgentType:IAgentType – Members	Typ
<code>Guid Id { get; }</code>	Property
Id des AgentTyp.	
<code>string TypeName { get; }</code>	Property
Name des AgentTyp.	

**Tabelle 27: Properties und Methoden für AgentType:IAgentType**

`IRandom` definiert diverse Methoden, um Zufallszahlen zu erzeugen. Als Basis wurde [8] verwendet und mit einigen weiteren Methoden (z.B. für die Triangular-Verteilung) ergänzt. Die Beschreibungen der Verteilungsfunktionen sind angelehnt an [9].

IRandom – Members	Typ
<code>double GetBeta(double a, double b)</code>	Methode
Die Beta-Verteilung ist eine kontinuierliche Verteilung, die sowohl obere als auch untere Schranken aufweist. Da viele reale Situationen auf diese Weise begrenzt werden können, kann die Beta-Verteilung empirisch verwendet werden, um die tatsächliche Verteilung zu schätzen ohne viele Daten zur Verfügung zu haben.	
<code>double GetCauchy(double median, double scale)</code>	Methode
Die Cauchy-Verteilung ist eine unbegrenzte kontinuierliche Verteilung, die einen scharfen zentralen Peak, aber signifikant breite Ausläufe aufweist. Die Ausläufe sind viel gewichtiger als die Ausläufe der Normalverteilung.	
<code>double GetChiSquare(double degreesOfFreedom)</code>	Methode
Die Chi-Quadrat-Verteilung ist eine kontinuierliche Verteilung, die auf der unteren Seite begrenzt ist. Sie wird vorwiegend für Hypothesentests und eher weniger für die Modellierung von Daten verwendet.	
<code>double GetExponential()</code>	Methode
Die Exponentialverteilung ist eine kontinuierliche Verteilung, die auf der unteren Seite begrenzt ist. Ihre Form ist immer gleich, beginnend bei einem endlichen Wert und kontinuierlich abnehmend mit grösserem x. Die exponentielle Verteilung nimmt schnell ab, wenn x zunimmt. Die Exponentialverteilung wird häufig verwendet, um die Zeit zwischen zufälligen Ereignissen darzustellen, z. B. die Zeit zwischen Ankünften an einer bestimmten Position in einem Warteschlangenmodell oder die Zeit zwischen Ausfällen in Zuverlässigkeitsmodellen. Sie wird z.B. auch verwendet, um die Servicezeiten einer bestimmten Operation darzustellen.	
<code>double GetExponential(double mean)</code>	Methode
Siehe <code>GetExponential()</code> – zusätzlich spezifizierter Mittelwert.	
<code>double GetGamma(double shape, double scale)</code>	Methode
Die Gamma-Verteilung ist eine kontinuierliche Verteilung, die an der unteren Seite begrenzt ist. Die Gamma-Verteilung wird verwendet, um z.B. Lebensdauern, Vorlaufzeiten, persönliche Einkommensdaten, eine Bevölkerung über ein stabiles Gleichgewicht oder Zwischenankunftszeiten darzustellen.	
<code>double GetInverseGamma(double shape, double scale)</code>	Methode
Reziproke Gamma-Verteilung. Sie wird vor allem für die Modellierung von unsicheren Grössen verwendet. Anwendungsgebiet ist u.a. Machine Learning.	
<code>double GetLaplace(double mean, double scale)</code>	Methode
Die Laplace-Verteilung, manchmal auch als doppelte Exponentialverteilung bezeichnet, ist eine unbegrenzte kontinuierliche Verteilung, die einen sehr scharfen zentralen Peak aufweist. Sie kann verwendet werden, um die Differenz zweier unabhängiger und gleichverteilter Exponentiale zu beschreiben. Sie wird auch in der Fehleranalyse verwendet.	
<code>double GetLogNormal(double mu, double sigma)</code>	Methode
Die Lognormalverteilung ist eine kontinuierliche Verteilung, die auf der unteren Seite begrenzt ist. Die Lognormal-Verteilung wird in vielen verschiedenen Bereichen verwendet, einschliesslich der Verteilung der Partikelgrösse in natürlich vorkommenden Aggregaten, Staubkonzentration in industriellen Atmosphären, die Verteilung von Mineralien in niedrigen Konzentrationen, Dauer der Abwesenheit von Krankheit, Arzt Beraterzeit, Lebensdauerverteilungen in Zuverlässigkeit, Einkommensverteilung, Mitarbeiterbindung und viele Anwendungen zur Modellierung von Gewicht, Grösse usw.	

IRandom – Members (Fortsetzung)	Typ
<b>double</b> GetNormal()	Methode
Die Normalverteilung (hier mit Mittelwert = 0 und Standardabweichung = 0) ist eine unbegrenzte kontinuierliche Verteilung. Wegen ihrer Eigenschaft, eine zunehmende Summe kleiner, unabhängiger Fehler darzustellen, findet die Normalverteilung viele Anwendungen in der Statistik. Die Normalverteilung wird häufig verwendet, um symmetrische Daten darzustellen, leidet aber darunter, dass sie in beiden Richtungen unbegrenzt ist. Wenn bekannt ist, dass die Daten eine untere Grenze haben, können diese besser durch geeignete Parametrisierung der Lognormal-, Weibull- oder Gamma-Verteilungen dargestellt werden. Wenn bekannt ist, dass die Daten sowohl obere als auch untere Grenzen haben, kann die Beta-Verteilung verwendet werden.	
<b>double</b> GetNormal( <b>double</b> mean, <b>double</b> standardDeviation)	Methode
Siehe GetNormal() – zusätzlich spezifizierter Mittelwert und Standardabweichung.	
<b>double</b> GetStudentT( <b>double</b> degreesOfFreedom)	Methode
Die t-Verteilung wird insbesondere für Hypothesentests und Konfidenzintervalle verwendet.	
<b>double</b> GetTriangular( <b>double</b> min, <b>double</b> max)	Methode
Die Dreiecksverteilung ist eine kontinuierliche Verteilung, die auf beiden Seiten begrenzt ist. Sie wird häufig verwendet, wenn keine oder nur wenige Daten verfügbar sind.	
<b>double</b> GetTriangular( <b>double</b> min, <b>double</b> max, <b>double</b> mode)	Methode
Siehe GetTriangular() – zusätzlich spezifizierter Modalwert (mode). Dies entspricht dem am häufigsten vorkommenden Wert.	
<b>double</b> GetTriangularAverage( <b>double</b> average, <b>double</b> variability)	Methode
Siehe GetTriangular() – erfüllt jedoch folgende Aussage: ungefähr XY (average) mit +/- Abweichung (variability).	
<b>double</b> GetUniform()	Methode
Die Gleichverteilung ist eine kontinuierliche Verteilung, die auf beiden Seiten begrenzt ist [0, 1]. Sie wird verwendet, um eine Zufallsvariable mit konstanter Wahrscheinlichkeit darzustellen.	
<b>double</b> GetUniform( <b>double</b> minValue, <b>double</b> maxValue)	Methode
Siehe GetUniform() – zusätzlich spezifiziertes Minimum und Maximum (double).	
<b>int</b> GetUniform( <b>int</b> minValue, <b>int</b> maxValue)	Methode
Siehe GetUniform() – zusätzlich spezifiziertes Minimum und Maximum (int).	
<b>double</b> GetWeibull( <b>double</b> shape, <b>double</b> scale)	Methode
Die Weibull-Verteilung ist eine kontinuierliche Verteilung, die auf der unteren Seite begrenzt ist. Sie wird oft verwendet, um die Festigkeit von Materialien zu modellieren. Weiter wird sie z.B. verwendet, um die Abnutzungslebensdauer in Bezug auf Zuverlässigkeit, Windgeschwindigkeit, Niederschlagsintensität, gesundheitsbezogene Probleme, Keimung, Dauer von industriellen Betriebsunterbrechungen, Migrationssysteme und Gewitterdaten darzustellen.	
<b>bool</b> RandomTrue( <b>double</b> p)	Methode
Erzeugt true mit der gegebenen Wahrscheinlichkeit p. Die Wahrscheinlichkeit von false ist entsprechend 1 – p.	

**Tabelle 28: Properties und Methoden für IRandom**

IMessage – Members		Typ
<code>List&lt;IAgent&gt; AllReceiver { get; }</code>	Gibt alle Empfänger der Nachricht zurück.	Property
<code>List&lt;IAgent&gt; AllReplyTo { get; }</code>	Gibt alle Empfänger der Antwortnachricht zurück.	Property
<code>string ConversationId { get; }</code>	Konversations-Id der Nachricht.	Property
<code>object Content { get; }</code>	Inhalt der Nachricht.	Property
<code>string InReplyTo { get; }</code>	Wenn es sich um eine Antwortnachricht handelt, wird hier die Identifikation, die in <code>ReplyWith</code> übermittelt wurde, übernommen.	Property
<code>bool IsBroadcast { get; }</code>	Handelt es sich um eine Broadcast-Nachricht ohne bestimmten Empfänger?	Property
<code>bool IsResponse { get; }</code>	Handelt es sich um eine Antwortnachricht?	Property
<code>Language Language { get; }</code>	Sprache der Nachricht.	Property
<code>string Ontology { get; }</code>	Kontext der Nachricht (siehe [2, S.22]).	Property
<code>Performative Performative { get; }</code>	Sprachaktverb (siehe [2, S.22]).	Property
<code>string ReplyWith { get; }</code>	Identifikation, die in die Antwortnachricht übernommen werden soll.	Property
<code>IAgent Sender { get; }</code>	Absender der Nachricht.	Property
<code>void AddReceiver(IAgent receiver)</code>	Fügt einen Empfänger zur Nachricht hinzu.	Methode
<code>void AddReplyTo(IAgent replyTo)</code>	Fügt einen Empfänger für die Antwort der Nachricht hinzu.	Methode
<code>void ClearAllReceiver()</code>	Löscht alle Empfänger.	Methode
<code>void ClearAllReplyTo()</code>	Löscht alle Antwort-Empfänger.	Methode
<code>IMessage CreateReply(object content)</code>	Erstellt eine Antwort mit dem Inhalt <code>content</code> .	Methode
<code>IMessage CreateReply(Performative performative)</code>	Erstellt eine Antwort mit dem Sprachverb <code>performative</code> (siehe [2, S.22]).	Methode
<code>IMessage CreateReply(Performative performative, Language language)</code>	Erstellt eine Antwort mit dem Sprachverb <code>performative</code> (siehe [2, S.22]) und der Sprache <code>language</code> .	Methode
<code>void RemoveReceiver(IAgent receiver)</code>	Entfernt den Empfänger <code>receiver</code> .	Methode
<code>void RemoveReplyTo(IAgent replyTo)</code>	Entfernt den Antwort-Empfänger <code>replyTo</code> .	Methode

**Tabelle 29: Properties und Methoden für IMessage**

Die Methode `void Move(IMovingLogic movingLogic)` ermöglicht die Definition einer komplexen Bewegungslogik. Die API für `IMovingLogic` sowie `ISteering` sind nachfolgend beschrieben. Weitere Informationen für die Verwendung von `IMovingLogic` und `ISteering` sind in Abschnitt 3.3.10 zu finden.

IMovingLogic – Members	Typ
MovingAction CurrentAction { <b>get</b> ; }	Property
Action, die dem Agenten zugewiesen werden soll.	
<b>void</b> DoSteer( <b>ISteering</b> steering, <b>IAgent</b> agent)	Methode
Ruft die Bewegungslogik auf.	
<b>bool</b> TryGetNewTimeInterval( <b>out double</b> newTimeInterval)	Methode
Wenn die Bewegungslogik ein neues TimeInterval definiert wird dieses hier zurückgegeben. Ansonsten gibt die Methode <b>false</b> zurück.	

Tabelle 30: Properties und Methoden für IMovingLogic

ISteering – Members	Typ
<b>double</b> TimeInterval { <b>get</b> ; <b>set</b> ; }	Property
TimeInterval für die Bewegungssteuerung.	
<b>void</b> Continue()	Methode
Die Bewegung soll ohne Änderung fortgesetzt werden.	
<b>void</b> DoNotMove()	Methode
Der Agent soll sich nicht bewegen.	
<b>void</b> FinishMove()	Methode
Die Bewegung soll beendet werden. Dies bewirkt, dass der Agent den ComplexGateway verlässt.	
<b>void</b> MoveTo( <b>Vector3</b> target, <b>double?</b> newVelocity = <b>null</b> )	Methode
Der Agent bewegt sich mit der Geschwindigkeit <b>velocity</b> zum 3D-Punkt <b>target</b> .	
<b>void</b> SetLocation( <b>Vector3</b> location)	Methode
Der Agent wird direkt an die Position <b>target</b> gesetzt.	

Tabelle 31: Properties und Methoden für ISteering

## Anhang B: Known Issues

Nachfolgende Auflistung enthält die aktuell bekannten Probleme der Implementierung. Diese betreffen nicht den Kern der Funktionalität und wurden aus diesem Grunde im Rahmen der Arbeit nicht adressiert.

- Wird in Simio eine neue Projektdatei geladen, nachdem die Agent Management Form bereits geöffnet wurde, wird diese nicht immer aktualisiert. In der Folge muss Simio beendet und neu gestartet werden.
- Wird in Simio ein Agent aus einem Model gelöscht, werden eventuell offene StateCharts in der Agent Management Form nicht automatisch geschlossen. Diese müssen manuell geschlossen werden.
- Wenn in der Agent Management Form nicht gespeicherte Änderungen vorhanden sind, werden diese nicht gespeichert, wenn Simio geschlossen wird. Es muss zuerst die Agent Management Form geschlossen und Simio erst anschliessend beendet werden.

## Anhang C: Planung

Aufgaben	Dauer	18.09.2017	25.09.2017	02.10.2017	09.10.2017	16.10.2017	23.10.2017	30.10.2017	06.11.2017	13.11.2017	20.11.2017	27.11.2017	04.12.2017	11.12.2017	18.12.2017	25.12.2017	01.01.2018	08.01.2018	15.01.2018	22.01.2018	29.01.2018	05.02.2018	12.02.2018	19.02.2018	26.02.2018	05.03.2018	12.03.2018	19.03.2018	26.03.2018	02.04.2018	09.04.2018	16.04.2018	23.04.2018	30.04.2018	07.05.2018	14.05.2018	21.05.2018	28.05.2018	04.06.2018	11.06.2018	18.06.2018	25.06.2018	02.07.2018	09.07.2018	16.07.2018	23.07.2018	30.07.2018	06.08.2018	
Aufgabenstellung, Vorarbeiten	40.50																																																
Planung, Konzeption • der kompletten Arbeit	20.25																																																
Konzeption und Aufbau der Modelle • Konkretisierung der Fragestellungen die den Modellen zugrundeliegen • Identifikation der Agenten und deren Anforderungen • Erstellung der Modelle in Simio	20.25																																																
Anforderungsanalyse anhand der Modelle • Identifikation der Entwicklungsanforderungen	40.50																																																
Generalisierung und Optimierung • Generalisierung der Anforderungen für den breiten Einsatz d.h. unabhängig der drei Problemstellungen • Optimierung der Anforderungen für Verwendung durch den User mittels minimalem Aufwand	81.00																																																
Konzeption der Entwicklungen am Behavior (Logik) • Wo sind am Behavior Anpassungen nötig • Recherche/ Entwurf Algorithmen • Klassenmodelle erstellen	60.75																																																
Konzeption/Design UI • Benötigte Funktionalität/Möglichkeiten der UI identifizieren • Design der UI	60.75																																																
Prototyping, Paper Studium • Erstellung von Prototypen anhand Konzeption	40.50																																																
Umsetzung • Implementierung • Tests	182.25																																																
Implementierung der Modelle für Verifikation • Getting Started • Anforderungsaspekte der Problemstellung	60.75																																																
Dokumentation • Erstellung der Dokumentation anhand den Notizen	162.00																																																
Korrekturlesen / Korrigieren	40.50																																																
Abschluss (Drucken, Binden, Abgabe)	20.25																																																
830.25																																																	

Zeitliche Aufteilung:  
 • 810h total  
 • ~20h/Woche  
 • 40 Wochen

## Anhang D: Protokolle

### C.1 19. September 2017

Teilnehmer: Andreas Rinkel, Thomas Kehl

Besprochene Themen: 1) Inhalt der Masterarbeit  
2) Weiteres Vorgehen

#### 1) Inhalt der Masterarbeit

Die Masterarbeit knüpft an die Projektarbeit II an. Das Ziel ist, die Möglichkeit zur agentenbasierten Modellierung und Simulation so in Simio zu integrieren, damit sie sich in die intuitive Modellierungsphilosophie von Simio einfügt.

Es werden drei Problemstellungen formuliert, die als Basis für die Umsetzung dienen.

#### 2) Weiteres Vorgehen

Es wird eine konkrete Aufgabenstellung formuliert. Als Basis dienen die drei Problemstellungen. Diese sollen in der Folge als Modelle in Simio umgesetzt werden.

### C.2 08. November 2017

Teilnehmer: Andreas Rinkel, Thomas Kehl

Besprochene Themen: 1) Beurteilung des aktuellen Standes  
2) Weiteres Vorgehen

#### 1) Beurteilung des aktuellen Standes

Die am 19.09.2017 formulierten Problemstellungen wurden zwischenzeitlich als Modelle in Simio umgesetzt. Diese Modelle wurden besprochen und beurteilt. Um die Komplexität der Modelle einzugrenzen, wurden für die Pfade der Agenten «Path»-Objekte verwendet. So kann auf die Agentenlogik fokussiert und die Bewegung der Agenten im «Freespace» vorderhand vernachlässigt werden. Das Objekt «Place» wurde erweitert, damit es einen Input- sowie Outputnode besitzt, an denen die Pfade mit dem Objekt verbunden werden können.

Die Modelle sind zweckmässig und eignen sich als Basis für die weitere Umsetzung.

#### 2) Weiteres Vorgehen

Auf Basis der Modelle wird nun das UI für die Agentenlogik entwickelt.

## C.3 21. März 2018

Teilnehmer: Andreas Rinkel, Thomas Kehl

Besprochene Themen: 1) Beurteilung des aktuellen Standes  
2) Weiteres Vorgehen

### 1) Beurteilung des aktuellen Standes

Die Implementierung wurde soweit abgeschlossen. Vorführung und Besprechung des StateChart-Designers sowie dessen Funktionalität anhand des Modells «Problemstellung: Produktion» (sowie dessen Simulation). Es wurde ein kompletter visueller StateChart-Designer implementiert. Im Gegenzug wird auf die konkrete Umsetzung der Problemstellungen 2 und 3 verzichtet.

Andres Rinkel ist begeistert von der Umsetzung.

### 2) Weiteres Vorgehen

Weiter soll nun die Dokumentation vervollständigt werden. Darüber hinaus sollen folgende Teile dokumentiert werden:

- Hello World-Sample. Beispielsweise in der Form, dass sich Agenten im 2D-Raum bewegen und sich kommunikativ Befehle erteilen.
- Walkthrough durch die Implementierung der Problemstellung «Produktion»
- Dokumentation der API für die Verwendung für Actions usw. im StateChart-Designer.

## Anhang E: Digitaler Anhang

Auf der beiliegenden CD befinden sich die Projektdaten mit dem Stand vom 11. August 2018. Der Inhalt der CD ist wie folgt strukturiert:

Name	Inhalt
Masterarbeit Thomas Kehl - Bericht.pdf	Masterarbeit – dieses Dokument.
SimioAgentLibraryInstaller.msi	Installer für SimioAgentLibrary. Für Installation siehe Kapitel 5.
AgentLibrary.sfp	Simio-Projektdatei welche die Simio-Objekte für die SimioAgentLibrary enthält. Diese ist in SimioAgentLibraryInstaller.msi enthalten und wird vom Installer installiert.
Verzeichnis SimioAgentLibrary_Bin	UserExtension SimioAgentLibrary in kompilierter Form (inkl. aller referenzierter 3rd-Party-Libraries). Dies dient der Vollständigkeit und ist in SimioAgentLibraryInstaller.msi enthalten und wird vom Installer installiert.
Verzeichnis SimioAgentLibrary_Dev	Projekt für Visual Studio 2017 mit der Implementierung der UserExtensions. Es ist zu beachten, dass für die Entwicklung die DevExpress-Suite benötigt wird (siehe 2.3 sowie [4]).
Verzeichnis StateMachineCode	C#-Code der StateMachine für die Modelle ConsumerMarket sowie Manufacturing. Für weitere Informationen siehe Abschnitt 3.1.4.3.
Verzeichnis SimioModels	Simio-Modelle ConsumerMarket und Manufacturing. Für weitere Informationen siehe Kapitel 6.2 und 6.3.

Tabelle 32: Struktur der Projektdaten-CD

Der Inhalt der CD befindet sich auch unter folgendem Link in komprimierter Form:

[http://bit.ly/MA\\_TK](http://bit.ly/MA_TK)

Thomas Kehl

Seite 96/96