HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Definition und Aufbau einer SCOR Simulationsbibliothek für Simio

Studienarbeit

Abteilung Informatik

Hochschule für Technik Rapperswil

Frühjahrssemester 2015

Autoren: Ernst Füllemann, Sandra-Olivia Müller

Betreuer: Prof. Dr. Andreas Rinkel

AUFGABENSTELLUNG SA/BA DEFINITION UND AUFBAU EINER SCOR SIMULATIONSBIBLIOTHEK FÜR SIMIO

HAUPTZIEL

Zur Vereinfachung der Simulation von Lieferketten (Supply Chains) ist eine Komponentenbibliothek für die Simulationssoftware Simio zu entwickeln. Die Simulations-Module der Library sollen aus dem SCOR- Modell (Supply Chain Operation Reference) abgeleitet werden. Die Arbeit soll folgende Punkte beinhalten:

- Analyse der Anforderungen an die Bibliothek
- Identifizierung von Basismodulen und deren Modellierung
- Beschreibung der wichtigsten internen Prozesse
- Umsetzung in Simio
- Nachweis der Funktionalität durch eine Prototype-Supply Chain in Simio
- Entwicklung und Umsetzung von Ideen zum Komponenten-Test in Simio
- Die Möglichkeit einige SCOR Metriken zu erfassen

NICHT TEIL DER ARBEIT

- Modellieren und Simulieren des Geldflusses

Unterschrift: 

SELBSTSTÄNDIGKEITSERKLÄRUNG FÜR STUDIENARBEIT

DEFINITION UND AUFBAU EINER SCOR SIMULATIONSBIBLIOTHEK FÜR SIMIO

Erklärung

Ich erkläre hiermit,

- dass ich die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt habe, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt ist oder mit dem Betreuer schriftlich vereinbart wurde,
- dass ich sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben habe.
- dass ich keine durch Copyright geschützten Materialien (z.B. Bilder) in dieser Arbeit in unerlaubter Weise genutzt habe.

Ort, Datum: Rapperswil, 29.05.2015

Name, Unterschrift: Ernst Füllemann

ABSTRACT

In dieser Arbeit wird eine Komponentenbibliothek zur Modellierung von Lieferketten (Supply Chain) in Anlehnung an das SCOR-Modell (Supply Chain Operation Reference) beschrieben. Ferner wird gezeigt, wie die konzeptuellen Überlegungen mit der Simulationssoftware Simio umgesetzt werden können.

Ausgehend vom Studium des abstrakten Referenzmodells SCOR werden die Basiskomponenten, Anforderungen und Prozesse definiert. In einem folgenden Schritt wird aufgezeigt, wie die ersten zwei Komponenten in Simio implementiert und verifiziert werden. Die Simio Komponentenbibliothek, im weiteren SCLib genannt, besteht aus den vier Basiskomponenten:

Rack	Lagert ein spezifisches Produkt. Kann mehrfach instanziiert werden um ein Lager nachzubilden.
Factory	Stellt Produkte aufgrund von eingehenden Bestellungen her.
Purchase	Entscheidet ob Produkte intern oder extern beschafft werden und bestellt diese bei der entsprechenden Instanz.
Selling	Kommuniziert mit Racks und entscheidet aufgrund der Priorität eingehender Orders wie weiter verfahren wird.

sowie drei Entities zur Beschreibung der Systemdynamik:

Bestellung (Order)	Speichert Produkttyp, Menge, Lieferzeit und Priorität.
Produkt (Product)	Speichert Herstellungszeitpunkt.
Request	Speichert Produkttyp und Lagermenge.

Abschliessend wird die Leistungsfähigkeit der SCLib anhand eines einfachen Beispiels aufgezeigt.

MANAGEMENT SUMMARY

AUSGANGSLAGE

Supply Chains sind sehr komplexe Systeme. Da die Analyse mit statischen Tests nicht gewinnbringend ist, werden sie meist von Hand optimiert. Ein effizienteres Werkzeug zur Optimierung ist Simulation. In der Simulation kann eine Supply Chain nachgebaut und unter rekonstruierbaren Bedingungen geprüft werden. Um den Simulationsprozess zu vereinfachen und schneller zu aussagekräftigen Ergebnissen zu kommen, ist eine Simulationsbibliothek zu entwickeln. Diese soll den schnellen Aufbau einfacher Supply Chains, sowie das entwickeln komplexer Systeme unterstützen.

TECHNOLOGIEN

VERWENDETE PROGRAMME

BIZAGI

Für die Modellierung der Aktivitätsdiagramme und Konzepte in BPMN wird Bizagi Modeler verwendet. Bizagi stellt alle für diese Arbeit benötigten Funktionen zur Verfügung. Zudem ist es Benutzerfreundlich sowie schnell und intuitiv erlernbar.

SIMIO

Simio ist eine Simulationssoftware mit grossem Funktionsumfang. In Simio können im Verhältnis zu anderen Programmen auf einfache Weise eigene Komponenten und Bibliotheken implementiert werden. Es stehen vielseitige Möglichkeiten zur Definition von Prozessen zur Verfügung. Simio bietet einfache Debugging Funktionen, wie Haltepunkte (Breakpoints) und Aufzeichnung (Tracing).

ERGEBNISSE

KONZEPT

Es wurde ein Konzept mit vier statischen und drei dynamischen Komponenten ausgearbeitet. Für diese wurden Anforderungen spezifiziert. Des Weiteren wurden die grundlegenden Prozesse, Entscheidungsabläufe sowie die Kommunikation zwischen den Komponenten mit Aktivitätsdiagrammen beschrieben.

SIMIO

Im Rahmen dieser Arbeit wurden die statischen Komponenten Rack und Factory sowie die dynamischen Komponenten Product, Order und Request implementiert. Anhand dieser Komponenten ist man als Benutzer in der Lage einen simplen SCOR Make-To-Stock Prozess zu erstellen und auszuwerten. Für die weiteren Komponenten Purchase und Selling wurden bereits Konzepte ausgearbeitet. Die implementierten Komponenten sind bereits darauf ausgelegt mit diesen konzeptionellen Komponenten zu interagieren.

AUSBLICK

Die Ergebnisse können im Rahmen einer Projekt- oder Studienarbeit z.B. um folgende Funktionalitäten erweitert werden:

- Geldfluss einer Supply Chain
- Erstellen und Anfordern von Offerten

DANKSAGUNG ERNST FÜLLEMANN

An dieser Stelle möchte ich mich bei allen Personen bedanken, die mir die Durchführung dieser Arbeit ermöglicht oder erleichtert haben. Besonderer Dank gilt unserem Betreuer Herr Prof. Dr. Rinkel für die Unterstützung die er erbracht und die Freizeit die er geopfert hat um uns bei der Problemlösung unter die Arme zu greifen.

Ein weiterer Dank gilt meiner Arbeitspartnerin Frau Olivia Müller für das angenehme Arbeitsklima das während der Arbeit geherrscht hat und die gute Kommunikation die zwischen uns stattgefunden hat.

Letztendlich möchte ich mich auch bei meiner Familie und meinen Freunden für ihre Unterstützung und Geduld bedanken. Das Gegenlesen konnte uns wichtige Einblicke auf die Arbeit geben, welche uns aus Entwicklersicht verborgen blieben. Besonders bedanken möchte ich mich an diesem Punkt speziell bei meiner Mutter, die es mir überhaupt erst ermöglicht hat, an der HSR zu Studieren und somit diese Arbeit zu vollenden.

TYPOGRAPHISCHE KONVENTION

In dieser Arbeit werden folgende typographischen Konventionen angewendet:

- Vorgegebene sowie selbst erstellte Objekte und Parameter in Simio: `Products`
- Aus der Benutzung des Simio Vokabulars hervorgehende Anglizismen: *batchen*
- Literaturverweise sind in eckigen Klammern angegeben: [42]

INHALTSVERZEICHNIS

EINLEITUNG

Dieser Arbeit zugrundeliegende Probleme sind die immer komplexer werdenden Handels- und Lieferungssysteme. Man kann nicht mehr davon ausgehen, dass Waren dort verwendet werden, wo sie hergestellt wurden. Vielfach können Kosten, Zeit und Ressourcen eingespart werden, wenn Arbeitsschritte wie z.B. die Produktion, Lagerung oder Versand ausgelagert werden. So entsteht ein komplexes, weltweites Netz aus Supply Chains.

Eine Fabrik, die keinen Nachschub an Rohstoffen erhält und deshalb still steht, kostet eine Firma ein Vermögen. Doch auch die Lagerkosten können untragbar werden, wenn ein übermässig grosser Vorrat an Rohstoffen immer zur Verfügung steht. Ebenso verhält es sich mit der Produktion bzw. Lagerung von fertigen Produkten. Ist die Nachfrage zu gross und kann nicht erfüllt werden springen Kunden ab, was zu Verlusten führt. Genau so unangenehm ist ein Lager voll von Produkten, die keiner kauft. Dabei ist zu beachten, dass in einer Lieferkette viele Firmen zwar voneinander abhängig sind, jedoch völlig eigenständig agieren können. Man spricht hier von der Gefahr des Bullwhip-Effekts[4]. Eine kleine Bewegung am einen Ende bewirkt grosse Bewegungen am anderen Ende.

Ein Werkzeugt um diese Komplexität überschaubar zu machen ist die Simulation. Mit Simulation ist hier das abbilden der realen Welt auf ein virtuelles Modell gemeint. Ein Modell ist immer eine Abstraktion. Je genauer abgebildet wird, umso grösser ist der Aufwand. Deshalb wird beim Modellieren darauf geachtet, so genau wie nötig und nicht so genau wie möglich zu abstrahieren. In einem solchen Modell können dann mögliche Szenarien simuliert werden. Die so geschaffene Testumgebung hat gegenüber der realen Welt den Vorteil, dass sie reproduzierbar ist. So kann bei gleichbleibender Ausgangssituation unterschiedliche Szenarien durchgespielt werden. Dies ermöglicht das Analysieren von Fehlern und das frühzeitige Erkennen von Risiken.

Die Wissenschaft vom Optimieren der Supply Chains nennt sich *Supply Chain Management* (SCM). Im Gegensatz zum Unternehmensmanagement befasst sich das SCM in den meisten Fällen nicht mit einer Firma sondern mit der ganzen Lieferungskette. Es gibt viele Ansätze für SCM. Ein Versuch, diese zu vereinheitlichen, ist das SCOR-Modell (Supply Chain Operation Reference). Das SCOR Modell beschreibt und standardisiert Prozesse und Messwerte in einer Supply Chain.

Das SCOR-Modell wird bisher selten im Zusammenhang mit Simulation verwendet. Zwar existieren viele theoretische Abhandlungen darüber wie SCOR in die Simulation eingebaut werden könnte, an konkreten Ansätzen mangelt es aber. Ein Programm für das Modellieren und Simulieren von Prozessen ist Simio. Simio steht für: ***SI****mulation* ***M****odeling framework based on* ***I****ntelligent* ***O****bjects* [5]. Es ist eine Plattform um aus statischen und beweglichen Elementen ein Modell zu erstellen. Für das Simulieren von Supply Chain eignen sich die Grundfunktionen von Simio sehr gut. Das Programm ist jedoch so allgemein gehalten, dass auch völlig andere Prozesse modelliert werden können. Um eine Supply Chain in Simio aufzubauen muss jedes Element von Grund auf zusammengebaut werden.

In dieser Arbeit wird als Erstes genauer auf die Themen Supply Chain und SCOR eingegangen. Es werden Grundkomponenten einer Supply Chain beschrieben und wie eine generische Funktionsbibliothek für den Aufbau von Supply Chain aussehen könnte. Diese Grundkomponenten wurden so aufgebaut, dass mit ihnen einige SCOR Prozesse modelliert und SCOR Metriken erhoben werden können. Es folgt eine Einführung in Simio. Ein Prototype der Funktionsbibliothek ist in Simio implementiert und für den Anwender beschrieben.

GRUNDLAGEN ZU SUPPLY CHAIN, SCOR UND SIMIO

SUPPLY CHAIN

Eine Supply Chain, *Supply Chain*, ist der Weg, den ein Produkt oder ein Service vom Zulieferer bis zum Kunden nimmt. Dazu gehören alle Personen, Bearbeitungsprozesse, Firmen, Ressourcen, Informationen, Rohstoffe oder vorgefertigte Komponenten, die an der Lieferung beteiligt sind.

Es wird zwischen externen und internen Supply Chains unterschieden. Eine externe Supply Chain beschreibt den Weg zwischen Lieferanten und Kunden wobei der Kunde auch wieder Lieferant für den nächsten Kunden sein kann. Als interne Supply Chain wird der Weg den das Produkt innerhalb einer Institution zurücklegt bezeichnet.

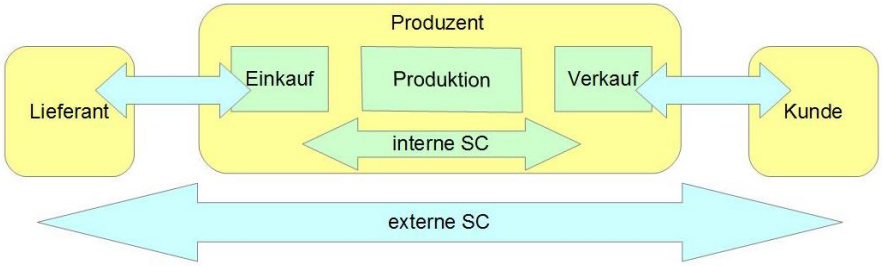


Abbildung 1 Interne und externe Supply Chain

In einer Supply Chain werden die drei Bereiche Material- oder Dienstleistungsfluss, Informationsfluss und Geldfluss berücksichtigt. In einer externen Supply Chain fliesst Material vom Hersteller zum Kunden und Geld vom Kunden zum Hersteller. Die Informationen fliessen erst vom Kunden zum Hersteller (z.B. in Form einer Bestellung) und später vom Hersteller zum Kunden (z.B. in Form eines Lieferscheins).

SUPPLY CHAIN-MANAGEMENT

Das Supply Chain-Management beschäftigt sich mit der Frage, wie man eine Supply Chain (intern oder extern) optimieren kann. Probleme mit denen sich das Supply Chain-Management beschäftig sind zum Beispiel:

- Das Verhältnis zwischen Kooperation und Wettbewerb zwischen den beteiligten Instanzen einer Supply Chain
- Die Verteilung der Wertschöpfungsanteile einer Supply Chain
- Transparenter, umfassender Informationsaustausch zwischen den beteiligten Instanzen
- Konfiguration und Planung der Prozessstrukturen
- Alternative Koordinationsformen
- Performance Management und Performance Measurement Systeme
- Planung und Entwicklung der Prozessstrukturen

SCOR

SUPPLY CHAIN COUNCIL

Das SCOR-Modell wurde vom Supply Chain Council (SCC) beschrieben. Das SCC ist eine Non-Profit-Organisation mit dem Ziel ihren Mitglieder-Organisationen zu helfe die Performance ihrer Supply Chains zu verbessern.

SCOR AUFBAU

„The Supply Chain Operations Reference (SCOR®) model provides a unique framework that links performance metrics, processes best practices, and people into a unified structure"

(Supply Chain Council, Inc. 2010)

Auf Deutsch: Das SCOR-Modell bietet eine einzigartige Umgebung welche Performanz-Metriken, Bewährte Vorgehensweisen, Prozesse und die Menschen die damit Arbeiten in eine einheitliche Struktur überträgt.

Diesem Satz soll nun eine aussagekräftigere Definition hinzugefügt werden. Im SCOR-Modell geht es darum eine Struktur für Supply Chains zu beschreiben und Kenngrössen zu Standardisieren. Ziele sind verbesserte Leistung und Vergleichbarkeit. Die beiden Hauptstrukturen, die im SCOR-Modell immer wieder vorkommen sind die Metriken und die Prozesse.

DAS SCOR PROZESS MODELL

Im SCOR Prozess Model werden fünf Hauptprozesse beschrieben. Diese sind:

- sP Plan
- sS Source
- sM Make
- sD Deliver
- sR Return

In Abbildung 2 kann man erkennen, dass jedes Mitglied einer Supply Chain diese fünf Prozesse berücksichtigen muss.

Abbildung 2 SCOR Level 1 Prozesse [2]

PLAN

Im *Plan* Prozess geht es darum die anderen Prozesse zu planen. Dazu gehört jeden Prozess einzeln zu planen, sowie das Zusammenspiel der Prozesse innerhalb der Institution. Ausserdem beschäftigt sich der *Plan* Prozess mit der Analyse von Fehlern und die Optimierung aufgrund früher gesammelter Erfahrungen.

SOURCE

Der *Source* Prozess beschreibt das Koordinieren der Ressourcen. Dies beinhaltet den Bestellvorgang, das Empfangen vom Gütern und Dienstleitungen, das Einlagern, Validierung von empfangenen Lieferungen und Priorisierung von Lieferungen.

MAKE

Im *Make* Prozess finden alle Ressource verändernden Aktivitäten sowie die Dienstleitungen statt. Er beinhaltet zum Beispiel Prozesse wie das Zusammenstellen, das Wiederverwerten oder das chemische Verändern von Produkten.

DELIVER

Der *Deliver* Prozess beschreibt alle zur Lieferung von Gütern notwendigen Schritte. Dies beinhaltet das Zusammenstellen von Produkten zu Lieferungen, das Erstellen von Ladelisten, das Verladen, das Transportieren sowie das Stellen von Rechnungen.

RETURN

Der *Return* Prozess befasst sich mit dem Zurückgeben von Gütern. Dies kann bei defekten oder mangelhaften Gütern Reparatur oder Recycling zur Folge haben (Die Güter fliessen dann in den *Make* Prozess zurück). Aber auch das Management eines angebotenen Probeverkaufs fällt unter den *Return* Prozess („Geld-zurück-Garantie“ oder Kleiderversandt, bei dem die Kleider erst gekauft und dann anprobiert werden).

PROZESS LEVEL

Die SCOR Prozesse sind in 3 Level aufgeteilt. Jeder Level beschreibt einen Prozess detaillierter. Auf Level 1 stehen die fünf Hauptprozesse *Plan, Make, Source, Deliver* und *Return*. In Abbildung 3 ist die Aufteilung der Levels am Beispiel des *Make* Prozesses erklärt. Im Anhang befinden sich Tabellen mit allen SCOR Prozessen.

Level 1	sM MAKE		
Level 2	sM1 Make-to-Stock	sM2 Make-to-Order	sM3 Engineer-to-Order
Level 3	sM1.1: Schedule Production Activities	sM2.1: Schedule Production Activities	sM3.1: Finalize Production Engineering
	sM1.2: Issue Product	sM2.2: Issue Product	sM3.2: Schedule Production Activities
	sM1.3: Produce and Test	sM2.3: Produce and Test	sM3.3: Issue Product
	sM1.4: Package	sM2.4: Package	sM3.4: Produce and Test
	sM1.5: Stage Product	sM2.5: Stage Finished Product	sM3.5: Package
	sM1.6: Release Product to Deliver	sM2.6: Release Finished Product to Deliver	sM3.6: Stage Finished Product
	sM1.7: Waste Disposal	sM2.7: Waste Disposal	sM3.7: Release Product to Deliver
			sM3.8: Waste Disposal

Abbildung 3 SCOR Make Level 1, 2 und 3 Prozesse [2]

METRIKEN

DIE SCOR METRIKEN

Eine der Hauptmotivationen um SCOR zu verwenden sind die Metriken. Das Wort Metrik wird in diesem Zusammenhang für Messgrösse oder Messwert verwendet. Die SCOR Metriken sind Zahlenwerte, mit denen sich verschiedene Supply Chains miteinander vergleichen lassen oder die Entwicklung einer Supply Chain beobachtet werden kann. Es existieren über 250 solcher Metriken. Diese sind in fünf Funktionseigenschaften (Performance Attributes) kategorisiert. In der nachfolgenden Tabelle sind die offiziellen Definitionen des Supply Chain Council aufgelistet.

Performance Attributes	**Definition**
Reliability	Die Fähigkeit zur erwarteten Ausführung von Aufgaben. *Reliability* konzentriert sich auf die Vorhersagbarkeit der Ergebnisse eines Prozesses. Typische Messgrößen für das *Reliability* Attribut sind: zur richtigen Zeit, die richtige Menge, die richtige Qualität.
Responsiveness	Die Geschwindigkeit mit der die Aufgaben durchgeführt werden. Die Geschwindigkeit, mit der eine Supply Chain Produkte an den Kunden liefert. Beispiele hierfür sind cycle-time Metriken.
Agility	Die Fähigkeit auf äußere Einflüsse zu reagieren, auf Marktveränderungen zu reagieren und Wettbewerbsvorteil zu gewinnen oder zu halten. Die SCOR *Agility*-Metriken beinhalten Flexibilität und Anpassungsfähigkeit.
Costs	Die Betriebskosten der Supply Chain-Prozesse. Dies beinhaltet die Arbeitskosten, Materialkosten, Management und Transportkosten. Typische Kostenmetrik sind die Kosten der verkauften Produkte
Assets Management Efficiency (Assets)	Die Fähigkeit Vermögenswerte effizienter zu nutzen. Zu Asset-Management-Strategien in einer Lieferkette gehören Bestandsreduzierung und Insourcing vs. Outsourcing. Metriken sind: Bestandsreichweite und die Kapazitätsauslastung.

Tabelle 1 Definitionen der Funktionseigenschaften [6]

Wie die Prozesse sind diese in drei Levels strukturiert:

- Level 1 – Organisation
- Level 2 – Prozess
- Level 3 – Diagnostik

Eine detaillierte Auflistung befindet sich im Anhang.

SCOR METRIKEN IN DER SIMULATION

Für die Analyse durch Simulation besonders geeignet sind alle zeit- oder mengenbezogenen Metriken. Für das Konzept der Simio Bibliothek wurden fünf spezifische Messwerte aus den Bereichen *Reliability, Responsiveness* und *Asset Management Efficiency* ausgewählt.

Tabelle 2 SCOR Metriken [6]

Performance Attributes	**Metrik**	**Einheit**
Reliability	*Perfect Order Fullfillment*	*Anzahl der beim ersten Versuch erfolgreichen Auslieferungen*
	Delivery Performance	$\frac{Perfect\ Order\ Fullfillment}{Totale\ Anzahl\ auslieferungen}$
Responsiveness	*Order fullfillment leadtimes*	*Durchnittliche Zeit zwischen Bestellung und Auslieferung*
Asset Management Efficency	*Inventory Days of Supply*	$\frac{Total\ gelagerte\ Waren}{Zeit}$
	Fixed Assets	Gelagerte Produkte

Da die Betachtung des Geldflusses nicht Teil dieser Arbeit ist und so noch nicht in der Bibliothek implementiert wurde, werden auch Metriken welche mit dem Geldfluss zu tun haben nicht untersucht. Metriken der Kategorie *Agility* sind ebenso nicht berücksichtigt. In einer Simulation sollen vergleichbare Werte produziert werden. Dynamische Änderungen sollen anhand verschiedener Simulationen analysiert werden können. Sie sind sinnvollerweise nicht Teil der Simulation selbst.

SIMIO

Simio ist eine Software zur Erstellung von Prozess- und eventbasierten Simulationen. Es unterstützt die objektbasierte Modellierung von 2D- und 3D-Komponenten sowie aufwändige Animationen und die Möglichkeit diese aufzunehmen. Simio verfügt über eine sehr umfangreiche graphische Oberfläche. Der grosse Funktionsumfang von Simio benötigt wie das Lernen einer neuen Programmiersprache Einarbeitungszeit. Die Software ist ein sehr grosses und mächtiges Tool. Sind die Kenntnisse zur Anwendung vorhanden, können selbst komplexeste Prozesse mit überschaubarem Aufwand realisiert werden.

Eine Unterstützung für Anwender bietet Simio über die `SimBits` an. Dies ist eine umfangreiche Bibliothek aus Vorlagen für konkrete Problemlösungen und kleinen Beispielanwendungen. Für Entwickler bietet sich der Simio Reference Guide an. Darin sind die Dokumentationen zu allen Objekten und Parametern enthalten. Für Probleme die sich nicht mit den implementierten Funktionen zu lösen lassen scheinen bietet Simio auch eine Programmierschnittstelle für C# mit API an.

STARTEN MIT SIMIO

Dieses Kapitel soll eine kurze Einführung zu Simio, seinen Funktionen und Anwendungsmöglichkeiten geben.

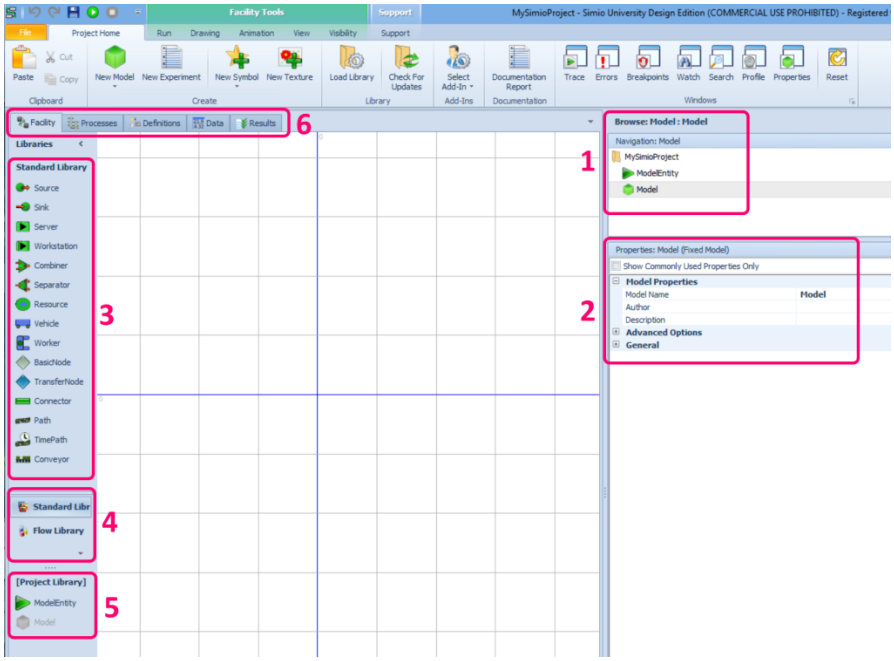


Abbildung 4 Simio nach dem starten

1. Im Navigationsfeld sind das Projekt und alle darin enthaltenen Modelle sichtbar. Standartmässig erstellt Simio nach dem Starten eines neuen Projektes (MySimioProject). Darin sind ein `ModelEntity` und ein Model vorhanden. Angewählt ist das Model, was bedeutet, dass alles was man im Fenster sieht sich auf das Model bezieht.
2. In diesem Feld sind die verstellbaren Parameter zum aktuell angewählten Objekt zu sehen. Dies ist zurzeit das `Model`. Es können aber auch Objekte angewählt werden, die sich im `Model` befinden. Die Parameter `(Properties)` sind in Kategorien aufgeteilt (z.B. `Model Properties`, `Advanced Options`, `General`)
3. Nach der Installation stellt Simio eine Standartbibliothek zur Verfügung. Sie ist die Grundlage für die meisten Modelle sowie auch für die SCLib. Zusätzlich zur Standard Library gibt es auch noch eine Flow Library zur Simulation von Flüssigkeit transportierenden Systemen.
4. In diesem Feld sind alle importierten Libraries sichtbar. Standartmässig sind dies die Standart- und die Flowlibrary. Die Flowlibrary dient zur Simulation von flüssigkeitsbasierten Systemen und wird hier nicht benötig. Die SCOR-Library soll dort sichtbar sein, wenn sie importiert wird.
5. In der Project Library stehen die `Models` zur Verfügung, welche sich auch im aktuellen Projekt befinden. Hier ist dies nur das `ModelEntity`.
6. Das weisse, karierte Feld mit den Libraries auf der linken Seite ist die Facility. Es ist das Hauptbearbeitungsfeld für das `Model`. Auf die weiteren Reiter wird später noch eingegangen.

SIMIO OBJEKTE

Simio ist auf intelligenten Objekten aufgebaut. Es werden fünf verschiedene Objektklassen unterschieden. `Model` und `ModelEntity` sind implementierte Bespiele für solche Objekte. `Model` ist aus der `Fixed Class` abgeleitet und `ModelEntity` aus der `Entity Class`. Die Tabelle 3 zeigt eine Übersicht über die fünf Objektklassen.

Tabelle 3 Simio Objekte

Fixed Class	Die `Fixed Class` entspricht einem System. Das reicht von sehr komplexen

	Systemen bis hin zu den simpelsten fixen Bestandteilen eines Systems. Der `Processor` ist eine bereits vorgefertigte `Fixed Class`, bei der schon ein Teil der Logik vordefiniert ist.
Entity Class	`Entities` stellen den beweglichen Teil eines Systems dar. Zum Beispiel die Kunden, die in einem Geschäft ein und ausgehen, die Flaschen, die in der Maschine abgefüllt werden oder die Autos auf einer Strasse.
Transporter Class	`Transporter` sind `Entities`, die andere `Entities` aufnehmen und transportieren können
Node Class	Die `Node` Objekte definieren die Start- und/oder Endposition für ein oder mehrere `Link` Objekte. Sie dienen ausserdem zum Transferieren in oder aus anderen Objekten.
Link Class	Die `Link` Objekte sind die Pfade in einem System. Sie verbinden die anderen Komponenten. `Entities` und `Transporter` bewegen sich auf den `Link` Objekten.

Um ein neues `Model` zu erstellen wählt man unter *Project Home – New Model* eine Klasse aus. Diese erscheint dann in der Navigation und kann nun bearbeitet werden.

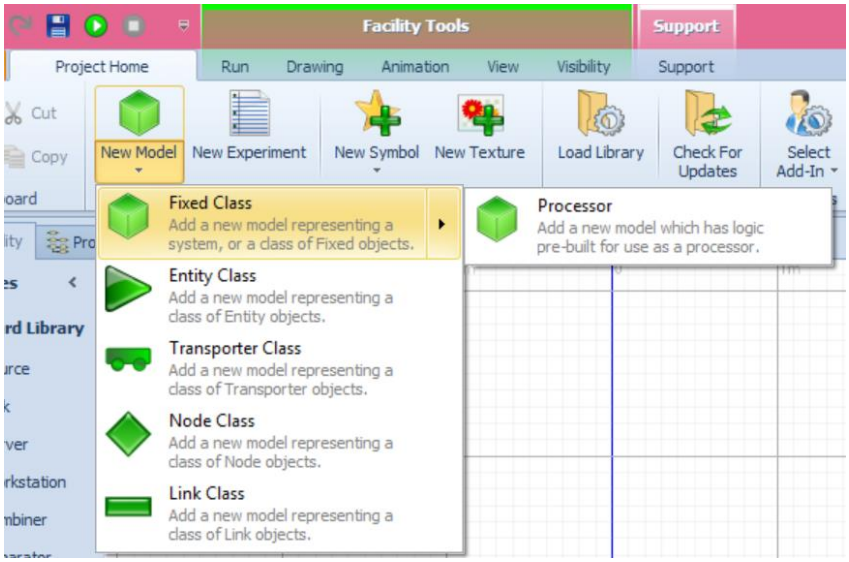


Abbildung 5 Komponentenklassen

SIMPLES MODEL

Das folgende Tutorial zeigt einige der Grundfunktionen und Begriffe von Simio auf.

LERNZIELE

- Ein einfaches `Model` erstellen
- Farben ändern
- `Events` erstellen

- Prozesse erstellen
- `Add-On Process Triggers`
- `Processing Time` beim `Server` ändern
- `States` definieren
- `StatusLabel`

TUTORIAL

1. Per drag&drop zweimal eine `Source`, eine `Sink` und einen `Sever` aus der Standartlibrary in das Feld ziehen.
2. Diese jeweils mit einem `Path` verbinden.
3. Zwei `ModelEntities` aus der ProjectLibrary in das Feld ziehen. Diese umbenennen in `MyEntity1` und `MyEntity2`.
4. Mit der Funktion Color eine Farbe auswählen und eines der `Entities` anklicken um es zu färben.

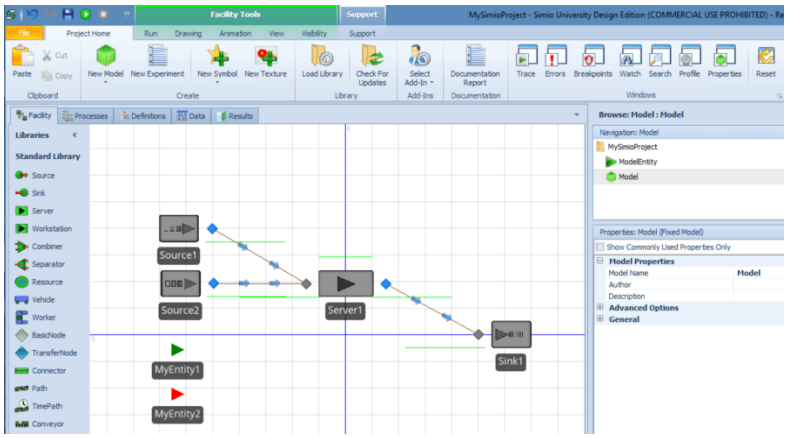


Abbildung 6 Simio Tutorial Schritt 4

5. `Source1` auswählen und bei den `Properties` unter *Entity Arrival Logic – Entity Type* im Dropdown-Menu `MyEntity1` auswählen.
6. Dasselbe mit `Source2` und `MyEntity2`.

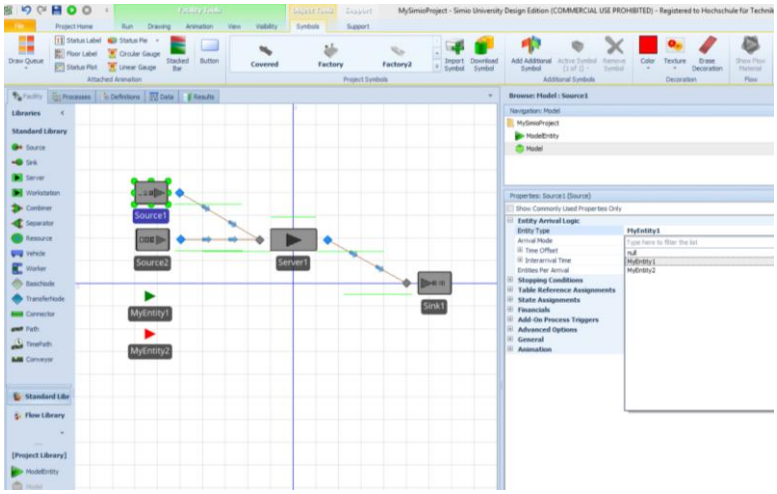


Abbildung 7 Simio Tutorial Schritt 6

7. Nun kann über die Reiter auf *Definitions – Events* gewechselt werden. Mit Klicken auf `Event` wird ein neuer `Event` erzeugt. Diesen kann man unter *General – Name* oder durch Rechtsklicken zu *CreateNew* umbenennen.

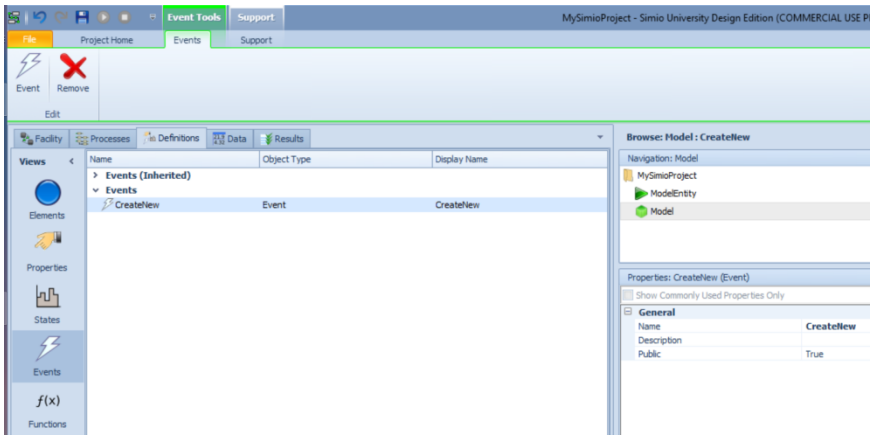


Abbildung 8 Simio Tutorial Schritt 7

8. Nun wechselt man wieder in der *Facility*-Ansicht und wählt den `OutputNode` von `Source1` an. *Unter Add-On Process Triggers* (bei den Properties) wird durch doppelklicken auf *Entered* ein neuer Prozess erzeugt und die Ansicht wechselt automatisch auf den *Process*-Reiter.
9. Dem neu erstellten Prozess *Output_Source1_Entered* fügt man nun den `Step Fire` hinzu. In der *Basic Logic – Event Name* wählt man den zuvor erstellten `Event`. (Der `Step Fire` muss angewählt sein)

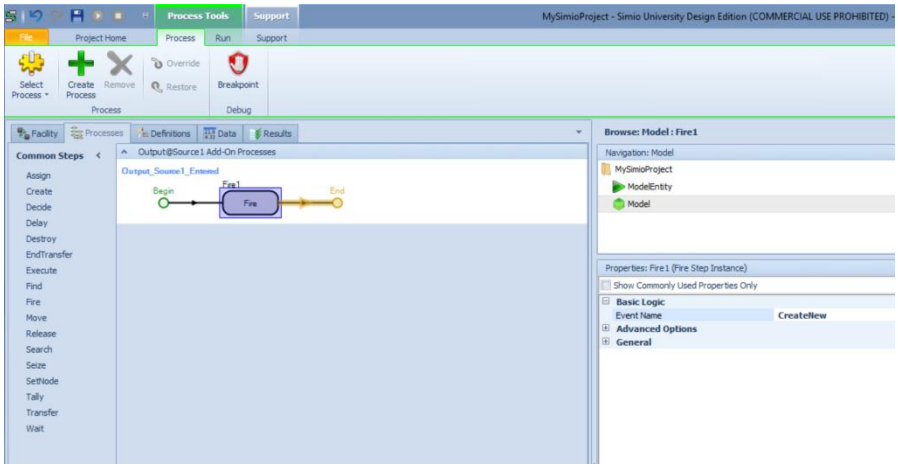


Abbildung 9 Simio Tutorial Schritt 9

10. Zurück in der *Facility* wählt man `Source2` an. Unter *Entity Arrival Logic* den *Arrival Mode* auf *On Event* und den *Trigger Event Name* auf den zuvor erstellten `Event` stellen.

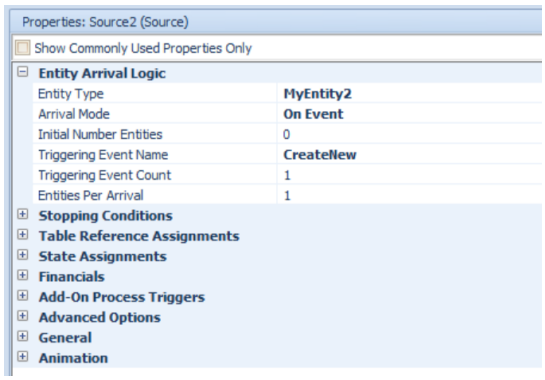


Abbildung 10 Simio Tutorial Schritt 10

11. Zum Schluss wird noch der `Server1` angewählt. Hier unter *Process Logic – Processing Time* den Wert Random.Exponential(0.1) eingeben.
12. Unter *Definitions – States* lassen sich lokale Variablen definieren, die zur Laufzeit geändert werden können. Um dies zu demonstrieren soll ein neuer `Integer State` erstellt werden (klicken auf *Definitions – States – Integer*)
13. Um diesem `State` nun einen Wert zu geben geht man zurück in die *Facility* und erstellt einen neuen *Add-On Process Trigger* Prozess für *Input@Sink1 Entered.*
14. Dem Prozess fügt man den `Step Assign` hinzu. *State Variable Name* ist der zuvor erstellte *IntegerState1.* Als *NewValue* wird *IntegerState1 + 1* eingetragen.

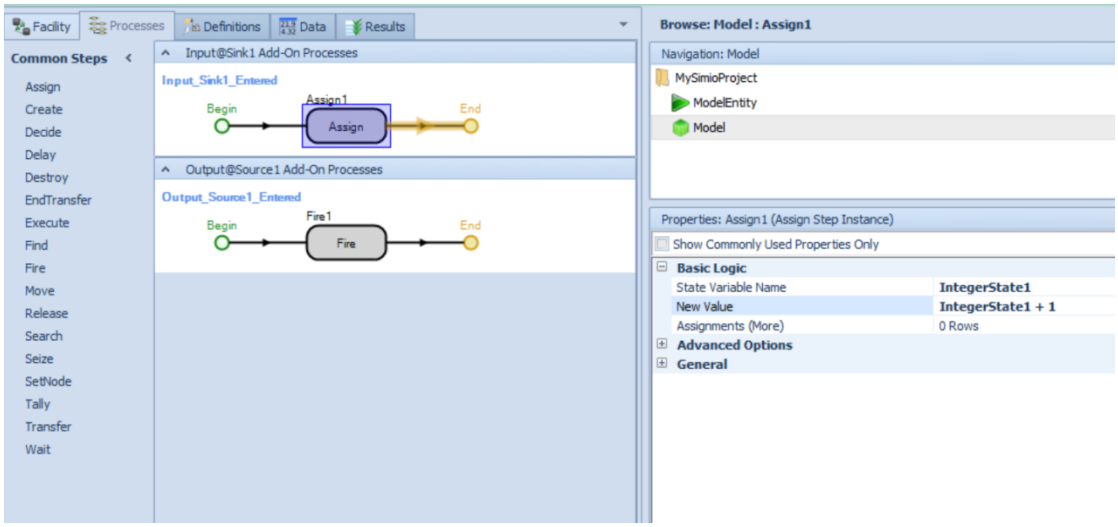


Abbildung 11 Simio Tutorial Schritt 14

15. Zurück in der *Facility* wählt man nun den übergeordneten Reiter *Animation* und erstellt ein *Status Label* mit der *Expression: IntegerState1.*

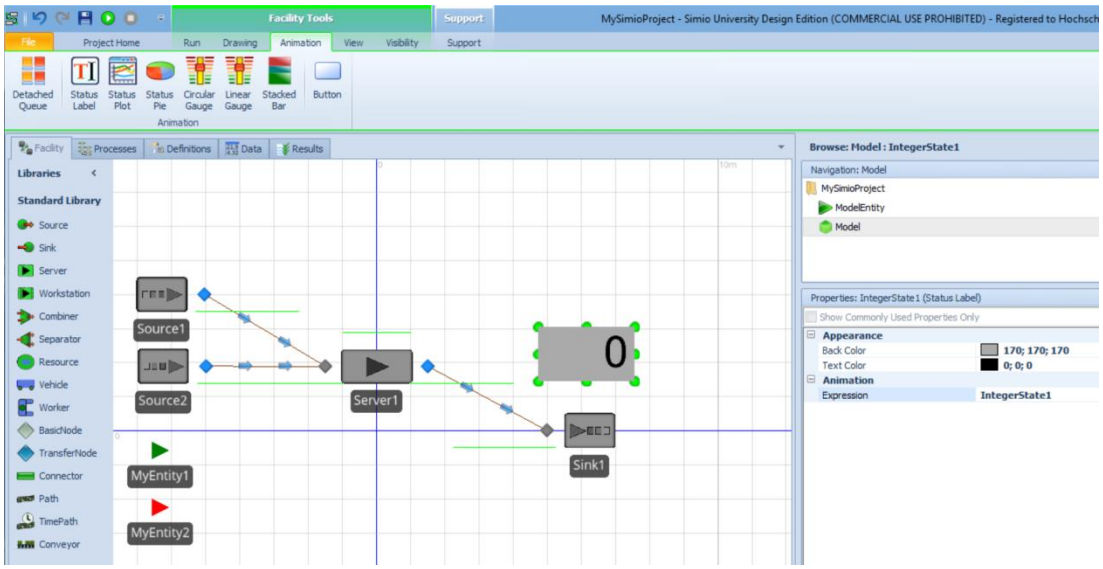


Abbildung 12 Simio Tutorial Schritt 15

16. Wenn man nun auf den grünen *Play* Button klickt um das `Model` zu starten kann man sehen, dass für jedes grüne `Entity` ein Rotes erstellt wird. Diese werden beim `Server` mit einer exponentiell verteilten Zeit von 0.1 Minute im `Server` bearbeitet und gehen danach weiter zur `Sink`. Auf dem `Status Label` wird angezeigt, wie viele `Entities` bereits bearbeitet wurden.

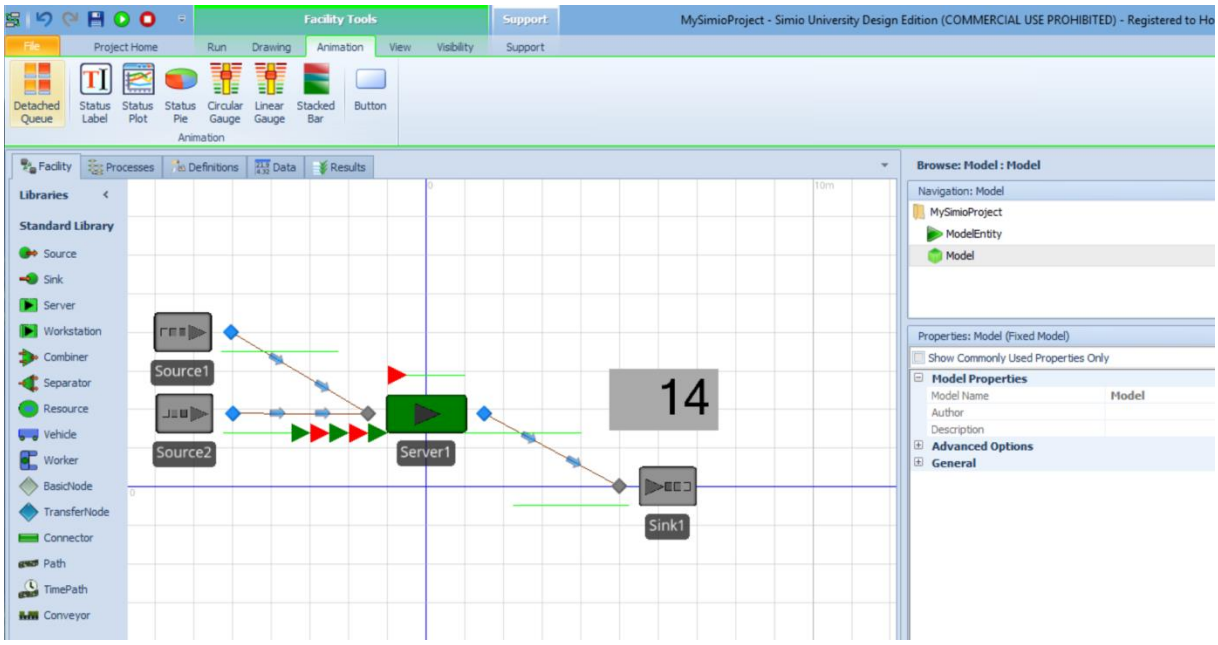


Abbildung 13 Simio Tutorial Schritt 16

EXISTIERENDE LÖSUNGEN ZUR SCOR SIMULATION

Zum Thema Simulation von SCOR getreuen Supply Chains gibt es eine Handvoll Research Papers, welche sich mit der Durchführbarkeit von SCOR Simulationssoftware auseinandersetzen. Allesamt kommen sie auf den Schluss, dass SCOR mit verschiedensten Ansätzen simulierbar ist, jedoch je nach Tiefe der Implementierung sehr arbeitsaufwändig werden kann. Explizite Anwendungssoftware gibt es bis zum heutigen Zeitpunkt nur eine, nämlich das IBM Smartscor.

IBM SMARTSCOR

IBM SmartSCOR umfasst die Methodologie und ein Framework zur Problemlösung von On-Demand Supply Chains und integriert das SCOR Modell sowie weitere Simulations- und Optimisierungstechniken.

IBM SmartSCOR führt die SCM-Transformation auf den zwei verschiedenen Levels supply chain strategy desgin/redesign sowie supply chain process improvement durch.

Supply chain strategy design/redesign transformiert eine Supply Chain fundamental durch Rekonfiguration der Produktion und des Vertriebsnetzes.

Supply chain process improvement hilft die darunterliegenden Business Prozesse zu optimieren.

SmartSCOR besteht aus den Modulen Supply Chain Network Optimization, Supply Chain Simulation und Supply Chain Process Analysis. Die einzelnen Module können einzeln oder im Zusammenspiel benutzt werden.

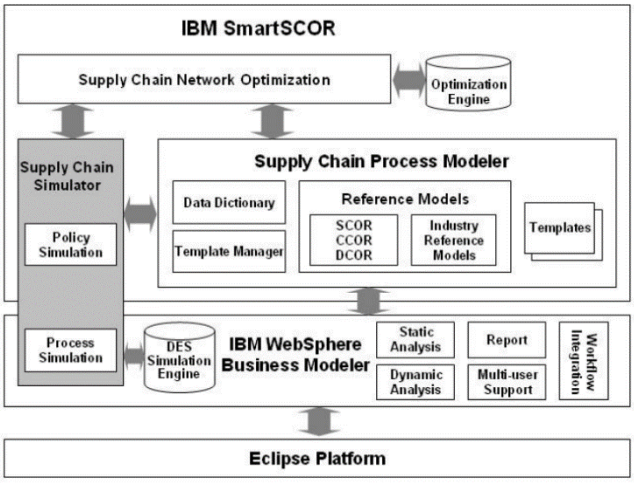


Abbildung 14 IBM SmartSCOR

Als Endbenutzer hat man über die IBM Websphere Zugriff auf den Business Modeler. Diesen kann man via Eclipse Plattform benutzen.

[3]

KONZEPTUELLER AUFBAU EINER SCOR-LIBRARY (KOMPONENTEN)

KOMPONENTEN

STATISCHE KOMPONENTEN

Um eine Bibliothek zu entwerfen stellt sich als Erstes die Frage, aus welchen Komponenten diese besteht. Für die SCLib ist es naheliegend, die Komponenten einer realen Supply Chain zu abstrahieren. Die Komponenten einer externen Supply Chain sind die am Wertschöpfungsprozess beteiligten Unternehmen und Institutionen. Jedes Unternehmen hat entsprechend seiner Spezialisierung Aufgaben zu erfüllen. Aus den SCOR Prozessen wurden folgende Aufgaben für ein Unternehmen im Bereich Güterumschlag abstrahiert:

- Einkaufen
- Verkaufen
- Lagern
- Produzieren
- Transportieren
- Verpacken und Entpacken

Nicht jedes Unternehmen beschäftigt sich mit all diesen Aufgaben. Je nach Ausrichtung des Unternehmens fallen gewisse Aufgaben stärker oder schwächer ins Gewicht oder gar ganz weg.

Ähnlich verhält es sich bei einem einzeln betrachteten Unternehmen. Bei einer internen Supply Chain werden die Prozesse innerhalb eines Unternehmens betrachtet. Ein Unternehmen besteht in der Regel aus mehreren Abteilungen. Jede Abteilung widmet sich einem oder mehreren Aufgabenbereichen. So können die oben genannten Aufgaben innerhalb eines Unternehmens in mehreren Abteilungen anfallen. Güter müssen nicht nur zwischen den Unternehmen transportiert werden, sondern auch zwischen den Abteilungen in einem Unternehmen. Jede Abteilung für sich hat wie das Unternehmen einen Hauptaufgabenbereich. Doch auch hier fallen zusätzliche Aufgaben an. Zum Beispiel ein Rohstofflager in einer Produktion. Das Lagern von Material ist nicht Hauptaufgabe der Produktionsabteilung. Je nachdem wie detailliert die Abteilung betrachtet werden soll, muss dieser Aspekt berücksichtig werden.

Auf Grund dessen wurde entschieden, die Komponenten der SCLib als konkreten Aufgabenbereich zu definieren. Abbildung 15 zeigt ein Unternehmen mit den Abteilungen Einkauf, Verkauf, Lager und Produktion. In den Komponenten Einkauf, Verkauf und Produktion gibt es jeweils ein Zwischenlager. Das Lager hat eine eigene Einkaufs- und Verkaufsabteilung.

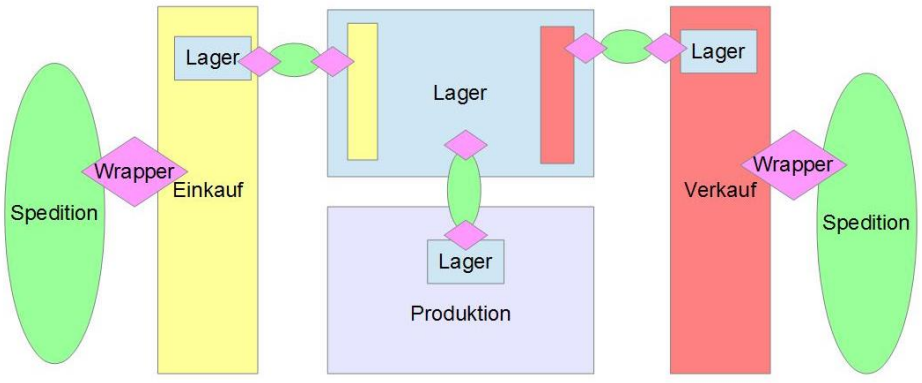


Abbildung 15 Erster Entwurf der einer Komponenten in möglichen Komposition

Die Komponente Spedition steht für den Transport innerhalt des Unternehmens und zu anderen Unternehmen. Vor und nach jedem Transport muss die transportierte Ware in entsprechenden Kombinationen verpackt werden. Dies kann eine Palette zwischen Hauptlager und/oder Produktion und ein Container zwischen Verkauf und nächstem Unternehmen sein. Die *Wrapper* Komponente steht für diesen Ver- und Entpackungsvorgang.

Die hier gezeigten Komponenten Wrapper und Spedition wurden in dieser Arbeit nicht weiter verfolgt. Simio implementiert in der Standardlibrary mit den *Transporter* und *Combiner* Klassen diese Funktionen bereits ausreichend.

DYNAMISCHE KOMPONENTEN

Zwischen den statischen Unternehmen und Abteilungen werden dynamische Einheiten ausgetauscht. Es wird hier in drei Bereiche unterteilt, welche im Supply Chain Management definiert sind.

INFORMATIONSFLUSS

Zum Informationsfluss gehören Anfragen, Offerten und Bestellungen. Alles was in Form eines Briefes übermittelt werden kann fällt in diesen Bereich.

MATERIALFLUSS

Zum Materialfluss gehören alle materiellen Güter, das heisst: Produkte sowie zu Lieferungen zusammengefasste Produkte.

GELDFLUSS

Zum Geldfluss gehören alle Gegenwerte für die gelieferten Güter, das heisst: Geld, Gutscheine usw.

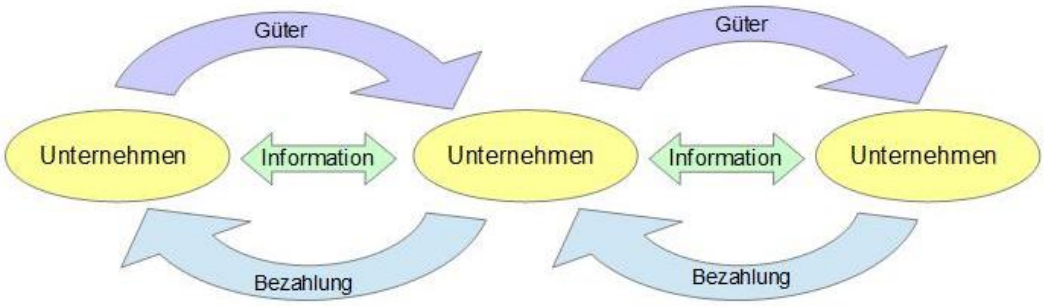


Abbildung 16 Informations-, Material und Geldfluss

ANFORDERUNGSSPEZIFIKATION

PRODUKTFUNKTION GESAMTE BIBLIOTHEK

Die Hauptfunktion der SCLib ist das Modellieren von Supply Chains. Mit modularen Bausteinen sollen Unternehmen und die Handelsaktivitäten zwischen ihnen abgebildet werden können. Die SCLib soll die Auswertung einer Supply Chain mit dem SCOR Modell unterstützen.

PRIMÄRE FEATURES

F01 Modulares zusammenstellen von Handels- und Produktionsunternehmen (interne SC)

F02 Verknüpfen von Unternehmen zu einer Supply Chain (Externe SC)

F03 Senden von Bestellungen

F04 Empfangen von Lieferungen

F05 Zusammenstellen von Lieferungen

F06 Lagerung

F07 Produktion von Produkten nach Bestellung

F08 Generieren von SCOR Metriken

F09 Modellieren von SCOR Prozessen

ERWEITERNDE FEATURES

F10 Kostenberechnung

F11 Reservierung von Arbeitskräften

F12 Anfordern und stellen von Offerten

BENUTZERCHARAKTERISTIK

Zum Zielpublikum gehören Anwender von Simio und Supply Chain Analysten.

EINSCHRÄNKUNGEN

Der konzeptuelle Beschrieb der SCLib ist allgemein gehalten. Die Implementation ist durch die Möglichkeiten von Simio eingeschränkt.

USE CASES

UC01 Modellieren eines Make-to-Stock Prozesses

UC02 Modelieren eines Make-to-Order Prozesses

UC03 Erheben der SCOR Metrik

ANFORDERUNGEN AN DIE KOMPONENTEN

EINKAUF – VERKAUF

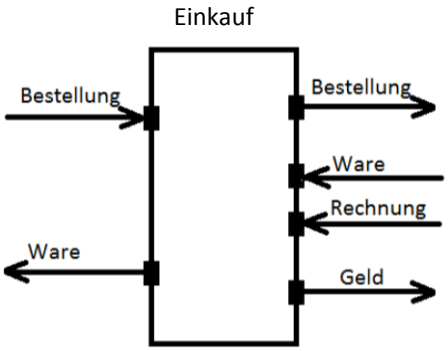


Abbildung 17 Einkauf Konzept

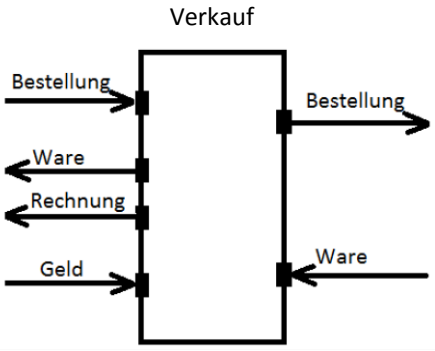


Abbildung 18 Verkauf Konzept

BESCHREIBUNG

Die Komponenten *Einkauf* und *Verkauf* bilden zusammen die Kommunikationsschnittstelle zwischen zwei Stationen einer Supply Chain. Sie können in beide Richtungen direkt aneinander gehängt werden oder einen Rahmen für die Komponenten *Lager* und *Produktion* bilden.

PROZESSE

Tabelle 4 Konzept für Prozesse in Einkauf/Verkauf

Einkauf	Verkauf
• Offerte einholen • Bestellen • Ware prüfen • Ware entgegen nehmen • Ware ablehnen / retournieren • Rechnung begleichen	• Offerte stellen • Bestellung bearbeiten • Lieferung bereitstellen • Rechnung stellen • Mahnung senden • Retournierte Ware entgegen nehmen

PARAMETER

Tabelle 5 Konzept für Parameter in Einkauf/Verkauf

Einkauf und Verkauf
• Ressourcen (Arbeitskräfte)

Einkauf und *Verkauf* sind Kommunikationskomponenten. Skalierbar sind sie über die Anzahl Ressourcen, die gleichzeitig an den Prozessen arbeiten.

STATISTIK

Tabelle 6 Konzept für Statistiken in Einkauf/Verkauf

Einkauf	Verkauf
• Offerten eingeholt/Zeit • Gesendete Bestellungen /Zeit • Eingehende Deliveries/Zeit • Auslastung • Offene Bestellungen • Abgeschlossene Bestellungen • Offene Rechnungen • Bezahlte Rechnungen	• Offerten gestellt / Zeit • Bestellungen eingegangen/ Zeit • Ausgehende Deliveries / Zeit • Auslastung • Offene Bestellungen • Abgeschlossene Bestellungen • Offene Rechnungen • Beglichene Rechnungen

TESTS

Input zum Testen ist immer eine Bestellung. Das zu überprüfende Resultat ist die gelieferte Ware. An alle zu testenden Ein- und Ausgänge kann direkt eine Quelle bzw. Senke angehängt werden.

PRODUKTION

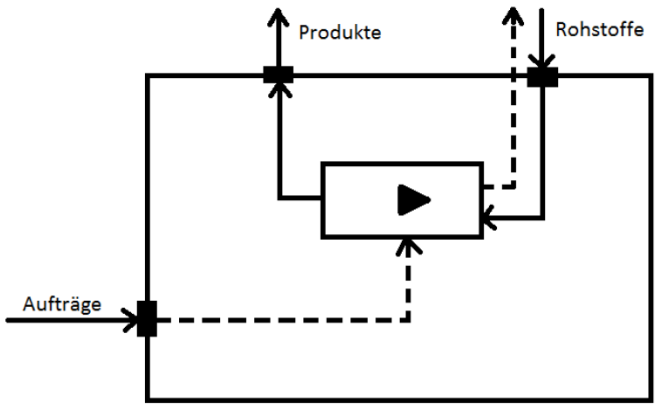


Abbildung 19 Produktion Konzept

BESCHREIBUNG

Die Komponente *Produktion* ist ein Platzhalter für den eigentlichen Produktionsprozess einer Firma. Sie nimmt Aufträge entgegen, fordert Rohstoffe an und produziert. Für diesen Vorgang (Austauschen von Rohstoffen und Produkten) ist die Komponente *Lager* zwingend. Aufträge können sowohl von einer *Lager*- als auch von einer *Verkaufs*-Komponente kommen.

PROZESSE

- Order entgegen nehmen
- Rohstoffe bestellen
- Rohstoffe entgegen nehmen
- Produzieren
- Produkte liefern
- Auftrag ablehnen

PARAMETER

- Produkt Typen
- Produktionsketten
- Ressourcen

STATISTIK

- Aufträge/ Zeit
- Produkte/ Zeit
- Auslastung
- Abgelehnte Aufträge/Zeit

TESTS

- Auftrag rein, Produkt raus (Beliebig viele Rohstoffe)
- Auftrag rein, keine Rohstoffe vorhanden Auftrag abgelehnt

LAGER

BESCHREIBUNG

Die Lager Komponente ist für die Lagerung von spezifischen Entities verantwortlich. Ein Lager kann für jeden Artikel einen Event auslösen welcher zur Nachbestellung von diesem verwendet werden kann. Zusätzlich kann es einen Event auslösen welcher kennzeichnet, dass das Lager überlastet ist. Ausserdem kann das Lager jederzeit über die Lagerbestände abgefragt werden.

PARAMETER

KAPAZITÄT

Die maximale Anzahl an Artikeln welche in das Lager eingelagert werden können.

ARTIKELLISTE

Dieser Parameter benötigt eine Liste in welcher folgende Elemente vorhanden sein müssen:

- Artikel
- Mindestlagerbestand
- Benötigte Slots

EIN-/AUSLAGERUNGSZEIT

Zeit welche für die Ein-/Auslagerung benötigt wird.

VERSCHLEISSWAHRSCHEINLICHKEIT

Dieser Parameter gibt die Wahrscheinlichkeit an mit welcher beim Ein- oder Auslagern der Artikel zerstört wird.

PROZESSE

- Erfolgreiche Einlagerung
- Ausschuss bei Einlagerung
- Erfolgreiche Auslagerung
- Ausschuss bei Auslagerung
- Lagerbestände abfragen
- Event auslösen: Mindestlagermenge unterschritten

AUSWERTUNG & STATISTIKEN

- Zeitliche Auslastung
- Räumliche Auslastung
- Fehlgeschlagene Einlagerungen
- Fehlgeschlagene Auslagerungen
- Durchschnittliche Lagerdauer eines Artikels

DYNAMISCHE KOMPONENTEN (ENTITIES)

Die Hauptfunktion der beweglichen Systemteile in einer Simulation ist das Austauschen von Informationen. Es bietet sich an die Entities als *DataObjcets* zu definieren. `DataObjects` sind Daten speichernde Objekte ohne eigene Logik. Sie werden als eine Art Container für Informationen verwendet, entscheiden jedoch nicht eigenständig über den weiterführenden Systemverlauf.

INFORMATIONSFLUSS

Tabelle 7 Konzept dynamische Komponenten 1

Order
• OrderID • Kunde • List<Artikel, Menge> • Liefertermin • Status [offen, inBearbeitung, geliefert, bezahlt]
Ladeliste
• LadelisteID • Sender • Empfänger • List<Deliveries>
Rechnung
• RechnungsID • OrderID • Betrag • Zahlungsfrist
Mahnung
• MahnungsID • Level [1, 2, 3] • RechnungsID • Strafbetrag • Zahlungsfrist

MATERIALFLUSS

Tabelle 8 Konzept dynamische Komponenten 2

Artikel
• ArtikelID • Preis pro Stück • Grösse • Gewicht
Rechnung
• RechnungsID • OrderID • Betrag • Zahlungsfrist

GELDFLUSS

Tabelle 9 Konzept dynamische Komponenten 3

Bezahlung
• Währung • Betrag

SCOR METRIKEN

Nicht alle SCOR Metriken eignen sich dafür als Produkt der SCLib angesehen zu werden. Die Metrik RS.3.129 Stock Shelf Cycle Time (Einlagerungszykluszeit) beispielsweise sollte vom Benutzer im Voraus definiert werden und als Eingangsparameter betrachtet werden. In der Tabelle 10 werden Metriken die erhoben werden könnten, auf die bisherigen Konzepte abgebildet. Alle Metriken aus dieser Tabelle wurden dem Supply Chain Operations Reference (SCOR) model. Overview - Version 10.0 [2] im Anhang entnommen.

Tabelle 10 Abbildung Metriken auf Komponenten

	Auswertung/Attribute	Metriken Input	Metriken Output
Einkauf			
	Importliste		RL.2.1 % of Orders Delivered in Full
	Produktionsliste		RL.2.2 Delivery Performance to Customer Receiving
			RL.2.4. Perfect Condition
Verkauf			
	Produkteliste		RL.2.1 % of Orders Delivered in Full
			RL.2.2 Delivery Performance to Customer Receiving
			RL.2.4. Perfect Condition
Factory			
	Production Time	RS.2.2 Make Cycle Time	
Rack			
	Gelagerte Produkte	RS.3.97 Pick Product from Backroom Cycle Time	AM.2.2 Inventory Days of Supply
	Produkteumschlag	RS.3.129 Stock Shelf Cycle Time	AM.2.5 Fixed Assets
	Auslastung		AM.2.8 Inventory
Product			
	Produktionszeit		
Order			
	Liefertermin SOLL		RL.1.1 Perfect Order fullfillment
	Liefertermin IST		RS.1.3 Order fullfillment Cycle Time
	Erstellungszeitpunkt		RL.2.4 Perfect Condition
	Priorität		

ABSTRAHIERTES MODELL

In einem zweiten Schritt geht es darum ein möglichst generisches Modell zu abstrahieren. Dieses Modell besteht aus den vier statischen Komponenten Einkauf, Verkauf, Lager und Produktion. Jede Komponente hat die Möglichkeit Bestellungen zu empfangen und in Form von Lieferungen zu Bearbeiten.

Um das Modell so simpel wie möglich zu halten werden ihm nur die dynamischen Komponenten Bestellung und Produkt hinzugefügt. Im Laufe der Implementation hat sich herausgestellt, das die Möglichkeit zum abfragen des Lagerbestandes benötigt wird. Dafür wurde das dynamische Objekt *Request* (Anfrage) definiert. Der Hauptfokus liegt auf dem dynamischen Element Bestellung.

Um den Produktfluss klar analysieren zu können muss stets klar ersichtlich sein, welches Produkt zu welcher Bestellung gehört. Die Produkte werden daher immer kombiniert mit einer Bestellung im System weitergegeben.

Die ganzen Prozesse innerhalb eines Unternehmens beginnen mit einer externen Bestellung. Der Verkauf nimmt die Bestellung entgegen. Er entscheidet, ob das gewünschte Produkt gelagert wird oder erst beschafft werden muss. Je nach dem schickt er eine Bestellung an die Komponente Lager oder Einkauf. Kommen die bestellten Produkte zurück, werden diese an den externen Kunden geschickt. Die Komponente Lager empfängt Bestellungen. Es ist nicht von Belang, ob diese von einer Verkaufs- oder einer Produktionskomponente kommen. Ist das Produkt in ausreichender Menge vorhanden, wird es an die bestellende Abteilung geliefert. Das Lager kann selbst Bestellungen senden, wenn ein Minimalbestand unterschritten wird. Wie das Lager empfängt auch die Einkaufskomponente intern versandte Bestellungen. Die Einkaufskomponente entscheidet, ob intern produziert werden kann oder extern bestellt werden muss. Je nachdem sendet sie eine interne Bestellung an die Produktionskomponente oder eine externe Bestellung an das nächste Unternehmen.

Abbildung 20 zeigt eine mögliche Kombination der Komponenten.

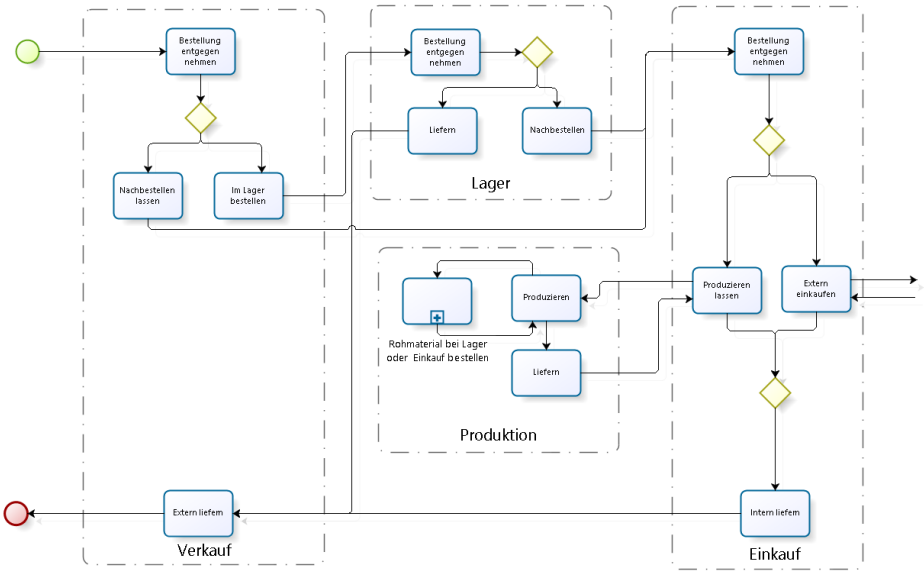


Abbildung 20 Komponentenübersicht

AUSBLICK ZUR UMSETZUNG

Um die Implementation international nutzen zu können, werden alle Komponenten auf Englisch bezeichnet. Im Weiteren werden die folgenden Begriffe verwendet:

Tabelle 11 Englisch/Deutsch abbildung der Komponenten

Verkauf	Selling
Einkauf	Purchase
Lager	Rack
Produktion	Factory
Bestellung	Order
Produkt	Product
Anfrage	Request

KOMPONENTEN

ENTITIES

Die dynamischen Teile von Simio sind die Objekte der `Entity` Klasse. Die dynamischen Komponenten der SCLib werden als `Entity` Klassen implementiert

Umgesetzt werden die `Entities` Order, Product und Request. Alle erben von dem von Simio zur Verfügung gestellten Standardentity `ModelEntity`

MODELENTITY

Als Vorlage für die speziellen `Entities` der SCOR-Library wurde das Bereits vorhandene `ModelEntity` gewählt, da es im Unterschied zu einem neu erstellten `Entity` bereits einige nützliche Funktionen beinhaltet. Erwähnenswert sind vor allem der `OnEnteredFreeSpace` Prozess sowie die `States` `Picture` und `Animation`.

Der vorhandene Prozess `OnEnteredFreeSpace` hat das Ziel, das Verhalten des `Entities` festzulegen wenn es zu einem undefinierten Ziel transferiert wird. Er dient dem abfangen von Fehlern.

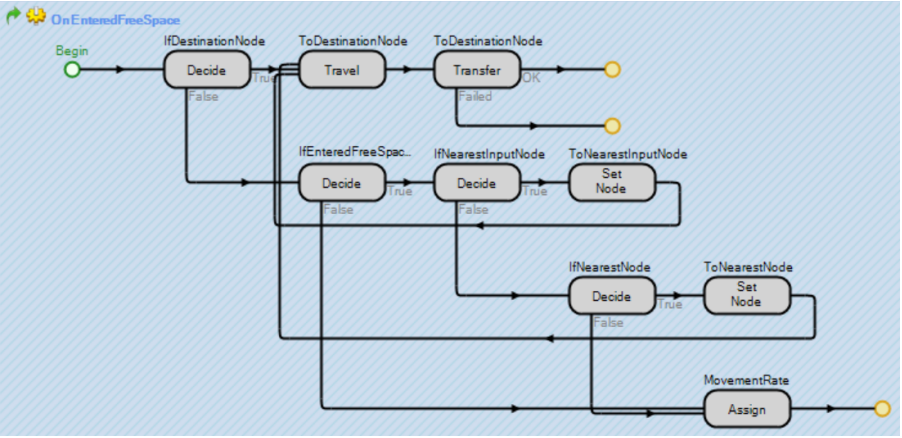


Abbildung 21 Vordefinierter Prozess beim ModelEntity

Zusätzlich zu einem leeren `Entity` hat das `ModelEntity` noch zwei `States`: `Picture` und `Animation`.

`Picture` ist ein `Realstate` (Eine Variable für eine reelle Zahl). In diesem `State` wird der aktuelle Symbolindex gespeichert. Das `Symbol` (Anzeigebild) des `Entities` kann über diesen Index während der Simulation verändert werden.

`Animation` ist eine Stringvariable. In ihr kann der Name einer `Animation` gespeichert werden. Dieser kann dann z.B. in einer Expression referenziert werden.

ORDER

Das `Entity Order` dient als Bestellung und/oder Lieferung. Die bestellten `Products` werden an das `Order Entity` angefügt und mit diesem transportiert. Die `Order` erbt von `ModelEntity` und besitzt all dessen Eigenschaften. Zusätzlich speichert es Informationen darüber was und wieviel Bestellt werden.

PROPERTIES

Über die Properties lässt sich der Ordertype beim instanziieren anpassen.

Tabelle 12 Order Properties

Name	Typ	Beschreibung
`OrderSizeExpression`	`Expression Property`	Die `OrderSizeExpression` dient dazu, `Orders` mit einer zufälligen Bestellmenge zu generieren. Es wird eine diskrete Randomfunktion übergeben, damit nur ganzzahlige Zufallswerte generiert werden. Standartwert: Random.Discrete(1, 0.8 , 3, 0.9, 2, 1)
`ProductType`	`Entity Property`	Dieses `Property` ist eine `Entity` Referenz. Sie Bestimmt, welches `Product` mit diesem Typ `Order` bestellt wird. Standartwert: null. Das `Property` muss beim instanziieren gesetzt werden.
`ExpectedDeliveryDelay`	Real `Property` Unit Type: Time	Hier kann angegeben werden, wie lange eine Lieferung für diesen `Order` Typ maximal dauern darf. Verglichen mit der effektiven Zeit die für die Lieferung benötigt wird, kann entschieden werden ob es eine „FirstTrySuccess“ Lieferung war. Standartwert: 7 Tage

STATES

Tabelle 13 Order States

Name	Typ	Beschreibung
`OrderSize`	Integer	Anzahl der bestellten `Products`
`RestOrderSize`	Integer	In der `RestOrderSize` wird die Anzahl im Lager verfügbarer `Products` gespeichert, wenn diese kleiner als die `OrderSize` ist. Je nach `DeliveryPriority` der `Order` wird entschieden, ob dieser Rest geliefert wird. Dies trifft auch zu wenn er nicht der `OrderSize` entspricht.

ParentOrderID	Real State Variable	Wenn eine statische Komponente aufgrund einer eingehenden Order selbst etwas bestellen muss, erstellt sie eine Kopie dieser Order. Als ParentOrderID wird die ID der originalen Order gespeichert.
Address	String	Im Address Property wird die Adresse der Komponente gespeichert welche die Order Erstellt hat. Andere Komponenten können darauf hin entscheiden, an wen die Lieferung gesendet wird.

PROZESSE

Der einzige Prozess dieser Komponente wird beim Initialisieren ausgeführt. Er speichert den errechneten Wert der OrderSizeExpression im State OrderSize.

PRODUCT

Wie die Order erbt auch das Entity Product vom ModelEntity. Dem Product wird keine weitere Funktionalität hinzugefügt. Wichtig ist nur, dass der EntityType vom Typ Product ist. So können andere Entities und Komponenten diesen ProductType abfragen.

REQUEST

Auch der Request erbt vom ModelEntity. Die Funktion dieses Entity liegt in der Übertragung von Informationen. Es kann in der Anwendung der Library beliebig erweitert werden. Die einzigen hinzugefügten States sind StockSize und ProductType.

STATES

Tabelle 14 Request States

Name	Typ	Beschreibung
StockSize	Integer	In diesem State wird der aktuelle Lagerbestand gespeichert, wenn der Request an ein Rack gesendet wird.
ProductType	EntityReferenceState	In diesem State wird eine Entity Referenz zu einem Product gespeichert. Wenn der Request an ein Rack gesendet wird und der ProductType mit dem StockableProduct des Racks übereinstimmt, schreibt das Rack seinen Lagerbestand in den StockSize State.

RACK

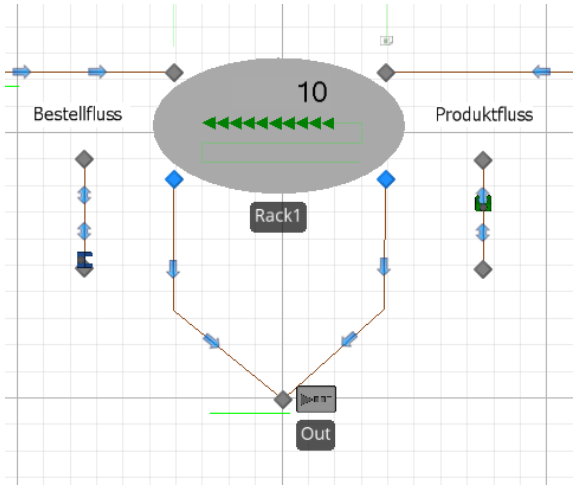


Abbildung 22 Rack

BESCHREIBUNG

Das `Rack` ist für die Lagerung eines spezifischen `Produktes (Product)` verantwortlich. Es verfügt über zwei Zugangs- sowie zwei Ausgangsknoten. Am Bestelleingang `(OrderInputNode)` können Bestellungen `(Orders)` abgegeben werden. Diese werden dann nach Möglichkeit mit den nötigen Produkten zusammengestellt und verlassen das `Rack` über den Produktausgang `(ProductOutputNode)`. Am Wareneingang `(ProductInputNode)` können Produkte `(Products)` mit den dazugehörigen Bestellungen `(Orders)` abgegeben werden. Diese werden dann nach Möglichkeit in das `Rack` eingelagert. Das `Rack` ist ausserdem im Stande eine Nachbestellung auszulösen, sobald eine spezifische Produkteschwelle unterschritten wird. Die Grösse des `Racks`, das spezifische Produkt, Ein- und Auslagerzeiten sowie weitere Parameter können vom Benutzer angegeben werden. Gehen Produkte, Bestellungen oder andere Objekte an den Zugangspunkten ein, welche nichts mit dem `Rack` zu tun haben, werden diese vom `Rack` ignoriert und weitergeleitet. Mehrere Racks können auch nebeneinander angeordnet und miteinander verbunden werden um ein Lager zu repräsentieren. Das `Rack` zeichnet Werte zur eigenen Auslastung sowie dem Produkteumschlag und der Lagerdauer von Produkten auf.

In den folgenden Erläuterungen werden nur noch die englischen Bezeichnungen welche in Simio benutzt werden angegeben.

PROZESSE

Nachfolgend sind alle konzeptionellen Abläufe in BPMN beschrieben.

- Einlagerung (Store)
- Auslagerung (Move Out)
- Event auslösen: Mindestlagermenge unterschritten (Reorder)
- Lagerbestände abfragen (Request entered)

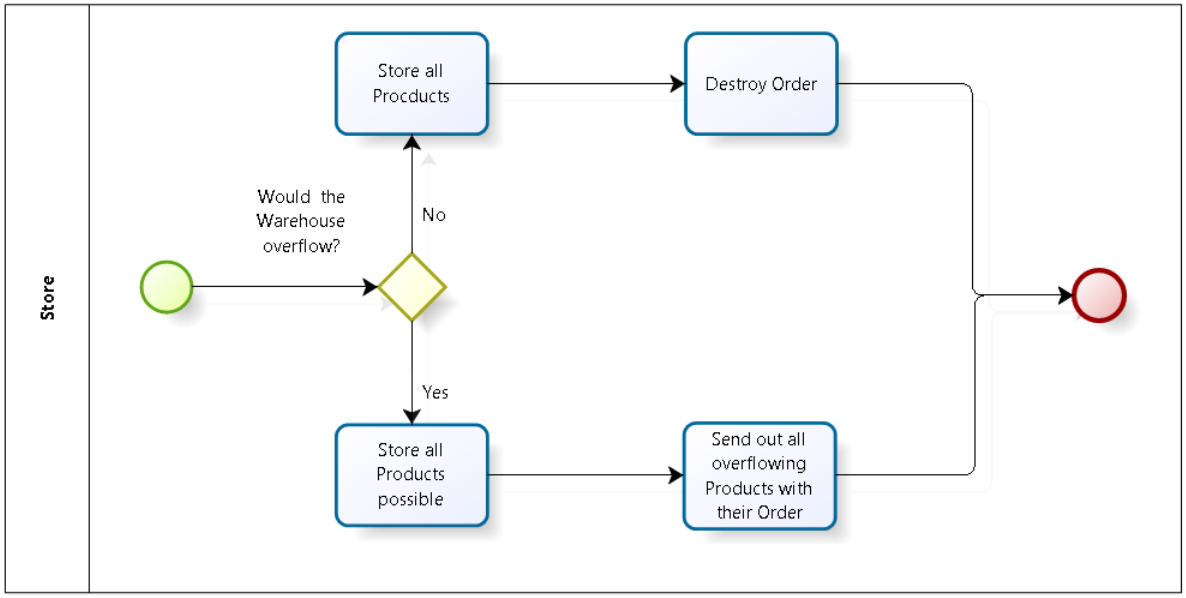


Abbildung 23 Store BPMN

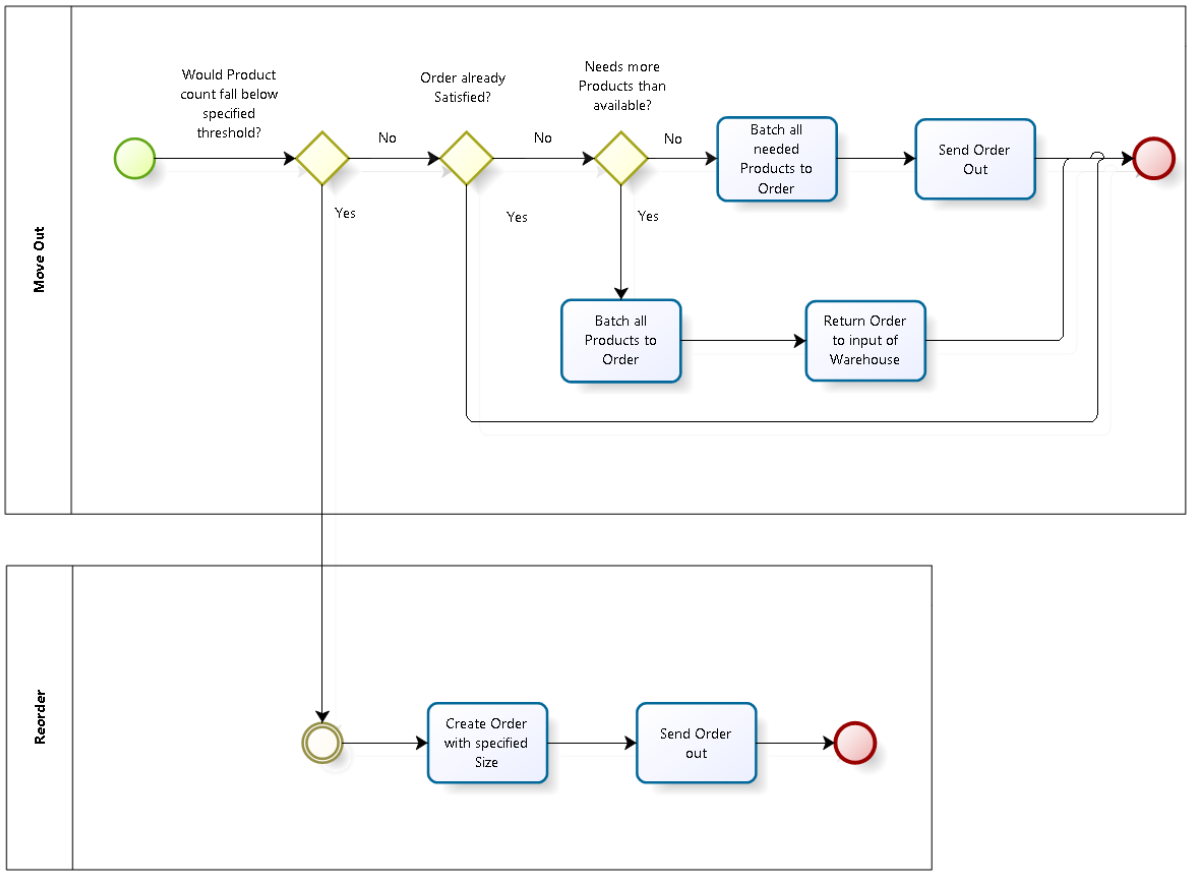


Abbildung 24 Move Out & Reorder BPMN

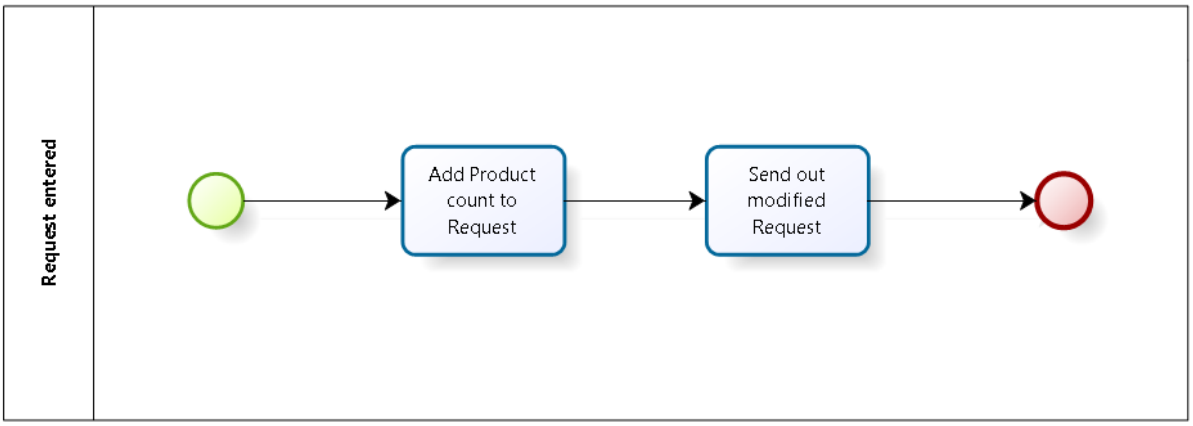


Abbildung 25 Request entered BPMN

UMSETZUNG IN SIMIO

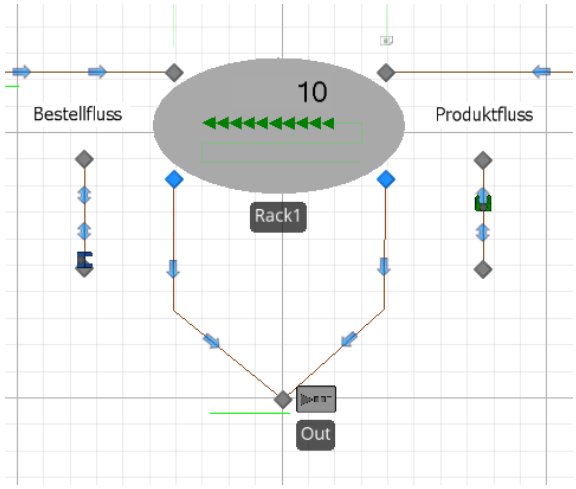


Abbildung 26 Rack

Das `Rack` stellt gegen aussen die vier Anschlussnodes `ProductInputNode`, `OrderInputNode`, `ProductOutputNode` sowie `OrderOutputNode` zur Verfügung.

`OrderInputNode`: An diesen `Node` werden auf `Products` wartende `Orders` gesendet. Dadurch werden `Products` ausgelagert.

`ProductOutputNode`: Von diesem `Node` aus werden die erfüllten `Orders` zurück ins System gesendet.

`ProductInputNode`: An diesen `Node` werden an `Orders` *gebatchte* `Products` gesendet. Dadurch werden `Products` eingelagert.

`OrderOutputNode`: Von diesem `Node` aus werden die überzähligen `Products` bei vollem `Rack` zurück ins System gesendet. Über diesen `Node` werden auch vom `Rack` erzeugte `Orders` und `Reorders` ins System gesendet.

ELEMENTE

BATCH `PRODUCTS` TO ORDER

Dieses `Batch Logic Element` dient dazu den `Batch` und `Unbatch Steps` in den Prozessen mitzuteilen welche Menge an `Products` zu den `Orders` werden sollen. Dieses `Element` stellt die Logik für den Normalfall zur Verfügung in dem genügend `Products` im `Shelf` vorhanden sind.

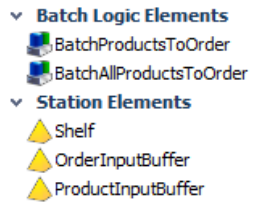


Abbildung 27 Elements

BATCH ALL `PRODUCTS` TO ORDER

Dieses `Batch Logic Element` dient im Gegensatz zum `BatchProductsToOrder Element` dazu, alle noch verfügbaren `Products` welche sich im `Shelf` befinden an die eingegangene `Order` zu *batchen*.

SHELF

Die `Shelf Station` enthält die gelagerten `Products` und stellt somit eine Art „Regal“ dar.

PRODUCT INPUT BUFFER

In der `ProductInputBuffer Station` werden `Orders` mit *gebatchten* `Products` zwischengelagert bevor sie in die `Shelf Station` eingelagert werden. Dies kann der Fall sein wenn die `StoreTime` höher als die Ankunftsrate der `Orders` ist oder wenn *geseizte* Ressourcen nicht schnell genug am gewünschten Ort ankommen.

ORDER INPUT BUFFER

In der `OrderInputBuffer Station` werden `Orders` mit oder ohne *gebatchten* `Products` zwischengelagert und warten darauf abgearbeitet zu werden. Die Differenz der `OrderSize` und den bereits zur `Order` *gebatchten* `Products` werden jeweils der `Order` hinzugefügt. Falls sich weniger `Products` im `Shelf` befinden als erforderlich werden alle verfügbaren `Products` zur `Order` hinzugefügt und die `Order` wird zurück in den `OrderInputBuffer` gesendet.

EIGENSCHAFTEN (PROPERTIES)

In diesem Kapitel werden die Eigenschaften des `Racks` erläutert. Die Eigenschaften oder auch `Properties` stehen dem Benutzer zur Verfügung um das `Rack` seinen Wünschen anzupassen. Alle Eigenschaften welche als `Fixed Class` geerbt wurden sind für das `Rack` auf invisible gesetzt. Somit sind diese Eigenschaften für Entwickler sichtbar und benutzbar, jedoch nicht für den Benutzer. Alle folgenden Eigenschaften wurden explizit für das `Rack` erstellt und können für zukünftige Erweiterungen benutzt werden.

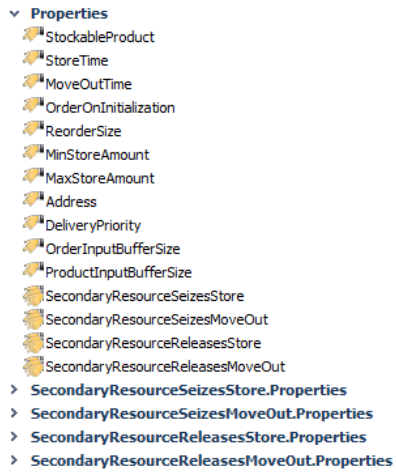


Abbildung 29 Properties Definitition

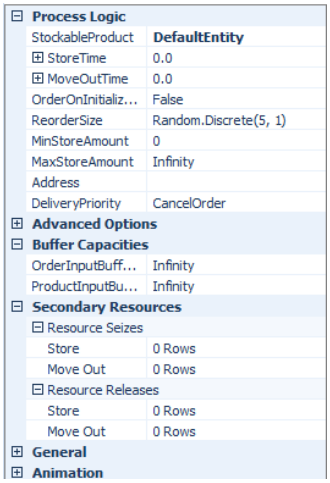


Abbildung 28 Property Window

STOCKABLE PRODUCT

Durch dieses `Entity Property` wird definiert welcher `Entity` Typ in der `Shelf Station` gelagert werden kann.

Default Units: **null**
Required Value: **False**

STORE TIME

Dieses `Expression Property` erlaubt dem Benutzer die Anpassung der benötigten Zeit zur Einlagerung eines `Products` in die `Shelf Station`.

Beispiel: Eine `Order` mit 5 gebatchten `Entities` wird aus dem `ProductInputBuffer` abgearbeitet. Die effektive Zeit diesen Vorgang zu durchlaufen beträgt somit 5 x `StoreTime`.

Default Value: **0.0**
Unit Type: **Time**
Default Units: **Seconds**
Required Value: **True**

MOVE OUT TIME

Dieses `Expression Property` erlaubt dem Benutzer die Anpassung der benötigten Zeit zur Auslagerung eines `Products` aus der `Shelf Station`.

Beispiel: Eine `Order` mit einer `OrderSize` von 5 wird aus dem `OrderInputBuffer` abgearbeitet. Die effektive Zeit diesen Vorgang zu durchlaufen beträgt somit 5 x `MoveOutTime`.

Default Value:	**0.0**
Unit Type:	**Time**
Default Units:	**Seconds**
Required Value:	**True**

ORDER ON INITIALIZATION

Dieses `Boolean Property` spezifiziert ob beim Start der Simulation eine `Reorder` mit `OrderSize` gleich des `MinStoreAmount` abgesendet wird oder nicht. Wenn das `Rack` nicht zusammen mit einer `Selling` Komponente benutzt wird kann die Nutzung dieses `Properties` notwendig sein. Das `Rack` sendet sonst nur eine spezifizierte `Reorder` aus, wenn ein gewisser Grenzwert unterschritten wird. Somit könnte unter Umständen nie etwas in das `Rack` eingelagert werden.

Default Units:	**False**
Required Value:	**True**

REORDER SIZE

Dieses `Expression Property` erlaubt es dem Benutzer eine Verteilungsfunktion oder einen festen Wert zu übergeben. Das `Property` wird dazu verwendet die `OrderSize` der `Reorder` zu bestimmen. In Kombination mit dem `MinStoreAmount Property` ist der Benutzer in der Lage den Bestand des `Racks` im Gleichgewicht zu halten.

Wichtig: Die übergebene Verteilungsfunktion muss eine diskrete Verteilungsfunktion sein bzw. der übergebene Wert ein Ganzzahliger.

Default Value:	**Random.Discrete(5, 1)**
Unit Type:	**Unspecified**
Required Value:	**True**

MIN STORE AMOUNT

Dieses `Integer Property` spezifiziert den zu unterschreitenden Grenzwert um eine `Reorder` ans System abzusenden. In Kombination mit dem `ReorderSize Property` ist der Benutzer in der Lage den Bestand des `Racks` im Gleichgewicht zu halten.

Wichtig: Wenn die Anzahl der `Products` in der `Shelf Station` den Wert des `MinStoreAmount Properties` unterschreitet, wird das `MinProductsThreshold Event` *getriggert*. Das bedeutet, dass es nur *getriggert* wird wenn von einem Wert grösser oder gleich `MinStoreAmount` auf einen Wert kleiner `MinStoreAmount` gewechselt wird.

Default Value:	**0**
Data Format:	**Integer**
Required Value:	**True**

MAX STORE AMOUNT

Dieses `Integer Property` spezifiziert die Grösse der `Shelf` Station. Falls zur Laufzeit versucht wird mehr `Products` einzulagern, werden die überschüssigen `Products` zurück an ihre `Order` *gebatcht* und verlassen das `Rack` über den `OrderOutputNode`.

Default Value: **Infinity**
Data Format: **Integer**
Required Value: **True**

ADDRESS

Dieses `String Property` spezifiziert die Adresse des `Rack` Objekts. Wird in diesem `Rack` Objekt eine `Order` oder `Reorder` erzeugt, wird der Wert dieses `Properties` als `Address` des erzeugten Objekts gesetzt.

Wichtig: Die Zuweisung eines Wertes an dieses `Property` ist optional. Wenn der Benutzer sich dazu entscheidet das `Address Property` zu benutzen, liegt es in seiner Verantwortung die Adressen eindeutig zu Verteilen.

Default Value: **-**
Required Value: **False**

DELIVERYPRIORITY

Dieses `List Property` spezifiziert die Priorität der `Orders` welche in diesem `Rack` erzeugt werden. Wird in diesem `Rack` Objekt eine `Order` oder `Reorder` erzeugt, wird der Wert dieses `Properties` als `Priority` gesetzt. Die möglichen Werte werden aus der `DeliveryPriorityList` entnommen.

Default Value: **CancelOrder**
Required Value: **True**

ORDER INPUT BUFFER SIZE

Dieses `Integer Property` spezifiziert die Grösse der `OrderInputBuffer Station`.

Wichtig: Dieser Wert muss grösser 0 sein, da sonst die `Orders` nicht abgearbeitet werden können.

Default Value: **Infinity**
Data Format: **Integer**
Required Value: **True**

PRODUCT INPUT BUFFER SIZE

Dieses `Integer Property` spezifiziert die Grösse der `ProductInputBuffer Station`.

Default Value: **Infinity**
Data Format: **Integer**
Required Value: **True**

SECONDARY RESOURCE SEIZES STORE/MOVE OUT

Die `SecondaryResourceSeizesStore` sowie `SecondaryResourceSeizesMoveOut` sind `Repeat Group Properties`. Sie bieten dem Benutzer eine Schnittstelle um Ressourcen wie z.B. einen `Worker` auf den entsprechenden Arbeitsvorgang zu *seizen*. Über diese Schnittstellen können auch Listen von mehreren Ressourcen gleichzeitig *geseized* werden. Diese `Properties` sowie alle Unterproperties wurden nach dem Vorbild des `Servers` aus der Standardlibrary von Simio erstellt.

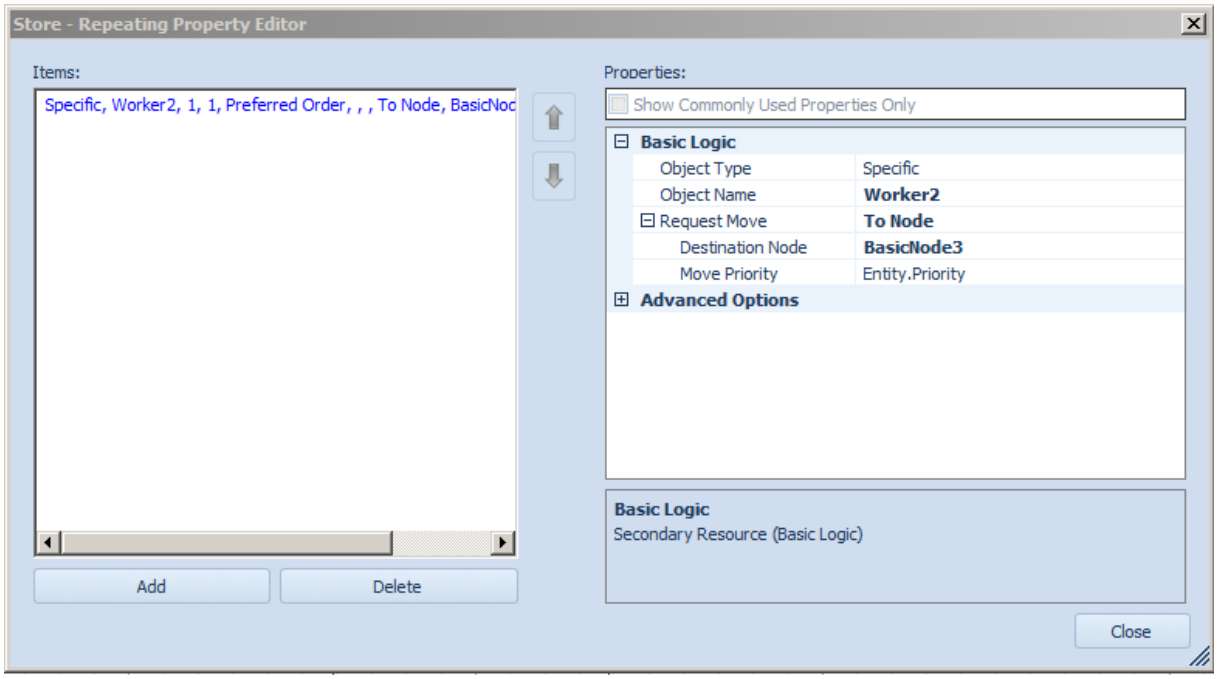


Abbildung 30 Seize Eingabefenster für Store Prozess

SECONDARY RESOURCE RELEASES STORE/MOVE OUT

Die `SecondaryResourceReleasesStore` sowie `SecondaryResourceReleasesMoveOut` sind `Repeat Group Properties`. Sie bieten dem Benutzer eine Schnittstelle um auf einen Arbeitsvorgang *geseizte* Ressourcen wieder zu *releasen*. Diese `Properties` sowie alle Unterproperties wurden nach dem Vorbild des `Servers` aus der Standardlibrary von Simio erstellt.

ZUSTÄNDE (STATES)

Abbildung 31 States

WORKING

Dieser `Boolean State` wird dazu benutzt den `MoveOutProcess` zu synchronisieren. Dieser ist für den Benutzer nicht sichtbar.

Public: **False**

EVENTS

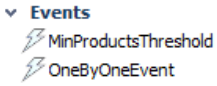


Abbildung 32 Events

MIN PRODUCTS THRESHOLD

Der `MinProductsThreshold` ist ein `Event` welcher *getriggert* wird, sobald die Anzahl `Products` in der `Shelf Station` den Wert in `MinStoreAmount` unterschreitet.

ONE BY ONE EVENT

Der `OneByOneEvent` wurde eingeführt um den `MoveOutProcess` zu synchronisieren. Das Grundprinzip besteht darin, dass der `MoveOutProcess` nach seiner Beendigung den nächsten `MoveOutProcess` starten kann. An allen anderen Stellen an denen ein `OneByOneEvent` *getriggert* wird, muss zuerst überprüft werden ob der `Working State` auf False gesetzt ist.

LISTS

DELIVERYPRIORITYLIST

Diese Liste spezifiziert die möglichen Werte für die in diesem `Rack` Objekt generierten `Orders`.

- CancelOrder
- RestOrder
- ReOrder

PROZESSE

In diesem Kapitel werden die Prozesse so wie sie effektiv in Simio implementiert sind beschrieben. Die Prozesse welche nur aus einem `End Transfer Step` bestehen oder keinen Beitrag zum Verständnis des Ablaufs aufweisen werden weggelassen. `Transfer Steps` werden nicht einzeln erklärt und zusammengehörende Funktionalitäten der `Steps` können zusammengefasst aufgezeigt werden.

ON RUN INITIALIZED

Dieser Prozess wird bei der Initialisierung der Simulation automatisch gestartet.

Decide Step ReorderOnInitialize: Hier wird das `Property OrderOnInitialization` überprüft. Ist dieses auf True gesetzt wird mit dem `Create Step` fortgefahren, ansonsten wird dieser ausgelassen und direkt zum `Wait Step` gesprungen.

Create Step CreateInitialOrder: Dieser `Step` dient dazu ein `Reorder Entity` zu erstellen. Das so erzeugte `Entity` wird an ein `Token` gebunden und zum `Assign Step` weitergeleitet.

Assign Steps AssignReorderSize, AssignAddress und Assign Priority: Diese drei `Assign Steps` setzen die Werte `OrderSize` und `Address` des erzeugten `Entities` auf `ReorderSize`, `Address` und `Priority` des `Racks`. Anschliessend wird das `Entity` aus dem `Rack` gesendet.

Wait Step WaitOneByOne: Dieser `Wait Step` leitet eine Endlosschleife ein. Diese Schleife wird durch das feuern eines `OneByOne Events` in Gang gesetzt und läuft jedes Mal einen Durchgang über den `Search Step` ab und steht beim `Wait Step` an bis der nächste `Event` eintrifft.

Search Step SearchOrdersInQueue: Dieser `Search Step` sucht ein `Entity` aus dem `OrderInputBuffer`. Wird ein `Entity` gefunden erzeugt der `Search Step` ein `Token`, bindet es an das `Entity` und startet damit durch den `Execute Step` den Prozess `MoveOut`.

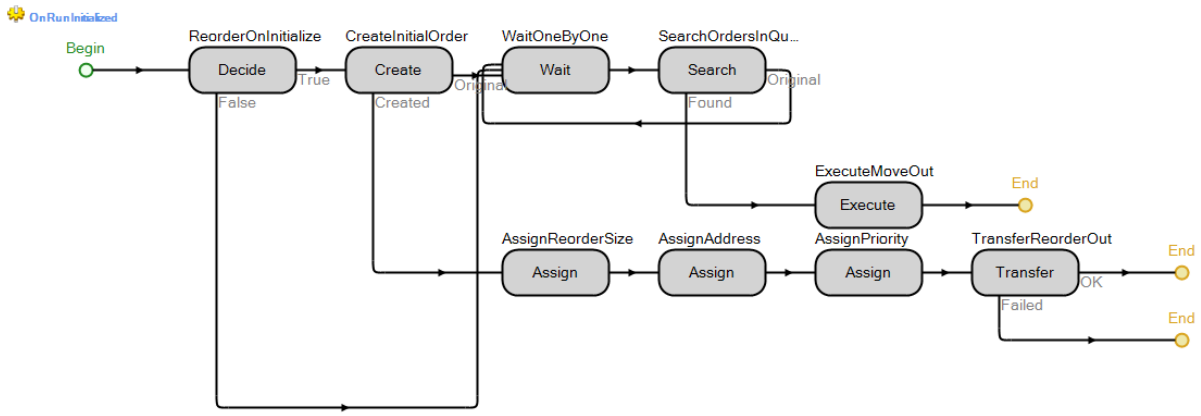


Abbildung 33 OnRunInitialized Prozess

ORDER INPUT ENTERED

Dieser Prozess ist als `Add-On Process` bei dem `OrderInputNode` registriert und wird jedes Mal gestartet, wenn ein beliebiges `Entity` das `Rack` über den `OrderInputNode` betritt. In diesem Prozess wird entschieden ob ein `Entity` überhaupt erst zum `OrderInputBuffer` transferiert wird.

Decide Step IsRequest und IsThisProductType: Zu Beginn wird entschieden ob das eingetroffene `Entity` ein `Request` ist oder nicht und ob der `ProductType` dem `StockableProduct` dieses `Racks` entspricht. Ist es ein `Request` für dieses `Rack` wird die aktuell gelagerte Anzahl `Products` auf `StockSize` addiert. Andernfalls verlässt das `Entity` das `Rack` auf direktem Weg. Ist das `Entity` kein `Request` wird mit dem `IsOrder Step` fortgefahren.

Decide Step IsOrder und IsThisProductType: In diesen beiden `Steps` wird überprüft, ob das eingetroffene `Entity` vom Typ `Order` ist und ob der `ProductType` dem `StockableProduct` dieses `Racks` entspricht. Trifft beides zu wird mit dem `Satisfied Step` fortgefahren. Sonst verlässt das `Entity` das `Rack` wieder.

Decide Step Satisfied: Dieser `Step` überprüft ob die Differenz der `OrderSize` der `Order` und der Anzahl *gebatchten* `Products` grösser als 0 ist. Ist sie grösser benötig diese `Order` keine weiteren `Products` und kann

das `Rack` wieder verlassen. Benötigt die `Order` weitere `Products` wird sie in den `MoveOut` Prozess weitergeleitet.

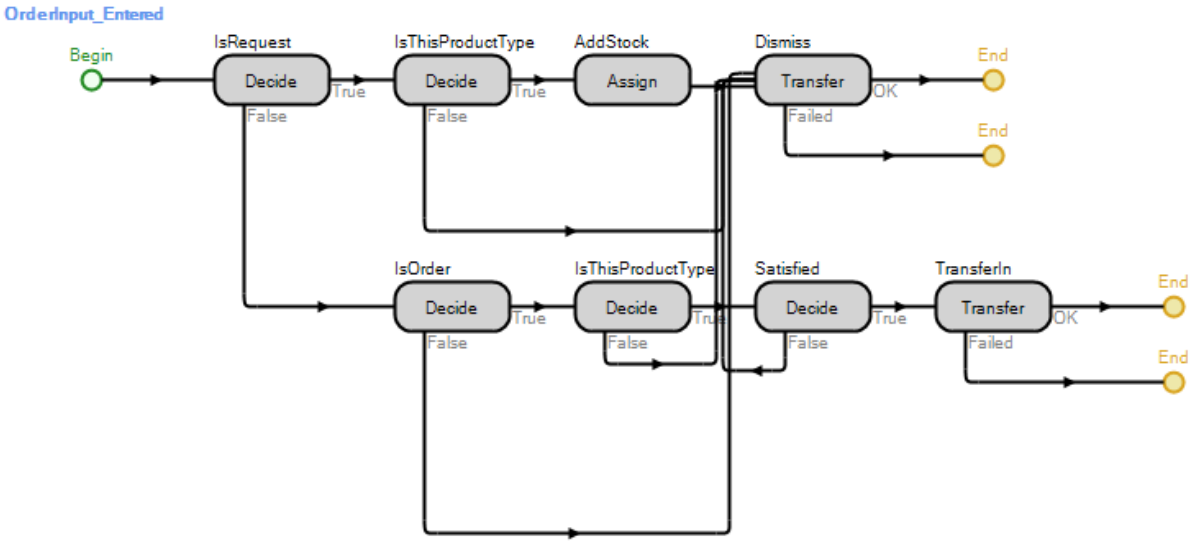


Abbildung 34 OrderInput_Entered Prozess

ORDER INPUT BUFFER ENTERED

Dieser Prozess wird gestartet sobald eine `Order` an den `OrderInputBuffer` gesendet wurde.

Decide Step Working und Fire Step FireOneByOne: In diesem `Decide Step` wird der `State Working` abgefragt. Ist dieser auf True gesetzt bedeutet das, dass ein `MoveOut` Prozess bereits in Gange ist und somit nicht angestossen werden muss. Ist der Wert auf False gesetzt wird ein `OneByOne Event` gefeuert von dem `Fire Step`.

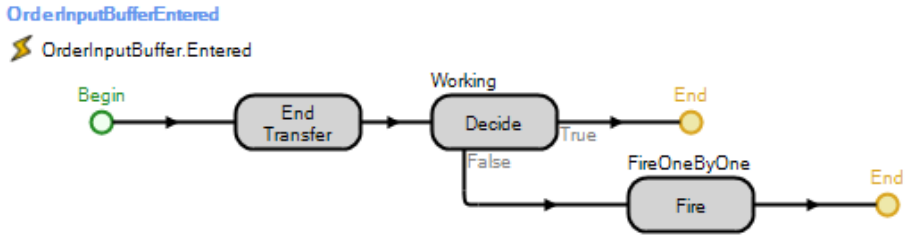


Abbildung 35 OrderInputBufferEntered Prozess

MOVE OUT PROCESS

Dieser Prozess wird in dem `OnRunInitialized` Prozess bei jedem Schleifendurchlauf gestartet in dem ein `Entity` aus dem `OrderInputBuffer` gefunden und entfernt wurde.

Assign Step Working: In diesem `Assign Step` wird der `State Working` auf True gesetzt. Dies hat zur Folge, dass nirgendwo anders ein weiterer `MoveOut` Prozess gestartet werden kann.

Seize Step SeizeMoveOutResources: Dieser `Step` wird dazu genutzt alle Ressourcen in der `Repeat Group SecondaryResourceSeizesMoveOut` *geseized* werden.

Decide Step FallBelowThreshold: In diesem `Decide Step` wird überprüft ob mit der Auslagerung der gewünschten `Products` die `MinStoreAmount` Schwelle unterschritten wird. Wenn ja wird im folgenden `Fire Step` das `MinProductsThreshold` gefeuert.

Delay Step MoveOutTime: Dieser `Delay Step` pausiert den Prozess für jedes `Product Entity` welches aus dem `Rack` ausgelagert wird um die `MoveOutTime`.

Decide Step IsSatisfied und folgende Search/Batch Steps: Dieser `Decide Step` entscheidet ob der Bedarf an `Products` der aktuellen `Order` abgedeckt werden kann. In beiden Fällen folgt ein `Search Step`. Der Unterschied ist nur, dass wenn der Bedarf mit den `Products` aus der `Shelf Station` abgedeckt werden kann, werden alle nötigen `Products` ausgelagert und an die `Order` *gebatched* und wenn nicht, werden alle ausgelagert welche in der `Shelf Station` sind.

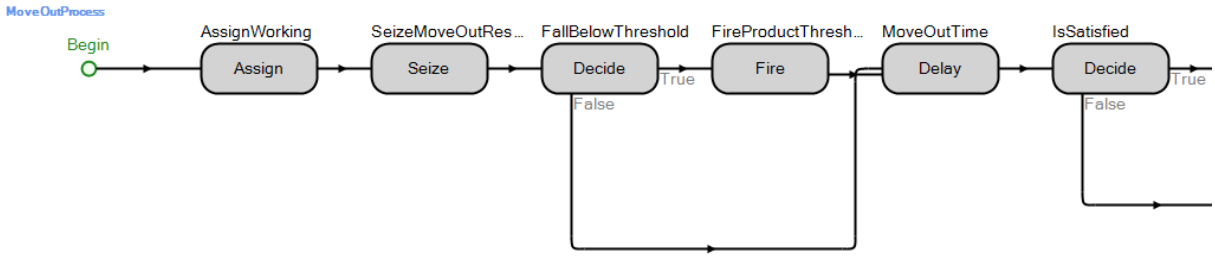


Abbildung 36 MoveOutProcess Part1

Release Step MoveOutResources: Dieser `Step` *released* alle Ressourcen welche im vorherigen `Seize Step` *geseized* wurden.

Assign Step NotWorking: Hier wird der `Working State` wieder auf False gesetzt, was bedeutet dass nun ein neuer `MoveOut` Prozess gestartet werden kann. Dies spielt jedoch keine Rolle mehr, da alle kritischen Schritte abgearbeitet wurden.

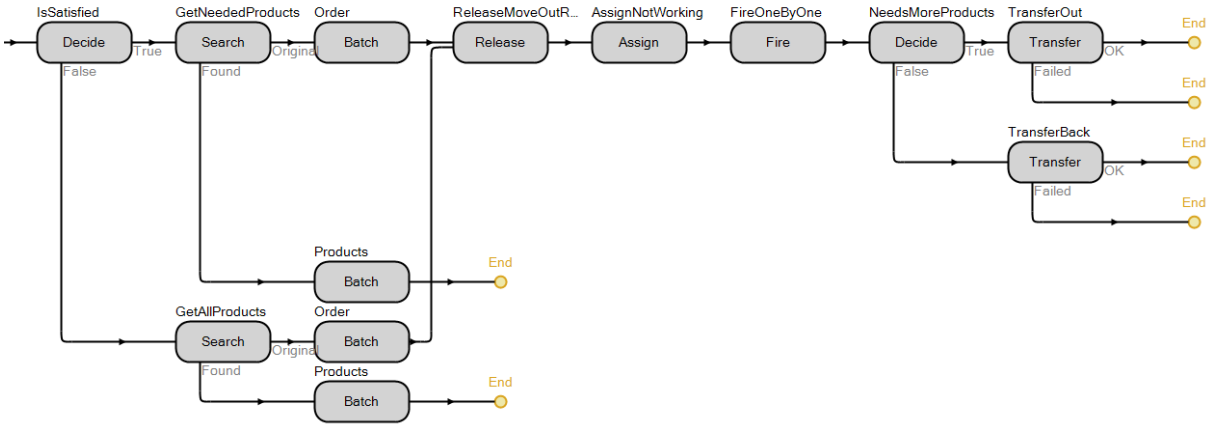


Abbildung 37 MoveOutProcess Part2

PRODUCT INPUT ENTERED

Dieser Prozess ist als `Add-On Process` bei dem `ProductInputNode` registriert und wird jedes Mal gestartet, wenn ein beliebiges `Entity` das `Rack` über den `ProductInputNode` betritt. In diesem Prozess wird Entschieden ob eine `Order Products` in dieses `Rack` einlagern darf.

Decide Steps IsProduct, IsThisProducType und HasProducts: In diesen `Steps` wird überprüft ob die eingetroffene `Entity` vom Typ `Order` ist, der `ProductType` dem `StockableProduct` entspricht und überhaupt `Product Entities` *gebatcht* hat. Falls einer dieser Fälle nicht zutrifft wird das `Entity` gleich wieder verworfen. Sind alle Überprüfungen erfolgreich verlaufen wird mit der `Order` ein `Store` Prozess ausgeführt.

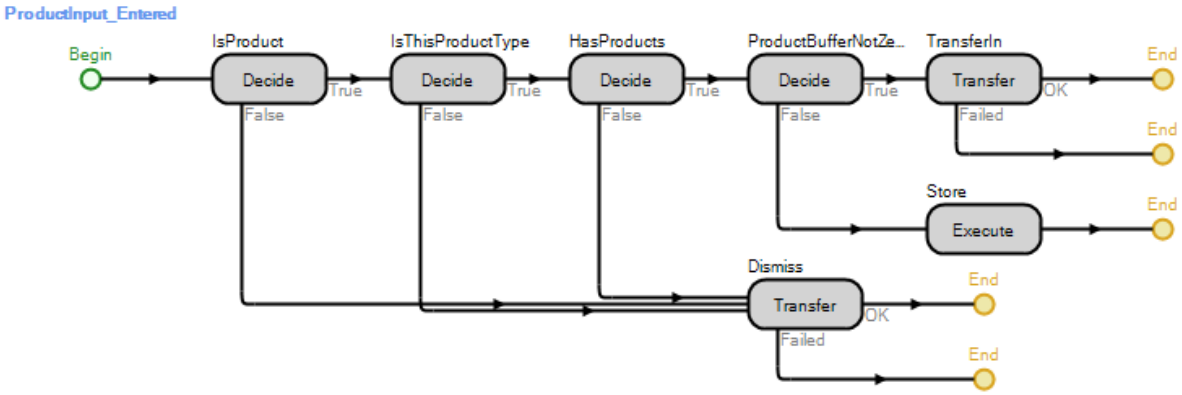


Abbildung 38 ProductInput_Entered Prozess

STORE PROCESS

Dieser Prozess wird aus dem `ProductInputBufferEntered` Prozess gestartet und dient dazu die `Entities` von einer `Order` zu *unbatchen* und in die `Shelf Station` zu transferieren.

Seize Step SeizeResourcesStore: Dieser `Step` wird dazu genutzt alle Ressourcen in der `Repeat Group` `SecondaryResourceSeizesStore` zu *seizen*.

Decide Step OverflowRack: Dieser `Deciede Step` überprüft ob der aktuelle Einlagerungsprozess das `Rack` überfüllen würde.

Delay und Unbatch Steps: Die folgenden `Delay Steps` pausieren den Prozess für jedes `Product Entity` welche an die `Order` *gebatched* ist um die `StoreTime`. Die `Unbatch Steps` *unbatchen* entweder alle oder alle nötigen `Products` um das `Rack` zu füllen aufgrund der Entscheidung des vorherigen `Decide Steps`.

Release Step StoreResources: Dieser `Step` *released* alle Ressourcen welche im vorherigen `Seize Step` *geseized* wurden.

Decide Step Working: Dieser `Step` überprüft den `Working State`. Ist er False wird ein `OneByOne Event` gefeuert um den `MoveOut` Prozess anzustossen.

Decide Step HasBatchedProducts: Wenn noch immer `Products` an die `Order` *gebatcht* sind, wird diese aus dem `Rack` gesendet. Wenn die `Order` leer ist, wird sie in dem `Rack` zerstört.

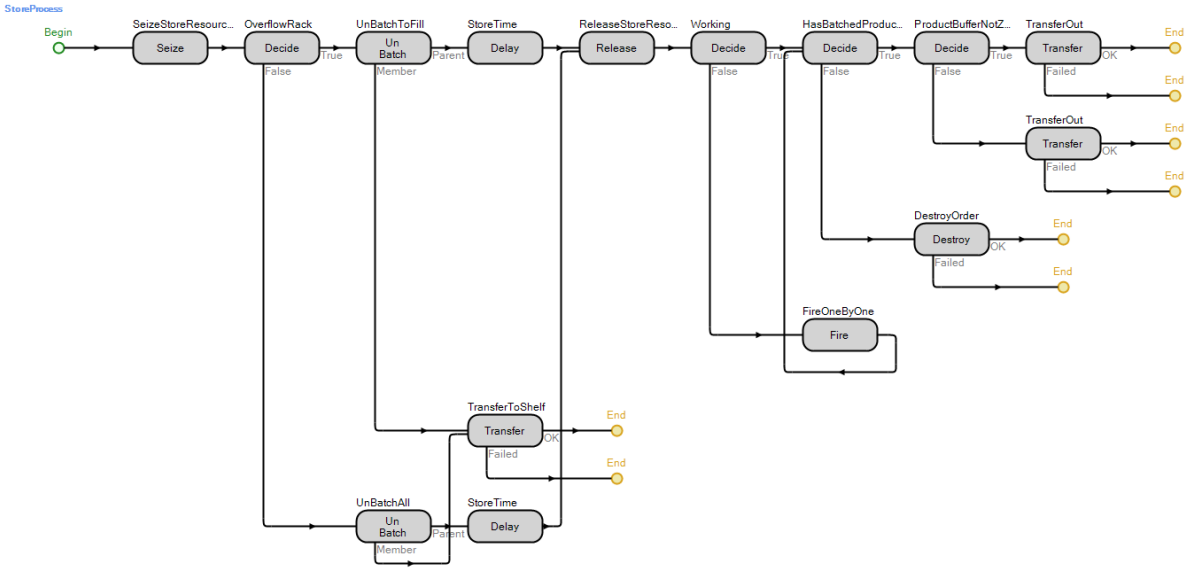
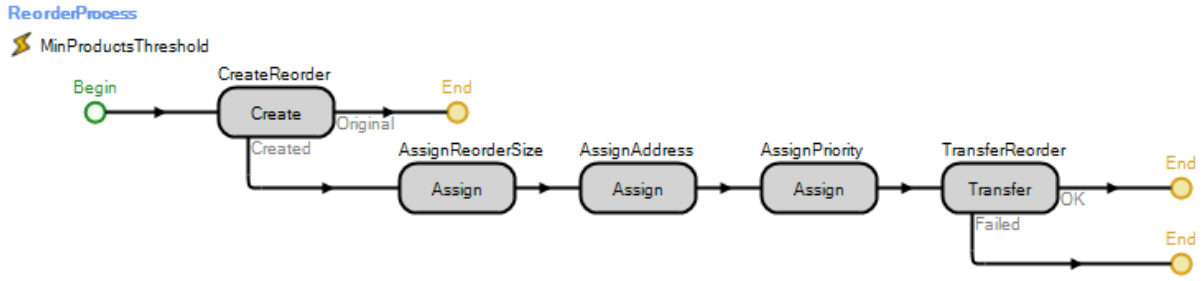


Abbildung 39 Store Prozess

REORDER PROCESS

Dieser Prozess wird durch das feuern eines `MinProductsThreshold Events` gestartet. Wie beim `RunOnInitialized` Prozess wird eine `Reorder` erzeugt. Die `Reorder` wird mit der Adresse des `Racks`, der `OrderSize` von `ReorderSize` und `Priority` zu versehen und an den `OrderOutputNode` transferiert.



AUSWERTUNG & STATISTIKEN

- Auslastung
- `Product`umschlag
- Lagerdauer eines `Products`

Shelf	Content	NumberInStation	Average	0.8189
			Maximum	11.0000
	HoldingTime	TimeInStation	Average (Mi...	0.2737
			Maximum (Mi...	4.9722
			Minimum (Min...	0.0006
	Throughput	NumberEntered	Total	4'308.0000
		NumberExited	Total	4'308.0000

Abbildung 40 Auswertung

Anhand dieser Werte können die SCOR Metriken Inventory Days of Supply, Fixed Asset Value und Order Fulfillment Cycle Time in einer Supply Chain gestützt werden.

ANWENDUNG

Anwendungsbeispiel Lager. Folgend ist erklärt wie man mit der `Rack` Komponente ein Lager für unterschiedliche `Products` anfertigen kann.

1. Die SCLib in ein Projekt laden.
2. Drei `Rack` Instanzen platzieren.
3. Mit `Paths` aus der `Standardlibrary` wie in Abbildung 41 verbinden.
4. Drei `Order` sowie drei `Product Entities` platzieren.
5. Für jede `Order` ein `Product` als `ProductType` anwählen.
6. In jedem `Rack` eines der `Products` als `StockableProduct` anwählen.
7. Sechs `Sourcen`, drei `Combiner` und eine `Sink` aus der `Standartlibrary` platzieren.
8. Die ersten drei `Sourcen` sollen die `Orders` produzieren. `Sourcen` wie in Abbildung 42 mit dem obersten `Rack` verbinden
9. Die zweiten drei `Sourcen` sollen die `Products` produzieren. `Sourcen` wie in Abbildung 43 mit den `Combiners` verbinden. Die OutputNodes der Combiners mit dem `ProductInputNode` des ersten `Racks` verbinden.
10. Die ersten drei `Sourcen` wie in Abbildung 44 mit den `Combiners` verbinden. Die so erstellten Pfade werden mit einer `Selection Weight` von 1.5 versehen. Bei den `Combiners` sollte darauf geachtet werden, dass die `Orders` mit den zugehörigen `Products` *gebatcht* werden.
11. Simulation Starten.

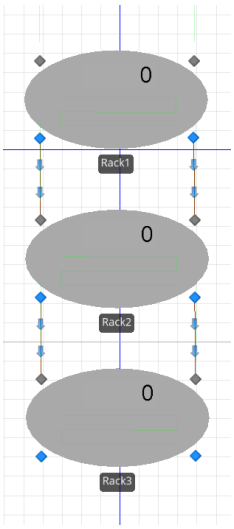


Abbildung 41 Racks verbinden

In dieser Anwendung wird ersichtlich, dass die Reihenschaltung der `Racks` von ausserhalb gleich benutzbar ist wie ein einzelnes `Rack`. `Entitys` welche nicht für ein `Rack` bestimmt sind werden weitergeleitet und gar nicht erst in den `InputBuffer` verschoben.

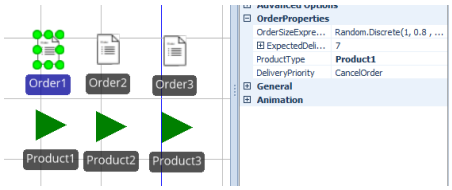


Abbildung 45 Orders und Products platzieren

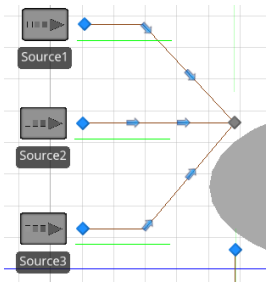


Abbildung 42 Sourcen mit Rack verbinden

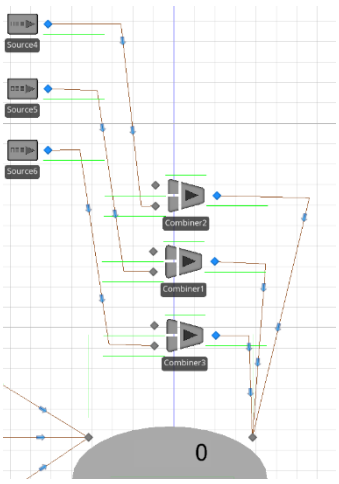


Abbildung 43 Combiners mit Rack und Sourcen verbinden

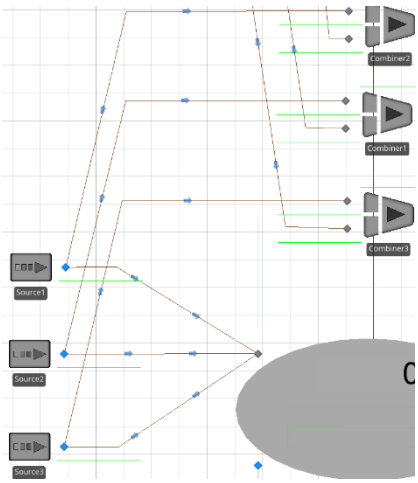


Abbildung 44 Restliche Sourcen mit Combiners verbinden

PROBLEME

RACE CONDITIONS

Während der Erarbeitung des `Racks` kam es öfters zu `Errors` in dem `MoveOut` Prozess. In den `Batch Steps` wurden Products von zwei unterschiedlichen `MoveOut` Prozessen gleichzeitig angefordert. Simio bietet in der Grundfunktionalität die `Steps` in den Prozessen an. Diese sind die kleinst möglichen Schritte für den Anwender. Da diese `Steps` nicht Elementar genug sind, ist jedoch in jeder Umsetzung ein gewisser Grad an Parallelität nicht zu vermeiden. Die in dieser Arbeit angewendete Lösung lässt eine gewisse Überlagerung von `MoveOut` Prozessen zu. Die kritischen abschnitte sind jedoch mit einer Art binärem Semaphor ausreichend abgesichert, damit sie nicht mehr mit Simulation provoziert werden können.

Für den Fall, dass in einer Simulation jedoch trotzdem ein solcher `Error` ausgelöst werden sollte bietet Simio die Möglichkeit Simulationen mit anderen Seeds zu initialisieren. Dies schränkt die Chance auf Probleme mit Race Conditions noch mehr ein.

FACTORY

EXTERN

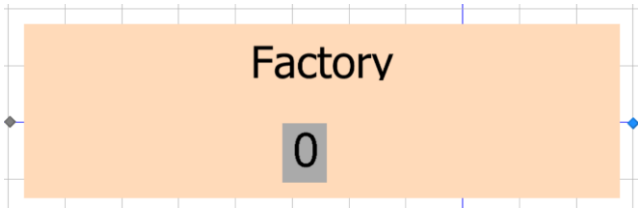


Abbildung 46 Factory externe Ansicht

Die `Factory` besteht aus einem beschrifteten Feld und einem `StatusLabel` das die jeweilige `OrderSize` anzeigt. Links ist ein `InputNode`. Dieser nimmt nur `Entities` vom Typ `Order` entgegen.

Aus der `Order` werden der `ProductType` und die `OrderSize` ausgelesen. In dieser ersten Version kann die `Factory` noch keine Rohstoffe bei anderen Komponenten bestellen. Aus dem `OutputNode` auf der rechten Seite wird die `Order`, kombiniert mit der gewünschten Anzahl `Entities` zurück ins System transferiert.

INTERN

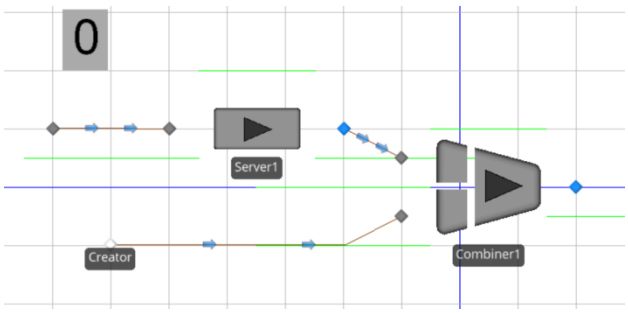


Abbildung 47 Factory interne Ansicht

Die Komponente `Factory` besteht aus einem `Server`, einem `Combiner` und einem `BasicNode` namens `Creator`. Wenn ein `Order Entity` eintrifft, wird daraus `EntityType` und `OrderSize` ausgelesen. Beim `CreatorNode`

werden anschliessend die gewünschten `Product Entities` in der gewünschten Anzahl erzeugt. Diese werden anschliessen mit dem `OrderEntity` kombiniert und zum `Output` weitergegeben.

PROPERTIES

PROCESSINGTIME

Das `ProcessingTime Property` definiert wie lange die Produktion für das Erstellen eines `Product Entities` benötigt. Das endgültige `Delay` für die Komponente ergibt sich aus der `OrderSize` der eingehenden `Order` multipliziert mit der `ProcessingTime`.

Der Unit Type ist Time. Als `DefaultUnits` sind Minuten definiert. Als Standartwert ist die Verteilfunktion

STATES

ACTUALORDERSIZE

Die `ActualOrderSize` ist ein Integer `State`, welcher die `OrderSize` der aktuell bearbeiteten `Order` zwischenspeichert. Die `ActualOrderSize` wird ausserdem im `StatusLabel` angezeigt.

PRODUCTTYPE

Der `ProductType` ist ein `Entity` reference `State`, welcher den `ProductType` der aktuell bearbeiteten `Order` zwischenspeichert.

PROZESSE

READORDER_PROZESS

`ReadOrder` Prozess wird vom `InputNode` auf den Event `Entered` getriggert. In den beiden `Assign Steps` werden die `OrderSize` und der `ProductTye` aus der `Order` ausgelesen und lokal zwischengespeichert.

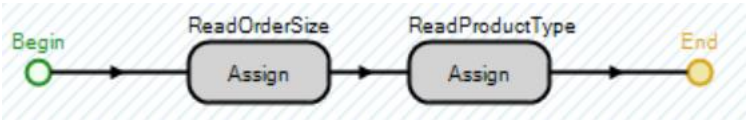


Abbildung 48 ReadOrder Prozess

CREATION_PROZESS

Der `Creation` Prozess wird vom `InputNode` auf den `Event Exited` getriggert. Im `Create Step` werden genauso viele neue `Entities` vom Typ `ProductType` erstellt, wie in `ActualOrderSize` angegeben. Diese werden dann zum `CreatorNode` transferiert.

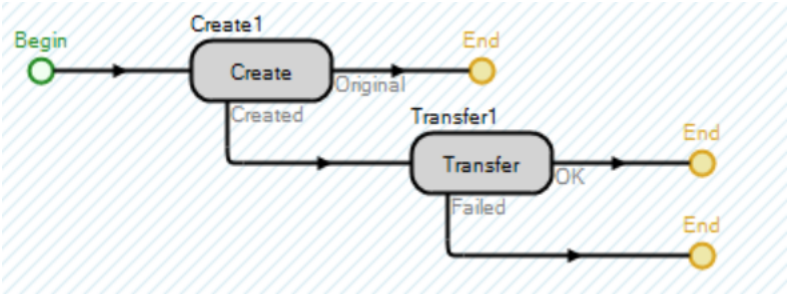


Abbildung 49 Creation Prozess

TESTS

Da Simio keine Testumgebung zur Verfügung stellt die sich mit einer Testumgebung von z.B. Programmiersprachen vergleichen liesse, ergeben sich folgende zwei Möglichkeiten die Komponenten isoliert zu testen.

Experimente Simio erlaubt es für jedes Modell Experimente zu erstellen. Diese sind besonders dafür geeignet Simulationsresultate zu erstellen und Werte darzustellen welche von bestimmten Wahrscheinlichkeiten abhängig sind.

Manuelles Testen Das manuelle Testen einer Komponente hat zur Folge, dass man die Tests nicht automatisieren kann. Jedoch gewinnt man mehr Einsicht in die Funktionsweise einer Komponente.

Der entscheidende Vorteil des manuellen Testens bringt jedoch die Reproduzierbarkeit von Fehlern. Zusammen mit der Tatsache, dass in den in dieser Arbeit erarbeiteten Komponenten Wahrscheinlichkeiten eine eher untergeordnete Rolle spielen fällt der Entscheid auf das manuelle Testen.

Zusätzlich werden Stresstests angefertigt welche die spezifischen Komponenten auslasten sollen. Diese werden gleich wie die manuellen Testumgebungen aufgebaut. Statt manuellen Auslösern sind Ankunftsraten für die `Sourcen` eingestellt.

RACK

MANUELLE TESTUMGEBUNG

Für die Umsetzung der manuellen Testumgebung wird ein neues Simio Projekt namens `ManualRackTesting` erstellt. In diesem Projekt wird die SCLib importiert und ein `Rack` erzeugt. Zusätzlich werden vier `Source` Objekte sowie ein `Combiner` Objekt und ein `Sink` Objekt aus der Standardlibrary platziert. Für jede `Source` muss ein `Button` und ein `Event` erstellt werden. Die `Buttons` feuern jeweils ein `Event` welcher in dem jeweiligen `Source` Objekt als `Triggering Event` übergeben wird. Anschliessend müssen noch drei `Orders` sowie ein `Product` instanziiert werden. Zuletzt werden die Objekte wie in Abbildung 50 miteinander verbunden. Die Objekte werden wie folgt eingestellt.

Tabelle 15 Manuelles Testen Rack Einstellungen

BigOrder	OrderSize	5
SmallOrder	OrderSize	1
OrderDummy	OrderSize	Beliebig
BigOrderSource	Entity Type	BigOrder
	Arrival Mode	OnEvent
	Triggering Event	BigOrderEvent
SmallOrderSource	Entity Type	BigOrder
	Arrival Mode	OnEvent
	Triggering Event	SmallOrderEvent
	Entities Per Arrival	3
OrderDummySource	Entity Type	BigOrder
	Arrival Mode	OnEvent
	Triggering Event	OrderDummyEvent
Combiner	-	-
Sink	-	-

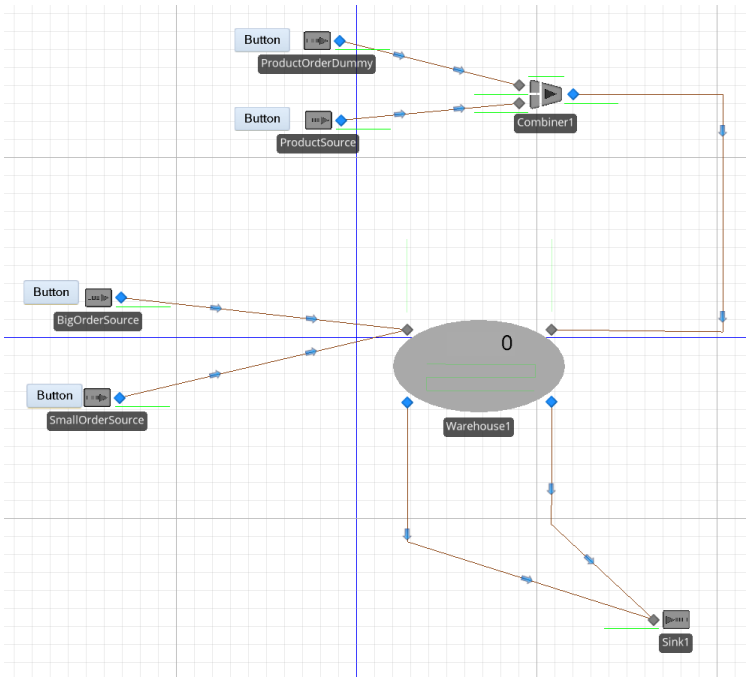


Abbildung 50 Manuelle Testumgebung

FACTORY

Die manuelle Testumgebung der `Factory` besteht aus zwei `Source` und zwei `Sink` Komponenten der Standardlibrary, sowie einer `Factory` Komponente aus der SCLib. Als dynamische Bestandteile werden die `Entities` `Order` und `Product` verwendet. Die `Orders` haben unterschiedliche `OrderSizes` und `ProductTypes`.

Tabelle 16 Manuelles Testen Factory Einstellungen

BigOrder	OrderSize	5
	ProductType	Product1
SmallOrder	OrderSize	1
	ProductType	Product2
BigOrderSource	Entity Type	BigOrder
	Arrival Mode	OnEvent
	Triggering Event	BigOrderEvent
SmallOrderSource	Entity Type	BigOrder
	Arrival Mode	OnEvent
	Triggering Event	SmallOrderEvent
Success_Sink	-	-
Fail_Sink	-	-
Product1	Farbe	Rot
Product2	Farbe	Grün

`Orders` können über einen `Button` erzeugt werden. Siehe Abbildung 52. Nach dem bearbeiten in der `Factory` wird Anzahl und Type der hinzugefügten `Products` mit den Angaben in der `Order` Verglichen sowie der `State` `Order.Korrekt` auf True oder False gesetzt. Durch diesen Wert wird dann entschieden, wie die `Order`

weitergeleitet werden soll. Ist `Order.Korrekt` auf True gesetzt, wird die Order zur `Success_Sink` weitergeleitet andernfalls zur `Fail_Sink`.

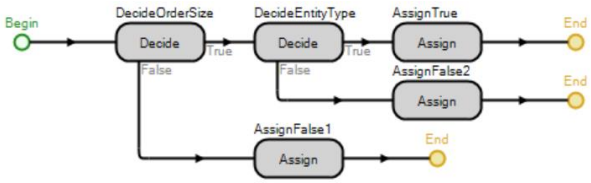


Abbildung 51 Validierung

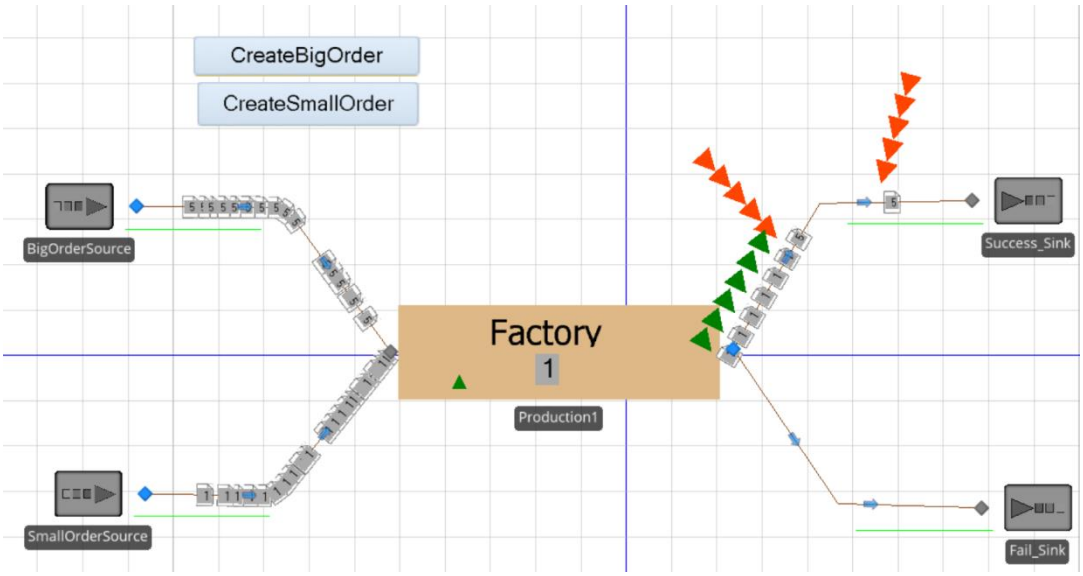


Abbildung 52 Testumgebung Factory

AUSBLICK

ALLGEMEIN

SUPPLY-CHAIN

Die Integration von Geldfluss welche explizit nicht Teil dieser Arbeit ist.

SIMIO

Da die Komponenten in erster Linie auf Supply-Chain und SCOR Anforderungen spezifiziert wurden, wurde während dieser Arbeit keinen Fokus auf Simio spezifische Funktionen gelegt. Alle Komponenten können daher in dieser Hinsicht verbessert bzw. hinzugefügt werden.

Beispiele

- `Input-/Outputbuffers`
- `Worktime Shedules`
- `Add-On Process Triggers`

DYNAMISCHE MANIPULATION

Mit `Events` und `Buttons` wäre es möglich zur Laufzeit `States` zu verändern. Somit könnte dem Benutzer eine Möglichkeit zur Anpassung gewisser Parameter zur Laufzeit gegeben werden.

RACK

STATIONEN

Die Stationen `Shelf`, `OrderInputBuffer` sowie `ProductInputBuffer` können um folgende Punkte erweitert werden.

- Dem Benutzer eine Expression zur Verfügung stellen mit welcher eine benutzerspezifische Reihenfolge für die Ein-/Austritte der `Entities` angegeben werden kann.
- Dem Benutzer eine Expression zur Verfügung stellen mit welcher eine Zeit angegeben werden kann nach welcher `Entities` automatisch aus der entsprechenden `Station` entfernt werden.

QUELLENVERZEICHNIS

LITERATUR

[1] Bolstorff, Peter A.; Poluha, Rolf G.; Rosenbaum, Robert G. (2007): Spitzenleistungen im Supply Chain Management. Ein Praxishandbuch zur Optimierung mit SCOR. Berlin, Heidelberg: Springer-Verlag Berlin Heidelberg.

[2] Supply Chain Council, Inc. (2010): Supply Chain Operations Reference (SCOR) model. Overview - Version 10.0. Hg. v. Supply Chain Council, Inc. Online verfügbar unter supply-chain.org/scor.]

[3] Jin Dong, Hongwei Ding, Changrui Ren, Wei Wang; IBM China Research Laboratory; Building 19 Zhongguancun Software Park, 8 Dongbeiwang West Road, Haidian District, Beijing 100094, P. R. China

WEBSITEN

[4] http://www.enzyklopaedie-der-wirtschaftsinformatik.de

[5] http://www.simio.com/

[6] http://www.apics.org/sites/apics-supply-chain-council

GLOSSAR

Batch, batchen	Zusammenfügen
Bullwhip-Effekt	Peitscheneffekt
Debugging	Fehlerbehebung
Entity	Objekt
Facility	Einrichtung, Umgebung
Factory	Fabrik
Institution	Einrichtung
Order	Bestellung
Product	Produkt
Purchase	Einkauf
Rack	Gestell
Request	Anfrage
Seeds	Saatkorn, Zahlenwert zur Euzeugung von Zufallszahlen
seize	Grösse
Selling	Verkauf
Service	Dienst, Dienstleistung
SimBits	Beispiel zu einem bestimmten Problem in Simio
Simio	Simulationssoftware
Sink	Senke
Slot	Platz, Niesche
Source	Quelle
Token	Marke, in Simio: Durchläufer in einem Prozess
Trigger	Auslöser