HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Definition und Aufbau einer SCOR Simulationsbibliothek für Simio

Bachelorarbeit

Abteilung Informatik

Hochschule für Technik Rapperswil

Frühjahrssemester 2015

Autoren:	Sandra-Olivia Müller, Ernst Füllemann
Betreuer:	Prof. Dr. Andreas Rinkel
Abgabe:	11. Juni 2015

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, die explizit in der Aufgabestellung erwähnt sind 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: 11.06.2015

Name, Unterschrift: 

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 eingehender 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 Lieferpriorität[1].
Produkt (Product)	Speichert Herstellungszeitpunkt.
Request	Speichert Produkttyp und Lagermenge.

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

[1] Die Lieferpriorität beschreibt, wie mit der Bestellung verfahren werden soll, wenn nicht genügend Waren im Lager vorhanden sind.

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[2] 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 viele 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, Factory, Selling und Purchase sowie die dynamischen Komponenten Product, Order und Request implementiert. Anhand dieser Komponenten ist man als Benutzer in der Lage, grössere Supply Chains zu erstellen und auszuwerten. Die Einsatzfähigkeit der Library wurde anhand des *Little Beergames* demonstriert.

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

[2] **Business Process Model and Notation**

DANKSAGUNG

Mein herzlicher Dank geht an Herr Prof. Dr. Andreas Rinkel, der uns bei dieser Arbeit vorbildlich unterstützt hat.

Auch möchte ich mich bei meiner Familie - meinen Eltern, meiner Schwester und meinem Partner - bedanken, die mich zu diesem Studium ermutigt haben und mich dabei zu jeder Zeit, mit allen Mitteln und vor allem bei dieser Arbeit unterstützten. Vielen Dank für eure Geduld und die Kraft, die ihr mir gegeben habt!

Nicht zuletzt möchte ich mich noch bei Ernst Füllemann für die angenehme und inspirierende Zusammenarbeit bedanken.

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

TECHNISCHER BERICHT

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.

Steht eine Fabrik still, weil kein Nachschub an Rohstoffen vorhanden ist, entstehen Kosten. Ebenso kostet es, Rohstoffe in so grossen Mengen zu lagern, dass sie immer zur Verfügung stehen. Das Gleiche gilt bei der Produktion bzw. Lagerung von fertigen Produkten. Ist die Nachfrage zu gross und kann nicht erfüllt werden, springen Kunden ab, was zu Verlusten führt. Lager voll von Produkten, die keiner kauft, führen ebenfalls zu Verlusten. Dabei ist zu beachten, dass in einer Lieferkette viele Firmen zwar voneinander abhängig sind, jedoch völlig eigenständig agieren können. Jede Aktion eines Unternehmens hat Reaktionen der Handelspartner zur Folge.

Ein Werkzeug, um diese Komplexität messbar 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 können 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 einer 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 es wird gezeigt, 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 (Lieferungskette) 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 vorgefertigten 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.

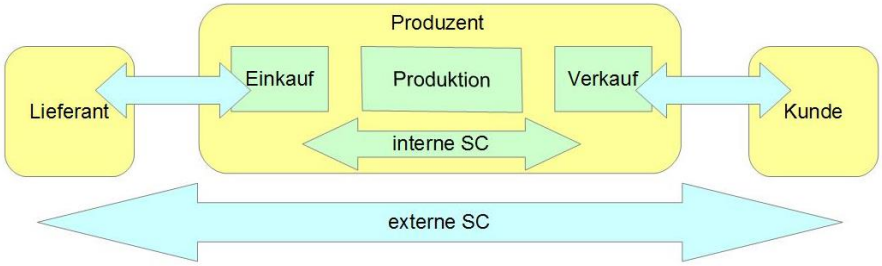


Abbildung 1 - Interne und externe Supply Chain

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
- Das Entwickeln von *Performance Management* und *Performance Measurement Systems*

Bei den letzten drei Punkten kann Simulation eine Unterstützung sein. Die entwickelte Library unterstützt die Konfiguration und Planung der Prozessstrukturen, in dem man Supply Chains darstellen und modellieren kann. Die Library ist so strukturiert, dass einige SCOR Metriken mit der Experiment Funktion in Simio berechnet werden können.

SCOR – SUPPLY CHAIN OPERATION REFERENCE

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 helfen, 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 berücksichtigt. Über den *Source*- und den *Deliver*-Prozess sind die Lieferanten mit den Kunden verbunden.

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 Ressourcen 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 Kleiderversant, 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.

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]

Auf Level 1 wird zwischen den fünf Hauptprozessen unterschieden. Der hier dargestellte Make-Prozess wird auf Level 2 in die drei Prozesse Make-to-Stock, Make-to-Order und Engineer-to-Order aufgeteilt. Dies sind drei Varianten des Make Prozesses. Die anderen Level 1 Prozesse sind auf Level 2 ebenfalls in drei bis fünf Varianten aufgeteilt. Auf Level drei werden diese Prozessvarianten Schritt für Schritt beschrieben. Dabei muss nicht jeder Level 3 Prozess in jedem Fall berücksichtigt werden. Vielmehr ist es eine Auflistung aller möglichen Unterprozesse.

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 aller SCOR Metriken und Prozesse befindet sich im *Supply Chain Operations Reference (SCOR) model. Overview - Version 10.0*. Dieses Dokument liegt der elektronischen Abgabe der Arbeit bei, oder ist online verfügbar, unter supply-chain.org/sco.

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.

Performance Attributes	Metrik	Einheit
Reliability	*Perfect Order Fullfillment*	Anzahl der beim ersten Versuch erfolgreichen Auslieferungen
	Delivery Performance	PerfectOrder Fullfillment pro totale Anzahl Auslieferungen
Responsiveness	*Order fullfillment leadtimes*	Durchschnittliche Zeit zwischen Bestellung und Auslieferung
Asset Management Efficency	*Inventory Days of Supply*	Total gelagerte Produkte pro Zeit
	Fixed Assets	Gelagerte Produkte

Tabelle 2 - SCOR Metriken [6]

Die Betrachtung des Geldflusses ist nicht Teil dieser Arbeit und ist noch nicht in der Bibliothek implementiert. Dies kann in einer erweiterten Implementation mit dem hinzufügen von Kosten-Properties in den Komponenten erreicht werden. Die den Geldfluss betreffenden Metriken werden hier aus diesem Grund 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 als kurze Filmsequenz 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 lösen lassen, bietet Simio auch eine Programmierschnittstelle für C# mit API an.

Simio arbeitet mit Objektorientierten Elementen. Es ist so aufgebaut, dass jedes Element ein eigenes Modul darstellt. Die mit den Elementen erstellten Module können wiederum als Elemente verwendet werden. So lassen sich ganze Projekte einfach als zusätzliche Library in ein neues Projekt importieren.

STARTEN MIT SIMIO

Dieses Kapitel soll eine kurze Einführung zu Simio, seinen Funktionen und Anwendungsmöglichkeiten geben. Für einen tieferen Einstieg in das Modellieren mit Simio eignen sich die Tutorials auf der offiziellen Simio Website[3].

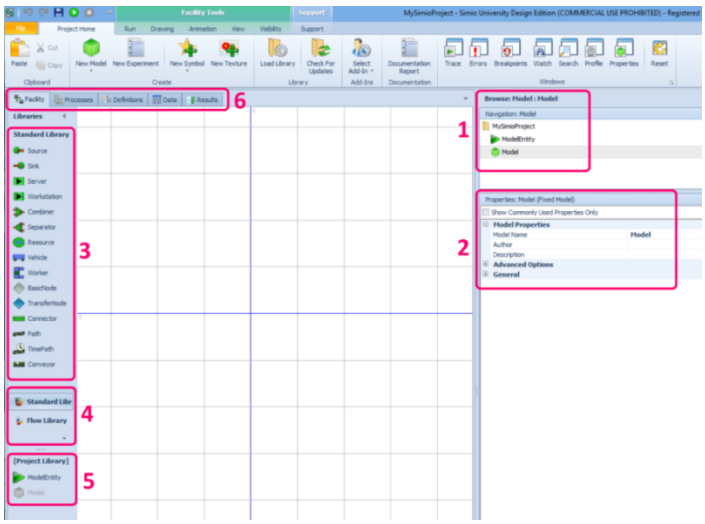


Abbildung 4 - Simio nach dem Starten

1. Im Navigationsfeld sind das Projekt und alle darin enthaltenen Modelle sichtbar. Standartmässig erstellt Simio nach dem Starten ein neues Projekt (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] http://www.simio.com/resources/videos/index.php

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.

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	Das `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.

Tabelle 3 - Simio Objekte

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.

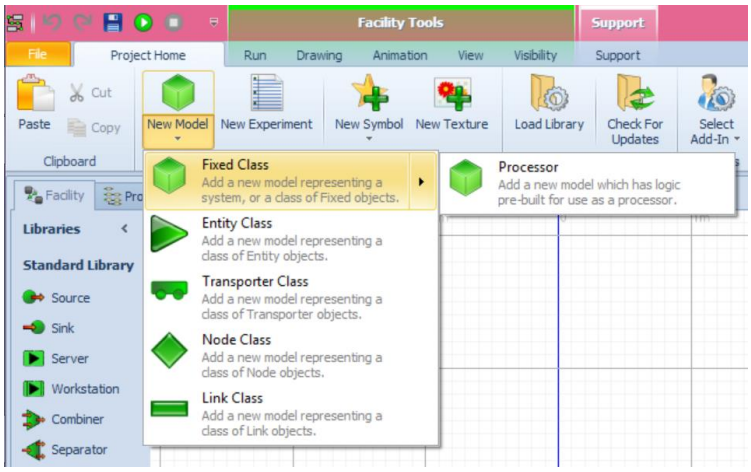


Abbildung 5 - Komponentenklassen

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. Die Schlussfolgerung all dieser Papers ist, dass SCOR mit verschiedensten Ansätzen simulierbar ist, jedoch je nach Tiefe der Implementierung sehr aufwändig werden kann. Ein solches *Research Paper* beschreibt das 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 Smart-SCOR 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

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

SUPPLY CHAIN PROCESS IMPROVEMENT

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

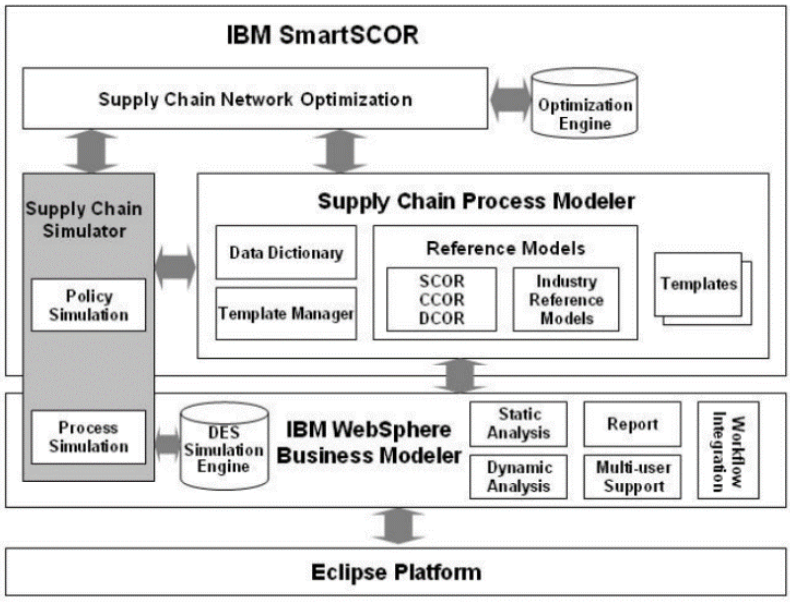


Abbildung 6 - IBM SmartSCOR Architektur [3]

SmartSCOR besteht aus den Modulen Supply Chain Network Optimization, Supply Chain Simulation und Supply Chain Process Analysis. Die Module können einzeln, oder im Zusammenspiel benutzt werden. Als Endbenutzer hat man über die IBM Websphere Zugriff auf den Business Modeler. Diesen kann man via Eclipse Plattform benutzen. [3] Im Vergleich zur erstellten SCLib, ist IBM Smartscor nicht in erster Linie auf die Simulation beschränkt. Vielmerh soll es bei der analyse und dem Management bereits existierender Supply Chain helfen.

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, schwächer oder gar nicht ins Gewicht.

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 7 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.

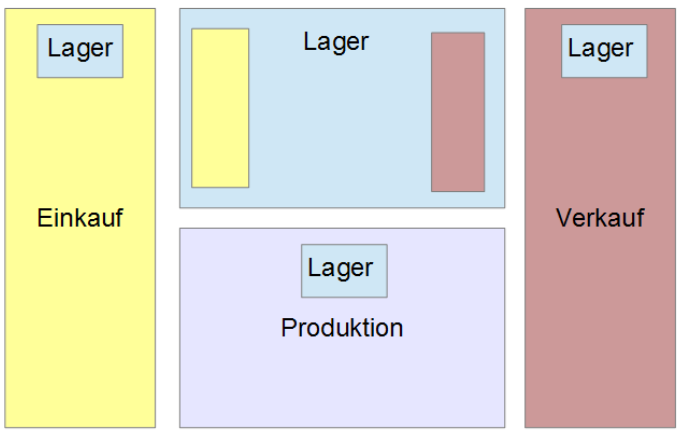


Abbildung 7 - Erster Entwurf der Komponenten

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.

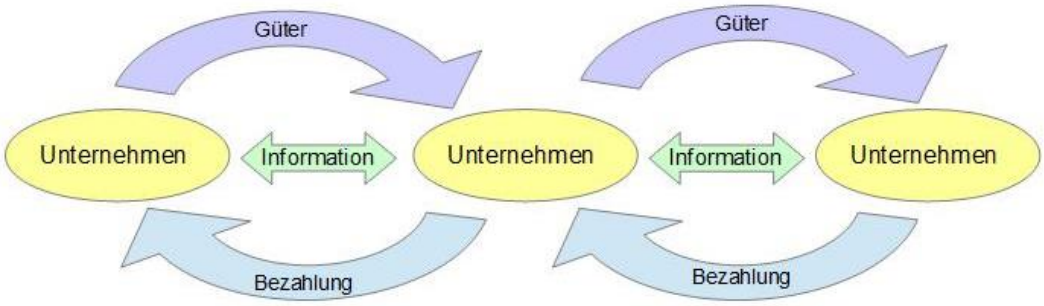


Abbildung 8 - Informations-, Material- und Geldfluss

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. Konzept wird um die Funktionalitäten

- Geldfluss
- Return Prozess
- Rohstoffe bestellen

reduziert und in einem ersten Schritt in Simio umgesetzt.

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

Im Folgenden werden die einzelnen Komponenten isoliert betrachtet.

EINKAUF – VERKAUF

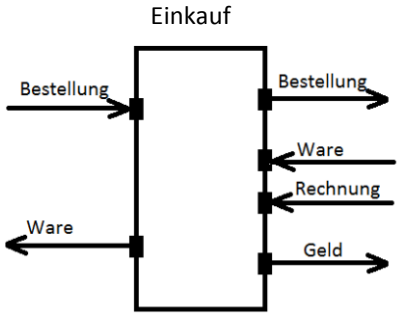


Abbildung 9 - Einkauf Konzept

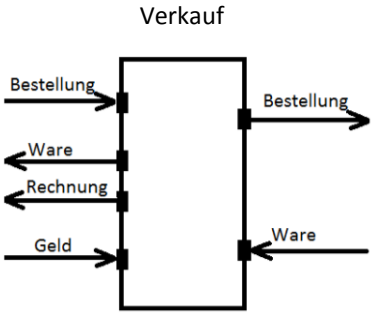


Abbildung 10 - Verkauf Konzept

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

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

Tabelle 4 - Konzept für Prozesse in Einkauf/Verkauf

PARAMETER

Einkauf und Verkauf besitzen Informationen darüber, was in ihrem Unternehmen verkauft wird, was produziert wird und was importiert wird. Anhand dieser Informationen wird über das Bearbeiten einer Bestellung entschieden.

STATISTIK

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

Tabelle 5 - Konzept für Statistiken in Einkauf/Verkauf

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

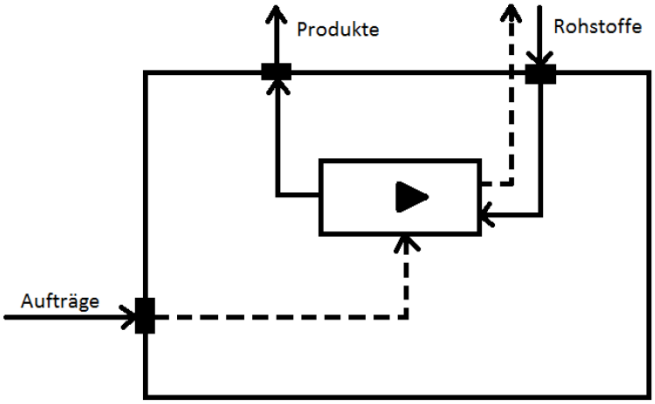


Abbildung 11 - Produktion Konzept

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
- Produktionszeit

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 für Nachbestellungen auslösen. Zusätzlich kann ein Event ausgelöst werden, wenn das Lager überlastet ist. Das Lager kann jederzeit über die Lagerbestände abgefragt werden.

PARAMETER

KAPAZITÄT

Die maximale Anzahl lagerbarer Artikel.

ARTIKELLISTE

Dieser Parameter benötigt eine Liste mit folgenden Elementen:

- Artikel
- Mindestlagerbestand
- Benötigte Plätze

EIN-/AUSLAGERUNGSZEIT

Zum Ein- und Auslagerung benötigte Zeit.

VERSCHLEISSWAHRSCHEINLICHKEIT

Dieser Parameter gibt die Zerstörungswahrscheinlichkeit der Artikel beim Ein- oder Auslagern an.

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

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

Tabelle 6 - Konzept dynamische Komponenten 1

MATERIALFLUSS

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

Tabelle 7 - Konzept dynamische Komponenten 2

GELDFLUSS

Bezahlung
• Währung • Betrag

Tabelle 8 - Konzept dynamische Komponenten 3

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 9 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 entnommen.

	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		

Tabelle 9 - Abbildung der Metriken auf die Komponenten

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, dass 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 12 zeigt eine mögliche Kombination der Komponenten und ihrer Prozesse. Diese BPMN-Grafik wurde mit dem BIZAGI Modler erstellt.

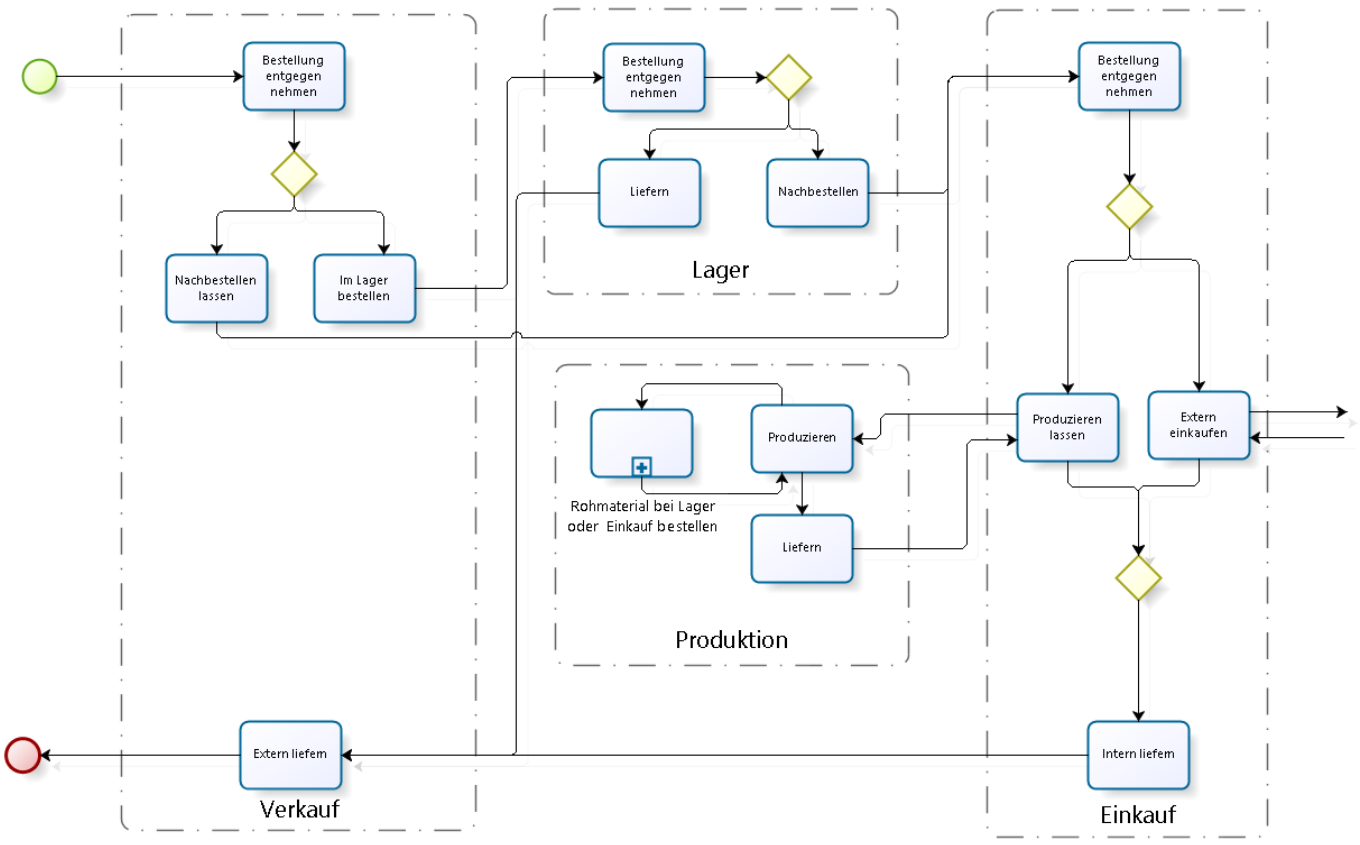


Abbildung 12 - Komponentenübersicht

AUSBLICK ZUR UMSETZUNG

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

Tabelle 10 - Englisch/Deutsch Abbildung der Komponenten

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

UMSETZUNG

IMPLEMENTIERUNG DER KOMPONENTEN

Im diesem Kapitel wird die Implementation der Library in Simio beschrieben. Es richtet sich an Simio-Fachkundige.

DYNAMISCHE KOMPONENTEN – ENTITIES

In allen statischen Komponenten muss mit denselben dynamischen Komponenten gearbeitet werden. Sie werden deshalb als erstes beschrieben. Die dynamischen Teile von Simio sind die Objekte der `Entity` Klasse. So werden auch die dynamischen Komponenten der SCLib als `Entity` Klassen implementiert.

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

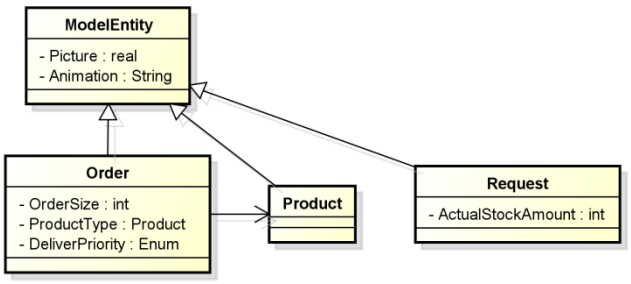


Abbildung 13 - Entity Class Diagramm

MODELENTITY

Als Vorlage für die speziellen `Entities` der SCLib wurde das Bereits vorhandene `ModelEntity` gewählt, da es im Unterschied zu einem neu erstellten `Entity` bereits nützliche Funktionen beinhaltet. Erwähnenswert sind vor allem 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. 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 wie viel bestellt wird.

PROPERTIES

Über die Properties lässt sich die Order beim Instanziieren anpassen.

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 instanzieren 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

Tabelle 11 - Order Properties

STATES

Die States sind zur Laufzeit veränderbare Variablen. Sie werden von den anderen Komponenten beschrieben oder ausgelesen.

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.

Tabelle 12 - Order States

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 bei der Anwendung der Library beliebig erweitert und verwendet werden. Die hinzugefügten `States` sind `StockSize` und `ProductType`. Standartmässiges Einsatzgebiet ist das Abfragen des Lagerbestandes in einem `Rack`.

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`.

Tabelle 13 - Request States

STATISCHE KOMPONENTEN

Die vier Statischen Hauptkomponenten sind `Selling`, `Purchase`, `Rack` und `Factory`. Sie können in vielen Varianten kombiniert werden.

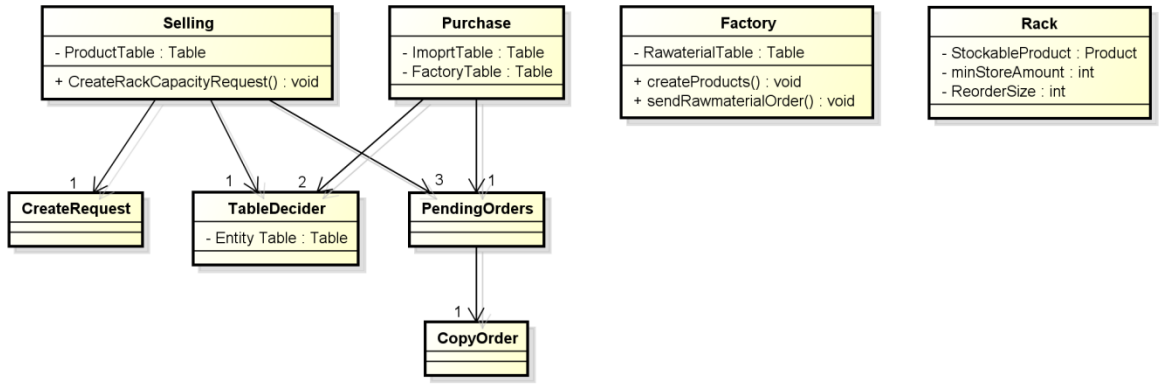


Abbildung 14 - Model Diagramm

Die Komponenten `Selling` und `Purchase` sind aus Teilfunktionen zusammengebaut. Diese Teilfunktionen sind als eigenständige Unterkomponenten implementiert und werden von den Komponenten `Purchase` und `Selling` ein, oder mehrmals verwendet.

UNTERKOMPONENTEN

TABLEDECIDER

Bei der Komponente `TableDecider` geht es darum, Orders auf zwei Ausgangsknoten zu verteilen. Das Kriterium hierfür ist der `ProductType` der Order.

Der `TableDecider` besitzt einen Eingangsknoten und zwei Ausgangsknoten, sowie eine Tabelle mit `Products`. Der `ProductType` einer eingehenden `Order` in dieser Tabelle gesucht. Da Simio keine *Contains*-Funktion für Listen anbietet, musste diese mit Hilfe von `Steps` zusammengebaut werden.

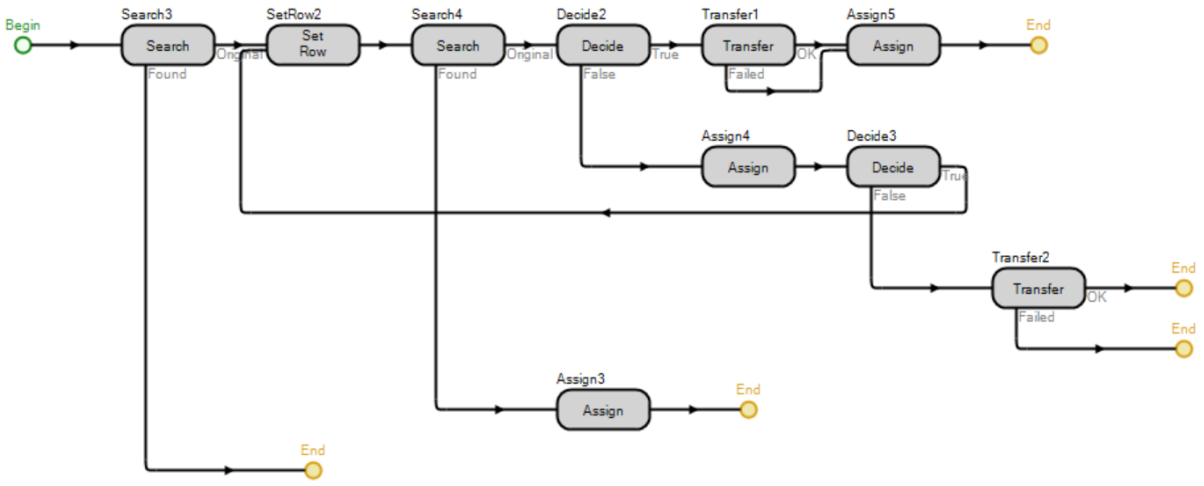


Abbildung 15 - TableDecider Prozess

Der Prozess aus Abbildung 15 wird gestartet, sobald eine `Order` in der Komponente ankommt. In einem ersten `Search Step` (`Search3`) wird die Anzahl in der Tabelle gespeicherter `Products` ausgelesen. Dieser Wert wird in der Variable `Count` gespeichert. Im darauffolgenden `SetRow Step` wird die Zeile mit dem Index `Count` in

der Tabelle angewählt. Im Step Search4 wird das in der angewählten Zeile gespeicherte Product mit dem ProductType der Order verglichen. Stimmen sie überein, wird der lokale Boolean Found auf wahr (True) gesetzt.

In einem nächsten Schritt (Decide2) wird entschieden, ob das Product bereits gefunden wurde oder nicht. Ist Found auf True wird die Order zum True-Ausgangskonoten transferiert und Found für die nächste Order wieder auf falsch (False) gesetzt. Wenn nicht, wird die Variable Count um 1 hinunter gezählt. Sind noch weitere Zeilen vorhanden (Count >= 0) geht der Prozess zurück zum SetRow Step und sucht in der nächsten Zeile. Bei Count<0 wird die Order zum False-Ausgangsknoten transferiert.

CREATEREQUEST

Das CreateRequest Modell hat die Aufgabe, zu einer eingehenden Order einen Request zu generieren. Dieser Request wird an ein Rack gesendet, um dort den Lagerbestand abzufragen. Kehrt der Request zurück, wird der ermittelte Lagerbestand mit der OrderSize verglichen. Die Orders werden danach sortiert, ob der Lagerbestand gross genug ist oder nicht.

Das Modell verfügt über zwei Stations. Die eine ist ein Eingangspuffer (InputBuffer) für die eingehenden Orders und die andere dient als Wartebereich (WaitingArea) für die aktuell zu bearbeitende Order. Tritt eine neue Order in den InputBuffer ein, wird als erstes ein Flag[4] gesetzt, damit keine Order gleichzeitig behandelt werden. Anschliessend wird der ProductType aus der Order ausgelesen und ein Request für diesen ProductType generiert. Die Order wird in die WaitingArea transferiert und wartet dort, bis der Request zurückkehrt.

Wird der Request an ein Rack mit gleichem ProductType gesendet, trägt dieses seinen aktuellen Lagerbestand ein. Kommt der Request mit dem ermittelten Lagerbestand zurück, wird dieser mit der OrderSize der wartenden Order verglichen. Ist der Lagerbestand grösser als die OrderSize geht die Order weiter zum Enough-Ausgangsknoten, andernfalls wird sie zum NotEnough-Ausgangsknoten transferiert. Der Request wird jeweils zerstört.

COPYORDER

Die Aufgabe der CopyOrder Komponente ist es, eine Kopie der eingehenden Order zu erstellen. Die Komponente hat einen Eingangs- und einen Ausgangsknoten. Um Race Conditions[5] zu verhindern, wurde hier mit einer Warteschlafe und einem InputBuffer gearbeitet. Diese Warteschlaufe startet mit dem Event OnRunInitialized, also wenn die Simulation zu laufen beginnt. Der Prozess geht als erstes in einen Wait Step (Wait1, Abbildung 16). Der Wait1 Step wartet auf den Event StartWorking. Sobald dieser Event gefeuert wird, geht der Prozess weiter zum Search Step (Search1). Es wird nach Entities im InputBuffer gesucht. Wird ein Entity gefunden, lauft dieses zum Exectue Step weiter und startet so den Kopierprozess. Der Originalprozess läuft wieder zurück in die Warteschlaufe bis zum nächsten StartWorikng Event.

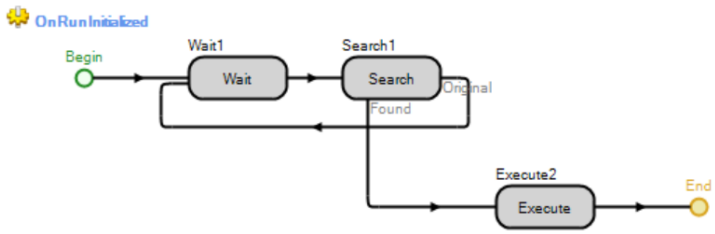


Abbildung 16 - Copy Order OnRunInitialized

[4] Marke, Markierung
[5] gleichzeitiges Zugreifen zweier Systeme auf denselben Wert

Das Feuern des `StartWorking Event` geschieht an zwei Orten. Zum einen im `InputBuffer_Entered` Prozess, also sobald ein `Entity` in den `InputBuffer` eintritt, und zum andern, sobald ein Kopiervorgang abgeschlossen wurde.

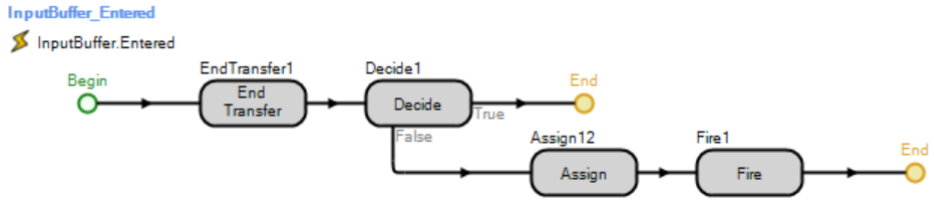


Abbildung 17 - InputBuffer Entered Prozess

Tritt eine `Order` in den `InputBuffer` ein, wird zuerst entschieden, ob der `Boolean IsWorking` auf `False` gesetzt ist (`Decide1`). Wenn dem so ist, wird er auf `True` gesetzt (`Assign12`), und der `StartWorking Event` wird gefeuert. Wie zuvor beschrieben, wird dann das nächste Element aus dem `InputBuffer` bearbeitet. Ist der `Boolean IsWorking` bereits auf `True`, bleibt die `Order` in der Warteschleife, bis der `Event` am Ende des Kopierprozesses gefeuert wird.

Der Kopierprozess erstellt eine neue `Order` mit demselben `ProductType` und derselben `OrderSize` wie das Original. Zusätzlich werden in der Kopie die ID des Originals unter `ParentOrderID` und die Adresse der Komponente in die neue `Order` gespeichert. Die Kopie wird an das Original gebunden (`batched`) und zum Ausgangsknoten weitergegeben.

PENDINGORDERS

In der `PendingOrders` Komponente geht es darum, Waren für eine Lieferung nachzubestellen. Die originale Bestellung wird mit Hilfe der zuvor beschriebenen `CopyOrder` Komponente kopiert und in einer `Station` gespeichert, währen die Kopie weitergeleitet wird. Kommt die Kopie zurück, so kann mit der `ParentOrderID` die originale `Order` aus der `Station` rausgesucht werden. Alle angehängten Produkte (`BatchMembers`) der Kopie werden daraufhin von ihr gelöst und an die originale `Order` angefügt. Die Kopie geht weiter in eine `Sink`, das Original wird zum Kunden weitergeleitet.

Jede `PendingOrders` Komponente benötigt eine `CopyOrder` Komponente. Die beiden Funktionen werden in diesem ersten Implementationsschritt als jeweils eigenes Modell aneinander gehängt. In einer späteren Implementation wäre es sinnvoll, diese beiden Funktionen in einer Unterkomponente zu vereinigen, da sie nur zu zweit verwendet werden können.

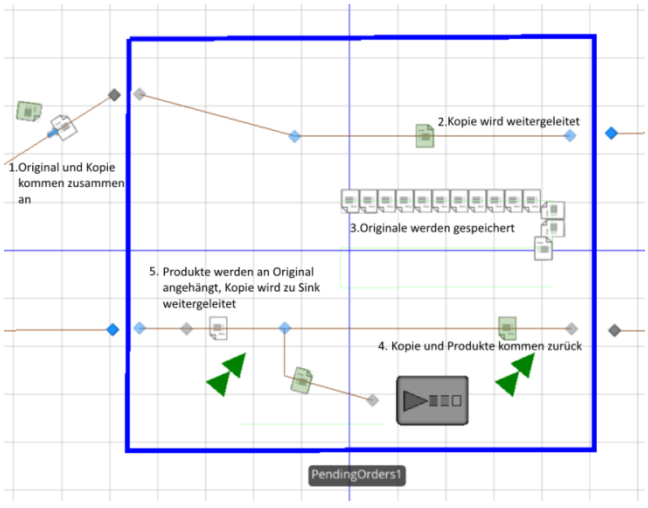


Abbildung 18 - Pending Orders

SELLING

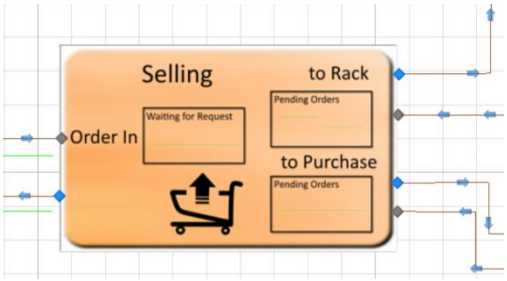


Abbildung 19 - Selling

BESCHREIBUNG

Die `Selling` Komponente ist die Schnittstelle zu externen Kunden. Sie nimmt Bestellungen entgegen und entscheidet, wie mit ihnen zu verfahren ist. Ist die Bestellung bearbeitet, wird diese mit den bestellten Produkten an den Kunden zurückgeschickt. Das Entgegennehmen und Zurückschicken von Bestellungen geschieht über den Eingangs und Ausgangsknoten bei „Order In“. Die Selling Komponente kann zusätzlich an ein `Rack` und eine weitere, `Orders` bearbeitende Komponente angeschlossen werden. Diese Anschlüsse sind mit „to Rack“ und „to Purchase“ bezeichnet. Die `Selling` Komponente generiert beim Eingehen einer `Order` einen `Request`. Dieser `Request` wird über den „to Rack“ Anschluss an ein angehängtes `Rack` gesendet und speichert dort den aktuellen Lagerbestand.

Der `to Purchase`-Anschluss versendet interne Bestellungen an die `Purchase` Komponente und empfängt die Lieferungen. Es ist möglich, am `to Purchase`-Anschluss eine andere Komponente anzuhängen, wie zum Beispiel eine `Factory`. Wichtig ist dabei nur, dass die Komponente `Orders` entgegen nehmen und weiterverarbeiten kann.

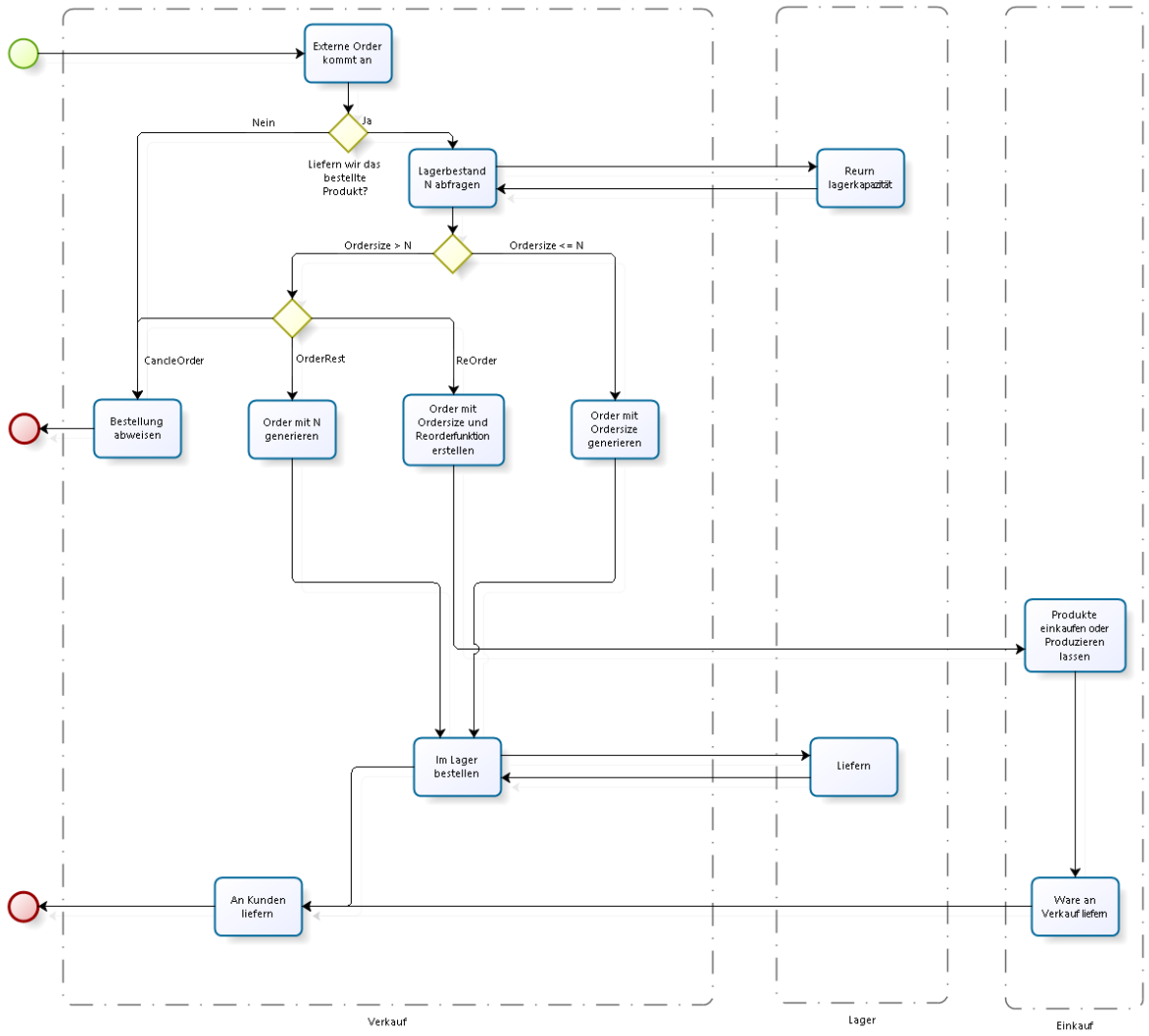


Abbildung 20 - Selling Prozessdiagramm

PROZESSSTRUKTUR UND EINGLIEDERUNG DER UNTERKOMPONENTEN

In Abbildung 20 ist der Prozessablauf in der Komponente `Selling` abgebildet. Ausgelöst wird der Ablauf durch eintreten einer `Order`. Als erstes wird in der `TabelDecider` Unterkomponente entschieden, ob das bestellte Produkt verkauft wird (Abbildung 21, 1). Steht es nicht zu verkauft, wird die `Order` abgelehnt. Das Ablehnen von Bestellungen geschieht in diesem ersten Schritt der Implementation durch das zurücksenden einer leeren `Order`. Wenn das bestellte `Product` verkauft wird, sendet die `Selling` Komponente einen `Request` ans Lager. Ist das `Product` in ausreichender Anzahl im Lager vorhanden, sendet die `Selling` Komponente eine Bestellung. Das Lager fügt die bestellten `Products` an die `Order` an und gibt diese zu `Selling` Komponente zurück, wo sie an den Kunden weitergeleitet wird (Abbildung 21, 3).

Für den Fall das der Lagerbestand zu niedrig ist, wird nach der `DeliverPriority` weiterentschieden. Die `DeliverPriority` ist ein `Property` der `Order`. Es bestimmt, wie wichtig die Bestellung ist und wie damit weiter verfahren werden soll.

DELIVERPRIORITY CANCELORDER

Ist die Bestellung vom Typ `CancelOrder`, wird sie bei zu niedrigem Lagerbestand abgewiesen. Der Kunde möchte die komplette Lieferung entweder sofort oder gar nicht. (Abbildung 21, 4)

DELIVERPRIORITY RESTORDER

Bei der Priorität `RestOrder` soll der restliche Lagerbestand geliefert werden, auch wenn er nicht der `OrderSize` entspricht. In diesem Fall schickt die `Selling` Komponente eine Bestellung ans Lager, um den gesamten restlichen Lagerbestand zu bestellen, den der `Request` zuvor ermittelt hat. Die `Order` wird dann mit weniger Produkten als bestellt an den Kunden zurückgeschickt. (Abbildung 21, 5)

DELIVERPRIORITY REORDER

Bei der Priorität `ReOrder` benötig der Kunde das Produkt unbedingt und nimmt eine längere Wartezeit in Kauf. Die `Selling` Komponente sendet in diesem Fall eine Bestellung zum `to Purchase`-Ausgang. Dort wird das Produkt entweder extern eingekauft oder produziert. Über die `Selling` Komponente wird die Lieferung dann an den Kunden weitergeleitet. (Abbildung 21, 6)

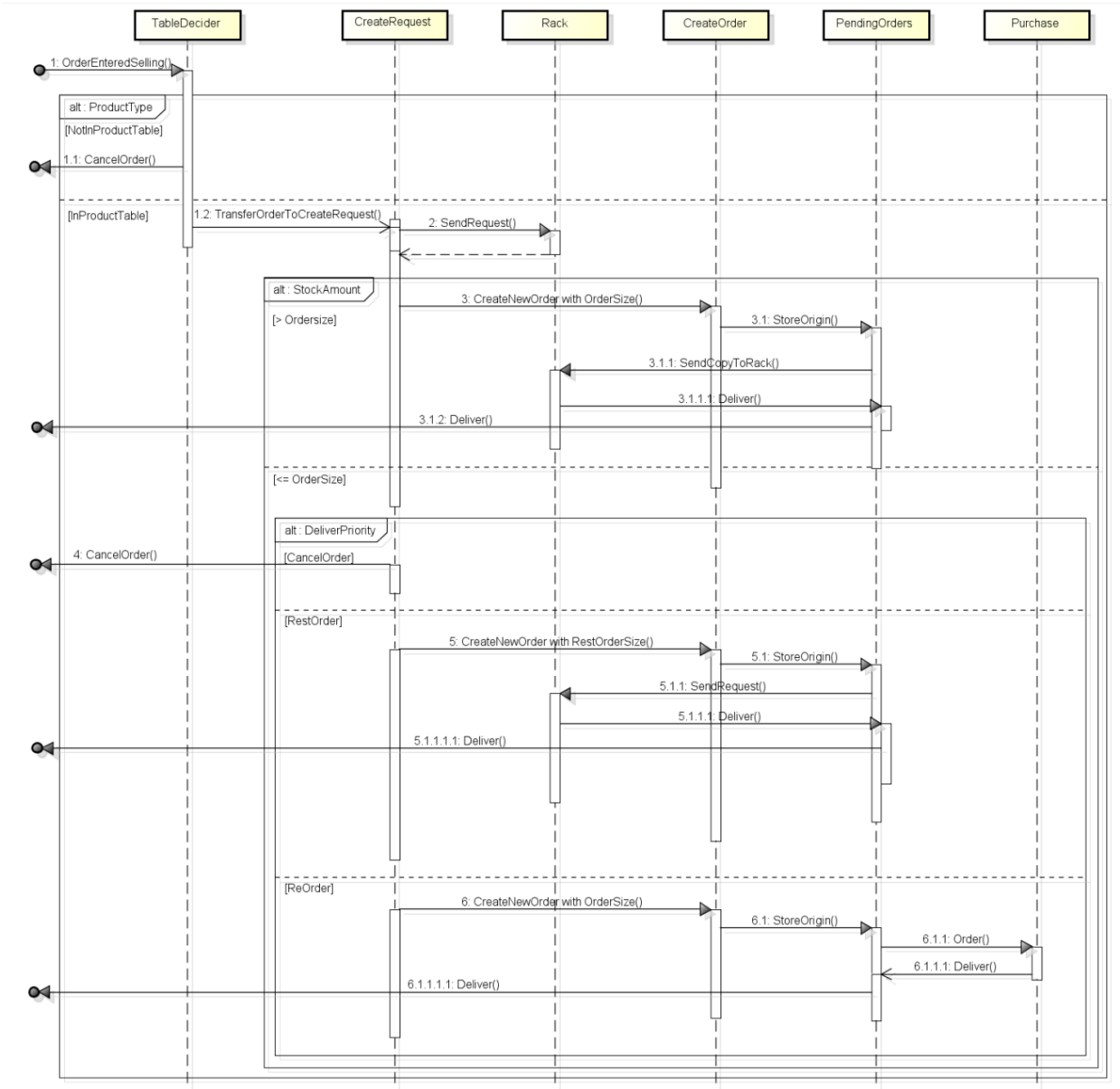


Abbildung 21 - Sequenzdiagramm der Komponente Selling

PROPERTIES UND EINSTELLUNGEN

PROPERTIES

Name	Typ	Beschreibung
`Address`	`String`	Im `Address Property` wird die Adresse der Komponente gespeichert. Diese Adresse wird in allen von der Komponente erstellten `Orders` gespeichert. Das `Property` dient dem `Routing` zwischen den Komponenten. Es muss nicht gesetzt werden, wenn kein `Routing` verwendet wird.
`ProductTable`	`RepetingGroup` vom Typ `Selling.ProductTable`	Die `ProductTable` wird der Komponente `TableDecider` übergeben. In diese Tabelle werden alle von der `Selling` Komponente zum Verkauf stehenden Produkte gespeichert. Die Tabelle muss übergeben werden. Wird sie leer übergeben, werden alle Bestellungen abgelehnt.

Tabelle 14 - Selling Properties

ANWENDUNG

Die Komponente `Selling` nimmt nur `Entities` vom Typ `Order` entgegen. Die an `OrderIn` angeschlossene Source oder Komponente muss zwingend `Orders` versenden. An den Anschlüssen `toRack` und `toPurchse` können ein `Rack` und eine `Purchase` Komponente angeschlossen werden (Siehe Abbildung 22).

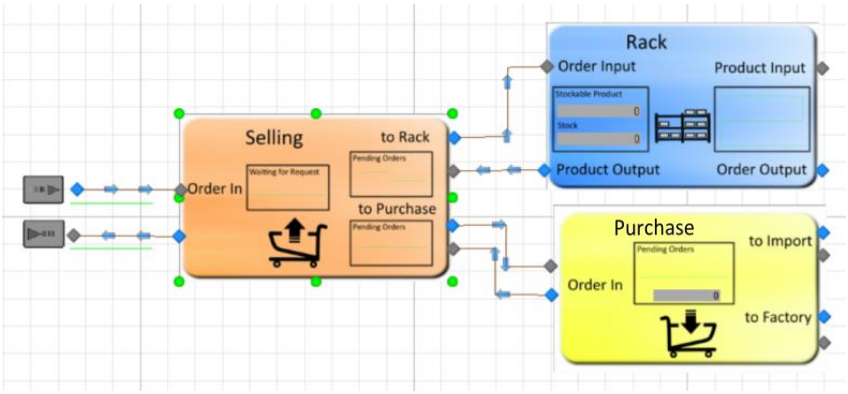


Abbildung 22 - Selling Variante 1

Es besteht auch die Möglichkeit, kein `Rack` anzuschliessen. In diesem Fall sind der Ausgangs- und der Eingangsknoten direkt miteinander zu verbinden (Abbildung 23). Ein ausgehender `Request` wird so direkt wieder zurückgeleitet und zeigt den Lagerbestand 0 an. Wichtig ist hier, dass keine Komponente angehängt wird, die nicht mit dem `Entity Request` umgehen kann. Am `toPurchase` Anschluss kann jede beliebige andere Komponente angeschlossen werden.

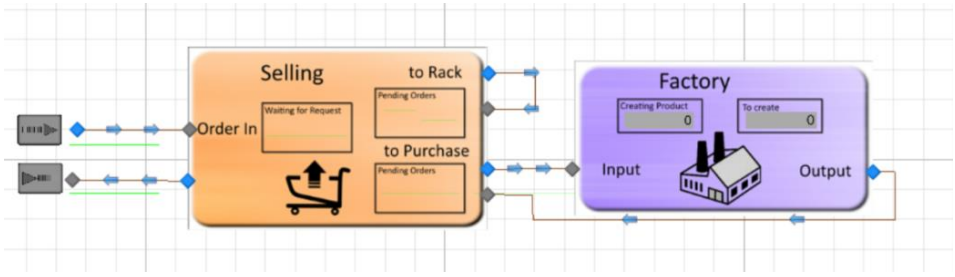


Abbildung 23 - Selling Variante 2

PURCHASE

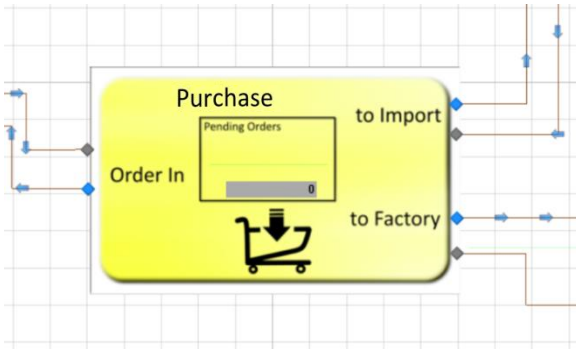


Abbildung 24 - Purchase

BESCHREIBUNG

Die Komponente `Purchase` ist die Schnittstelle zu externen Verkäufern. Sie nimmt interne Bestellungen entgegen und entscheidet, ob das bestellte Produkt eingekauft oder produziert werden soll. Der Anschluss an die internen Komponenten bilden die `OrderIn`-Knoten. Über den `toImport`-Anschluss gehen Bestellungen und Lieferungen an und von externen Verkäufern. Über den `toFactory`-Anschluss ist der Weg für Bestellungen und Lieferungen zur eigenen Produktion.

Die Komponente `Purchase` nimmt als `Properties` zwei Tabellen entgegen. Anhand dieser Tabellen wird entschieden, ob ein Produkt eingekauft oder selbst produziert wird.

PROZESSSTRUKTUR UND EINGLIEDERUNG DER UNTERKOMPONENTEN

Der Prozess in der `Purchase` Komponente beginnt mit dem Eintreten eines `Order Entity's`. Die `Order` wird von eine `CopyOrder` Komponente entgegengenommen, kopiert (Abbildung 25, 1) und an die `PendingOrders` Komponente weitergegeben. Anschliessen wird im ersten `TabelDecider` entschieden, ob das bestellte `Product` produziert wird oder nicht. Steht das `Prduct` auf der `FactoryTable`, wird die Order zur `Factory` weitergeleitet. Andernfalls geht es weiter zum zweiten `TableDecider`. Hier wird der `ProductType` der Order in der `ImportTable` gesucht. Steht das `Product` auf der `ImportTable`, wird die `Order` zum externen Lieferanten weitergeleitet. Andernfalls geht sie zurück zur `PeningOrder` Komponente (Abbildung 25, 3).

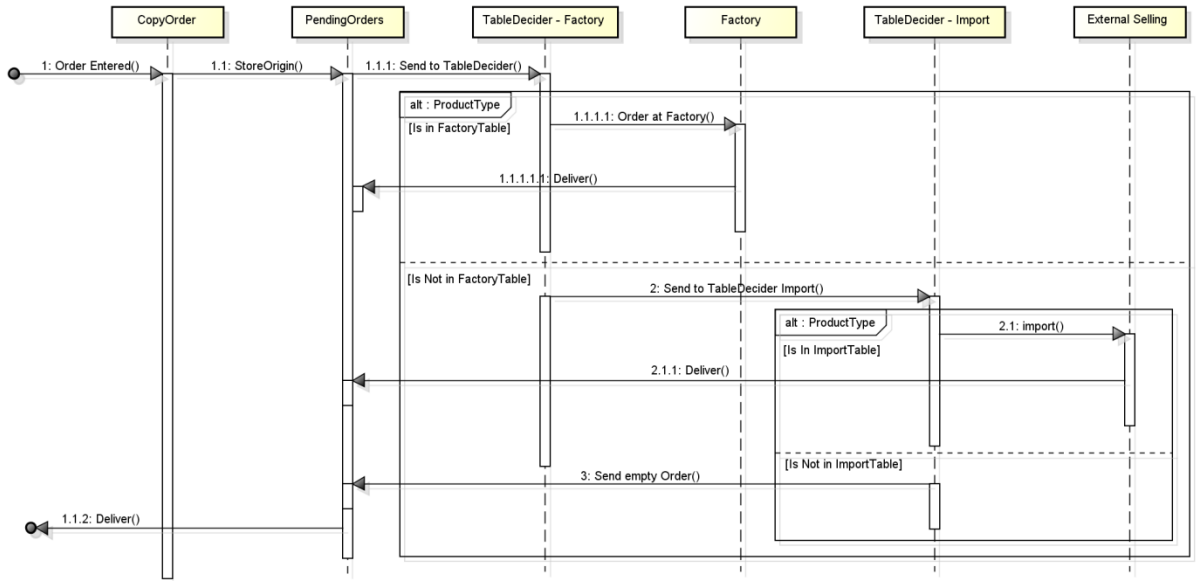


Abbildung 25 - Sequenzdiagramm der Komponente Purchase

Wie bei der `Selling` Komponente, ist das Ablehnen einer `Order` durch das Zurücksenden der leeren `Order` dargestellt. In diesem ersten Implementationsschritt wird das Ereignis einer abgelehnten Bestellung noch nicht abgefangen. Diese Erweiterung kann zur Zeit mit den Komponenten der Standartlibrary hinzugefügt werden. Für weitere Ausarbeitungen der Library wäre die Erweiterung in der `Purchase` Komponente im Prozess „Lieferanten Suchen“ zum implementieren (Abbildung 26).

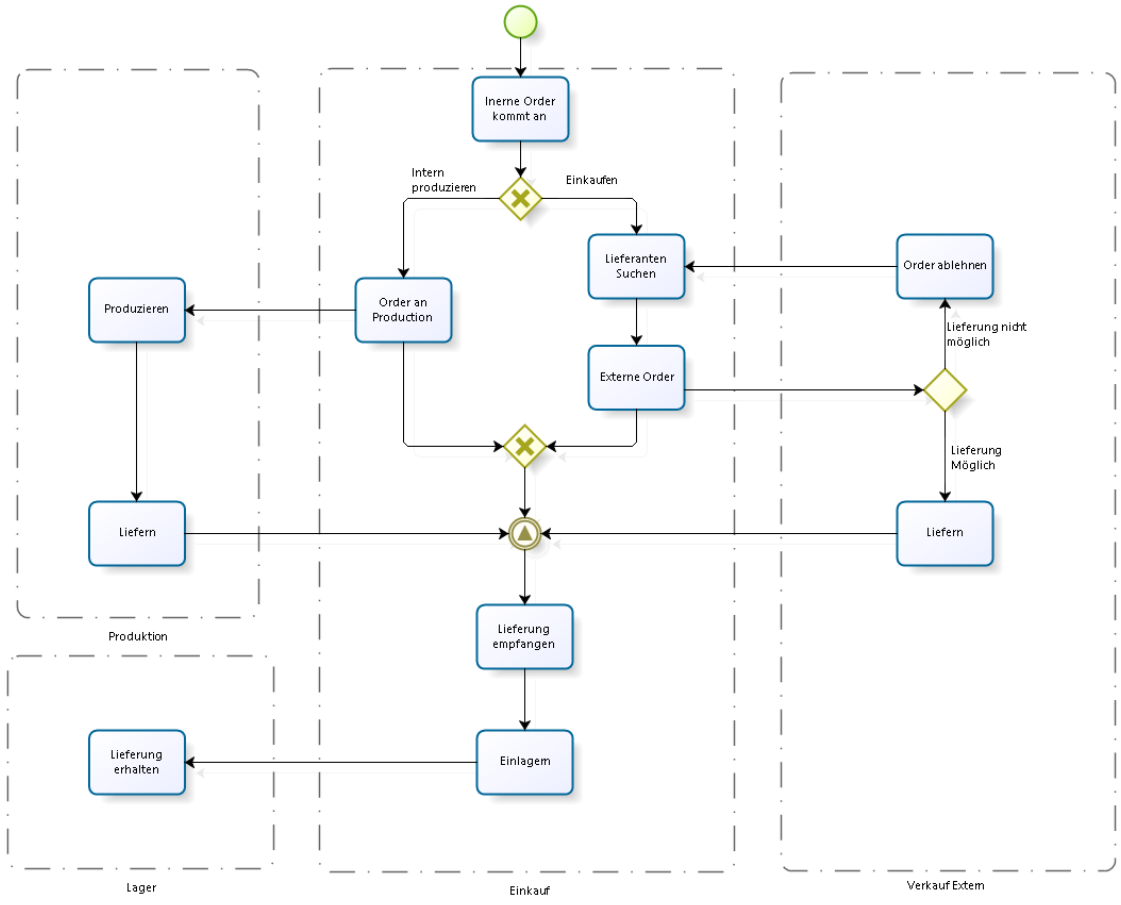


Abbildung 26 - Purchase Prozessdiagramm

PROPERTIES UND EINSTELLUNGEN

PROPERTIES

Name	Typ	Beschreibung
`Address`	`String`	Im `Address Property` wird die Adresse der Komponente gespeichert. Diese Adresse wird in allen von der Komponente erstellten Orders gespeichert. Das Property dient dem Routung zwischen den Komponenten. Es ist nicht gesetzt werden, wenn kein Rounting verwendet wird.
`FactoryTable`	`RepetingGroup` vom Typ `Purchase.FactoryTable`	Die `FactoryTable` wird der ersten TableDecider Komponente übergeben. In diese Tabelle werden alle intern produzierten Produkte gespeichert. Die Tabelle muss übergeben werden und mindestens ein Product enthalten.
`ImportTable`	`RepetingGroup` vom Typ `Purchase.ImportTable`	Die `ImportTable` wird der ersten TableDecider Komponente übergeben. In diese Tabelle werden alle intern produzierten Produkte gespeichert. Die Tabelle muss übergeben werden und mindestens ein Product enthalten.

Tabelle 15 - Purchase Properties

ANWENDUNG

An den `OrderIn` Anschluss können alle `Orders` versendenden Komponenten angeschlossen werden. Typischerweise wären dies `Rack` oder `Selling` Komponenten. An die Anschlüsse `toImport` und `toFactory` können alle beliebigen Komponenten angeschlossen werden. Einzige Voraussetzung ist, dass sie das `Entity Order` entgegen nehmen. `Orders` deren `ProductType` in der `ImportTable` befindet, werden zum `toImport` Anschluss weiter geleitet. Ebenso verhält es sich mit der FactoryTable und dem „to Factory“ Anschluss.

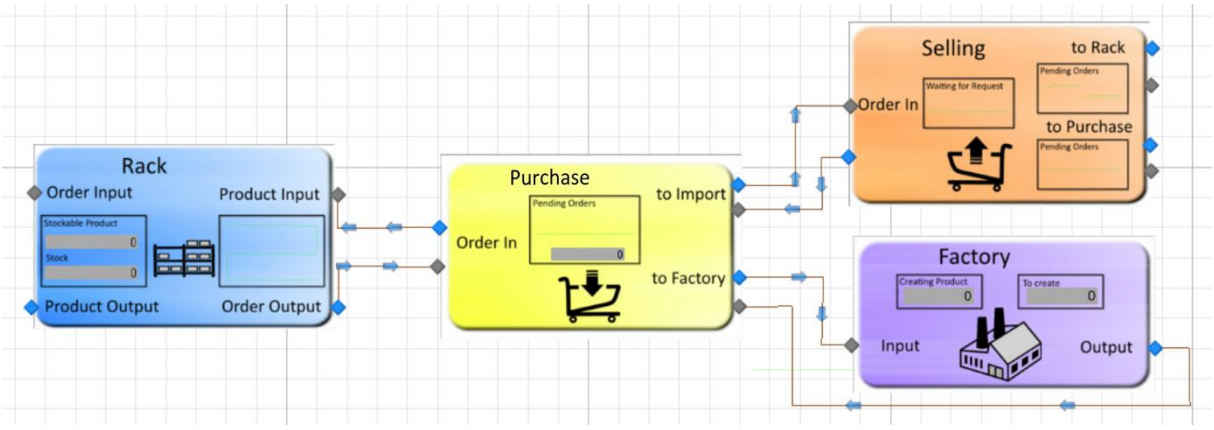


Abbildung 27 - Purchase Einsatzvariante

RACK

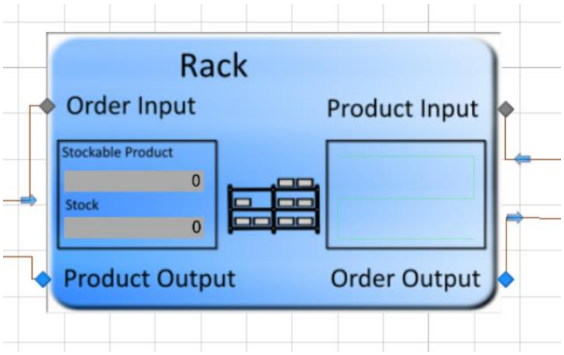


Abbildung 28 - Rack

BESCHREIBUNG

Das `Rack` ist für die Lagerung eines spezifischen `Products` 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 spezifischer Lagerbestand 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, 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.

PROZESSSTRUKTUR

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.

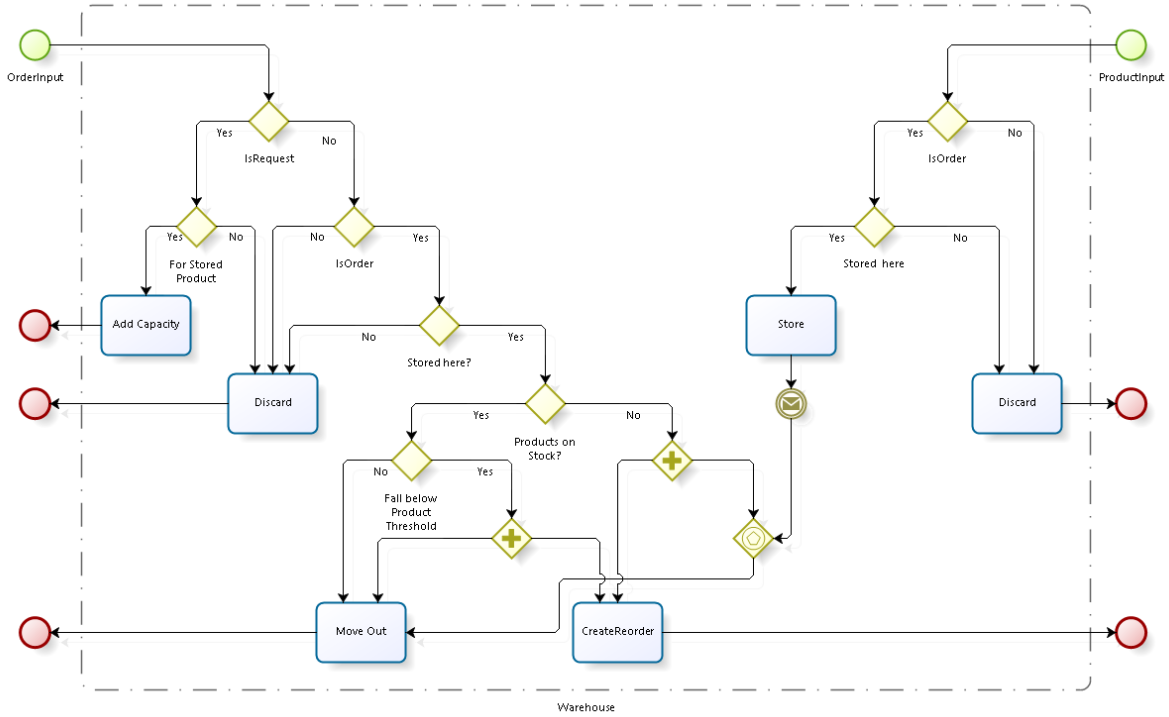


Abbildung 29 - Prozesse Rack

Jedes eingehende `Entity` wird als erstes auf seinen `EntityType` geprüft. Ist es ein `Request` für den gespeicherten `ProductType`, wird der aktuelle Lagerbestand im State `ActualStockSize` eingetragen und der `Request` zum `ProductOutputNode` geschickt. Bei einer `Order` wird ebenfalls der `ProductType` mit dem `StockableProduct` verglichen. `Orders` und `Requests` deren `ProductType` nicht in diesem `Rack` gespeichert wird, sowie alle anderen `Entities` werden direkt weitergegeben.

Bei `Orders` mit dem gleichen `ProductType` wird geprüft, ob der Lagerbestand gross genug ist. Ist dies der Fall, werden die bestellten `Products` an die `Order` angehängt und diese an die bestellende Komponente zurückgesendet. Ist der Lagerbestand zu klein, oder fällt nach der Bestellung unter den Mindestbestand, so wird eine Nachbestellung ausgelöst. Die Nachbestellung wird über den `OrderOutputNode` an die Einkaufskomponente geschickt.

Am `ProductInputNode` eingehende Orders, deren `Products` im betreffenden `Rack` gespeichert werden, wird von der Lieferung getrennt und zerstört. Die Lieferung wird eingelagert. Wartende Bestellungen für die zu wenig `Products` im `Rack` waren werden nun weiter abgearbeitet.

Eine ausführliche Beschreibung der Rack Prozesse befindet sich im Anhang.

PROPERTIES UND EINSTELLINGEN

PROPERTIES

Name	Typ	Beschreibung
`Address`	`String`	Im `Address Property` wird die Adresse der Komponente gespeichert. Diese Adresse wird in allen von der Komponente erstellten Orders gespeichert. Das Property dient dem Routung zwischen den Komponenten. Es ist nicht gesetzt werden, wenn kein Rounting verwendet wird.
`StockableProduct`	`Entity Reference`	Durch dieses `Entity Property` wird definiert, welcher `Entity` Typ in der `Shelf Station` gelagert werden kann.

MoveOutTime	Expression	Dieses Expression Property erlaubt dem Benutzer die Anpassung der benötigten Zeit zur Auslagerung eines Products aus der Shelf Station. Unit Type: Time Default Units: Seconds Required Value: True
StoreTime	Expression	Dieses Expression Property erlaubt dem Benutzer die Anpassung der benötigten Zeit zur Einlagerung eines Products in die Shelf Station.
OrderOnInitialisation	Boolean	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.
ReOrderSize	Expression	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. ein Integer. Default Value: Random.Discrete(5, 1)
MinStoreAmount	Integer	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.
MaxStoreAmount	Integer	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.
DeliverPriority	List Property	Dieses List Property spezifiziert die Priorität der Orders welche in diesem Rack erzeugt werden. Wird in diesem Rack eine Order oder Reorder erzeugt, wird der Wert dieses Properties als DeliveryPriority gesetzt. Die möglichen Werte werden aus der DeliveryPriorityList entnommen.

Tabelle 16 - Rack Properties

ANWENDUNG

Das `Rack` wird je an eine bestellende und eine liefernde Komponente angeschlossen (Abbildung 30). Auf der einen Seite nimmt es Bestellungen entgegen und liefert Produkte, auf der anderen Seite versendet es Bestellungen und nimmt Produkte entgegen um diese einzulagern.

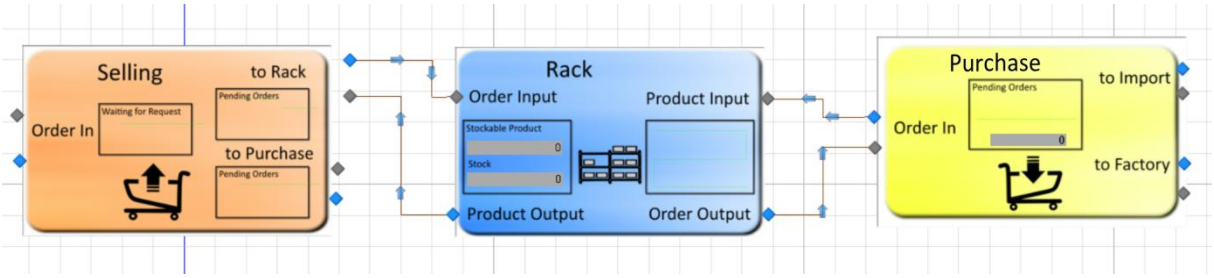


Abbildung 30 - Rack Variante 1

In Abbildung 31 wird ersichtlich, dass die Reihenschaltung der `Racks` von ausserhalb gleich benutzbar ist wie ein einzelnes `Rack`. `Entities` welche nicht für ein `Rack` bestimmt sind, werden weitergeleitet und gar nicht erst in den `InputBuffer` verschoben.

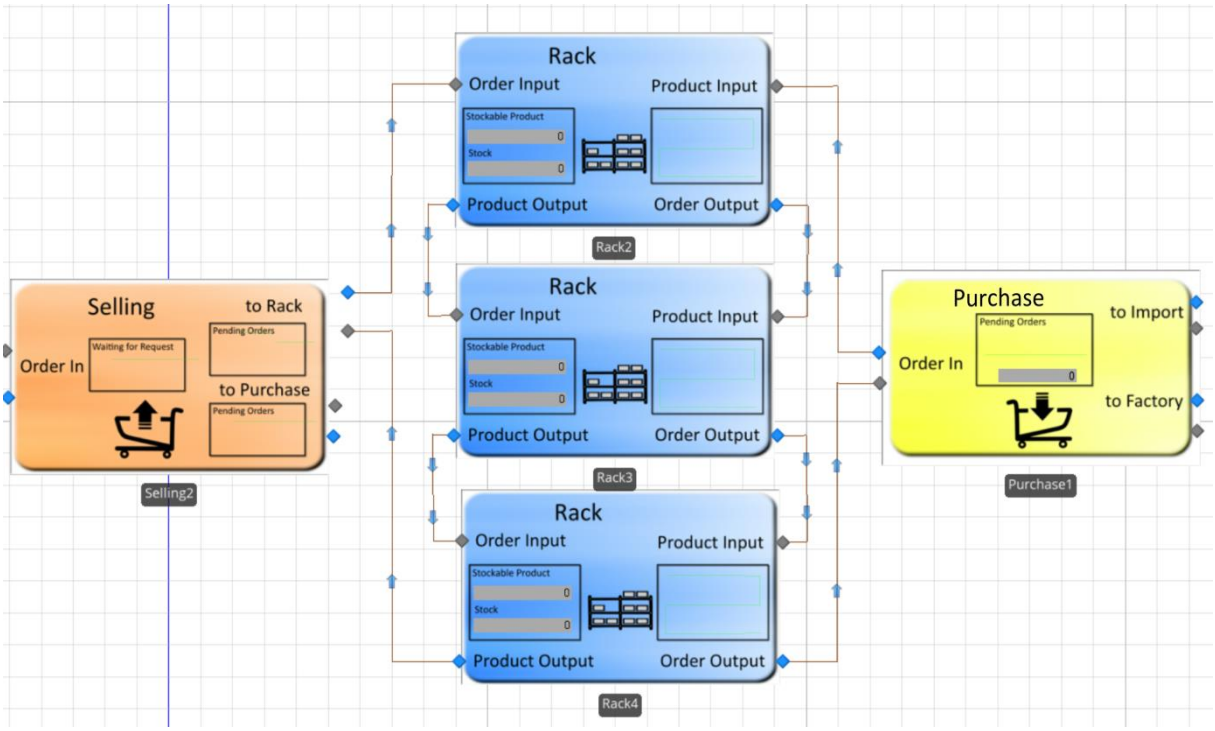


Abbildung 31 - Rack Variante 2

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 kleinsten 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 durch die 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.

AUSWERTUNG & STATISTIKEN

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.

FACTORY

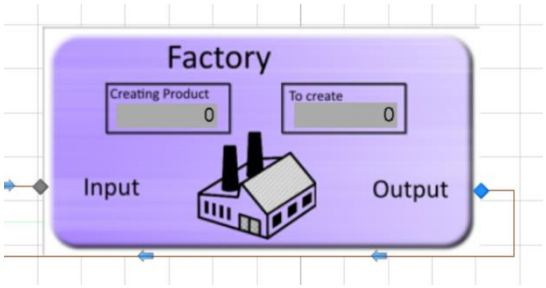


Abbildung 32 - Factory

BESCHREIBUNG

Die Factory hat die Aufgabe, bestellte Produkte herzustellen. Dazu werden `ProductType` und `OrderSize` aus der `Order` ausgelesen. Aus dem `OutputNode` auf der rechten Seite wird die `Order`, kombiniert mit der gewünschten Anzahl `Entities` zurück ins System transferiert. Das Bestellen und Konsumieren von Rohstoffen wurde in diesem ersten Schritt noch nicht implementiert. Diese Funktionalität kann der Komponente zum Beispiel mit Hilfe der Unterkomponenten `CopyOrder`, `PendingOrders` und `TableDecider` hinzugefügt werden.

STRUKTUR

INTERN

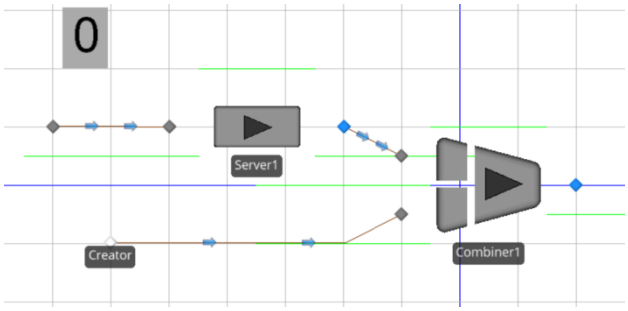


Abbildung 33 - Factory interne Ansicht

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

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.

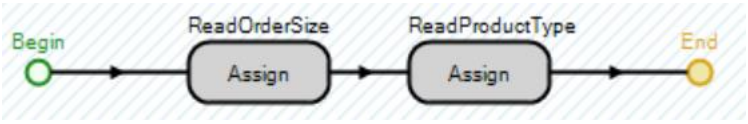


Abbildung 34 - ReadOrder Prozess

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.

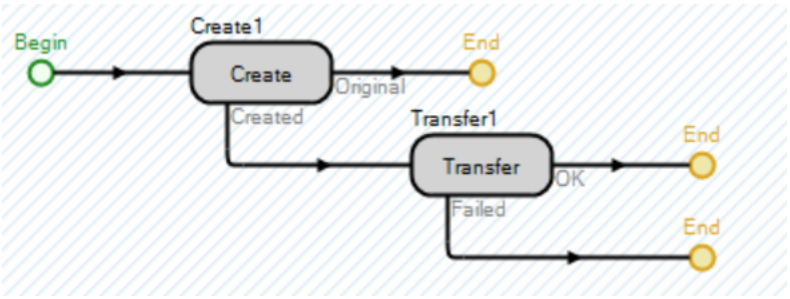


Abbildung 35 - Creation Prozess

ANWENDUNG

Die Factory Komponente wird an eine Order versendende Komponente angeschlossen. Diese können eine Selling, Purchase oder Rack Komponente sein.

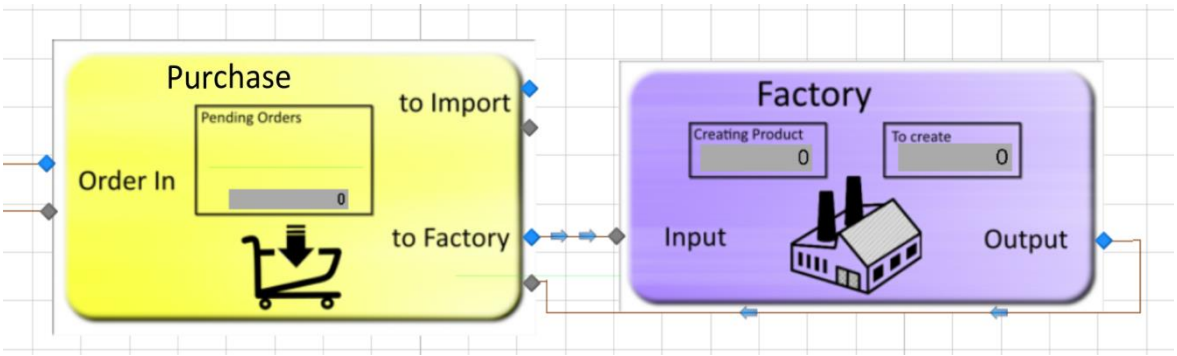


Abbildung 36 - Factory Anschlussvariante

PROTOTYPE – LITTLE BEERGAME

Zur Demonstration der erstellten Library wurde eine kleine Version des Beergames (Spiel zur Veranschaulichung von Supply Chains) erstellt. Dieses Spiel kam bereits vor Abgabe dieser Bachelorarbeit erfolgreich bei Schulungen zum Supply Chain Management zum Einsatz. Da diese Schulungen für ein internationales Publikum gehalten wurden, sind der Prototype sowie die Aufgaben dazu, in Englisch beschrieben.

WECOME TO THE LITTLE BEERGAME!

The Little Beergame is all about beer. You have a supply chain with six members. They all produce, buy or consume beer. You want to find out, how they can do it with the highest efficiency.

There are two kind of Beer in this example, the PaleAle and the LagerBeer.

THE MEMBERS

PUB

The Pub serves LagerBeer and PaleAle. It buys them at the ReSeller. You can change the InterArrivalTime of the Orders and the OrderSize of both beers.

BAR

The Bar serves LagerBeer and PaleAle as well. It buys them at the ReSeller. You can change the InterArrivalTime of the Orders and the OrderSize of both beer.

RESTAURANT

The Restaurant just serves LagerBeer. It buys it directly at the BigBrewery. You can change the InterArrivalTime of the Order and the OrderSize.

RESELLER

The ReSeller sells LagerBeer and PaleAle to the Bar and the Pub. It buys both beer from the BigBrewery. The ReSeller has a Rack for each beer. You can change the ReorderSize and the minimum StoreAmount of each Rack.

BIGBREWEY

The BigBrewery sells LagerBeer and PaleAle. It buys the PaleAle from the SmallBrewery and produces the LagerBeer for them self in it's two Factories. You can change the ProcessingTime of both Factories.

SMALLBREWERY

The SmallBrewery is selling PaleAle which is produced in their own Factory. You can change the ProcessingTime of the Factory.

EXERCISE

As default the order sources of Pub, Bar and Restaurant are joined to the interactive buttons on their left. The InterArrivalTimes has no influence. You can play with the changeable values and the buttons, to get a feeling for the flows within the supply chain.

If you want to run the model automatically or start an experiment, change the ArrivalMode of the sources from OnEvent to InterarrivalTime.

Use the Experiments function of Simio to Optimize the values of interest.

VARIABLE PARAMETTERS

You can adjust the system with the changeable values described before.

Reorder Sizes	
ReOrderSize_Rack1_RS	ReOrdersize of the ReSellers Rack 1
ReOrderSize_Rack2_RS	ReOrdersize of the ReSellers Rack 2
Minimum Store Amount	
MinStoreAmount_Rack1_RS	Minimum Store Amount of the ReSellers Rack 1
MinStoreAmount_Rack2_RS	Minimum Store Amount of the ReSellers Rack 2
Factory Processing Time	
ProcessingTime_Factory1_BB	Processing Time of the BigBrewery's Factory 1
ProcessingTime_Factory2_BB	Processing Time of the BigBrewery's Factory 2
ProcessingTime_Factory_SB	Processing Time of the SmallBrewery's Factory
Order InterArrival Times	
InterarrivalTime_PA_Bar	InterarrivalTime of the Bar's PaleAle Order
InterarrivalTime_LB_Bar	InterarrivalTime of the Bar's LagerBeer Order
InterarrivalTime_PA_Pub	InterarrivalTime of the Pub's PaleAle Order
InterarrivalTime_LB_Pub	InterarrivalTime of the Pubs's LagerBeer Order
InterarrivalTime_LB_Res	InterarrivalTime of the Restaurants's LagerBeer Order
Order Sizes	
OrderSize_PA_Bar	OrderSize of the Bar's PaleAle Order
OrderSize_LB_Bar	OrderSize of the Bar's LagerBeer Order
OrderSize_PA_Pub	OrderSize of the Pub's PaleAle Order
OrderSize_LB_Pub	OrderSize of the Pubs's LagerBeer Order
OrderSize_LB_Restaurant	OrderSize of the Restaurants's LagerBeer Order

Tabelle 17 – Little Beergame Variable Raremetters

VALUES OF INTEREST

Now we want to know how efficient our supply chain is. In this Little Beergame we want to spectate the utilization of the Racks and the Factories and never the last the "SCOR metric RS.1.3 Order fullfillment Cycle Time" for the Beer-deliveries.

In the Beergame Simio Model we have different Properties to show these values:

Utilisation of the Racks	
Util_Rack1_RS	Utilizsation of the ReSellers Rack1
Util_Rack2_RS	Utilizsation of the ReSellers Rack2
Utilization of the Factories	
Util_Factory1_BB	Utilizsation of the BigBrewery's Factory1
Util_Factory2_BB	Utilizsation of the BigBrewery's Factory2
Util_Factory_SB	Utilizsation of the SmallBrewery's Factory
Fullfillment Cycle Time of the LagerBeer	
DeliverTime_Pub_LB	DeliverTime of the LagerBeer ordered from the Pub
DeliverTime_Bar_LB	DeliverTime of the LagerBeer ordered from the Bar
DeliverTime_Res_LB	DeliverTime of the LagerBeer ordered from the Restaurant
Fullfillment Cycle Time of the LagerBeer	
DeliverTime_Pub_PA	DeliverTime of the PaleAle ordered from the Pub
DeliverTime_Bar_PA	DeliverTime of the PaleAle ordered from the Bar

Tabelle 18 – Little Beergame Values of Interest

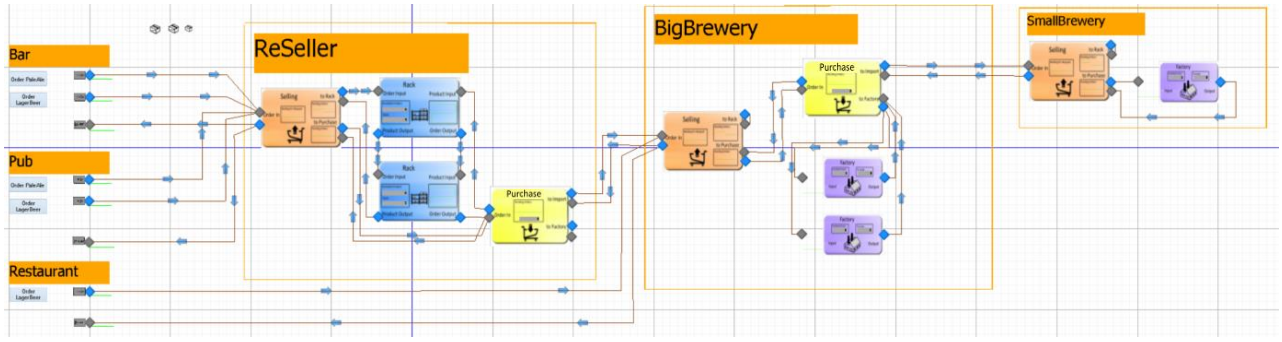


Abbildung 37 – Little Beergame komplett

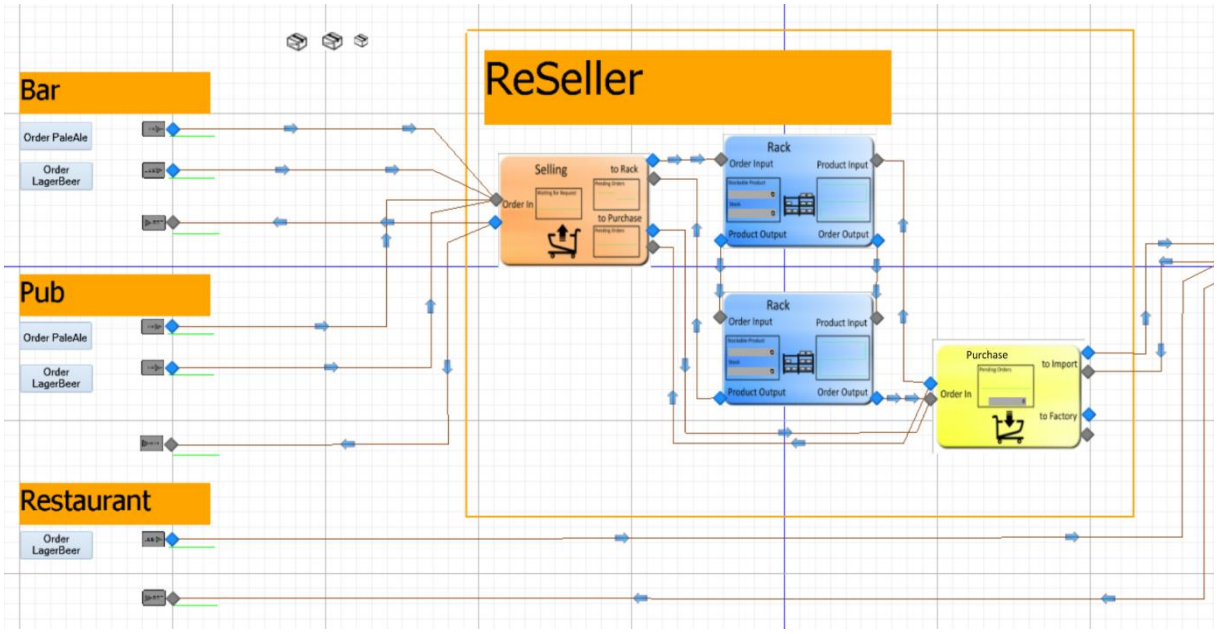


Abbildung 38 - Little Beergame Kunden und ReSeller

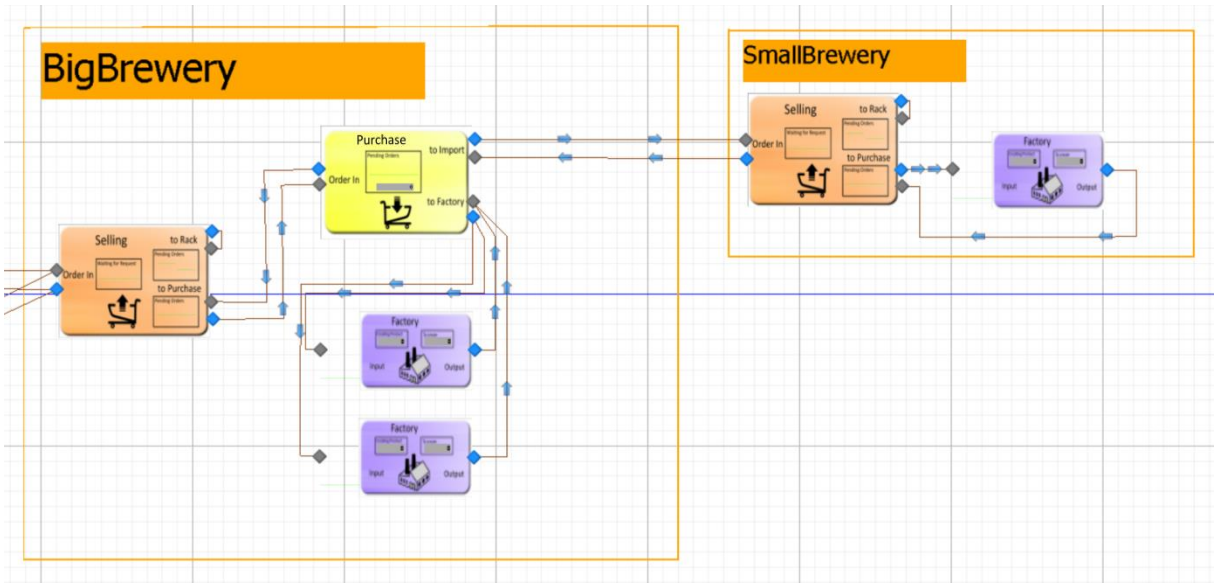


Abbildung 39 – Little Beergame kleine und grosse Brauerei

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 untergeordnete Rolle spielen, fällt der Entscheid auf das manuelle Testen.

In den Manuellen Tests wird eine künstliche Simulationsumgebung geschaffen. Die Komponenten sind isoliert voneinander. So kann das verhalten jeder Komponente einzeln Beobachtet werden. Diese Art zu testen hat sich sehr bewährt und die Entwicklung von Anfang an unterstützt. Die vier Testumgebungen sind nachfolgend einzeln beschrieben.

RACK TESTUMGEBUNG

BESCHREINUNG

Für die Umsetzung der manuellen Testumgebung wird ein neues Simio-Projekt namens `SCLib_Test` erstellt. In diesem Projekt wird die SCLib importiert und ein Model Namens `Rack_Test` 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 40 miteinander verbunden. Die Objekte werden wie folgt eingestellt.

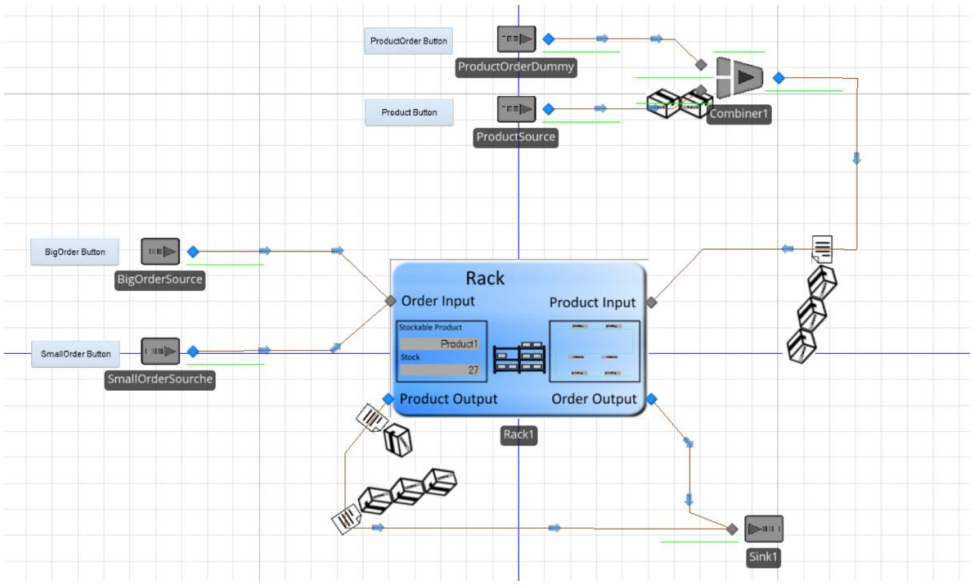


Abbildung 40 - Rack Testumgebung

ELEMENTE

Name	Property	Value
BigOrder	OrderSize ProductType	5 Product1
SmallOrder	OrderSize ProductType	1 Product1
DummyOrder	OrderSize ProductType	5 Product1
DummyOrderSource	EntityType EntityPerArrival	DummyOrder 1
BigOrderSource	EntityType EntityPerArrival	BigOrder 1
SmallOrderSource	EntityType EntityPerArrival	SmallEntity 3
Combiner1	BatchQuantity	Order.OrderSize

Tabelle 19 - Rack Testumgebung Elemente

FACTORY TESTUMGEBUNG

BESCHREIBUNG

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`.`Orders` können über einen `Button` erzeugt werden. 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`.

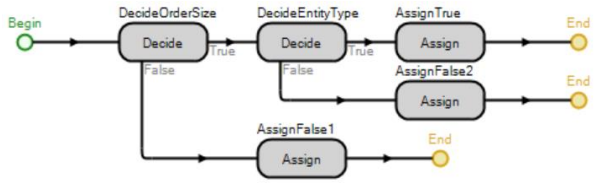


Abbildung 41 - Validierung

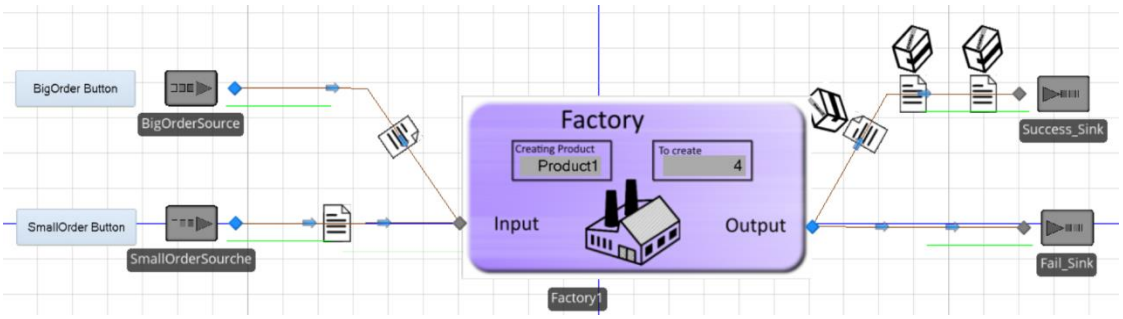


Abbildung 42 - Factory Testumgebung

ELEMENTE

Name	Property	Value
BigOrder	OrderSize ProductType	5 Product1
SmallOrder	OrderSize ProductType	1 Product2
BigOrderSource	EntityType EntityPerArrival	BigOrder 1
SmallOrderSource	EntityType EntityPerArrival	SmallEntity 3

Tabelle 20 - Factory Testumgebung Elemente

SELLING TESTUMGEBUNG

BESCHREIBUNG

Für die `Selling` Komponente wird ein Test Modell mit dem Namen `Selling_Test` erstellt. Am `OrderIn`-Anschluss werden zwei Sources und eine Sink angeschlossen. Der `ToRack`-Anschluss ist mit einem `Server` und einem `Combiner` verbunden. `Requests` werden zum `Server` geleitet, `Orders` zum `Combiner`. Der `Server` trägt in jedem `Request` einen Wert für den aktuellen Lagerbestand an. Dieser ist grösser als die `OrderSize` der `SmallOrder` und kleiner als die `OrderSize` der `BigOrder`. Der `toPurchase` Anschluss ist mit einem weiteren `Combiner` verbunden. Eine dritte `Source` (`ProductSource`) ist an die beiden `Member`-Anschlüsse der `Combiner` gehängt.

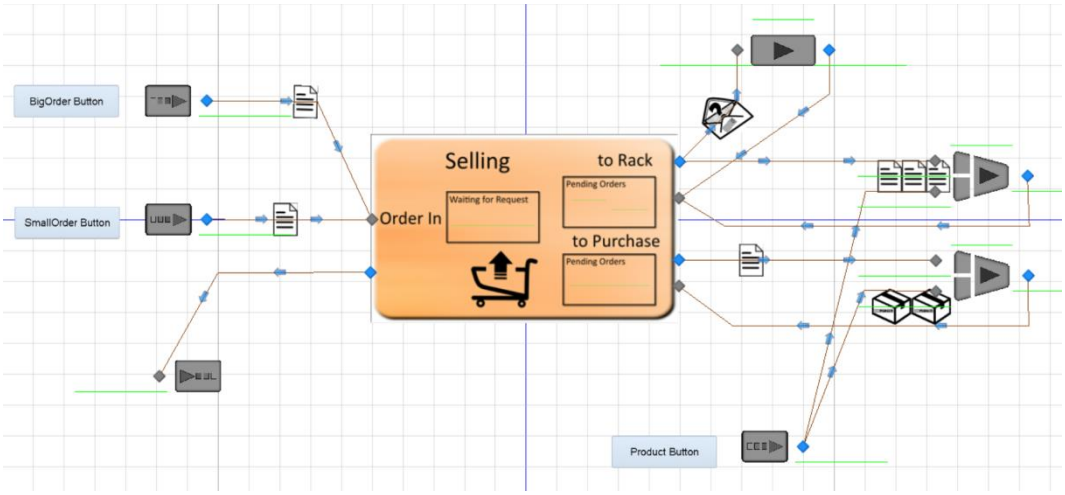


Abbildung 43 - Selling Testumgebung

ELEMENTE

Name	Property	Value
BigOrder	OrderSize ProductType	5 Product1
SmallOrder	OrderSize ProductType	1 Product1
BigOrderSource	EntityType EntityPerArrival	BigEntity 1
SmallOrderSource	EntityType EntityPerArrival	SmallEntity 3
Combiner1	BatchQuantity	Order.OrderSize
Combiner2	BatchQuantity	Order.OrderSize

Tabelle 21 - Selling Testumgebung Elemente

PURCHASE TESTUMGEBUNG

BESCHREIBUNG

Auch die `Purchase` Komponente wird in einem eigenen Modell getestet. Am `OrderIn`-Anschluss werden zwei `Sources` und eine `Sink` angehängt. Der `toImport`- und `toFactory`-Anschluss ist jeweils mit einem `Combiner` verbunden. Die `Member`-Anschlüsse der `Combiner` sind je mit einer `Source` verbunden. Beide stellen ein anderes `Product` her. Die beiden `Orders` bestellen diese unterschiedliche `Products`. Sie sollen über die Purchase Komponente zum richtigen `Combiner` geleitet werden.

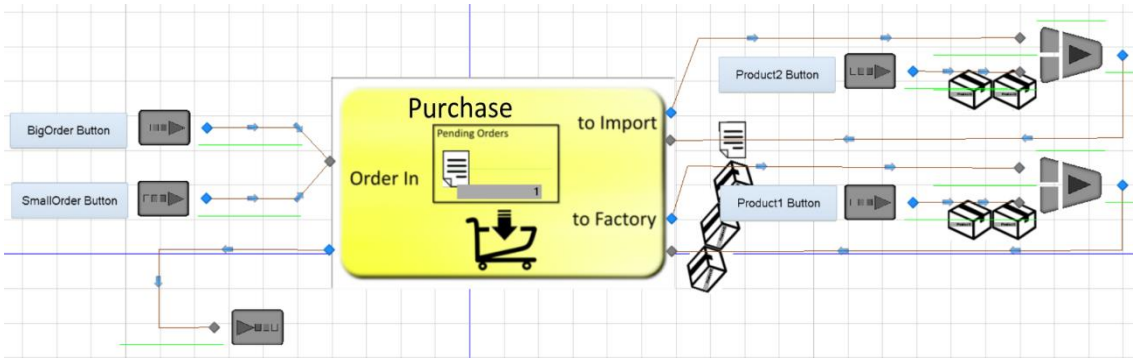


Abbildung 44 - Purchase Testumgebung

ELEMENTE

Name	Property	Value
BigOrder	OrderSize ProductType	5 Product2
SmallOrder	OrderSize ProductType	1 Product1
BigOrderSource	EntityType EntityPerArrival	BigEntity 1
SmallOrderSource	EntityType EntityPerArrival	SmallEntity 3
Combiner1	BatchQuantity	Order.OrderSize
Combiner2	BatchQuantity	Order.OrderSize
Product1Source	EntityType EntityPerArrival	Product1 1
Product2Source	EntityType EntityPerArrival	Product2 1

Tabelle 22 - Purchase Testumgebung Elemente

SCHLUSSFOLGERUNG

ERGEBNIS

Ergebnis dieser Arbeit ist eine Simulationsbibliothek für Simio. Die Bibliothek beinhaltet die Statischen Komponenten

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 Lieferpriorität.
Produkt (Product)	Speichert Herstellungszeitpunkt.
Request	Speichert Produkttyp und Lagermenge.

Mit diesen Komponenten lassen sich beliebige Supply Chains modellieren und nach ausgewählten SCOR Metriken bewerten.

Zur Demonstration der Library wurde eine vereinfachte Form des Beergames erstellt. Dieser Prototype kam bereits erfolgreich an Schulungen zum Thema Supply Chain Management zum Einsatz.

FAZIT

Das Ziel der Arbeit wurde mit einem zufriedenstellenden Ergebnis erreicht. Die erstellte Library ist aber noch stark ausbaufähig. Simulation von Supply Chains ist ein grosses Forschungsgebiet. Neueinsteigern in dieses Gebiet soll diese Arbeit eine Einstiegs Hilfe sein.

AUSBLICK

GELDFLUSS

Der Geldfluss und die Berechnung diverser Kosten (z.B. Lagerkosten) wären ein wichtiger Erweiterungspunkt für die SCLib. Eine Möglichkeit diese Erweiterung umzusetzen wäre das Hinzufügen weiterer Properties zu den bestehenden Komponenten.

OFFERTEN

Das Anfordern und Ausstellen von Offerten wäre vor allem bei komplexeren Supply Chains eine wichtige Funktion. Eine Möglichkeit diese Funktion zur SCLib hinzuzufügen wäre über eine Erweiterung der Request Komponente.

KONSUMIEREN VON ROHSTOFFEN

Das Anfordern und Konsumieren von Rohstoffen für die Produktion neuer Produkte wäre ein wichtiger Teil für die Komponente Factory. Eine Möglichkeit zur Umsetzung wäre eine Erweiterung der Komponente Factory um die Möglichkeit Bestellungen zu versenden. Ausserdem müssten die Produkte wissen, welche Rohstoffe sie benötigen.

ABGELEHNTE BESTELLUNGEN

Die Handhabung abgelehnter Bestellungen hängt mit der Möglichkeit Offerten zu stellen zusammen. Die anfragende oder bestellende Komponente muss in der Lage sein einen neuen Lieferanten auszusuchen, wenn eine Bestellung abgelehnt wurde.

ANHANG

PERSÖNLICHER BERICHT

Die Arbeit „Definition und Aufbau einer SCOR Simulationsbibliothek für Simio“ war für mich vor allem durch das Lernen im Umgang mit Simio geprägt. Zu Beginn der Arbeit konnte ich mir nicht vorstellen, wie viele Möglichkeiten Simio zur Verfügung stellt und welches Potential dahinter steht.

Ein sehr spannender Teil dieses Projektes war auch der konzeptionelle Teil. Ich habe hier viel über die Formulierung von Aktivitätsdiagrammen und dem Arbeiten mit der Modellierungssprache BPMN gelernt. Das Arbeiten am Konzept bestand aus vielen spannenden Diskussionen und Entwürfen, die uns Schritt für Schritt zu unserer Lösung führten.

Das Zusammenschliessen einer Studien- und einer Bachelorarbeit, hat auch meiner Sicht gut funktioniert. Die unterschiedlichen zeitlichen Anforderungen konnten durch entsprechende Aufgabenverteilungen ohne Probleme erfüllt werden. Zwischen den Teammitgliedern war stets eine enge Zusammenarbeit und gute Kommunikation vorhanden.

Ich bin mit den erreichten Ergebnissen der Arbeit zufrieden. Der erarbeitete Prototyp ist einsatzfähig und wurde bereits vor der Abgabe der Bachelorarbeit erfolgreich an einer Schulung zum Thema Supply Chain Management verwendet. Ich kann mir vorstellen in Zukunft weitere Projekte mit Simio zu machen und dort mein während der Bachelorarbeite gesammelten Wissen ein zu bringen.

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 Chin 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
Wrapper	Verpacker

ABKÜRZUNGEN

BPMN	Business Process Model and Notation
ID	Identifikation
MTO	Make to Order
MTS	Make to Stock
SCC	Supply Chain Council
SCLib	Supply Chain Library
SCM	Supply Chain Management
SCOR	Supply Chain Operation Reference
Usw.	Und so weiter
z.B.	Zum Beispiel

TABELLENVERZEICHNIS

BILDERVERZEICHNIS

RACK DETAILBESCHREIBUNG

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.

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.

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 45 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)

WORKING

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

Public: `False`

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`.

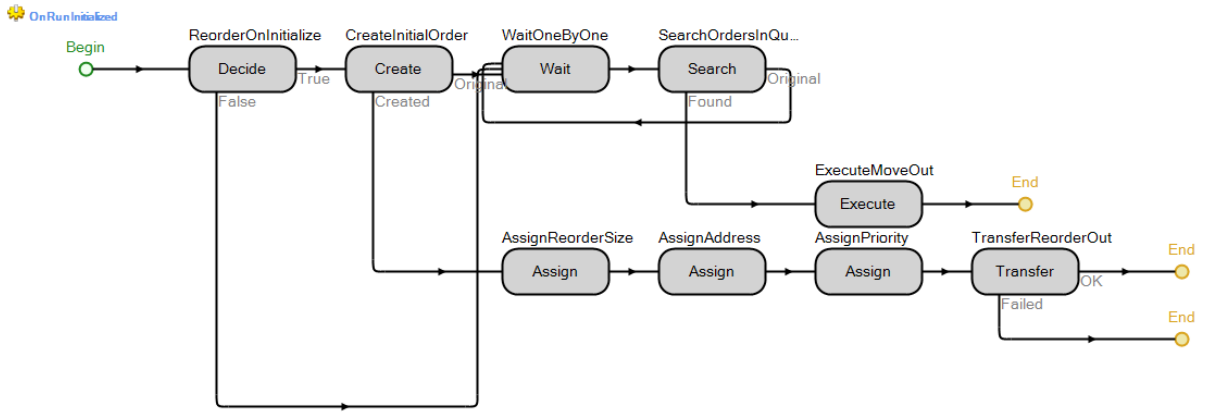


Abbildung 46 - OnRunInitialized Prozess

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.

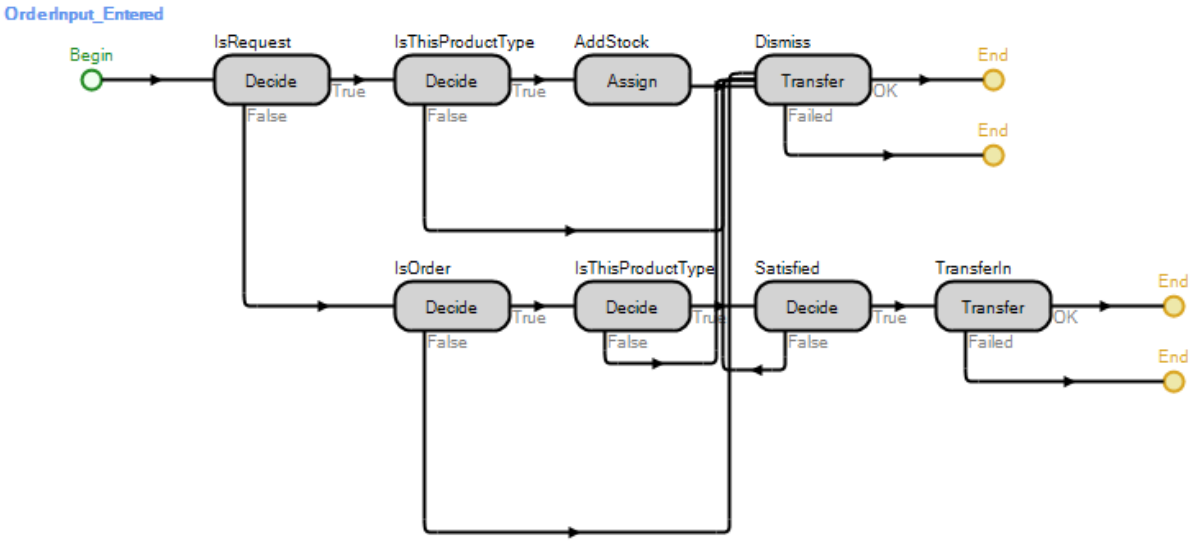


Abbildung 47 - OrderInput_Entered Prozess

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`.

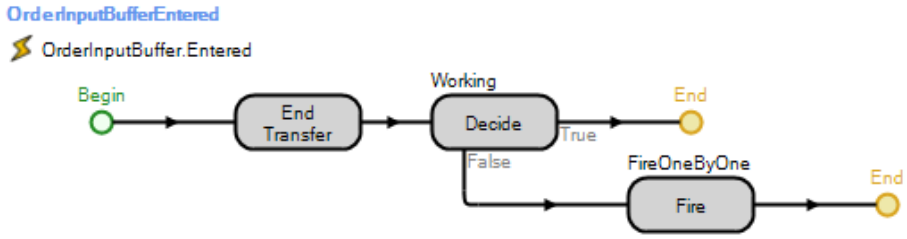


Abbildung 48 - OrderInputBufferEntered Prozess

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.

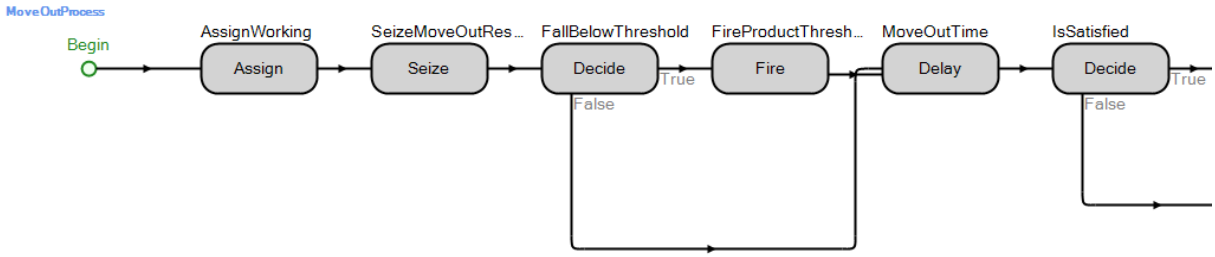


Abbildung 49 - MoveOutProcess Part1

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.

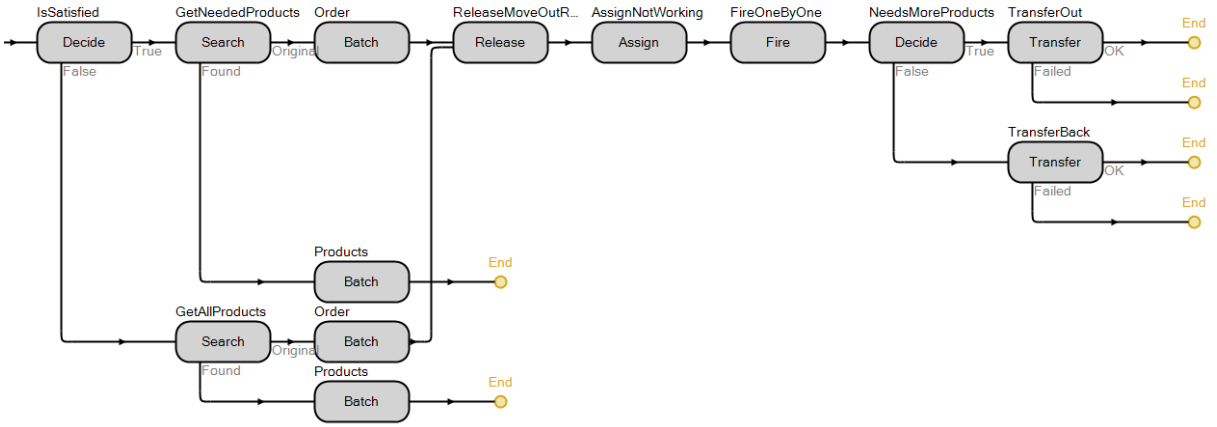


Abbildung 50 - MoveOutProcess Part2

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.

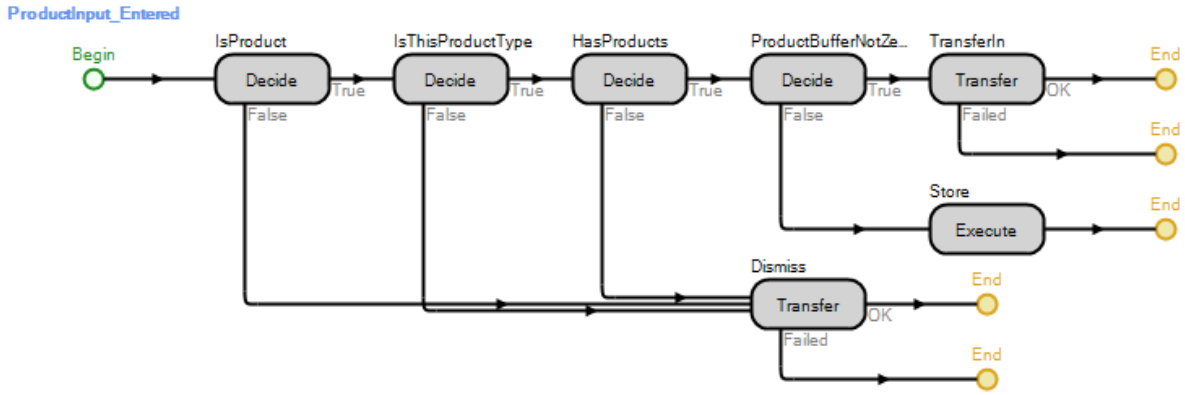


Abbildung 51 - ProductInput_Entered Prozess

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.

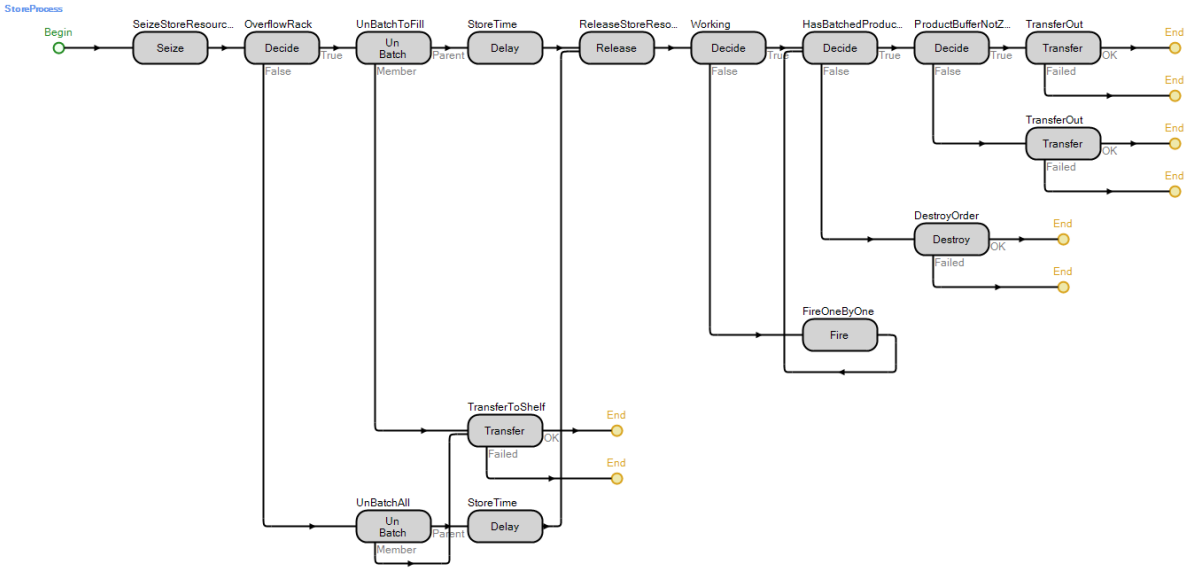
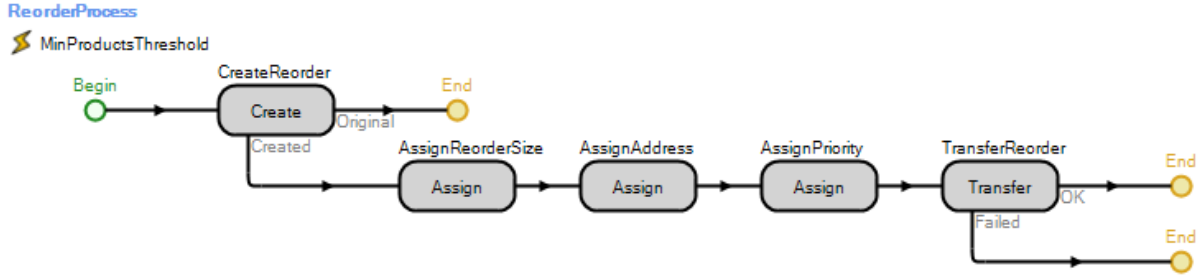


Abbildung 52 - Store Prozess

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 53 - 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.

PROJEKTPLAN

TEAM, ROLLEN, VERANTWORTLICHKEITEN

TEAMMITGLIEDER

Olivia Müller smueller@hsr.ch

Ernst Füllemann efuellem@hsr.ch

BETREUER

Prof. Dr. Andreas Rinkel, Professor an der HSR und Institutsparter des ITA (Institut für Internet-Technologien und -Anwendungen) arinkel@hsr.ch

ROLLENVERTEILUNG

Die Teammitglieder erarbeiten unterschiedliche Diplomarbeiten.

Olivia Müller: Bachelorarbeit

Ernst Füllemann: Studienarbeit

VERANTWORTLICHKEITEN

Beide Teammitglieder sind selbst dafür verantwortlich, dass Sie die von ihrer jeweiligen Arbeit gestellten Anforderungen erfüllen. Dies beinhaltet die rechtzeitigen Abgaben der Dokumente und Poster sowie das leisten der geforderten Arbeitszeit.

Die Aufteilung der Aufgaben und Arbeitspakete übernehmen die Teammitglieder gemeinsam.

MEILENSTEINE

ÜBERSICHT

Datum	23.2.-1.3	2.3.-8.3.	9.3.-15.3.	16.3.-22.3	23.3.-29.3.	30.3.-5.4.	6.4.-12.4.	13.4-19.4.	20.4.-26.4.	27.4.-3.5.	4.5.-10.5.	11.5.-17.5.	18.5.-24.5	25.5.-31.5.	1.6.-7.6.	8.6.-14.6
Projektwoche	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Meilenstein	KickOff	MS1				MS2					MS3			MS4		MS5

DETAILLIERTE AUSFÜHRUNG DER MEILENSTEINE

Meilenstein	Abgabedatum	Deliveries
MS1	Do 05.03.2015	• Projektplanung • Ziele formuliert • Big Picture zum Thema
MS2	Do 02.04.2015	Konzeptionelle Lösung für das Problem
MS3	Do 07.05.2015	Implementation der Library in Simio abgeschlossen
MS4	Fr 29.05.2015	SA: Abgabe des Berichtes an den Betreuer bis 17.00 Uhr
MS5	Fr 12.06.2015	BA: Abgabe des Berichts und des A0 Posters an den Betreuer bis 12.00 Uhr, Präsentation 16.00-20.00 Uhr

ENTSCHEIDUNGSMANAGEMENT

Datum	Gefällte Entscheidung	Begründung
04.03.2015	Für Modellzeichnungen wird so weit wie möglich das Opensource-Tool Modelio verwendet. Andernfalls wird auf Visio ausgewichen.	• Opensource • Möglichkeit für ActivityDigramme • Visio: Beiden Teammitgliedern bekannt
12.03.2015	Bizagi statt Modelio und Visio	• Bessere Darstellung für Prozessabläufe
12.03.2015	Konzept festgelegt (siehe Sitzungsprotokoll 12.03.2015, Konzept)	• 6 Komponenten
18.03.2015	Für Materielle Entities wird der Begriff „Artikel“ verwendet	• Überbegriff für Produkt und Rohstoff • Ähnlich wie der Englische Begriff „article“
18.03.2015	Die Entity Produkt und Rohstoff werden nicht unterschieden und unter dem Begriff Product zusammengefasst.	• Ähnliche bis gleiche Parameter. Unterscheidung unnötig. • In SCOR verwendeter begriff
26.03.2015	Die Komponente Lager muss nur einen ProductTyp fassen können. Dafür soll eine zweite „Combiner“ entwickelt werden. Mit diesem ist es möglich komplexe Lagerstrukturen zusammen zu setzen	• Komplexität verringert in dem ein grosses Problem in zwei kleine aufgeteilt wurde • Simio kann nicht zur Laufzeit dynamisch neue Komponenten platzieren/initialisieren und benutzen und somit kann kein Lager mit einer variablen Anzahl an Produkten mit vernünftigem Aufwand erstellt werden.
26.03.2015	Konzentration auf die Entities Product, Order und Delivery	• Genügen für einen ersten Prototyp
15.04.2015	Hinzunahme des Entities Request	• Entity zur Implementierung von Informationsaustausch (Messaging, Abfragen) • Erweiterbar für zukünftige Projekte
15.04.2015	Product Entities werden nur innerhalb von Komponenten einzeln angetroffen. Im Scope der Supply-Chain ist jedes Product an eine zugehörige Order angehängt.	• Jede Komponente nimmt nur Order Entities entgegen. Einzige Ausnahme: Rack, nimmt auch Requests entgegen.
02.05.2015	Manuelle Tests in eigenen Projekten für isolierte Komponenten	• Experimente könnten zwar für Tests benutzt werden geben jedoch zu wenig Einsicht in die Funktionsweise einer Komponente

ZEITMANAGEMENT

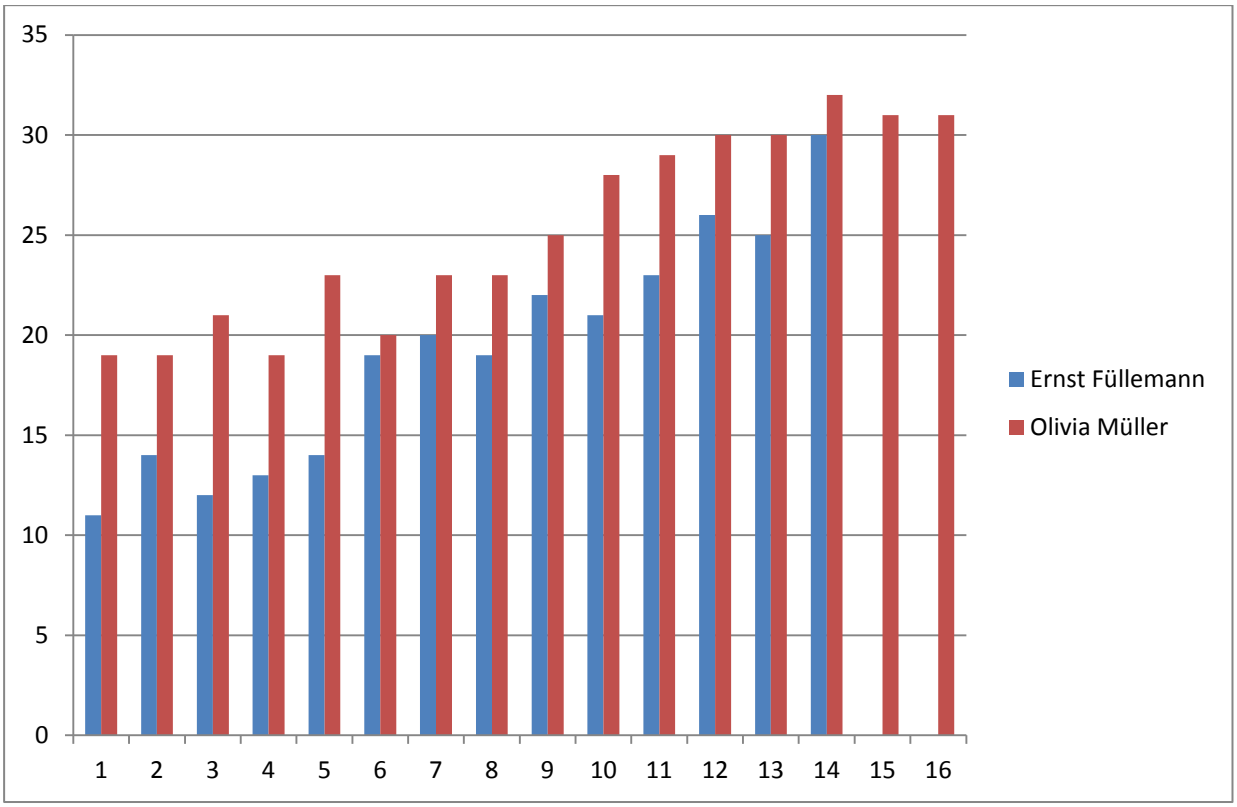
DURCHSCHNITTLICHE STUNDEN PRO WOCHE

Ernst Füllemann: 19,2

Olivia Müller: 25,7

STUNDEN PRO WOCHE

Projektwoche	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	**Total**
Ernst Füllemann	11	14	12	13	14	19	20	19	22	21	23	26	25	30	0	0	**269**
Olivia Müller	19	19	21	27	23	20	23	27	25	28	29	30	34	35	31	31	**403**



STUNDEN ERNST FÜLLEMANN

Projektwoche	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
Einarbeitung																	
Einarbeiteung SCOR	6	3	8	5													**22**
Einarbeitung Simio	3	3	2	8	2			5	4		3						**30**
Einarbeitung SCM	2	6	2														**10**
Konzeption																	
Prozesse Modelieren					2	3							7				**12**
SCOR Metriken					3	2	7										**12**
Implementation																	
Rack					5	11	10	12	13	17	10	13	6	3			**100**
Factory																	**0**
Entities											5						**5**
Selling																	**0**
Purchase																	**0**
LibGesamt													1	7			**8**
Testing																	
Rack					2	3	1	2	1	3	3	2	1	4			**22**
Factory																	**0**
Entities																	**0**
Selling																	**0**
Purchase																	**0**
LibGesamt														4			**4**
Dokumentation		2					2		4	1	2	11	10	12			**44**
Total	**11**	**14**	**12**	**13**	**14**	**19**	**20**	**19**	**22**	**21**	**23**	**26**	**25**	**30**	**0**	**0**	**269**

STUNDEN OLIVIA MÜLLER

Projektwoche	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
Einarbeitung																	
Einarbeiteung SCOR	6		8	5				2									**21**
Einarbeitung Simio	8	5		6	5	3				6	2						**35**
Einarbeitung SCM	5	8	2														**15**
Konzeption																	
Prozesse Modelieren		3	11	2	5	3	2		8	9							**43**
SCOR Metriken					7	2			5								**14**
Implementation																	
Rack																	**0**
Factory						6	8	4					7				**25**
Entities				6	4	3	2	4				2					**21**
Selling								8			6	8	6				**28**
Purchase								2	8	7	8	10					**35**
LibGesamt														8	16	9	**33**
Testing																	
Rack																	**0**
Factory						3							5				**8**
Entities					2		6										**8**
Selling								3			4	6	3				**16**
Purchase										3	7						**10**
LibGesamt													4	6	6	8	**24**
Dokumentation		3					5		4	3	2	4	5	18	9	14	**67**
Total	**19**	**19**	**21**	**19**	**23**	**20**	**23**	**23**	**25**	**28**	**29**	**30**	**30**	**32**	**31**	**31**	**403**