Implementierung eines Supply-Chain-Management-Simulationstools

Bachelorarbeit

Studiengang Informatik
OST – Ostschweizer Fachhochschule
Campus Rapperswil-Jona

Herbstsemester 2020

Autoren:	Fabienne Lienhard, Jana Kravarik
Betreuer:	Prof. Dr.-Ing. Andreas Rinkel
Experte:	Knut Schmahl
Gegenleserin:	Prof. Dr. Nathalie Weiler
Abgabe:	15.01.2021

ABSTRACT

Ausgangslage und Problem

Eine Supply Chain hat zum Ziel, komplexe Vorgänge einer Lieferkette zu beschreiben. Darunter fallen der Informations-, Waren-, Geld- und Personenfluss. Die Analyse und Optimierung von Supply Chains ist ein wichtiger Punkt in der Industrie. Dafür wird oft auf Simulation zurückgegriffen, doch sind Simulationen nicht immer optimal ausgelegt dafür, die Domäne von Supply Chains darzustellen.

Die Simulationssoftware Simio enthält zum Beispiel Komponenten, die vor allem für einzelne Prozessschritte gedacht sind und daher ungeeignet dafür, komplexe Stationen einer Supply Chain abzubilden. Jedoch bietet Simio die Möglichkeit, Komponenten anzupassen oder neue zu definieren. Zum Abbilden einer solchen Supply Chain müssen also komplexere Bausteine geplant und in Simio umgesetzt werden.

Ziel

Der finale Prototyp ermöglicht das Abbilden von Supply Chains mit bis zu 10 verschiedenen Produkten, mehren Kunden, Warenhäusern und zwei verschiedenen Herstellertypen. Verschiedene Key Performance Indikatoren ermöglichen die Beurteilung der Supply Chain.

Methode und Vorgehen

Die Arbeit ist die Fortführung der Studienarbeit «Modell Baukasten Supply-Chain» und setzt sich zusammen aus einem theoretischen und einem praktischen Teil.

Ergebnisse

Das Ergebnis der Arbeit sind drei aufeinander aufbauende Prototypen mit inkrementellem Funktionsumfang. Sie ermöglichen das Darstellen und Analysieren einer Supply Chain, die aus drei Stationen besteht: Kunde, Lager und Hersteller. Der Informations- und Warenfluss sind ebenfalls abgebildet. Handbücher zur Bedienung und Anpassung, sowie eine ausführliche technische Dokumentation sind Bestandteil der Abgabe. Im Ausblick der Arbeit ist besprochen, wie das Projekt weitergeführt werden könnte.

Ausblick

Im Ausblick wird besprochen, wie der Prototyp weiter ausgebaut und verbessert werden kann.

DANKSAGUNG

Hier möchten wir uns bei allen bedanken, die uns während dieser Arbeit unterstützt haben und für uns da waren.

An erster Stelle ist dies unser Betreuer Professor Andreas Rinkel, der uns mit seiner kompetenten Betreuung während der Bachelorarbeit, wie zuvor schon während der Studienarbeit, stets zur Seite stand.

Unserem Experten Knut Schmahl gebührt ebenfalls ein grosses Dankeschön, für seinen Input. Mit seinem Fachwissen hat er uns einen guten Einblick in die Supply Chain Domaine gegeben.

Danken möchten wir auch Simio für die Studentenversion, die sie uns während dieser Zeit kostenlos zur Verfügung gestellt haben.

Zu guter Letzt danken wir auch herzlichst unseren Familien, Freunden und Haustieren, die uns immer motivieren konnten und uns zum Lachen gebracht haben, wenn wir es brauchten.

INHALTSVERZEICHNIS

1 EINLEITUNG

Auftrag

Im Rahmen der Arbeit ist das in der Studienarbeit «Modell Baukasten Supply Chain» entwickelte Konzept zur Simulation von Supply Chains in Simio prototypisch umzusetzen. Dazu gehört die Planung und Definition der Prototypenfolge mit inkrementellem Funktionsumfang.

Ziel

Die während der vorangegangen Studienarbeit definierten Prototypen werden weiter ausgearbeitet und die gestellten Anforderungen verfeinert. Anschliessend sind die geplanten Prototypen in Simio zu implementieren.

Der finale Prototyp3 ermöglicht das Abbilden von Supply Chains mit bis zu 10 verschiedenen Produkten, mehren Kunden, Warenhäusern und zwei verschiedenen Herstellertypen. Verschiedene Key Performance Indikatoren ermöglichen die Beurteilung der Supply Chain.

Die umgesetzten Prototypen werden mit einem Handbuch zur Bedienung sowie Anpassung und Erweiterung geliefert.

Vorgehen bei der Umsetzung

Die Prozesse der Supply Chain, die im Bericht diskutiert werden, werden mit der BPMN Schreibweise abgebildet. Die den Bericht begleitende technische Dokumentation zeigt anhand von Bildern die Prozessschreibweise, die Simio selbst verwendet.

Überblick über die Arbeit

Das Interesse am Thema Supply Chain und Optimierung ist auch bei anderen Hochschulen vorhanden. Bereits beim ersten Diskussionsgespräch vom 21. September 2020 sind mehrere Hochschulen vertreten gewesen. Während der Arbeit involviert waren Prof. Dr. Robert Grüter, Hochschule Bonn-Rhein-Sieg, Experte Knut Schmahl, Lufthansa Industry Solutions Hamburg und Prof. Dr.-Ing Andreas Rinkel sowie weitere Angehörige der Ostschweizer Fachhochschule.

Zuerst wird der Arbeitsauftrag klarer definiert, also die Anforderungen an die verschiedenen Prototypen und der Lieferumfang festgelegt.

Anschliessend werden die drei Prototypen aufeinander aufbauend umgesetzt und die Überlegungen zu den Umsetzungen dokumentiert. Wobei der Funktionsumfang mit jedem Prototyp ansteigt.

Im Ausblick wird besprochen, was für weiterführende Arbeiten möglich ist. Begleitet wird der Bericht von den Handbüchern zur Bedienung und Anpassung, sowie der technischen Dokumentation, die einen ausführlichen Blick auf die Umsetzung in Simio ermöglicht.

2 BEGRIFFE UND DEFINITIONEN

In diesem Kapitel werden die verwendeten Begriffe und Definitionen erläutert.

2.1 DARSTELLUNG

Um die Leserlichkeit zu erhöhen, sind bestimmte Begriffe speziell hervorgehoben. Diese Hervorhebung soll dem Leser verdeutlichen, in welcher Kategorie der Begriff einzuordnen ist. Unterschieden wird zwischen drei Kategorien:

Simio Begriffe Begriffe, die für Stationen und Kategorien gelten, die von Simio verwendet werden.

Zum Beispiel State, Server, Process, ...

Stationsnamen Während der Arbeit neu definierte und erstellte Simio Objekte.

Zum Beispiel *ProducerAdvanced, Warehouse, Unpack, …*

Eigennamen Während der Arbeit neu definierte Namen für Prozesse, Prozessschritte, Listen, States, Events, Properties und Elemente.

Zum Beispiel *OpenOrder, RememberOpenOrder, InitialAvailability*

2.2 SUPPLY CHAIN

An dieser Stelle sei auf das Kapitel 2 Supply Chain Grundlagen der vorangegangen Studienarbeit verwiesen, um eine ausführlichere Beschreibung zu erhalten. Für die in dieser Arbeit umgesetzten Prototypen stellt sich die Supply Chain aus drei verschiedenen Hauptstationen zusammen.

1. Dem Kunden, der Produkte einkaufen möchte und Bestellungen verschickt.
2. Dem Warenhaus, das Bestellungen entgegennimmt, dieser erfüllt oder Waren nachbestellt, um sie erfüllen zu können.
3. Dem Produzenten, der Bestellungen entgegennimmt und Produkte herstellt, um die Nachfrage zu stillen. Auch der Produzent verschickt Bestellungen für Rohmaterialien, die er für die Produktion benötigt.

Die Stationen sind verbunden mit einem Informations- und einem Warenfluss.

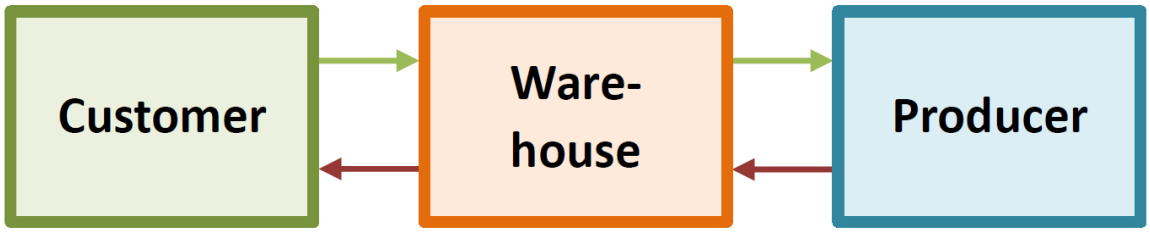


Abbildung 1: Supply Chain, vereinfachte Ansicht

2.3 ENGLISCH-DEUTSCHE BEGRIFFE

In den nachfolgenden Kapiteln werden für die einzelnen Komponenten und Kernelemente die englischen Begriffe, sowie Abkürzungen verwendet. Nachfolgend eine Übersicht der Begriffe mit der deutschen Übersetzung und einer kurzen Beschreibung.

Begriff	Übersetzung	Beschreibung
BPMN	*BPMN*	Steht für Business Process Model and Notation. Dies ist eine grafische Notation, um Prozesse zu beschreiben.
Batch	*Auflage*	Produktionsauflage, die während einem Produktionsvorgang hergestellt wird.
CostCenter	*Kostenstelle*	Einheit, an die bestimmte Kosten verrechnet werden.
Counter	*Zähler*	Nummer, welche erhöht wird, um etwas zu zählen.
Customer	*Kunde*	Während Arbeit definierte Station, welche dem Endkunden einer Supply Chain entspricht.
Data Table	*Datentabelle*	Data Table, ist eine von Simio zur Verfügung gestellte Tabellenart. Diese Tabellen können ausgefüllt, importiert, exportiert und als Table Objekte verwendet werden.
Events	*Ereignis*	Beim Auftreten einer bestimmten Gegebenheit, kann ein Event ausgelöst werden und an einem anderen Ort registriert werden. Dies erlaubt, auf etwas zu reagieren, das in einem anderen Teil der Simulation passiert.
FinalProduct	*Endprodukt*	Ein Produkt, das aus etwas anderem hergestellt wird. Bei den umgesetzten Prototypen handelt es sich dabei um ein Produkt, welches aus zwei Rohmaterialien bei einem *ProducerAdvanced* hergestellt wird.
Flag	*Kennzeichen/ Merker*	Kennzeichen, welches als Boolean (*True* / *False*) gespeichert wird. Einfacher Status mit zwei Zuständen.
InputNode	*Eingangsknoten*	Schnittstelle, über die Einheiten eine Station betreten können.
Lists	*Listen*	Listen in Simio sind einfache Aufzählungen von entweder Strings, Knoten oder Objekten. Sie können von Properties und States als zugelassene Wertelisten verwendet werden.
Loop	*Schlaufe*	Ein Ablauf, der wieder zurück auf seinen eigenen Anfang führt. Meist wird dieser so lange ausgeführt, bis eine bestimmte Bedingung erfüllt ist.
Node	*Knoten*	Verbindungsknoten sind Simio Objekte, die das Verbinden mit Pfaden erleichtern.
OutputNode	*Ausgangsknoten*	Schnittstelle, über die Einheiten eine Station verlassen können.

Order / OrderDecline	*Bestellung / Bestellungsabsage*	Eine *Order* ist eine Bestellung, die von einer Station an eine andere Station geschickt wird und diese auffordert der ersten Station Produkte zu liefern. Die *OrderDecline* Nachricht wird als Reaktion auf eine Bestellung versendet, die die Station nicht erfüllen kann.
Pack	*Verpackungsstation*	Während der Arbeit erstellte Station, die als Teilbaustein einer anderen Station verwendet wird. Sie hat die Aufgabe, Produkte auszupacken.
Path	*Pfad*	Verbindungsweg zweier Knoten, die für Verbindungen zwischen Stationen und innerhalb von Stationen verwendet wird.
Producer	*Hersteller*	Während der Arbeit erstellte Station, welche einem Hersteller in einer Supply Chain entspricht.
ProducerAdvanced	*Erweiterter Hersteller*	Während der Arbeit erstellte Station, welche eine Erweiterung der *Producer* Station ist.
Product (Entity)	*Produkt (Einheit)*	Während der Arbeit erstellter Objekttyp, der Produkte simuliert.
ProductionSize	*Produktionsmenge*	Steht für die Anzahl, wie oft die Produktion angestossen werden soll.
ProductionUnit	*Produktionseinheit*	Während der Arbeit erstellte Station, die ein Teilbaustein der Station *ProducerAdvanced* ist. Die Station enthält den Herstellungsprozess, um aus zwei Produkten ein anderes Produkt herzustellen.
Property	*Eigenschaft*	Eigenschaften einer Station oder eines Simio Objektes. Eigenschaften haben einen Typ (z.B. Integer) und einen Wert (z.B. 20).
Process	*Ablauf*	Vorgegebene oder während der Arbeit definierte Abläufe, welche immer in der gleichen Reihenfolge am gleichen Punkt der Simulation ausgeführt werden. Diese Abläufe sind über einzelne Schritte genauer definiert.
RawMaterial	*Rohmaterial*	Produkte, die für die Herstellung eines anderen Produktes benötig und dabei verbraucht werden.
ReceivingArea	*Wareneingang*	Während der Arbeit erstellte Station, die als Annahmestelle für Sendungen verwendet wird.
Server	*Server*	Simio Objekt, welches Entities entgegennimmt und verarbeitet.
ShippingArea	*Wareausgang*	Während der Arbeit definierte Station, die Sendungen aus einer Station verschickt. Die *ShippingArea* enthält eine *Pack* Station, welche die Produkte verpackt.
State	*Zustand*	Zustände können wie Eigenschaften auf Simio Objekten definiert werden. Im Gegensatz zu den Eigenschaften können diese aber während der Simulation dynamisch gesetzt werden.

Storage-Management	*Lagerverwaltung*	Die Lagerverwaltung ist eine Zusatzfunktion auf dem *ProducerAdvanced*. Ist sie aktiviert so verwaltet der *ProducerAdvanced*, wie viele Produkte sich in seinem Lager befinden. Der Lagerbestand wird dann immer nach dem Erfüllen einer Bestellung überprüft. Liegt er unter einem definierten Schwellwert, so wird eine ebenfalls definierte Menge an Produkten bei der Produktion in Auftrag gegeben.
StoreProducer	*Herstellerlager*	Während der Arbeit erstellte Station, die ein Teilbauteil einer *ProducerAdvanced* Station ist. Die Station lagert Produkte ein und aus.
StoreWarehouse	*Lagerhauslager*	Während der Arbeit definierte Station, die ein Teilbaustein einer *Warehouse* Station ist. Die Station lagert Produkte ein und aus.
Supplier	*Lieferant*	Ein Lieferant beschreibt eine *Warehouse*, *Producer* oder *ProducerAdvanced* Station. Sie versorgen andere Stationen mit Produkten.
Supply Chain	*Lieferkette*	Siehe Kapitel 2.2.
TargetStock	*Zielbestand*	Gibt die Menge der Produkte an, welcher der *ProducerAdvanced* nach dem Erfüllen einer Bestellung noch im Lager haben muss. Dieser Wert wird nur verwendet, wenn die Lagerverwaltung aktiviert ist. (siehe auch Lagerverwaltung)
Unpack	*Auspackstation*	Während der Arbeit definierte Station, welche ineinander verpackte Produkte trennt.
Warehouse	*Lagerhaus / Lager*	Während der Arbeit erstellte Station, die einem Lager und einer Verkaufsstation entspricht.

Tabelle 1: Englisch-Deutsche Begriffe mit Beschreibung

3 VORÜBERLEGUNGEN

Vor der Umsetzung in Simio ist der Arbeitsauftrag genauer zu definieren. Dieses Kapitel widmet sich dem Ausarbeiten des Auftrags, sowie den Überlegungen, die zu den Umsetzungen der einzelnen Prototypen führten.

3.1 DEFINITION ARBEITSAUFTRAG

Das Ziel der Arbeit ist das Planen und Umsetzen dreier Prototypen einer einfachen Supply-Chain, aufbauend auf dem «Prototyp0» der vorangegangen Studienarbeit «Modell Baukasten Supply-Chain».

Die drei Prototypen bauen alle aufeinander auf und sollen die folgenden Anforderungen erfüllen:

Prototyp1

- Supply Chain kann beliebig mit weiteren *Warehouses* erweitert werden
- Verpackungstypen Boxen und Container, abhängig von Empfänger:
    - Boxen für Haushalte (Waste), Container für Industrie (wiederverwertbar)
- Kosten von KPIs sollen berechnet und ausgelesen werden können
- CostCenter hinterlegen auf Stationen

Prototyp2

- Verschiedene Produkttypen sollen unterstützt werden
- Zusätzlich zum *Warehouse* soll es einen neuen *Producer* Typen geben, welcher aus verschiedenen Produkten ein Produkt macht
    - z.B. Wasser und Hopfen zur Herstellung von Bier
- Die Supply Chain soll auf Grund einer Entscheidungsmatrix automatisch den Supplier wechseln können, bei z.B. bei Ausfällen oder günstigeren Konditionen

Finaler Prototyp3

- Schlussendlich soll eine Beispiel Supply-Chain aus den erstellten Komponenten erstellt werden, welche auf Karten und Transportwegen über reale Strassen aufgebaut ist
- CostCenter separat pro Station, sowie eine Übersicht darüber erstellen, welche Informationen ausgelesen werden können
- Optimierung des Informationsflusses
- Finale optische Überarbeitung
- Leeres Grundmodell mit bereits erzeugten notwenigen Data Tables zur schnellen Handhabung

Zusätzlich soll eine Technische Dokumentation über alle Prototypen erstellt werden, sowie Handbücher, wie diese erweitert und bedient werden.

3.2 PROTOTYP1

Im ersten Teil der Arbeit wird ein auf Prototyp0 aufbauender Prototyp1 erstellt. Die Supply Chain ist um beliebig viele *Warehouses* und *Customers* erweiterbar. Bestellungen können in zwei verschiedenen Verpackungen verpackt werden, abhängig davon, ob ein *Warehouse* oder ein *Customer* beliefert wird. Die Implementation von CostCentern ermöglicht das Auslesen von einfachen Kostenfunktionen.

3.2.1 ÜBERLEGUNGEN

3.2.1.1 Skalierbarkeit der Supply Chain

Als mittlere Station ist das *Warehouse* die Station, welche mehrmals hintereinandergeschaltet werden kann, um die Supply Chain beliebig zu verlängern. Die Lieferung soll bestimmte Informationen von der Bestellung entgegennehmen, unabhängig davon, wer die Bestellung versendet hat. Dazu gehören, wohin die Lieferung geht, wie viel geliefert werden soll und wie die Produkte verpackt werden sollen. Beim Prototyp0 sind die zum Beispiel Produkte noch unverpackt an den Kunden gesendet worden, in Prototyp1 soll dies nur noch verpackt geschehen.

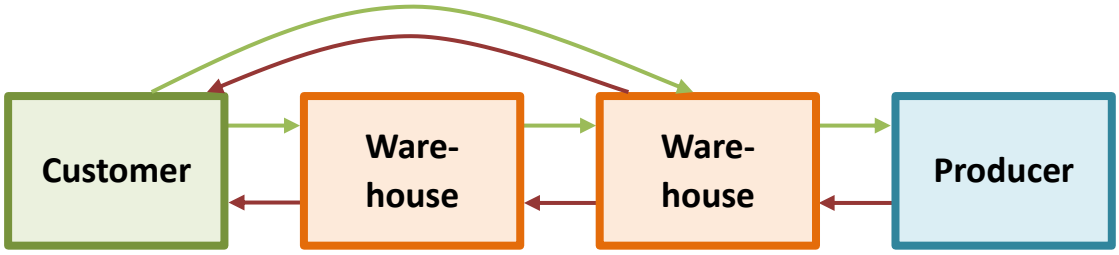


Abbildung 2: supply chain

Um die verschiedenen Verpackungsarten zu unterstützen wird beim Erweitern des *Warehouses* gleichzeitig auch die Modellierung überarbeitet. Ziel ist es, *Warehouses* in kleinere Teilstationen zu unterteilen, welche auch in anderen Stationen verwendet werden können. Der Orderfluss (Management) bleibt auf der *Warehouse* Ebene. Folgende Unterstationen sollen vom *Warehouse* separat abgebildet werden:

Store Im Store werden die Produkte und Verpackungen gelagert. Im Prototyp1 besteht es aus 3 Lagereinheiten (Servern) wobei je eines Produkte, Container und Carton lagert.

ReceivingArea Die *ReceivingArea* nimmt Lieferungen entgegen, packt diese aus und leitet sie weiter an den Store. Das Auspacken geschieht mithilfe einer *Unpack* Station.

ShippingArea In der *ShippingArea* werden Produkte mithilfe einer *Pack* Station in Verpackungen verpackt. Die verpackten Produkte werden anschliessend verschickt.

Pack/Unpack Die *Pack*/*Unpack* Stationen sind simpel aufgebaute Stationen, welche Produkte in Verpackungen verpacken, respektive auspacken können. Sie übernehmen auch das Schreiben des Container Labels, welches die Bestellmenge speichert.

Das Aufteilen des *Warehouses* in mehrere Unterstationen, vereinfacht das Wiederverwenden der einzelnen Stationen für andere Stationen und hält das *Warehouse* als eigene Station übersichtlicher.

3.2.1.2 Verschiedene Verpackungstypen

Bisher waren Verpackungen *OrderEntities*. Beim Einführen eines zweiten Verpackungstypen ist klar geworden, dass Verpackungen bei den *Warehouses* auch als Produkte gelten sollten und gelagert, nachbestellt und verschickt werden können. Zu diesem Zweck wird die *ProductEntity* überarbeitet. Die festgelegten Verpackungstypen für Prototyp1 sind Container und Carton.

Container Wiederverwendbarer Verpackungstyp, wird für den Versand vom *Producer* zum *Warehouse* und zwischen *Warehouses* verwendet.

Carton Verbrauchbarer Verpackungstyp, wird für den Versand zum *Customer* verwendet und nach Gebrauch beim Empfänger entsorgt.

Welcher Verpackungstyp beim Einpacken verwendet wird, hängt davon ab welcher Verpackungstyp von der *Order* festgelegt wurde. Der *Producer* ist dabei als Quelle für *Customer* noch vernachlässigt worden und unterstützt in Prototyp1 nur das Verpacken in Container.

Die Logik, um im Store Verpackungen und Produkte aufzuteilen, unabhängig vom Produkttyp ist so hinterlegt, dass in der Verpackungsliste Container, Carton und *NonPackaging* definiert sind. *NonPackaging* umfasst hier alle anderen Produkte, die selbst nicht auch Verpackung sein können.

Unpack/*Pack* sind umgesetzt worden, um in einer zukünftigen Version zu ermöglichen, dass Verpackungen auch in andere Verpackungen verpackt werden können, also zum Beispiel Cartons in Container.

Im Prototyp1 können Container und Cartons noch nicht nachbestellt werden. Dafür besteht ein Anfangsbestand auf den *Warehouses*. Die Logik, um Verpackungen beim Hersteller zu bestellen wird erst im Laufe des Prototyp2 umgesetzt. Dort soll anhand einer Entscheidungsmatrix entschieden werden, bei welchem Hersteller was zu welchen Preisen bestellt wird.

3.2.1.3 Kosten

Für das Einführen von CostCentern zum Berechnen von Kosten sind in einem ersten Schritt KPIs festgelegt worden, die für das Betrachten der Supply Chain interessant sind. Nicht alle der folgenden KPIs sind tatsächlich relevant für die Implementation von Kosten, geben aber einen guten Überblick darüber, welche Kosten relevant sind.

General	Zeit	**Supply-Chain-Cycle Time**	wie lange die Abwicklung eines Kundenauftrags dauern würde, wenn alle Bestände zum Zeitpunkt der Auftragserteilung bei null wären
General	Prozentzahl	**Perfect Order Index**	die fehlerfreie Auftragsabwicklung im gesamten Supply-Chain-Prozess
General	prozentzahl	**Fill Rate**	Bedarfsdeckungsrate: Höhe der Kundennachfrage, die durch verfügbare Lagerbestände gedeckt werden kann.
General	Anzahl in Zeitraum	**Inventory Turnover**	Lagerumschlag: Wie häufig wird der gesamte Lagerbestand in einem bestimmten Zeitraum verkauft.
General	Prozentzahl	**On time shipping**	Wie viel der Lieferungen innerhalb eines festgelegten Zeitraums sind

Finance	Kosten	**Produkt Rendite**	Herstellungskosten / Verkaufskosten
Finance	Zeit	**Cash-to-Cash time**	ist ein Maß für den Zeitraum zwischen der Bezahlung von Leistungen beim Zulieferer und dem Eingang der Zahlungen von Kunden
Green	Kosten	**Waste**	
Green	Kosten	**Verpackung**	Wert von Verpackungen welche einmal verwendet werden
Green	Kosten	**Abgelaufen**	Wert von Produkten, welche ablaufen, bevor sie verkauft werden können.
Green	Wert oder Kosten	**CO_2**	Co2 Ausstoss während dem Transport, wird aus der Transportdauer und dem Transportweg zusammengesetzt

Tabelle 2: KPI

Folgende KPIs werden in Kosten angegeben und sollen daher auch als Kosten abgebildet werden.

Produktrendite Die Produkt Rendite besteht aus den für die Produktbeschaffung oder Herstellung entstandenen Kosten unter Berücksichtigung der entstandenen Einnahmen. Im Prototyp1 werden diese Kosten über die gesamte Supply Chain angegeben, in Zukünftigen Versionen soll es dann so erweitert werden, dass jede Station eine eigene Produkt Rendite errechnet.

Waste Verpackungen Die Kosten, welche durch das Entsorgen von Karton entstehen. Dieser Wert wird auf jeder Station berücksichtig und kann in zukünftigen Versionen so erweitert werden, dass entstehender Abfall an eine Entsorgungsstelle weitergegeben werden und dort die anfallenden Kosten berechnet werden.

Waste Abgelaufen Die Kosten, die durch abgelaufene Produkte entstehen, bevor sie den Kunden erreichen. Der Wert ist interessant, das Einführen eines Ablaufdatums für Produkte ist jedoch aus dem Scope von Prototyp1 genommen und wird allenfalls in einem späteren Prototyp erneut aufgegriffen.

CO_2 CO_2 wird als durch CO_2 Abgaben entstehende Kosten an verschiedenen Bereichen der Supply Chain berechnet und besteht aus mehreren einzeln auslesbaren Werten. Im Prototyp1 werden dabei die Werte *StorageCost, PackagingCost* und *MaterialCost* berücksichtigt. Die Kosten, die für den Transport auftreten, sind im Prototyp1 noch nicht umgesetzt.

Abgeleitet von diesen Ansprüchen an die Kostenabblidung sind die folgenden CostCenter definiert:

CostCenterWastePacking	Definiert auf: *Customer*, *Warehouse*
CostCenterProductIncome	Definiert auf: *Customer* **Bemerkung**: Soll im nächsten Prototyp auf allen Stationen definiert werden.
CostCenterCO2Material	Definiert auf: *Producer*
CostCenterProduction	Definiert auf: *Producer* **Bemerkung**: Soll im nächsten Prototyp auch für den neuen *Producer* implementiert werden.
CostCenterCO2Packaging	Definiert auf: *Producer*, *Warehouse*
CostCenterStorage	Definiert auf *Warehouse* **Bemerkung**: Soll in der nächsten Version auch für den neuen *Producer* implementiert werden.

Standard Simio Stationen, wie zum Beispiel Server, bieten bereits eine Möglichkeit, Kosten an ein CostCenter weiter zu verrechnen. Pro Station sind jedoch nur einem CostCenter Kosten verrechenbar. Um diese Einschränkung zu umgehen, werden weitere Kosten via eines Assign Schritts in einem Prozess an weitere CostCenter verrechnet.

3.3 PROTOTYP2 UND PROTOTYP3

Im zweiten Teil der Arbeit wird ein auf Prototyp1 aufbauender Prototyp2 erstellt. Ein erweiterter *Producer* soll ermöglicht werden, der innerhalb der Supply Chain platzierbar ist, Materialien bestellt, diese weiterverarbeitet und andere Stationen mit den neu erstellen Produkten beliefert.

Designtechnisch sind Prototyp2 und Prototyp3 sehr ähnlich und demzufolge in einem Kapitel zusammenführend erläutert.

3.4 ÜBERLEGUNGEN

3.4.1 PRODUCERADVANCED UND DIE PRODUCTIONUNIT

Um einen komplexeren Produktionsprozess in der Supply Chain darzustellen, wird in Protoyp2 ein zusätzlicher *Producer*, der *ProducerAdvanced*, erstellt. Anders als der *Producer*, soll dieser zwei Rohmaterialien zu einem neuen Produkt verarbeiten.

Der *ProducerAdvanced* ist ähnlich aufgebaut wie das *Warehouse* und wird um eine *ProductionUnit* erweitert. Der innere Aufbau der Station ist in Abbildung 3 zu sehen.

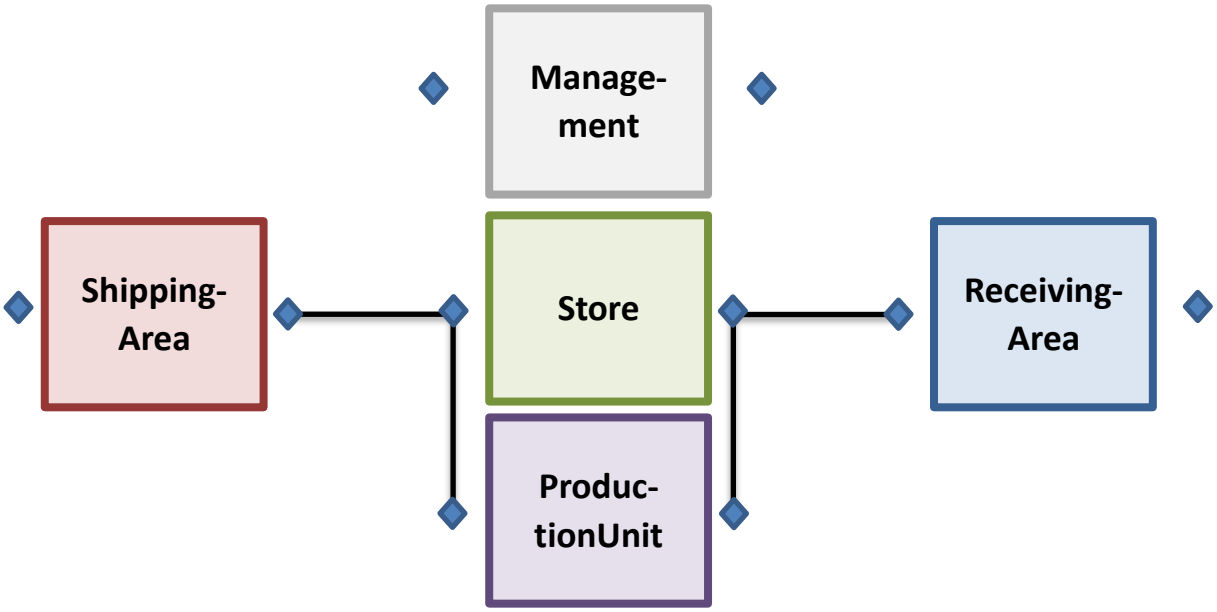


Abbildung 3: Prototyp2, ProducerAdvanced Aufbau

Der interne Store muss in der Lage sein, verschiedene Produkte zu verwalten, von denen eines das in der *ProductionUnit* hergestellte ist. Für den Anstoss der Produktion gibt es zwei verschiedene Möglichkeiten:

1. Die Produktion wird ausgelöst durch eine Bestellung, die explizit angibt, wie viel produziert wird.
2. Die Produktion wird solange fortgeführt wie Rohmaterial an Lager vorhanden ist.

Für Möglichkeit 1 – die Produktion auf Bestellung – ist der Management Prozess verantwortlich. Wohingegen für den 2. Fall ein vom Management Prozess abgetrennter, neuer Prozess definiert werden muss. Es ist auch eine Hybridlösung der beiden möglich; zum Beispiel wird, solange Ressourcen vorhanden sind, eine bestimmte Anzahl produziert. Diese Herstellungsanzahl kann bei plötzlich hoher Nachfrage dynamisch angepasst werden.

In der Realität werden meist mehr als nur zwei Rohmaterialien für die Produktion verwendet, die *ProductionUnit* kann jedoch nur für eine fest vorgegebene Anzahl Rohmaterial erstellt werden.

3.4.2 PRODUCERADVANCED IM MODELL UNTERBRINGEN

Der Fokus liegt zunächst darauf einfache Grundstationen mit überschaubaren Prozessen zu erarbeiten, die bei Bedarf für ein komplexeres System erweiterbar sind. Für Prototyp2 bedeutet dies ein an das *Warehouse* angelehnter *ProducerAdvanced* mit einer neuen Teilstation, der *ProductionUnit*.

Hierzu müssen einige Änderungen am Prototyp1 vorgenommen werden. Die Einführung mehrerer Produkte und die Anpassung des Systems, die zusätzliche Anzahl an Produkten zu verwalten.
Dies bedeutet, den Management Prozess so anzupassen, dass er dynamisch nach dem zu bestellenden Produkttyp sucht, auf die Notwendigkeit einer Nachbestellung überprüft und, falls notwendig, auch nachbestellt.

Um auf der *ProductEntity* genauer zu definieren, um was für einen Produkttyp es sich handelt und auf der *OrderEntity* anzugeben, welches Produkt bestellt wird, werden Listen geführt. Listen in Simio sind einfache Arrays. Mit der Einführung dieser Arrays, können verschiedene EntityTypen im Modell über den Integerwert der Position im Array miteinander verglichen werden. Dies verlangt, dass die verwendeten Listen auf der *OrderEntity* und der *ProductEntity* exakt gleich sortiert sein müssen.

Auch muss die Suche nach beliebigen Produkten am gleichen Ort ermöglicht sein. Die Store Unit des *Warehouses* und des *ProducerAdvanced* wird dahingehend angepasst, dass alle Produkttypen, *Packaging* oder *NonPackaging*, in einem Server gelagert werden. Es wird ein allgemeines Inventar über alle Produkte geführt, dass als Label im Modell selbst ersichtlich ist.

3.4.3 EINBINDEN EINER EINFACHEN PRODUCTIONUNIT

Die Produktion findet in der *ProductionUnit* statt und wird angestossen durch den *Production_Processing* Prozess auf dem *ProducerAdvanced*. Die *ProductionUnit* selbst hat keinen Einfluss darauf wann und wie viel produziert wird, nur darauf welches Produkt hergestellt wird und wie lange dieser Vorgang dauert.

3.4.4 MANAGEMENT PROZESS

Durch die steigende Komplexität des Management Prozesses wurde dieser in verschiedene Teilprozesse unterteilt.

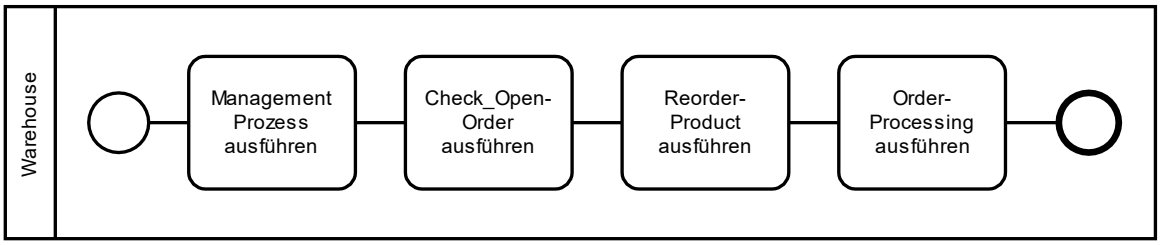


Abbildung 4: Warehouse, neuer Management Ablauf

Management Prozess
- Überprüft, ob Bestellung erfüllt werden kann
- Versendet *Orders*
- Versendet *OrderDeclines*
- Löst Nachbestellungen aus

Check Open Order
- Überprüft ob bereits eine Nachbestellung für ein Produkt verschickt wurde

Reorder Product
- Wählt den Zulieferer aus

Order Processing
- Erstellt und versendet *Orders*

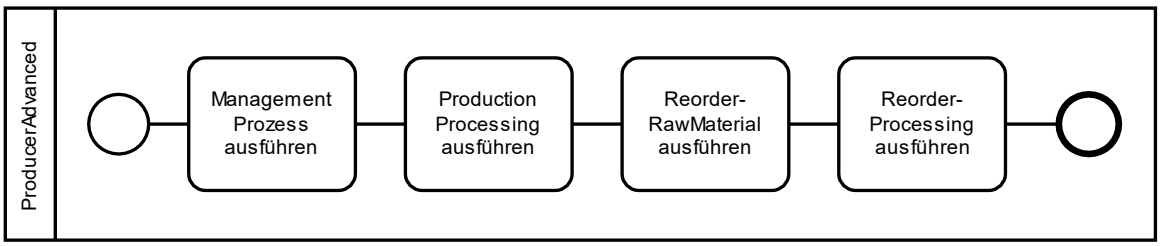


Abbildung 5: ProducerAdvanced, vereinfachter Ablauf

Management Prozess
- Überprüft, ob Bestellung erfüllt werden kann
- Stösst Production an
- Löst Nachbestellungen von Verpackungen aus

Production Processing
- Überprüft den Rohmaterial Lagerbestand
- Löst Nachbestellungen für Rohmaterial aus
- Wartet auf Nachbestellungen und sendet Rohmaterial an die ProductionUnit

Reorder RawMaterial
- Wählt den *Producer* aus

Reorder Processing
- Erstellt und versendet die *Order*

Die Prozesse *Check_OpenOrder*, *Reorder_Product*, *Order_Processing*, *Production_Processing* und *Reorder_RawMaterial* sind produktspezifisch und wurden für jeweils 10 Produkte definiert. Die *Warehouse* Station ist so definiert, dass es zehn verschiedene Produkte verwalten kann. Die *ProducerAdvanced* Station kann zwei verschiedene *RawMaterials* und ein *FinalProduct* verwalten. Die dafür verantwortlichen Prozesse sind so aufgebaut, dass sie einfach erweitert werden können, sollte die Anzahl der benötigten Produkte für nachfolgende Prototypen steigen. Das automatische Generieren dieser Prozesse ist nicht möglich. Ein selbst geschriebenes Plugin wäre nötig, welches dies ermöglicht.

Der *ProducerAdvanced* ist so definiert, dass das stationseigene Management auch den Lagerbestand für *FinalProduct* verwaltet. Ist die Option für die Lagerverwaltung aktiviert, werden zusätzlich die nachfolgenden Schritte ausgeführt:

- Der Lagerbestand wird mit einem Zielbestand verglichen.
- Liegt der momentane Lagerbestand unter diesem Zielwert, wird die Produktion angestossen.
- Die Standardproduktionsgrösse wird hergestellt.

Die Standardproduktionsgrösse sowie der Zielbestand können angegeben werden.

3.5 PRODUCER AUSWAHL

Eine der grössten Neuerungen für Prototyp2 ist, dass der *Producer*, bei dem bestellt werden soll, via eines Prozesses aus einer Liste mit mehreren Producern ausgewählt werden kann. Vor dem Bestellvorgang wird ein zusätzlicher Prozess aufgerufen, der vordefinierte Data Tables durchsucht und den ersten *Producer* auswählt, der verfügbar ist. Es werden die ersten drei Einträge der Data Tables durchsucht. Sollte das Modell mit mehreren Producern erweitert werden, kann dies im Prozess erweitert werden.

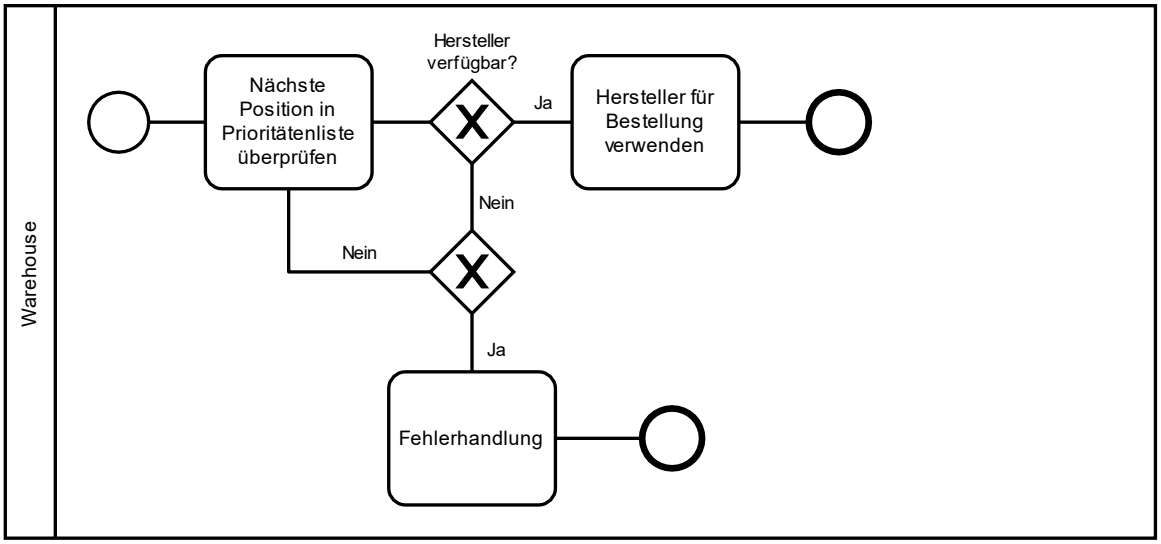


Abbildung 6: Warehouse, Herstellerwahl

Auf dem *Producer* gibt der State *Availability* an, ob ein *Producer* verfügbar ist. Problematisch ist, dass der Prozess den ObjektTypen des Producers braucht, um den State abzufragen. Dieser ist unterschiedlich, jenachdem, ob es sich um einen *Producer* oder einen *ProducerAdvanced* handelt.

Im Prototyp3 wird dieses Problem gelöst, indem der ganze Aufruf mit *Producer*, ProducerType und *Availability* aus der Data Table gelesen wird. Ebenfalls wird das *Warehouse* mit dem State *Availability* ergänzt.

	Supplier	Address	Expression Supplier
1	ProducerWater2	Input_Management@ProducerWater2	ProducerWater2.Producer.Availability
2	ProducerWater3	Input_Management@ProducerWater3	ProducerWater3.Producer.Availability
3	ProducerWater	Input_Management@ProducerWater	ProducerWater.Producer.Availability
4	NoSupplier	Input_Management@NoSupplier	NoSupplier.Producer.Availability
5	NoSupplier	Input_Management@NoSupplier	NoSupplier.Producer.Availability
6	NoSupplier	Input_Management@NoSupplier	NoSupplier.Producer.Availability

Abbildung 7: Hersteller Tabelle

3.6 ALLES IN ZAHLEN

Wird die Produktion angestossen, so wird diese für eine festgelegte Produktionsmenge ausgelöst. Die Produktion kann auf zwei unterschiedliche Arten angestossen werden:

1. Von einer Bestellung, die mehr Produkte benötigt als der momentane Lagerbestand erfüllen kann.
2. Vom Lagerverwaltungstask, sofern die Lagerverwaltung aktiviert ist.

Die Produktionsmenge wird immer aufgerundet, um nie weniger Produkte als benötigt zu produzieren. Erreicht wird dies durch das Addieren von 0.5, da Simio nach der folgenden Regel rundet:

0.0.. – 0.5 wird abgerundet, 0.51 – 0.99.. wird aufgerundet.

Um zu bestimmen, wie viel produziert wird, werden folgende Berechnungen ausgeführt:

Ausgelöst von Bestellung

$$ProductionSize = \frac{(NumberOrdered - CurrentInventory)}{FinalProductProductionBatchSize} + 0.5$$

Ausgelöst von Lagerverwaltung

$$ProductionSize = \frac{StandardProductionSize}{FinalProductProductionBatchSize} + 0.5$$

Begriffserklärungen

ProductionSize	Wie viele Batches von Produkten hergestellt werden
NumberOrdered	Die bestellte Menge des Produkts
CurrentInventory	Lagerbestand des bestellten Produkts.
FinalProductProductionBatchSize	Wie viele Finale Produkte, in einem Herstellungslauf hergestellt werden.
StandardProductionSize	Wenn die Lagerverwaltung aktiviert ist und der Lagerbestand unter dem Zielbestand ist, wird diese Anzahl an Produkten hergestellt.

Tabelle 3:Begriffserklärungen

3.7 BESONDERHEITEN ZUR UMSETZUNG IN SIMIO

Bei der Umsetzung des Prototyp2 sind mehrere Stolpersteine in der Modellierung aufgetreten. Einige davon sind auf Einschränkungen von Simio zurückzuführen, andere entstanden durch die gewählten Architekturentscheidungen der Arbeit.

3.7.1 PROZESSE: WAITS AND LOOPS

Eins der wiederkehrenden Probleme ist durch die Limitation des Wait Schrittes in Prozessen entstanden. Beim *Producer* ist das Design so gewählt, dass mehrmals auf vorherige Schritte gewartet wird.

Während des Produktionsprozesses wird darauf gewartet, dass die Produktion abgeschlossen ist und die neu hergestellten Produkte im Lager eintreffen. Ebenfalls zum Herstellungsprozess gehört das Nachbestellen von Rohmaterialien falls nötig. Auch bei diesem Schritt wird darauf gewartet, dass diese Rohmaterialien ankommen.

Dieses Warten wird mit einem Wait Schritt umgesetzt. Dieser wartet auf ein Event, welches das Lager beim Eintreffen der jeweiligen Produkte auslöst. Der Wait Schritt kann auf ein oder mehrere Events warten.

Problematisch ist, wenn zwei Produkte zeitlich sehr nah, fast gleichzeitig, ankommen, denn jede Ankunft löst ein Event aus. Wenn diese Events nun so schnell nacheinander ausgelöst werden, bleibt dem Prozess keine Zeit einen Counter für beide Events hochzuzählen und registriert nur den ersten. Der Workaround, der dafür in Prototyp2 angewendet wird, fügt eine Verzögerung von einer Sekunde direkt nach dem Wait ein.

Bei Produkten, die aus der Produktion ins Lager kommen, tritt ein ähnliches Problem auf. Die Verzögerung zwischen den Ankünften ist jedoch höher und in diesem Fall mit einer Wait Loop lösbar. Ein Counter wird verwendet, der jeweils nach dem Eintreffen eines Events um eins erhöht wird. Dieser Counter wird verglichen mit der erwarteten Anzahl, dieser Vergleich wird so oft wiederholt, bis er der erwarteten Menge entspricht. Dann wird der Counter zurückgesetzt und der Loop verlassen.

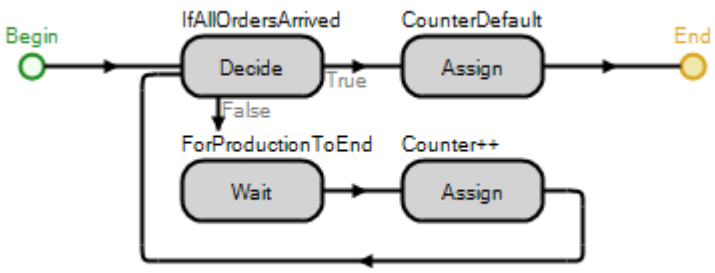


Abbildung 8: Wait Loop

Beide Lösungsansätze sind nicht optimal und sollten in einer Fortführung weiter durchdacht werden.

Bei der Verzögerung nach der Ankunft ist nicht garantiert, dass die Dauer der Verzögerung für jeden Fall ausreicht. In den meisten Fällen ist die Verzögerung länger als notwendig, was die Dauer des Produktionsprozesses unnötig verlängert.

Beim Wait Loop kann es sein, dass die Ankunft eines Produktes in die Phase zwischen den Loops fällt. Dies hat zur Folge, dass das Event ausgelöst wird, während der Prozess nicht im Wait Schritt ist und das Event somit nicht gezählt werden kann.

Im Prototyp3 ist dieses Problem mit einer Verzögerung gelöst worden. Nach der Ankunft eines Produkts im Lager wird ein Event ausgelöst, woraufhin der Lagerstand überprüft wird, ob alle benötigten Produkte im Lager sind. Ist dies nicht der Fall, wird eine Sekunde gewartet und die Überprüfung wiederholt, solange bis alle Produkte im Lager sind.

3.7.2 BESTELLUNGSHANDHABUNG

Im Prototyp2 kann sowohl das *Warehouse* als auch der *ProducerAdvanced* nur jeweils eine Bestellung gleichzeitig verarbeiten. Das Parallelisieren von Bestellungen ist dadurch eingeschränkt, dass die Produkte im Lager via Search Schritt gesucht und transportiert werden. Beim Parallelisieren von Bestellungen kann es vorkommen, dass mehrere Search Schritte gleichzeitig das gleiche Produkt oder Verpackung markieren und verschieben möchten. Was zu einem Fehler führt, der die Simulation abbricht. Mit den in der Arbeit erstellten Prototypen ist eine Erweiterung auf parallele Bestellverarbeitung daher nicht empfohlen.

3.7.3 OPENORDER HANDHABUNG BEIM WAREHOUSE

Bei Prototyp0 und Prototyp1 kann das *Warehouse* jeweils nur einen Produkttypen und Karton nachbestellen. Mit einem jeweiligen *OpenOrder* State merkt sich das *Warehouse*, ob das Produkt bereits nachbestellt worden ist. Beim Prototyp2 wird diese Funktionalität so erweitert, dass das *Warehouse* die Verwaltung und Nachbestellung von bis zu zehn Produkten handhaben kann und diese unabhängig voneinander nachbestellen kann.

Im Prototyp2 besteht ein Fehler mit diesem System: Beim Ankommen einer *OrderDecline* Entität wird der *OpenOrder* State von dem Produkt zurückgesetzt, das zuletzt vom *Warehouse* bestellt worden ist. Es wird nicht überprüft, für welches bestellte Produkt die *OrderDecline* Nachricht wirklich ist. In Prototyp3 wird dies behoben, indem der betreffende Produkttyp von der *OrderDecline* Nachricht in den Zwischenspeicher geschrieben wird, und erst dann der Prozess ausgeführt wird, der die betreffenden *OpenOrders* zurücksetzt.

3.7.4 DROP DOWN VON LIST PROPERTIES

Wird bei einem List Property eine Liste hinzugefügt, erstellt das List Property eine Caption Liste welche angibt, unter welchem Namen die Listeneinträge angezeigt werden. Beim Hinzufügen werden die Namen aus der Liste übernommen. Wird die Liste jedoch später angepasst, werden die Captions nicht automatisch aktualisiert. Entweder müssen bei den List Properties die Captions manuell angepasst werden oder die Liste vom List Property entfernt und neu hinzugefügt werde.

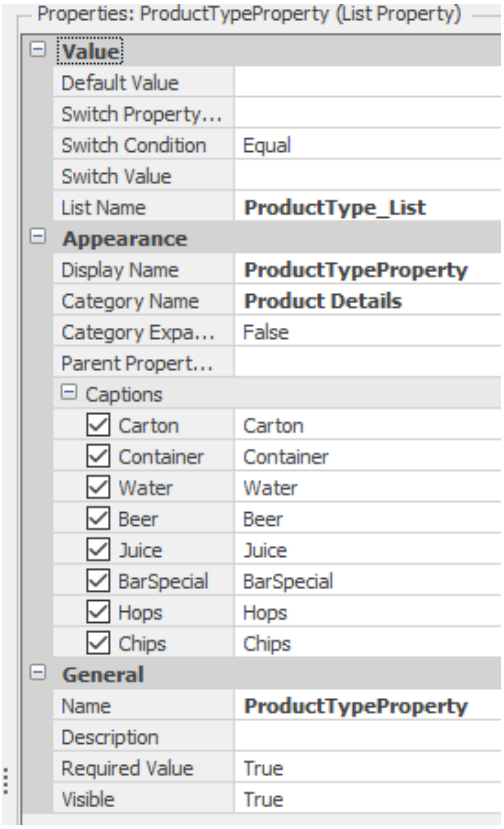


Abbildung 9: Captions

3.7.5 WARNUNGEN

Wenn beim *ProducerAdvanced* mehrere Produkte genau gleichzeitig ankommen, wirft die Simulation eine Fehlermeldung. Dieser Fehler ist darauf zurückzuführen, dass der Inventory Prozess, der bei der Ankunft von Paketen aufgerufen wird, zweimal gleichzeitig versucht wird aufzurufen. Indem sichergestellt wird, dass die Zulieferer für die Rohmaterialien nicht genau gleichweit vom *ProducerAdvanced* entfernt sind, kann dieses Problem umgangen werden.

4 SPEZIFIKATION DER PROZESSABLÄUFE EINER SUPPLY CHAIN

Eine Supply Chain besteht aus verschiedenen voneinander abhängigen Stationen. Bei jeder dieser Stationen passieren bestimmte Abläufe. In diesem Kapitel sind diese Abläufe anhand von BPMN genauer definiert.

4.1 WAREHOUSE

4.1.1 ÜBERSICHT

Das *Warehouse* verfügt über mehrere komplexe Abläufe. Diese sind in zwei Hauptprozesse unterteilbar. Der erste der beiden Abläufe ist verantwortlich für die Handhabung von Bestellungen und Erfüllung respektive Nichterfüllung dieser. Er setzt sich aus mehreren Teilprozessen zusammen wie in Abbildung 10 aufgeführt.

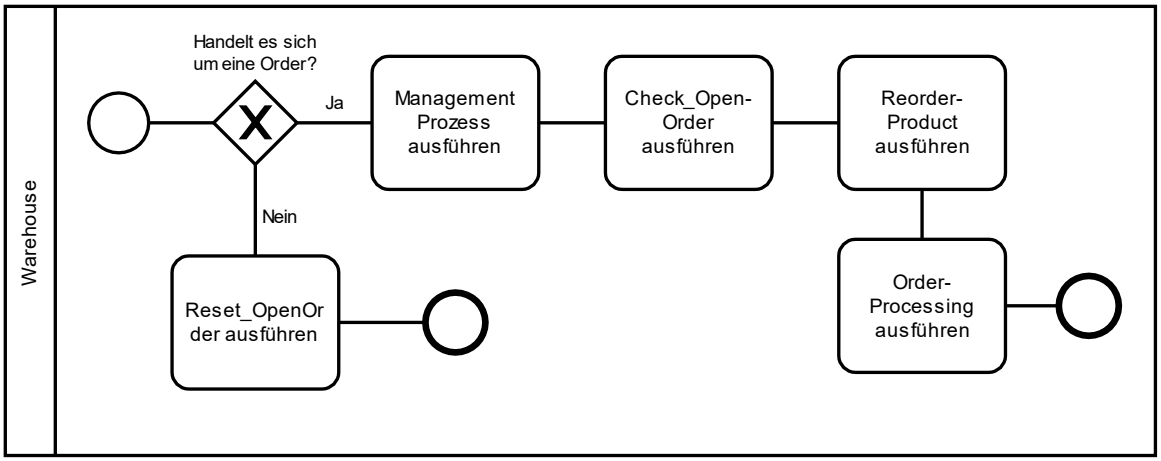


Abbildung 10: Warehouse Übersicht

Aus den nachfolgenden Abläufen ist ersichtlich, dass beim *Warehouse* erst etwas passiert, wenn eine *Order*, eine *OrderDecline* Nachricht oder eine Lieferung beim *Warehouse* eintreffen. Einzige Ausnahme ist das Verrechnen der Lagerungskosten an das entsprechende CostCenter. Sobald Produkte im *StoreWarehouse* gelagert werden, werden die Lagerungskosten gezählt.

4.1.2 MANAGEMENT PROZESS

Der Management Prozess ist der wichtigste und komplexeste Ablauf im *Warehouse*. Er bearbeitet ankommende Bestellungen und löst die Lieferung respektive eine *OrderDecline* Nachricht aus. Im Falle einer *OrderDecline* Nachricht, also wenn das *Warehouse* die Bestellung nicht erfüllen kann, wird eine Nachbestellung des betreffenden Produkts ausgelöst.

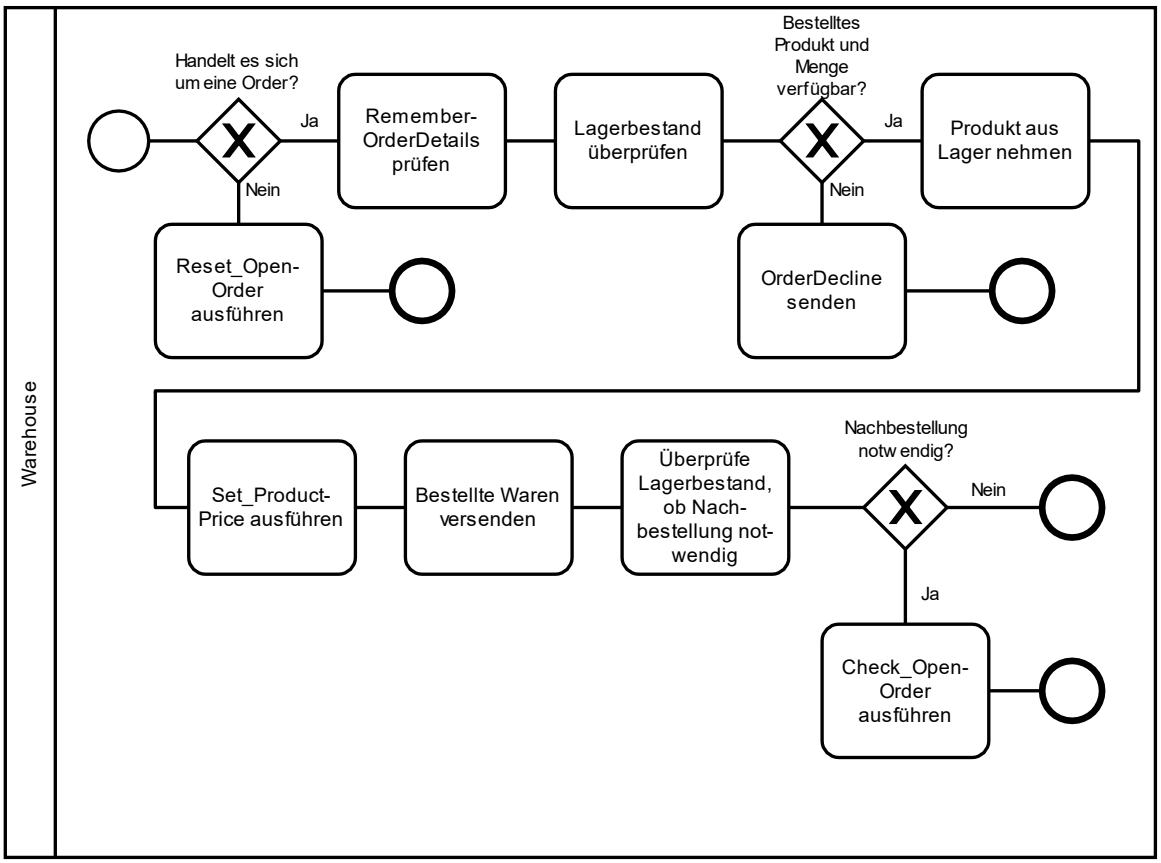


Abbildung 11: Warehouse Management Prozess

4.1.2.1 Set_ProductPrice

Wird eine Lieferung versendet, ruft der Management Prozesses den *Set_ProductPrice* Prozess auf. Dieser füllt den Preis basierend auf den Benutzereingaben für das bestellte Produkt aus und schreibt diesen auf die *ProductEntity*.

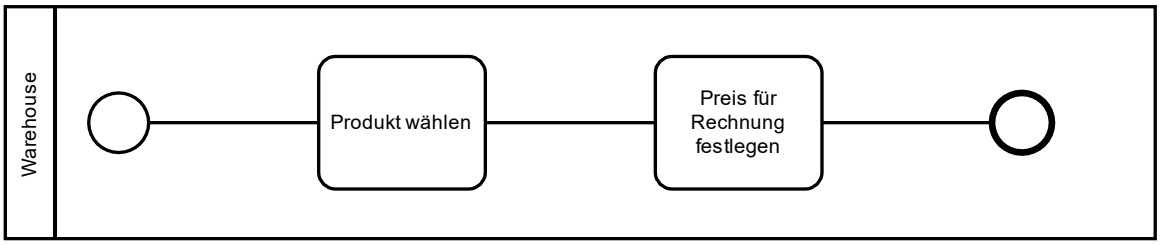


Abbildung 12: Warehouse Set_ProductPrice

4.1.2.2 Reset_OpenOrder

Ist die eingegangene Bestellung vom Typ *OrderDecline*, wird der Prozess *Reset_OpenOrder* ausgeführt. Dieser setzt das *OpenOrder* Kontrollflag für das in der *OrderDecline* angegebene Produkt zurück.

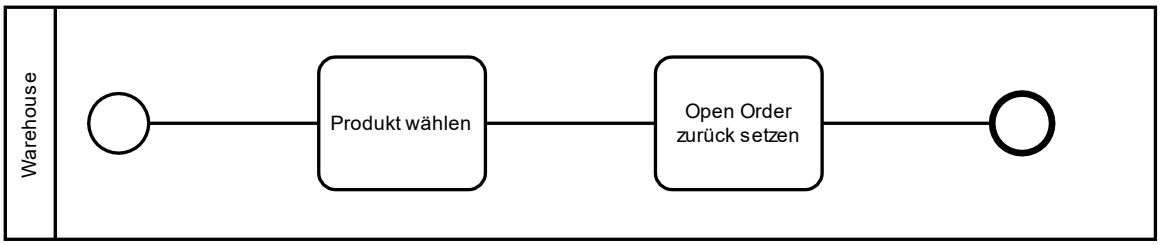


Abbildung 13: Warehouse Reset_OpenOrder

4.1.2.3 Check_OpenOrder

Kommt eine normale *Order* im *Warehouse* an, gibt es drei Endzustände für das *Warehouse*, wie in Abbildung 11 zu erkennen ist.

1. Die Bestellung kann nicht erfüllt werden, da der Lagerbestand für das Produkt nicht ausreichend ist. In diesem Fall wird eine *OrderDecline* Nachricht an den Bestellenden zurückgesendet.
2. Die Bestellung kann erfüllt werden und anschliessend sind noch genug Produkte im Lager, die über dem benutzerdefinierten Schwellwert liegen. In diesem Fall endet der Ablauf nach der Überprüfung, ob eine Nachbestellung notwendig ist.
3. Die Bestellung kann erfüllt werden, der Lagerbestand ist anschliessend aber unter dem Schwellwert. Nach dem Überprüfen, ob eine Nachbestellung notwendig ist, wird der *Check_OpenOrder* Prozess ausgeführt.

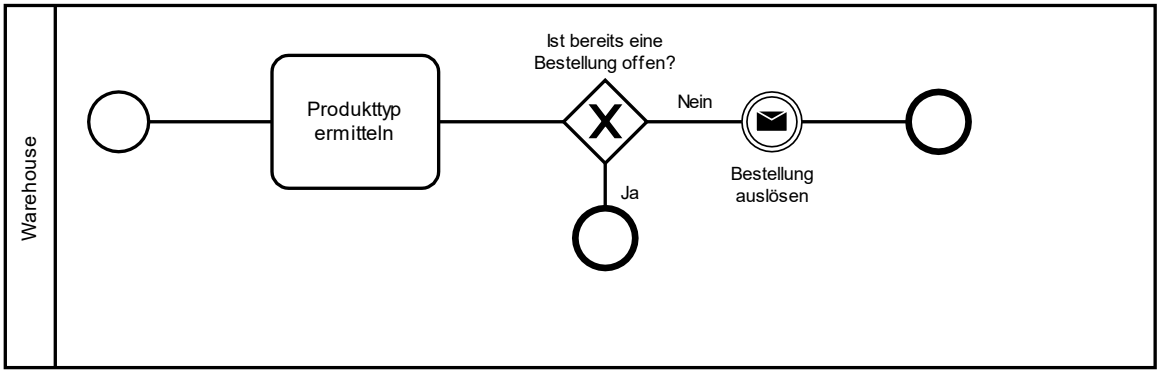


Abbildung 14: Warehouse Check_OpenOrder

Dieser Prozess überprüft das *OpenOrder* Kontrollflag, also ob bereits eine Bestellung für das ausgegangene Produkt versendet wurde. Ist dies nicht der Fall, wird die Bestellung des Produktes ausgelöst.

4.1.2.4 Reorder_Product

Der Nachbestellablauf entscheidet, bei welchem *Producer* nachbestellt wird. Er ist auch der Prozess, welcher die Bestellung abbricht, sollten keine Hersteller für das gewünschte Produkt verfügbar sein. Ist dies der Fall wird ein Kontrollflag auf *False* gesetzt, und es wird zum vorherigen Prozess zurückgegangen. Nach der Ausführung des Subprozesses wird dann überprüft, ob die Nachbestellung erfolgreich war. Ist dies nicht der Fall, wird der Prozess an dieser Stelle beendet.

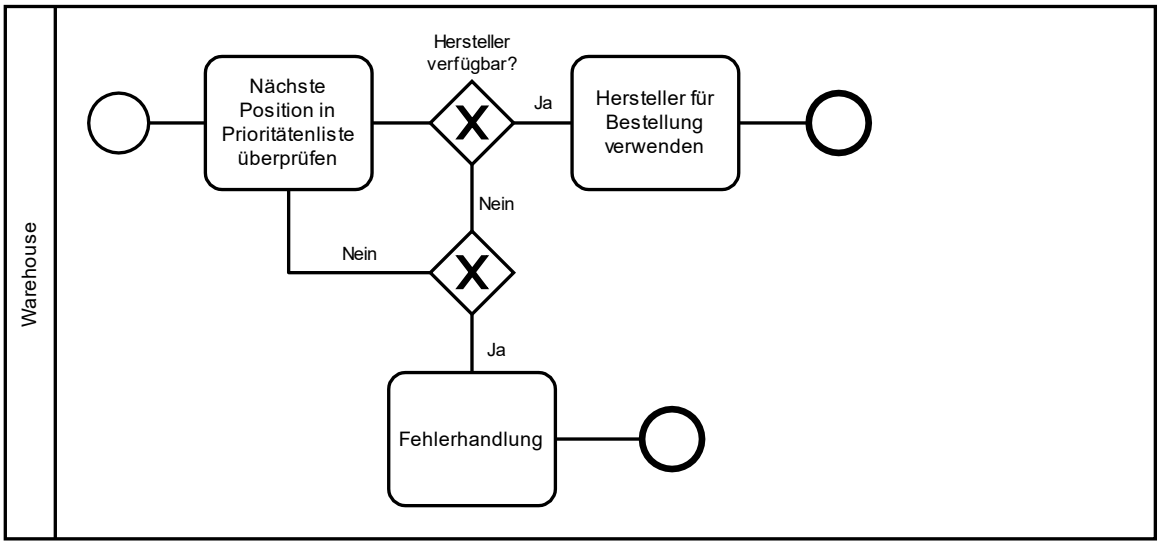


Abbildung 15: Warehouse Reorder Product

4.1.2.5 Order_Processing

Nach dem die *Order* erstellt wird, werden die benötigten Parameter auf die *Order* geschrieben und verschickt. Dazu gehören:

- Welches Produkt wird bestellt.
- Wie viel wird bestellt.
- In welcher Verpackung soll das Produkt geliefert werden.
- Wohin die *Order* verschickt wird.
- An welchen Stationseingang die Produkte gesendet werden sollen.
- An welchen Stationseingang *OrderDecline* Nachrichten gesendet werden sollen.

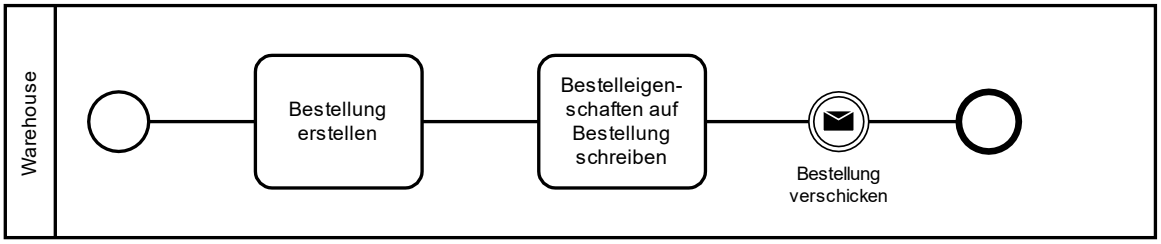


Abbildung 16: Warehouse Order_Processing

4.1.3 WARENFLUSS

Ein weiterer komplexer Prozess des *Warehouses* ist für die Bearbeitung ankommender Pakete verantwortlich. Die verschiedenen Schritte dieses Prozesses sind verteilt auf die einzelnen Teilstationen, aus denen das *Warehouse* besteht. Die Sendung wird von der *ReceivingArea* entgegengenommen und die Kosten an die entsprechenden CostCenter verrechnet. Dann werden Produkte von den Verpackungen getrennt (ausgepackt) und in den Store weitergeleitet.

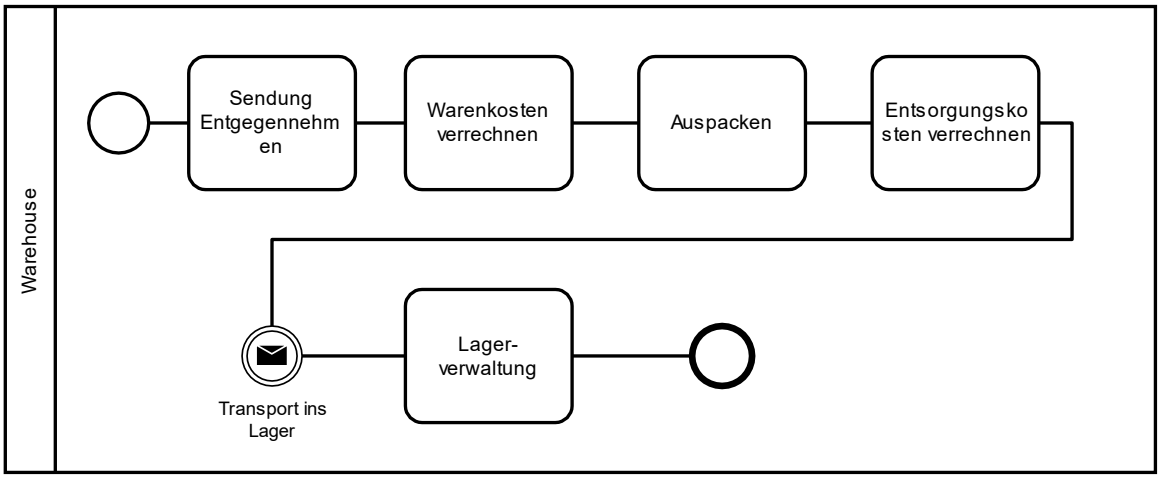


Abbildung 17: Warehouse Warenfluss

4.2 CUSTOMER

4.2.1 INFORMATIONSFLUSS

Dieser Prozess widmet sich dem Erstellen von Bestellungen, dazu gehört das Ausfüllen der Bestelldetails und das Verschicken der Bestellung.

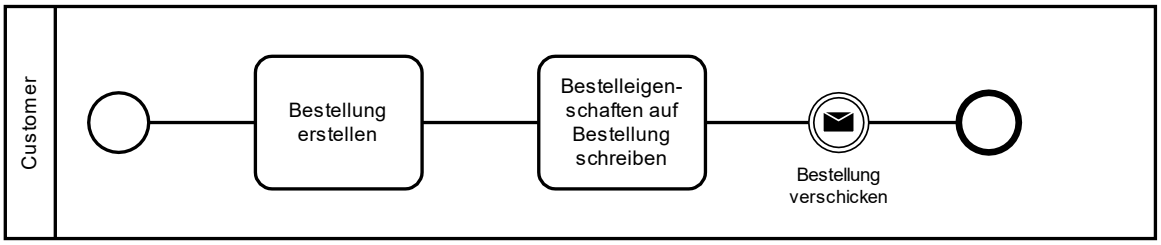


Abbildung 18: Customer Informationsfluss

Anders als zum Beispiel *Warehouse* und *ProducerAdvanced* reagiert der *Customer* nicht speziell auf *OrderDecline* Nachrichten. Diese werden lediglich gelöscht.

4.2.2 WARENFLUSS

Wenn Lieferungen entgegengenommen werden, werden diese ausgepackt und die Kosten an die entsprechenden CostCenter verrechnet.

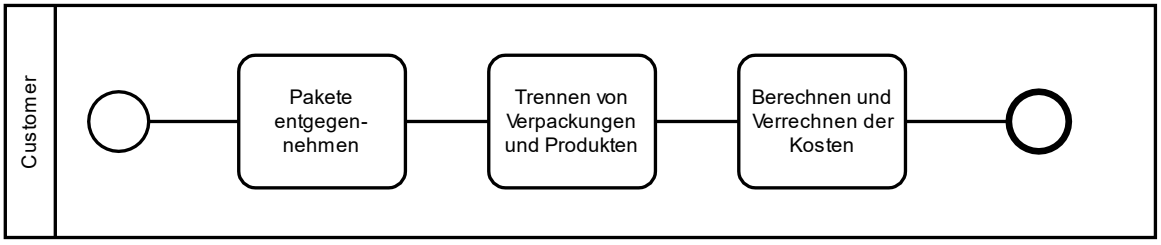


Abbildung 19: Customer Warenablauf

4.3 PRODUCER

4.3.1 ÜBERSICHT

Der *Producer* generiert Produkte, deren Herstellung nicht Teil der eigentlichen simulierten Supply Chain sind. Die Logik für die Beschaffung von Rohmaterialien für den *Producer* wird vernachlässigt, stattdessen besteht die Möglichkeit über die Angabe von fixen Herstellungskosten. Der

Producer dient in der Simulation der Erzeugung von Rohmaterialien für den *ProducerAdvanced*. Er kann Produkte und Verpackungen erzeugen, diese ineinander verpacken und verschicken.

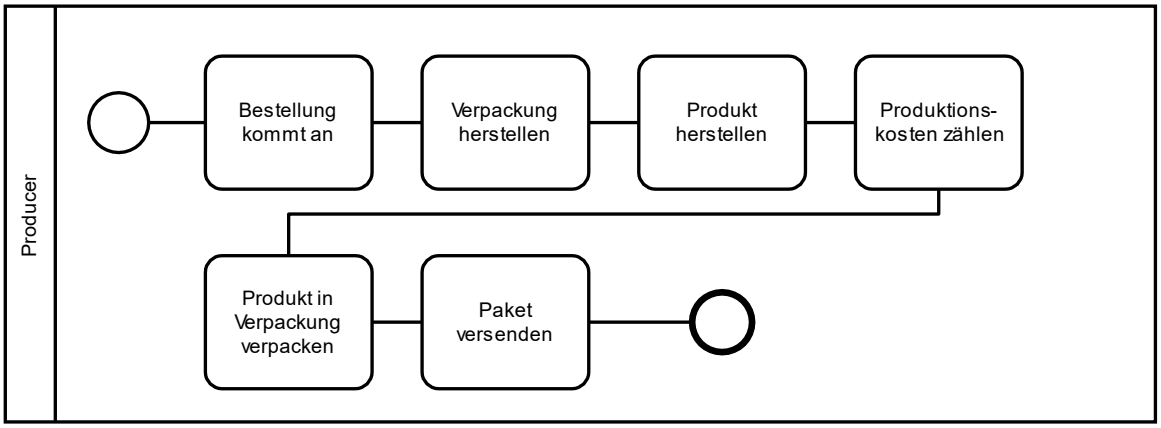


Abbildung 20: Producer Übersicht

Anders als beim *ProducerAdvanced* ist der Herstellungsprozess unabhängig von der Bestellmenge und dem bestellten Produkttyp.

4.4 PRODUCERADVANCED

4.4.1 ÜBERSICHT

Wie auch das *Warehouse* gehört der *ProducerAdvanced* zu den zwei grossen Stationen, die innerhalb der Supply Chain verwendet werden. Die Prozesse des *ProducerAdvanced* setzen sich aus drei Hauptprozessen zusammen: Einem Bestell-, einem Produktions- und einem Warenflussprozess. *Warehouse* und *ProducerAdvanced* unterscheiden sich darin, wie die einzelnen Abläufe zusammenhängen. Beim *Warehouse* laufen die Prozesse solange getrennt, bis der Bestellprozess Produkte aus dem Lager für den Versand markiert. Beim *ProducerAdvanced* hingegen wird der Produktionsprozess vom Warenflussprozess ausgelöst.

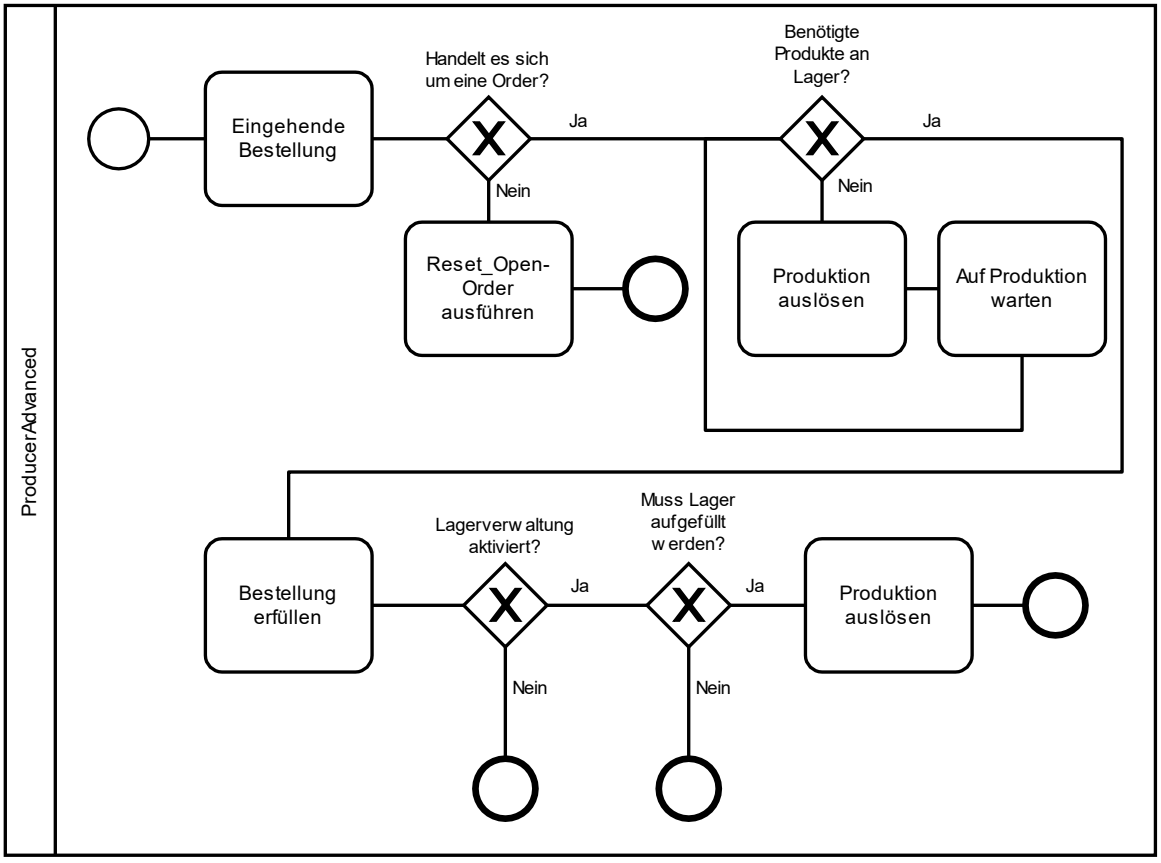


Abbildung 21: ProducerAdvanced Übersicht

4.4.2 INFORMATIONSFLUSS

Der Bestellprozess beschreibt, was passiert, wenn eine *Order* beim *ProducerAdvanced* eintrifft. Wie schon beim *Warehouse* gibt es auch beim *ProducerAdvanced* verschiedene Szenarien zu handhaben.

1. Die Bestellung kann nicht erfüllt werden, da der Lagerbestand für das Produkt nicht ausreichend ist. In diesem Fall wird der Produktionsprozess ausgelöst.
2. Die Bestellung kann erfüllt werden und anschliessend sind noch genug Produkte im Lager, die über dem benutzerdefinierten Schwellwert liegen. In diesem Fall endet der Ablauf nach der Überprüfung, ob eine Nachproduktion notwendig ist.
3. Die Bestellung kann erfüllt werden, der Lagerbestand ist anschliessend aber unter dem Schwellwert und die Lagerverwaltung ist aktiviert. Der Produktionsprozess wird ausgelöst.

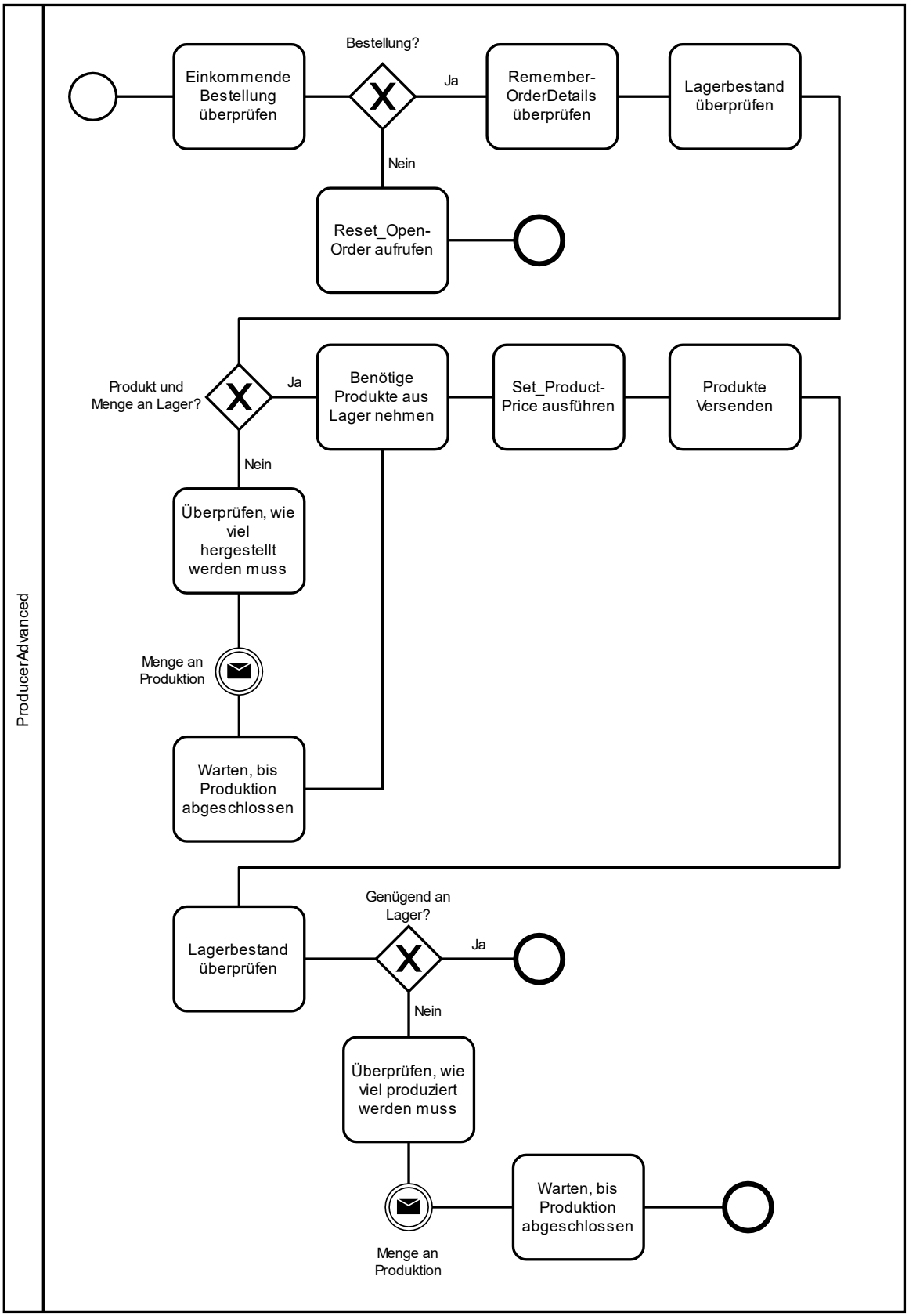


Abbildung 22: ProducerAdvanced Bestellablauf

4.4.3 PRODUKTION

Sind die bestellten Produkte an Lager, werden diese verrechnet und verschickt. Sind aber nicht genug an Lager, wird berechnet, wie viele Produkte hergestellt werden müssen, um die Bestellung zu erfüllen, und die Produktion ausgelöst.

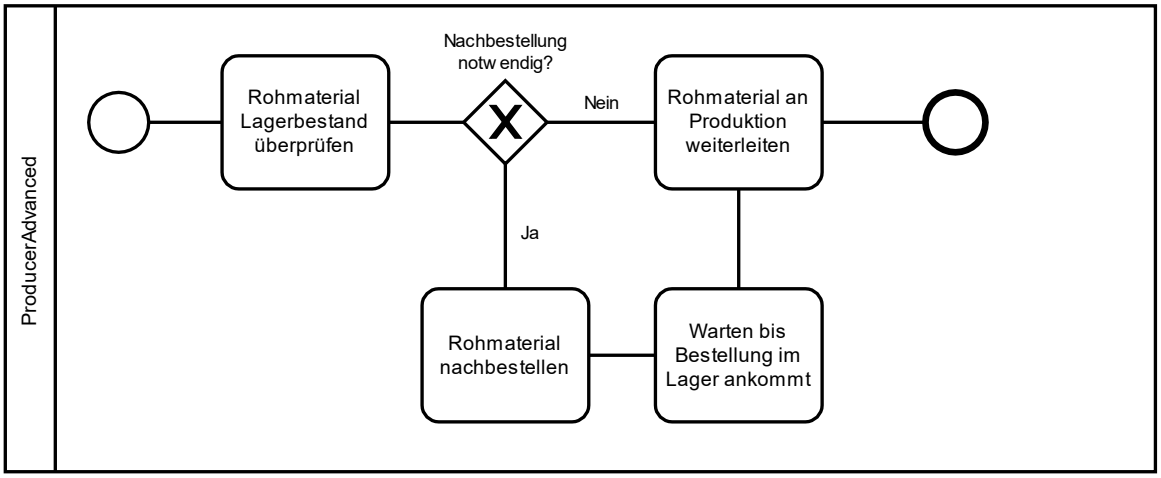


Abbildung 23: ProducerAdvanced Production

4.4.4 SUPPLIERCHOICE

Wird die Produktion ausgelöst, wird zunächst überprüft, ob die benötigten Rohmaterialien im Lager vorrätig sind. Ist dies nicht der Fall, wird der Bestellprozess ausgelöst und die Produktion pausiert, bis die bestellten Materialien im Lager eintreffen.

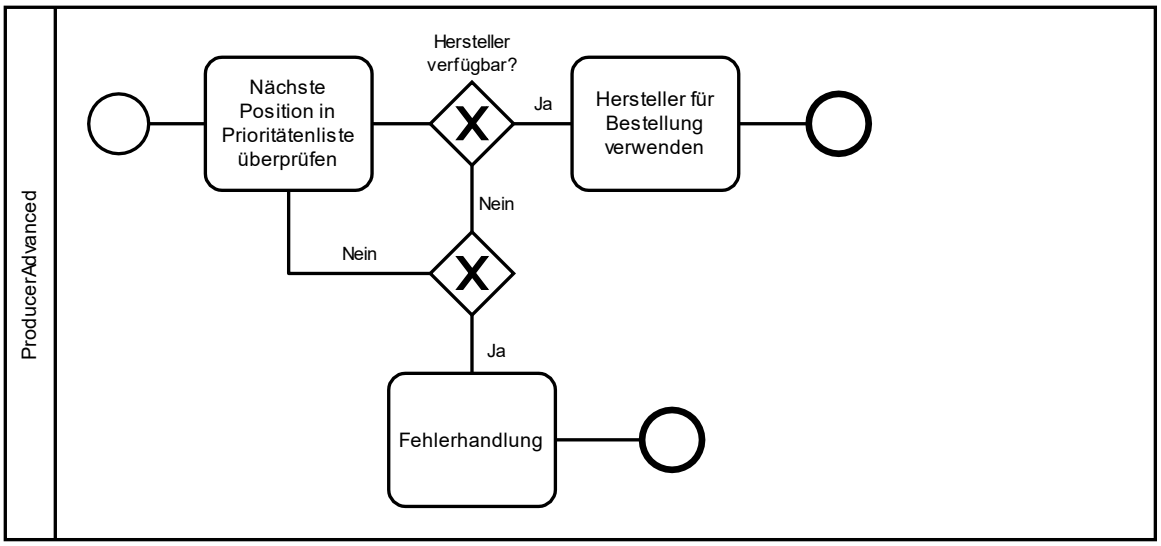


Abbildung 24: ProducerAdvanced SupplierChoice

Beim Nachbestellen von Rohmaterialen wird, wie schon beim *Warehouse*, ein Lieferant aus einer gewerteten Prioritätenliste ausgewählt. Der Prozess überprüft die ersten drei Einträge dabei auf Verfügbarkeit. Sollte keiner der Lieferanten verfügbar sein, so wird der Produktionsprozess abgebrochen.

4.4.5 LAGERVERWALTUNG

Lieferungen an den *ProducerAdvanced* werden von der *ReceivingArea* entgegengenommen, ausgepackt und die Rohmaterialien ans Lager weitergeleitet. In diesem Fall wird das Inventar für die betreffenden Produkte hochgezählt und die *ProductionUnit* über das Eintreffen der Produkte

informiert. Wenn Produkte verschickt werden oder Rohmaterialien an die Produktion weitergeleitet werden, wird das Inventar für die entsprechenden Produkte runtergezählt.

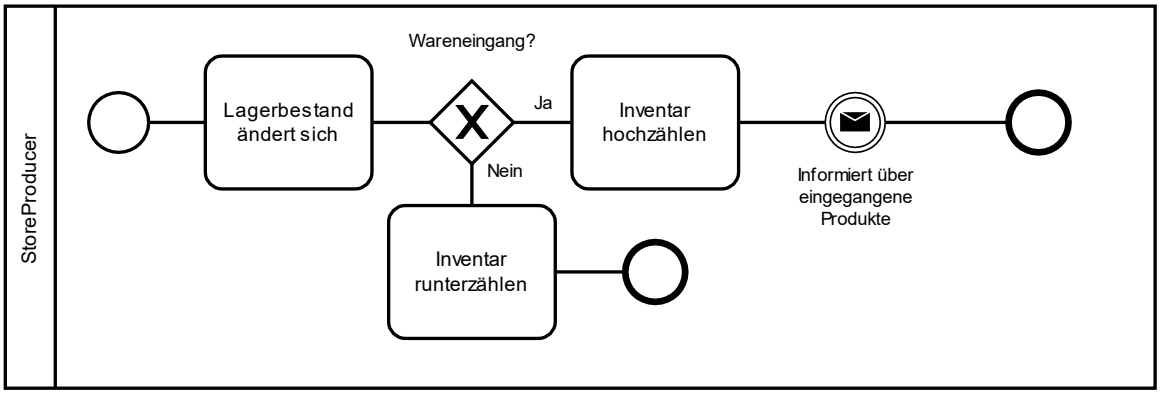


Abbildung 25: StoreProducer Lagerverwaltung

4.4.6 PRODUCTIONUNIT PRODUCTION

Ist die benötigte Menge an Rohmaterialien für die Produktion vorhanden, werden diese an die *ProductionUnit* gesendet und der Produktionsprozess wird gestartet.

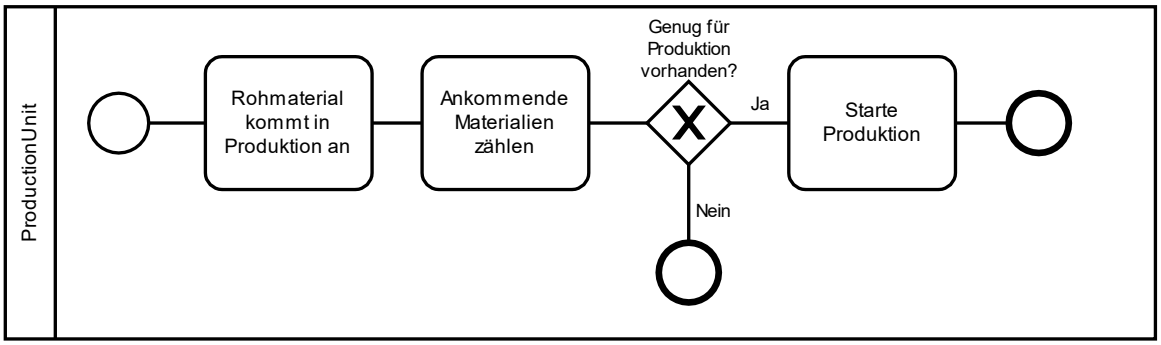


Abbildung 26: ProductionUnit Production

Es werden alle Rohmaterialien, die an die *ProductionUnit* gesendet werden, zu Produkten verarbeitet. Die erzeugten Produkte werden zurück ans Lager gesendet und der Management Prozess informiert.

5 ZUSAMMENFASSUNG UND AUSBLICK

In diesem Kapitel werden die gewählten Umsetzungen rückblickend besprochen und auf Features eingegangen, die nicht wie geplant umsetzbar waren. Am Ende des Kapitels wird angeschaut, was für zukünftige Arbeiten möglich ist und was dabei beachtet werden sollte.

5.1 ZUSAMMENFASSUNG

5.1.1 RÜCKBLICK AUF DIE GEWÄHLTE ARCHITEKTUR

Die gewählte Architektur der umgesetzten Prototypen basiert auf der Architektur, die während der vorangegangenen Studienarbeit vordefiniert worden ist. In diesem Kapitel sind Überlegungen geäussert, wie diese Architektur in zukünftigen Supply Chain Simulationen überarbeitet werden kann.

5.1.1.1 Vereinfachter Materialfluss

In den Prototypen 1-3 wird der Materialfluss tatsächlich logisch abgebildet: Jedes Produkt, das im System vorhanden ist, ist auch durch eine einzelne Entität dargestellt. Eine Vereinfachung für grössere Simulationen könnte sein, Produkte nur als Werteangaben auf den Sendungen zu Simulieren. Diese Sendungen müssten als neue Entity definiert werden.

5.1.1.2 Material und Inventory Elemente in Simio

Simio verfügt über Material und Inventory Elemente. Ein Inventory Element übernimmt die Verwaltung von Materialien, die an verschiedenen Orten gelagert werden, aber zum gleichen Inventar gehören. Die Art, wie Inventory Elemente Bestellungen und Nachbestellungen verarbeiten, ist nicht kombinierbar mit der in den Prototypen verwendeten Architektur. In den umgesetzten Prototypen werden Produkte von Station zu Station bewegt und dort jeweils zwischengelagert. Das Simio Element für Inventories funktioniert so, dass gleiche Materialien, die an verschiedenen Orten gelagert werden, zum gleichen Verwaltungsbereich gehören. Falls ein ähnliches Projekt durchgeführt wird, kann es von Wert sein, die Architektur um die Inventarverarbeitung dahingehend anzupassen, dass die Simio Elemente Inventory und Material verwendet werden können.

5.1.1.3 Bestellungsverarbeitung

Die umgesetzten Prototypen sind so eingeschränkt, dass die Stationen jeweils nur eine ankommende Bestellung verarbeiten können. Wenn mehrere Bestellungen parallel verarbeiten werden, entstehen Probleme mit den Search Schritten, die Produkte im Lager für den Transport suchen und markieren. Verdeutlicht wird dies in Abbildung 27. Es gibt einen kurzen Zeitraum, in dem ein Prozess (rot) die vordersten Produkte im Lager für den Transport markiert, jedoch noch nicht transferiert, da dies von einem Transfer Schritt übernommen wird. Greift nun ein anderer Prozess (lila) via eines Search Schritts auf dieselben Produkte zu, entsteht ein Fehler, da versucht wird, die gleichen Produkte zu markieren. Eine saubere Parallelisierung ist somit nicht möglich.

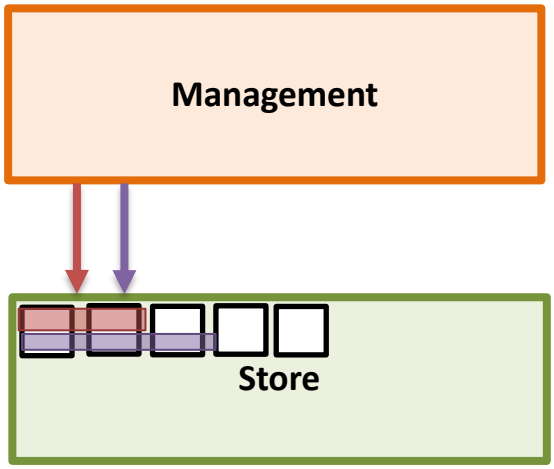


Abbildung 27: Search

Beim *Warehouse* werden Bestellungen entweder erfüllt (eine Lieferung wird versendet) oder abgesagt (eine *OrderDecline* Nachricht wird versendet). Eine Möglichkeit, dies zu umgehen ist, das Management mit einem zweiten Server zu erweitern. Dieser würde ankommende Bestellungen zwischenspeichern, die im Moment nicht erfüllt werden können. Der neue Server dient also als Warteschlange, die nicht erfüllbare Bestellungen zu einem späteren Zeitpunkt nochmals durch das eigentliche Management laufen lässt. Die zwischengespeicherten Bestellungen müssten regelmässig darauf überprüft werden, wie lange sie bereits offen sind. Wenn eine festgelegte Dauer überschritten wird, werden dann doch *OrderDecline* Nachrichten an den Bestellenden zurückgesendet.

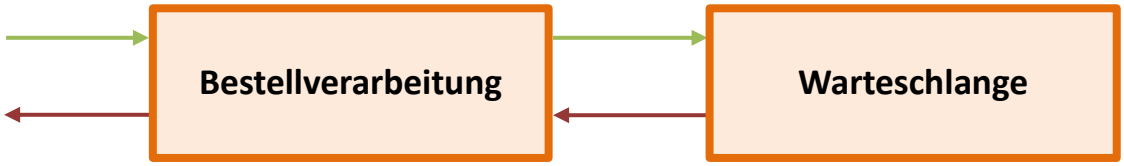


Abbildung 28: Zwischenspeicher

5.2 Weggelassene Features

Die nachfolgend besprochenen Features sind zum Teil deswegen nicht umgesetzt worden, da die Version, die für diese Arbeit zur Verfügung gestellt wurde, einige Features, nicht inbegriffen hat, die dafür von Relevanz gewesen wären. Andere sind an Einschränkungen gescheitert, die Simio von Haus aus mit sich bringt und für die eventuell selbst geschriebene PlugIns nötig wären, die den Umfang dieser Arbeit aber gesprengt hätten.

Eine entscheidende Option, die in der Education Version leider fehlte, ist die Möglichkeit Output Listen zu erstellen.

5.2.1 Dynamisches Label

Die im Modell verwendeten Label zur Darstellung der Lagerinhalte in den einzelnen Stationen ist nicht dynamisch umsetzbar. Eine schöne Lösung wäre gewesen, die Produktnamen aus der auf der *ProductEntity* hinterlegten Liste auszulesen. Simio bietet jedoch nicht die Möglichkeit List String Values auszulesen. Dies wäre eventuell mit einer C# Ergänzung möglich.

5.2.2 DUE TIME

Die Due Time, also die Überprüfung, ob Produkte in einem festgelegten Zeitfenster geliefert werden oder diesen überschreiten, war wegen fehlender Output Listen in der Education Version nicht möglich. Die Entreprise Edition oder ein Workaround wäre notwendig.

Der Workaround kann zum Beispiel so aussehen:

1. *DueTime* auf *Customer* hinterlegen
2. *CreationTime* auf *Order* schreiben
3. *CreationTime* auf Stationen zwischenspeichern
4. *CreationTime* auf Pakete übertragen
5. *Customer* bei ankommenden Paketen *CreationTime* mit *CurrentTime* vergleichen und Check machen, ob *DueTime* erfüllt oder nicht erfüllt ist für dieses Paket.

Dieser Workaround erlaubt es, Statistiken darüber zu führen, für wie viele Pakete die Due Time erfüllt wurde, gibt aber keinen Einblick über die Höhe der zeitlichen Überschreitungen von einzelnen Paketen.

5.2.3 DASHBOARDS

Das Erzeugen von Dashboards für zum Beispiel die Darstellung von Key Performance Indikatoren war nicht möglich, da hierfür Output Listen nötig wären.

Dashboards, die mit der Education Version umsetzbar wären, müssen auf Tally und State Observation Logs basieren.

5.2.4 HINTERLEGEN VON KARTEN

Für finalen Prototyp3 war eine Beispiel Supply Chain geplant mit einer hinterlegten Map, um reale Routen abbilden zu können. Dies ist tatsächlich einfach umsetzbar und wurde zu Anfang der Arbeit bereits angeschaut, um zu klären, ob und wie das möglich wäre. Diese Funktion war zum Zeitpunkt der Umsetzung von Prototyp3 nicht mehr möglich. Vermutlich sorgt ein damit verbundener Service für Probleme, die Priorität, dieses Problem zu beheben ist jedoch nicht hoch genug.

5.3 AUSBLICK

5.3.1 WIE WEITER MIT DER BESTEHENDEN ARCHITEKTUR

Die verwendete Architektur verarbeitet zwei verschiedene Bestellarten. Die Beim *Warehouse* verwendete Variante eins, ist es Bestellungen zu erfüllen, falls die gewünschten Produkte an Lager sind und eine *OrderDecline* Nachricht zu senden, falls nicht. Das Versenden von *OrderDecline* Nachrichten löst eine Nachbestellung für die fehlenden Produkte aus.

Die Zweite Variante wird beim *ProducerAdvanced* verwendet und garantiert das Erfüllen der Bestellung auch bei fehlendem Produkt im Lager. Fehlt das Produkt, so wird dessen Herstellung veranlasst. Selbst wenn dafür zuerst Rohmaterial nachbestellt werden muss.

Ein Sonderfall ist, wenn beim *ProducerAdvanced* Verpackungen bestellt werden. In diesem Fall wird Variante 1 verwendet, die Bestellung also mit einer *OrderDecline* Nachricht abgesagt, wenn nicht genug vorrätig sind.

Die *OrderDecline* Nachrichten, welche der bestellenden Station mitteilen, dass eine Bestellung nicht erfüllt werden kann, erfüllen eine ähnliche Funktion wie die *Availability* Überprüfung. Die *Availability*

Überprüfung geschieht jeweils vor Abschicken einer Bestellung, wohingegen *OrderDecline* Nachrichten nach Ankunft einer Bestellung als Antwort zurückgeschickt werden.

Beide Varianten sind nicht ideal, da sie zu langen Verzögerungen führen. Für den Bestellenden wäre es optimal, bereits vor der Bestellung zu wissen, ob ein Zulieferer die Bestellung sofort erfüllen kann. In zukünftigen Versionen ist es zu empfehlen, die Funktionalität der *OrderDecline* Nachrichten auf die *Availability* umzulagern.

5.3.2 Erweiterung des Availability-States

Die Availablity Properties und States auf den Stationen *ProducerAdvanced*, *Producer* und *Warehouse* ermöglichen während der Simulation eine sofortige Verfügbarkeitsprüfung. Eine vorstellbare Erweiterung ist, den Verfügbarkeitsstatus nicht nur über die gesamte Station anzugeben, sondern auch für einzelne Produkte. Ein dynamisches Setzen des Status während der Simulation wäre ebenfalls von Vorteil.

Beim *Warehouse* kann dies zum Beispiel so gelöst werden, indem der Verfügbarkeitsstatus auf *False* gesetzt wird, sobald der Lagerbestand für ein Produkt unter einen gewissen Schwellwert fällt. Der Nachteil dieses Ansatzes ist, dass möglicherweise ein Restbestand im *Warehouse* zurückbehalten wird. Bei diesem Ansatz steigt allerdings die Gefahr, eine Bestellung zu erhalten, die mehr Produkte verlangt als im Lager vorrätig sind. Ein Lösungsansatz für beide Probleme ist, den *Availability* Status nicht als Boolean sondern als Integerwert zu führen. So kann bei einer Bestellung überprüft werden, ob beim Hersteller genug Produkte an Lager sind. Auch dieser Ansatz ist nicht optimal, da es immer vorkommen kann, dass mehrere Kunden fast gleichzeitig bestellen und der Wert der verfügbaren Produkte erst angepasst wird, nachdem die Bestellung erhalten und verarbeitet worden ist. *OrderDecline* Nachrichten können so zwar reduziert werden, werden aber auch bei optimiertem Aufbau immer vorhanden sein.

5.3.3 Zwischenspeichern der Bestellungen beim ProducerAdvanced

Wie in Kapitel 5.2.1 besprochen, werden Bestellungen beim *ProducerAdvanced* immer erfüllt. Ausgenommen dann, wenn Verpackungen nachbestellt werden müssen. Ein realistischerer Ansatz ist, eine Warteschlange für Bestellungen zu erstellen, die das Zwischenspeichern von Bestellungen ermöglicht. Bestellungen, welche aufgrund von fehlenden Verpackungen nicht erfüllt werden können, werden solange zwischengespeichert, bis die nachbestellten Verpackungen eingetroffen sind.

5.3.4 Erweiterung des Modells automatisieren

Im Rahmen der Arbeit wurden Simio Features darauf untersucht, das automatische Generieren von Simio Objekten ermöglichen zu können. Ursprünglich geplant war, die Erweiterung von Prozessen und Listen beim Einbringen von weiteren Produkten zu vereinfachen.

Ohne zusätzliche Erweiterungen, zum Beispiel externe Plugins für Simio, bietet Simio lediglich die Möglichkeit, Elemente und Objekte (Stations, Nodes & Paths) zu erstellen. Für das Erweitern des Modells um zusätzliche Produkte, müssen jedoch Prozesse, Properties und States generiert werden. Vor allem die Prozesse sind komplex und es wäre von Vorteil, wenn die Produktabhängigen Teile automatisch erzeugt werden könnten.

Zum Beispiel die Nachbestellprozesse, welche den ersten verfügbaren Zulieferer aus einer Liste aussuchen.

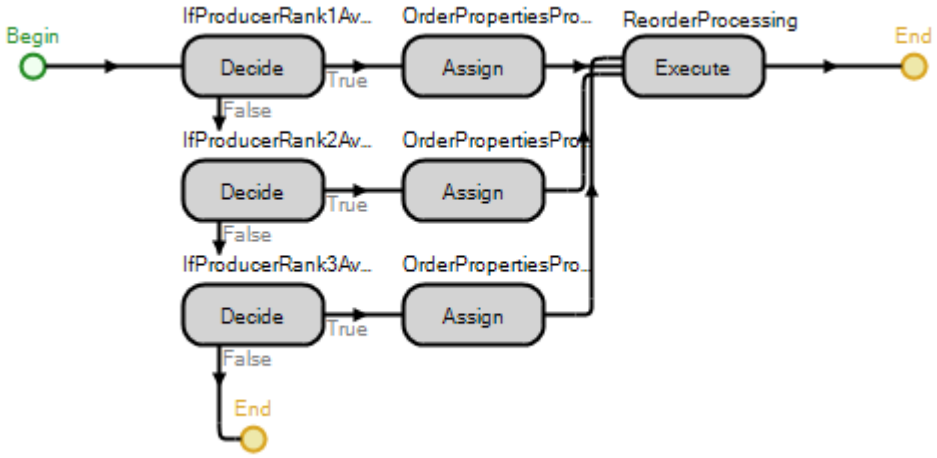


Abbildung 29: Chose Supplier Prozess

Der in Abbildung 29 gezeigte Prozess muss jedes zusätzliche Produkt neu erstellt werden. Es ist auch für jedes neue Produkt eine zusätzliche Tabelle notwendig, die Auskunft über die Zulieferer gibt. Für diese Schritte ist auch die Anpassung der Entscheidungsmatrix notwendig. Wurde das Excel ergänzt, so kann die Tabelle mit richtigen Spaltentypen vorbereitet und aufs Excel verlinkt werden.

	Supplier	Address	Expression Supplier
1	ProducerWater2	Input_Management@ProducerWater2	ProducerWater2.Producer.Availability
2	ProducerWater3	Input_Management@ProducerWater3	ProducerWater3.Producer.Availability
3	ProducerWater	Input_Management@ProducerWater	ProducerWater.Producer.Availability
4	NoSupplier	Input_Management@NoSupplier	NoSupplier.Producer.Availability

Abbildung 30: Supplier Table

Ist die Tabelle erstellt, muss auf dem *Warehouse* ein neues Table Property erstellt werden. Dann erst wird der Prozess erstellt. Eine Auflistung der notwendigen Angaben und eine genauere Beschreibung ist im Handbuch beschrieben.

5.3.5 AUTOMATISCHE BERECHNUNG VON LIEFERMENGEN

Im Prototyp3 werden sowohl beim *Warehouse* als auch beim *ProducerAdvanced* für Nachbestellungen fixe Liefermengen verwendet. Diese werden am Anfang der Simulation vom Benutzer festgelegt. Optimal wäre, wenn diese Liefermengen nicht statische, sondern dynamische Werte sind, welche während der Simulation vom Modell selbst berechnet werden. Nachfolgend sind zwei einfache Ansätze besprochen, wie dies umgesetzt werden könnte.

Zum einen könnten die Nachbestellmengen auf den Mengen der eingehenden Kundenbestellungen basieren. So wird sichergestellt, dass eine einzelne Nachbestellung ausreicht, um den Bedarf des Kunden zu erfüllen.

Ebenfalls möglich ist es, die Anzahl Nachbestellungen für ein Produkt während eines bestimmten Zeitraumes zu zählen. Basierend darauf werden dann Anpassungen an den Bestellmengen vorgenommen. Zum Beispiel wird gezählt, dass ein Produkt dreimal täglich nachbestellt wird. Nun kann entweder die Bestellgrösse um einen fixen Wert erhöht werden oder direkt verdreifacht werden, bis nur noch einmal pro Tag nachbestellt wird.

In weiterführenden Schritten können die Werte noch genauer berechnet werden. Anstatt die Häufigkeit von Bestellungen pro Tag zu berechnen, kann ein Mittelwert über alle Bestellungen über einen längeren Zeitraum erstellt werden. Zum Beispiel aus Rücksicht auf saisonale Unterschiede, wird der Mittelwert pro Kalenderwochen erstellt.

5.3.6 Überarbeitung Lagerungskosten

Im Prototyp3 werden die Lagerungskosten linear pro Produkt im Lager über den gesamten Zeitraum der Lagerung berechnet. Die verursachten Kosten werden über die CostCenter des Stores verrechnet. Dies ist ein einfacher Ansatz eines komplexeren Problems: Es wird davon ausgegangen, dass jeder Produkttyp die gleichen Lagerungskosten verursacht. In der Realität ist dies nicht der Fall. Verschiedene Produkte verursachen verschiedene Lagerkosten und auch die Dauer der Lagerung hat einen nicht linearen Einfluss auf die Kosten.

Es gibt verschiedene Möglichkeiten, wie diese Ungenauigkeit in zukünftigen Versionen verringert werden kann.

Ein allgemeiner Kostenwert kann verwaltet werden. Dieser berechnet die durchschnittlichen Lagerkosten pro Stunde über alle Produkte, welche sich derzeit im Lager befinden.

Eine andere Möglichkeit ist, den Vorgang der Lagerkosten Berechnung in einem Prozess abzubilden und diesen regelmässig auszuführen. Der Prozess würde dabei das Inventar pro Produkt auslesen und mit den produktspezifischen Lagerungskosten berechnen.

5.3.7 Entsorgungsmanagement

Das Entsorgungsmanagement ist nur rudimentär umgesetzt. Die Entsorgungskosten entstehen beim Entsorgen von einmal verwendbaren Karton-Verpackungen und werden als Fixkosten pro zerstörte Verpackung an das CostCenter für Entsorgungskosten verrechnet.

In zukünftigen Versionen kann das Entsorgungsmanagement weiter ausgebaut werden. Zum Beispiel kann eine allgemeine Entsorgungsstelle als zusätzliche Station im System definiert werden. Mit diesem Ansatz können Transportkosten und Entsorgungskosten dynamischer berechnet werden. Transportkosten könnten zum Beispiel auf der Distanz zwischen der Entsorgungsstelle und den betroffenen Stationen basieren. Weiter könnten die Entsorgungskosten dynamisch während der Simulation von der Entsorgungsstelle angepasst werden. Bereits gebrauchte Verpackungen können als «entsorgungsbereit» markiert werden und direkt and die Entsorgungsstelle gesendet werden.

Um zum Beispiel verderbliche Produkte abbilden zu können, kann das System durch das Integrieren von Ablaufzeiten auf den Produkten erweitert werden. Bei den entsprechenden Produkten wird bei der Erstellung ein Ablaufdatum als Property gespeichert. An verschiedenen Stellen in der Simulation muss ein Check vorgenommen werden, der überprüft, ob das Datum überschritten wurde. Das Produkt muss dann entsorgt werden, sobald dieses Ablaufdatum überschritten wird.

6 VERZEICHNISSE

6.1 ABBILDUNGSVERZEICHNIS

6.2 TABELLENVERZEICHNIS

6.3 QUELLENVERZEICHNIS

LLC, S., 2021. *Simulation Modeling With Simio: A Workbook | Simio*. [online] Simio.com. Available at: <https://www.simio.com/publications/index.php> [Accessed 21 September 2020 – 7. Dezember 2020].

Demo.bpmn.io. 2020. *BPMN Editor | Demo.Bpmn.Io*. [online] Available at: <https://demo.bpmn.io/s/start> [Accessed 8 November 2020].

Simio Forum. 2021. *Difference Between Entity, Model Entity And Defaultentity*. [online] Available at: <https://www.simio.com/simio-forum/topic/1356-difference-between-entity-model-entity-and-defaultentity/> [Accessed 5 October 2020].

Simio Forum. 2021. *[CLOSED] Creating Label That Shows List String*. [online] Available at: <https://www.simio.com/simio-forum/topic/495-closed-creating-label-that-shows-list-string/> [Accessed 20 November 2020].

Simio Forum. 2021. *Entity Selection From Queue*. [online] Available at: <https://www.simio.com/simio-forum/topic/2816-entity-selection-from-queue/> [Accessed 6 October 2020].

Simio Forum. 2021. *Multiple Waits*. [online] Available at: <https://www.simio.com/simio-forum/topic/3296-multiple-waits/> [Accessed 20 October 2020].

Simio Forum. 2021. *Extracting Calendar Attributes From Timenow Variable*. [online] Available at: <https://www.simio.com/simio-forum/topic/384-extracting-calendar-attributes-from-timenow-variable/> [Accessed 25 November 2020].

3dwarehouse.sketchup.com. 2021. *House | 3D Warehouse*. [online] Available at: <https://3dwarehouse.sketchup.com/model/95fe34021c0b02d467e6ecb93056efe6/House> [Accessed 30 November 2020].

Copniker, D., 2021. *Warehouse | 3D Warehouse*. [online] 3dwarehouse.sketchup.com. Available at: <https://3dwarehouse.sketchup.com/model/c72c7401dd88007c102960100812011b/Warehouse> [Accessed 30 November 2020].

3dwarehouse.sketchup.com. 2021. *Unsupported Browser | 3D Warehouse*. [online] Available at: <https://3dwarehouse.sketchup.com/model/f1f337988b9bfd6d8e8f4bdb41ca564b/factory> [Accessed 30 November 2020].

6.3.1 SIMBIT VORLAGEN

Im Laufe der Arbeit sind die nachfolgenden, von Simio bereit gestellten SimBit Beispiele zu Rate gezogen worden:

- Financials.spfx
- InventoryAndMaterials.spfx

7 ANHANG

I. ARBEITSAUFTRAG

Bearbeitung durch

Fabienne Lienhard	fabienne.lienhard@ost.ch
Jana Kravarik	jana.kravarik@ost.ch

Termine

Ausgabe:	21. September 2020
Abgabe:	15. Januar 2021

Betreuung

Prof. Dr.-Ing Andreas Rinkel	andreas.rinkel@ost.ch

Experte

Knut Schmahl	knut.schmahl@yahoo.com

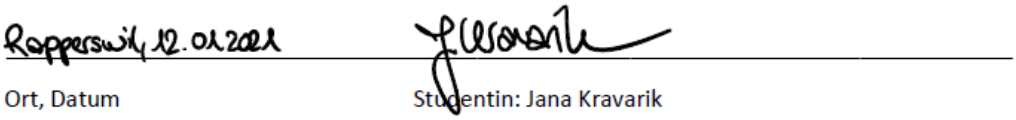
Gegenleserin

Prof. Dr. Nathalie Weiler	nathalie.weiler@ost.ch

Aufgabe

Im Rahmen der Arbeit ist das in der Studienarbeit «Modell Baukasten Supply Chain» entwickelte Konzept zur Simulation von Supply Chains in Simio prototypisch umzusetzen. Dazu gehört die Planung und Definition der Prototypenfolge mit inkrementellem Funktionsumfang.

Ort, Datum — Studentin: Fabienne Lienhard



Ort, Datum — Studentin: Jana Kravarik

Ort, Datum — Betreuer: Andreas Rinkel

Implementierung eines Supply-Chain-Management-Simulationstools

Handbuch

Studiengang Informatik
OST – Ostschweizer Fachhochschule
Campus Rapperswil-Jona

Herbstsemester 2020

Autoren:	Fabienne Lienhard, Jana Kravarik
Betreuer:	Prof. Dr.-Ing. Andreas Rinkel
Experte:	Knut Schmahl
Gegenleserin:	Prof. Dr. Nathalie Weiler
Abgabe:	15.01.2021

INHALTSVERZEICHNIS

8 EINLEITUNG

Dieses Dokument dient als Anleitung zur Bedienung, sowie Anpassung der in der Bachelorarbeit «Implementierung eines Supply-Chain-Management-Simulationstools» entwickelten Prototypen 1 – 3 und dem dazugehörenden Excel Decision_Matrix.

Wichtig: Es hat sich gezeigt, dass in unterschiedlichen Simio Versionen die Funktionalität der Prototypen beeinträchtigt sein kann. Es ist daher empfohlen, beim Bedienen der Prototypen die folgende Simio Version und Lizenz zu benutzen:

Version: 12.205.21521

License: Student

9 BEGRIFFE UND DEFINITIONEN

In diesem Kapitel werden die verwendeten Begriffe und Definitionen erläutert.

9.1 DARSTELLUNG

Um die Leserlichkeit zu erhöhen, sind bestimmte Begriffe speziell hervorgehoben. Diese Hervorhebung soll dem Leser verdeutlichen, in welcher Kategorie der Begriff einzuordnen ist. Unterschieden wird zwischen drei Kategorien:

Simio Begriffe Begriffe, die für Stationen und Kategorien gelten, die von Simio verwendet werden.

Zum Beispiel State, Server, Process, ...

Stationsnamen Während der Arbeit neu definierte und erstellte Simio Objekte.

Zum Beispiel *ProducerAdvanced, Warehouse, Unpack, …*

Eigennamen Während der Arbeit neu definierte Namen für Prozesse, Prozessschritte, Listen, States, Events, Properties und Elemente.

Zum Beispiel *OpenOrder, RememberOpenOrder, InitialAvailability*

9.2 SUPPLY CHAIN

An dieser Stelle sei auf das Kapitel 2 Supply Chain Grundlagen der vorangegangen Studienarbeit verwiesen, um eine ausführlichere Beschreibung zu erhalten. Für die in dieser Arbeit umgesetzten Prototypen stellt sich die Supply Chain aus drei verschiedenen Hauptstationen zusammen.

4. Dem Kunden, der Produkte einkaufen möchte und Bestellungen verschickt.
5. Dem Warenhaus, das Bestellungen entgegennimmt, dieser erfüllt oder Waren nachbestellt, um sie erfüllen zu können.
6. Dem Produzenten, der Bestellungen entgegennimmt und Produkte herstellt, um die Nachfrage zu stillen. Auch der Produzent verschickt Bestellungen für Rohmaterialien, die er für die Produktion benötigt.

Die Stationen sind verbunden mit einem Informations- und einem Warenfluss.

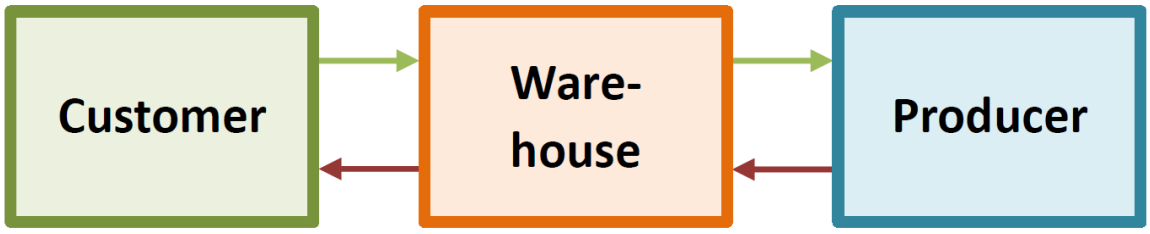


Abbildung 31: Supply Chain, vereinfachte Ansicht

9.3 ENGLISCH-DEUTSCHE BEGRIFFE

In den nachfolgenden Kapiteln werden für die einzelnen Komponenten und Kernelemente die englischen Begriffe, sowie Abkürzungen verwendet. Nachfolgend eine Übersicht der Begriffe mit der deutschen Übersetzung und einer kurzen Beschreibung.

Begriff	Übersetzung	Beschreibung
BPMN	*BPMN*	Steht für Business Process Model and Notation. Dies ist eine grafische Notation, um Prozesse zu beschreiben.
Batch	*Auflage*	Produktionsauflage, die während einem Produktionsvorgang hergestellt wird.
CostCenter	*Kostenstelle*	Einheit, an die bestimmte Kosten verrechnet werden.
Counter	*Zähler*	Nummer, welche erhöht wird, um etwas zu zählen.
Customer	*Kunde*	Während Arbeit definierte Station, welche dem Endkunden einer Supply Chain entspricht.
Data Table	*Datentabelle*	Data Table, ist eine von Simio zur Verfügung gestellte Tabellenart. Diese Tabellen können ausgefüllt, importiert, exportiert und als Table Objekte verwendet werden.
Events	*Ereignis*	Beim Auftreten einer bestimmten Gegebenheit, kann ein Event ausgelöst werden und an einem anderen Ort registriert werden. Dies erlaubt, auf etwas zu reagieren, das in einem anderen Teil der Simulation passiert.
FinalProduct	*Endprodukt*	Ein Produkt, das aus etwas anderem hergestellt wird. Bei den umgesetzten Prototypen handelt es sich dabei um ein Produkt, welches aus zwei Rohmaterialien bei einem *ProducerAdvanced* hergestellt wird.
Flag	*Kennzeichen/ Merker*	Kennzeichen, welches als Boolean (Wahr / Falsch) gespeichert wird. Einfacher Status mit zwei Zuständen.
InputNode	*Eingangsknoten*	Schnittstelle, über die Einheiten eine Station betreten können.
Lists	*Listen*	Listen in Simio sind einfache Aufzählungen von entweder Strings, Knoten oder Objekten. Sie können von Properties und States als zugelassene Wertelisten verwendet werden.
Loop	*Schlaufe*	Ein Ablauf, der wieder zurück auf seinen eigenen Anfang führt. Meist wird dieser so lange ausgeführt, bis eine bestimmte Bedingung erfüllt ist.
Node	*Knoten*	Verbindungsknoten sind Simio Objekte, die das Verbinden mit Pfaden erleichtern.
OutputNode	*Ausgangsknoten*	Schnittstelle, über die Einheiten eine Station verlassen können.

Order / OrderDecline	*Bestellung / Bestellungsabsage*	Eine *Order* ist eine Bestellung, die von einer Station an eine andere Station geschickt wird und diese auffordert der ersten Station Produkte zu liefern. Die *OrderDecline* Nachricht wird als Reaktion auf eine Bestellung versendet, die die Station nicht erfüllen kann.
Pack	*Verpackungsstation*	Während der Arbeit erstellte Station, die als Teilbaustein einer anderen Station verwendet wird. Sie hat die Aufgabe, Produkte auszupacken.
Path	*Pfad*	Verbindungsweg zweier Knoten, die für Verbindungen zwischen Stationen und innerhalb von Stationen verwendet wird.
Producer	*Hersteller*	Während der Arbeit erstellte Station, welche einem Hersteller in einer Supply Chain entspricht.
ProducerAdvanced	*Erweiterter Hersteller*	Während der Arbeit erstellte Station, welche eine Erweiterung der *Producer* Station ist.
Product (Entity)	*Produkt (Einheit)*	Während der Arbeit erstellter Objekttyp, der Produkte simuliert.
ProductionSize	*Produktionsmenge*	Steht für die Anzahl, wie oft die Produktion angestossen werden soll.
ProductionUnit	*Produktionseinheit*	Während der Arbeit erstellte Station, die ein Teilbaustein der Station *ProducerAdvanced* ist. Die Station enthält den Herstellungsprozess, um aus zwei Produkten ein anderes Produkt herzustellen.
Property	*Eigenschaft*	Eigenschaften einer Station oder eines Simio Objektes. Eigenschaften haben einen Typ (z.B. Integer) und einen Wert (z.B. 20).
Process	*Ablauf*	Vorgegebene oder während der Arbeit definierte Abläufe, welche immer in der gleichen Reihenfolge am gleichen Punkt der Simulation ausgeführt werden. Diese Abläufe sind über einzelne Schritte genauer definiert.
RawMaterial	*Rohmaterial*	Produkte, die für die Herstellung eines anderen Produktes benötig und dabei verbraucht werden.
ReceivingArea	*Wareneingang*	Während der Arbeit erstellte Station, die als Annahmestelle für Sendungen verwendet wird.
Server	*Server*	Simio Objekt, welches Entities entgegennimmt und verarbeitet.
ShippingArea	*Wareausgang*	Während der Arbeit definierte Station, die Sendungen aus einer Station verschickt. Die *ShippingArea* enthält eine *Pack* Station, welche die Produkte verpackt.
State	*Zustand*	Zustände können wie Eigenschaften auf Simio Objekten definiert werden. Im Gegensatz zu den Eigenschaften können diese aber während der Simulation dynamisch gesetzt werden.

Storage-Management	*Lagerverwaltung*	Die Lagerverwaltung ist eine Zusatzfunktion auf dem *ProducerAdvanced*. Ist sie aktiviert so verwaltet der *ProducerAdvanced*, wie viele Produkte sich in seinem Lager befinden. Der Lagerbestand wird dann immer nach dem Erfüllen einer Bestellung überprüft. Liegt er unter einem definierten Schwellwert, so wird eine ebenfalls definierte Menge an Produkten bei der Produktion in Auftrag gegeben.
StoreProducer	*Herstellerlager*	Während der Arbeit erstellte Station, die ein Teilbauteil einer *ProducerAdvanced* Station ist. Die Station lagert Produkte ein und aus.
StoreWarehouse	*Lagerhauslager*	Während der Arbeit definierte Station, die ein Teilbaustein einer *Warehouse* Station ist. Die Station lagert Produkte ein und aus.
Supplier	*Lieferant*	Ein Lieferant beschreibt eine *Warehouse*, *Producer* oder *ProducerAdvanced* Station. Sie versorgen andere Stationen mit Produkten.
Supply Chain	*Lieferkette*	Siehe Kapitel 2.2.
TargetStock	*Zielbestand*	Gibt die Menge der Produkte an, welcher der *ProducerAdvanced* nach dem Erfüllen einer Bestellung noch im Lager haben muss. Dieser Wert wird nur verwendet, wenn die Lagerverwaltung aktiviert ist. (siehe auch Lagerverwaltung)
Unpack	*Auspackstation*	Während der Arbeit definierte Station, welche ineinander verpackte Produkte trennt.
Warehouse	*Lagerhaus / Lager*	Während der Arbeit erstellte Station, die einem Lager und einer Verkaufsstation entspricht.

Tabelle 4: Englisch-Deutsche Begriffe mit Beschreibung

10 BEDIENUNG: PROTOTYP1

Prototyp1 ermöglicht dem Benutzer eine einfache Supply Chain zu erzeugen mit beliebig vielen *Customers*, *Warehouses* und einem *Producer* am Ende der Supply Chain. Es werden der Bestellungs- und der Warenfluss abgebildet.

Customer können ihre Waren bei einem *Warehouse* bestellen und *Warehouses* wiederum können ihre Waren entweder bei einem anderem *Warehouse* oder dem *Producer* bestellen.

Es gibt die Möglichkeit, verschiedene Produkte und Verpackungen zu definieren, der *Producer* kann jedoch nur einen Produkttyp produzieren. Die *Warehouses* verfügen über ein geteiltes Lager und können Produkte, Karton und Container separat einlagern und nachbestellen.

Die Produkte werden entweder in Karton oder in Containern zwischen den Stationen verschickt. Container sind wiederverwendbar und werden nach Gebrauch beim Empfänger eingelagert. Kartons werden entsorgt.

Es sind folgende CostCenter implementiert, die Angaben über verschiedene Kosten ermöglichen:

Cost Center	Aufruf
CO_2 Kosten bei der Verpackung	*WareHouse1*.**CostCenterCO2Packaging.Cost**
CO_2 Kosten bei der Produktherstellung	*Producer1*.**CostCenterCO2Material.Cost**
Lagerkosten auf Zeit	*WareHouse1*.**CostCenterStorage.Cost**
Materialkosten	*Producer1*.**CostCenterCO2Material.Cost**
Produktionskosten	*Producer1*.**CostCenterProduction.Cost**
Produkteinkommen	*Customer1*.**CostCenterProductIncome.Cost**
Produktertrag	*Customer1*.**CostCenterProductIncome.Cost** *-Producer1*.**CostCenterProduction.Cost**
Entsorgungskosten für Karton	*Customer1*.**CostCenterWastePackaging.Cost** *Warehouse1*.**CostCenterWastePackaging.Cost** *Producer1*.**CostCenterCO2Packaging.Cost**

Tabelle 5: Cost Center Prototyp1

Eine einfache Supply Chain, die mit Prototyp1 aufgebaut werden kann, kann zum Beispiel so aussehen:

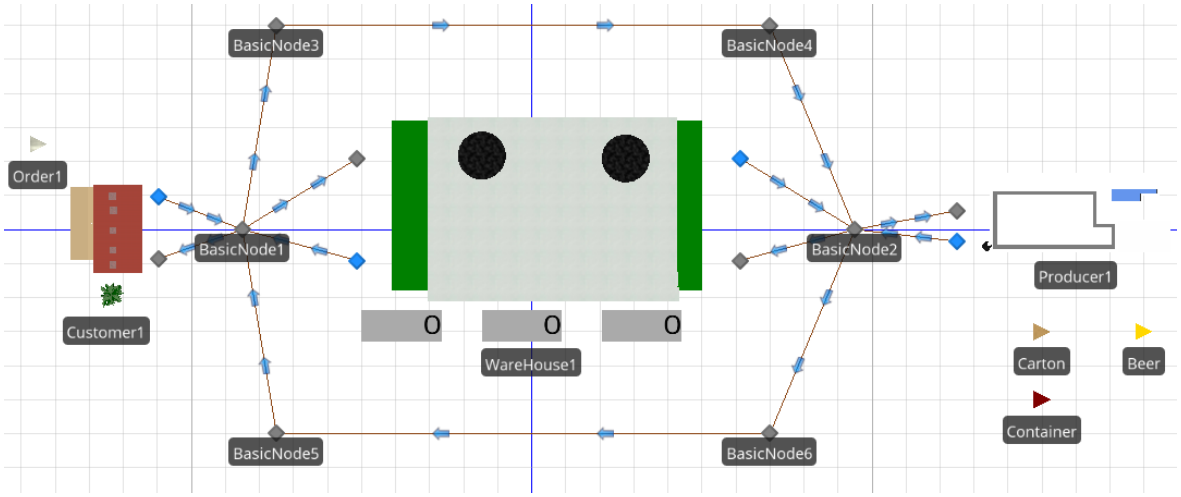


Abbildung 32: Beispiel Supply Chain Prototyp1

Es ist wichtig, dass das Wegesystem so aufgebaut ist, dass von jedem Input und Output jeder andere Input und Output erreicht werden kann, damit die Entities ihren Weg finden.

In den nachfolgenden Kapiteln wird auf die notwendigen Einstellungen der einzelnen Model Entities eingegangen.

Anmerkung: Beim Durchlaufen der Simulation wird folgende Warnung erscheinen, diese kann ignoriert werden.

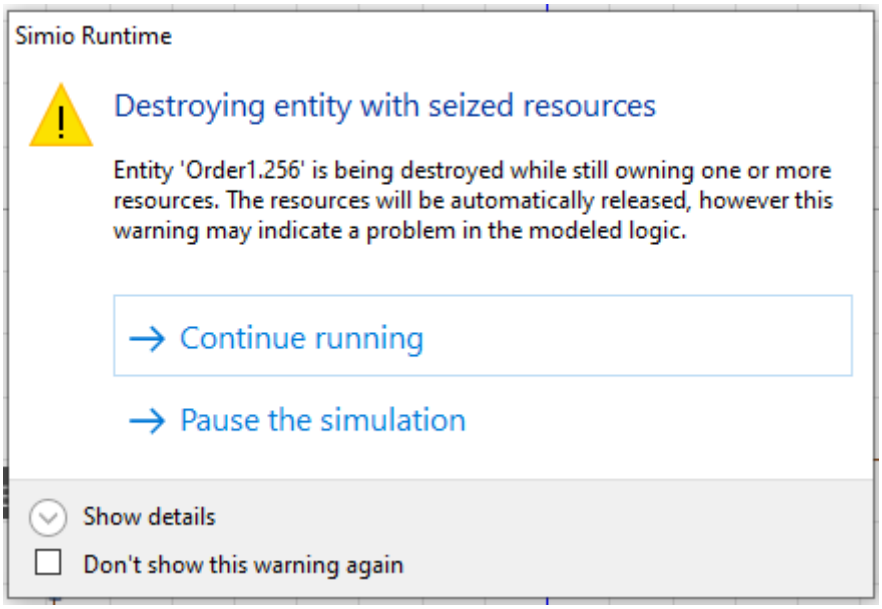
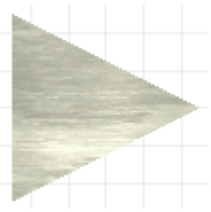


Abbildung 33: Fehlermeldung Prototyp1

10.1 ORDER



General	
Name	Order 1
Description	
Public	True
Report Statistics	True
InitialOrderAddressNode	null
InitialDeliveryAddressNode	null
InitialNumberOrdered	1
PackagingType	Carton
OrderedProductType	Beer
⊞ Physical Characteristics	

InitialOrderAddress-Node	Defaultwert für die Adresse, bei der bestellt wird. Dieser Wert kann leer gelassen werden.
InitialDeliveryAddress-Node	Defaultwert für die Adresse, zu der die Produkte geliefert werden. Dieser Wert kann leer gelassen werden.
InitialNumberOrdered	Wie viele Produkte mit einer *Order* bestellt werden.
PackagingType	In was die Produkte eingepackt werden sollen.
OrderedProductType	Welches Produkt mit dieser *Order* bestellt wird.

Tabelle 6: Order Einstellungen Prototyp1

10.2 PRODUCT

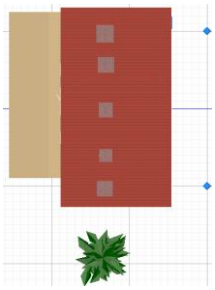
Anmerkung: Produkte und Verpackungen, die im Modell verwendet werden, müssen alle als einzelne *Product Entity* in das Modell reingezogen und definiert werden.

General	
Name	Beer
Description	
Public	True
Report Statistics	True
PackagingTypeProperty	**NonPackaging**
ProductTypeProperty	**Beer**
⊞ Physical Characteristics	

PackagingTypeProperty	Hier wird definiert, ob das Produkt selbst auch Verpackung sein kann. Bsp.: *Beer* soll hier auf *NonPackaging* gesetzt werden, *Container* auf *Container*.
ProductTypeProperty	Hier wird definiert, um was für ein Produkt es sich handelt.

Tabelle 7: Product Einstellungen Prototyp1

10.3 CUSTOMER



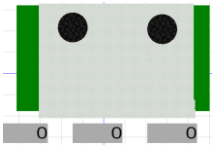
General	
Name	Customer1
Description	
Public	True
OrderEntityType	**Order1**
⊞ OrderInterarrivalTime	**2**
OrderMaximumArrivals	**50**
CustomerAddress	**Input_Customer@Customer1**
OrderAddress	**Input_Management@WareHouse1**
⊞ ProductPrice	**10**
⊞ PackagingWasteCost	**2**
⊞ Physical Characteristics	

OrderEntityType	Welche *Order* versendet werden soll. Über diese *Order* ist das bestellte Produkt, sowie die Menge der bestellten Produkte pro *Order* definiert. **siehe Order*
OrderInterarrivalTime	In welcher Zwischenankunftszeit die *Orders* versendet werden.
OrderMaximumArrivals	Wie viele *Orders* maximal versendet werden im Laufe der Simulation.
CustomerAddress	Zu welcher Adresse die bestellten Produkte oder abgelehnten Bestellungen geschickt werden, also die InputNode des *Customers*. Diese Angabe überschreibt die *InitialDeliveryAdd_Node* auf der *Order*.
OrderAddress	Bei welcher Station die Produkte bestellt werden, also die InputNode eines *WareHouses*. Diese Angabe überschreibt die *InitialDelivery-Add_Node* auf der *Order*.
ProductPrice	Der Verkaufspreis für die bestellten Produkte.
PackagingWasteCost	Wie viel es kostet, die Verpackungen zu entsorgen, in denen die Produkte ankommen.

Tabelle 8: Customer Einstellungen Prototyp1

10.4 WAREHOUSE

Anmerkung: Die Zahlen unterhalb des *Warehouses* stehen für 1. den Lagerbestand der Produkte, 2. den Lagerbestand der Kartons und 3. den Lagerbestand für Container.

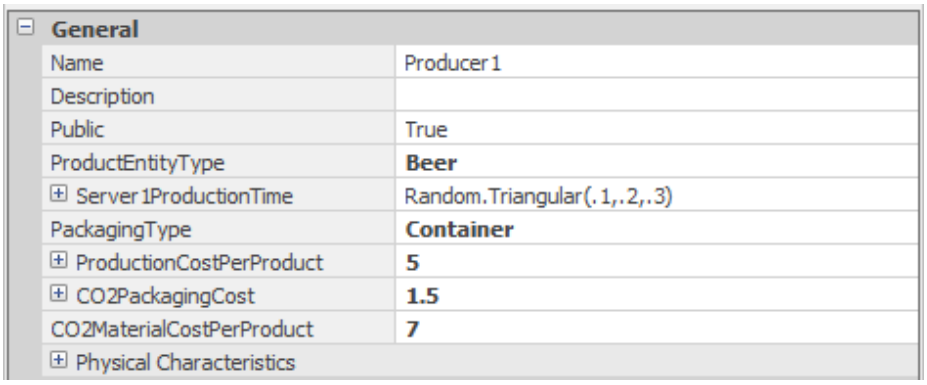


General	
Name	WareHouse 1
Description	
Public	True
ProductType	**Beer**
InitialNumberOfProducts	**0**
InitialReorderPoint	**50**
ChosenProducer	**Input_Producer@Producer1**
CartonEntity	**Carton**
NumberOrdered	150
ContainerEntity	**Container**
ReturnAddress	**Input_Management@WareHouse1**
⊞ PackagingWasteCost	**2**
OrderSizeCarton	50
⊞ CO2PackagingCost	**0.5**
⊞ StorageCostPerProduct	**0.2**
⊞ Physical Characteristics	

ProductType	Welcher Produkttyp in diesem *Warehouse* eingelagert und nachbestellt wird.
InitialNumberOfProducts	Der Anfangsbestand der Produkte im Lager
InitialReorderPoint	Bei welchem Lagerbestand das *Warehouse* Waren nachbestellt.
ChosenProducer	Wo Produkte bestellt werden. Dies kann das Input_Management-Node eines *WareHouses* oder der InputNode eines *Producers* sein.
CartonEntity	Hier soll das *Carton Entity* gewählt werden, damit im Lager die Einlagerung von bestellten Kartons funktioniert.
NumberOrdered	Wie viele Produkte mit einer Bestellung nachbestellt werden.
ContainerEntity	Hier soll das *Container Entity* gewählt werden, damit im Lager die Einlagerung von Containern funktioniert.
ReturnAddress	Zu welcher Adresse, abgelehnte *Orders* zurückgesendet werden, dies ist das eigene Input_ManagementNode des *Warehouses*.
PackagingWasteCost	Die Entsorgungskosten für Produkte, die in einem Karton verpackt geliefert wurden.
OrderSizeCarton	Wie viele Kartons mit einer Bestellung nachbestellt werden.
CO2PackagingCost	Die CO_2 Kosten die für das Verpacken von Produkten anfallen.
StorageCostPerProduct	Die Lagerkosten die für ein Produkt über einen gewissen Zeitraum anfallen.

Tabelle 9: Warehouse Einstellungen Prototyp1

10.5 PRODUCER



ProductEntityType	Welches Produkt der *Producer* herstellt.
Server1ProductionTime	Die Produktionszeit, die der *Producer* für ein Produkt benötigt.
PackagingType	In welchen Verpackungen der *Producer* seine Lieferungen verschickt. Für Prototyp1 werden diese einfach erzeugt, man geht von einem unendlichen Lagerbestand für Container aus.
ProductionCostPerProduct	Die Produktionskosten, die der *Producer* für ein Produkt hat.
CO2PackagingCost	Die CO_2 Kosten, die für das Verpacken von Produkten anfallen.
CO2MaterialCostPerProduct	Die CO_2 Kosten, die entstehen, für die Produktion eines Produkts.

Tabelle 10: Producer Einstellungen Prototyp1

11 BEDIENUNG: PROTOTYP2

Anmerkung: *Da der Prototyp2 ein Zwischenprototyp ist, ist die Handhabung dieses Prototyps komplexer als für den finalen Prototyp geplant ist. Ein eigener Aufbau eines Modells empfiehlt sich nicht. Stattdessen sollte zum Nachvollziehen der Funktionalität der bereits vorgefertigte Modellaufbau «ExampleSupplyChain» verwendet werden.*

Prototyp2 ermöglicht dem Benutzer eine Supply Chain zu erzeugen mit mehreren nacheinander geschachtelten *Customers*, *Warehouses*, *Products* und Producern. Es werden der Informations- und der Warenfluss abgebildet.

Wie bei Prototyp1, können *Customer* ihre Waren bei einem *Warehouse* bestellen und *Warehouses* wiederum können ihre Waren entweder bei einem anderem *Warehouse* oder dem *Producer* bestellen. Neu gibt es die Möglichkeit, einen *ProducerAdvanced* in die Supply Chain einzubinden, der *RawMaterial* bestellt und daraus ein FinalProduct herstellt (zum Beispiel bezieht er *Water* und *Hops* und produziert daraus *Beer*). Ebenfalls hinzugekommen ist die Möglichkeit anhand Entscheidungsmatrixen im Excel eine Prioritätenliste zu erzeugen, mit welcher entschieden wird, welche Produkte bei welchem *Producer* bestellt werden. Und welche *Producer* angesteuert werden, wenn einer oder mehrere *Producer* in der Simulation ausfallen.
Es gilt folgende Einschränkung: Sofern mit der «Decision_Matrix.xlsx» gearbeitet wird, sind maximal 20 *Producer* und 10 *Products* umsetzbar, ansonsten die Excel Datei zu erweitern. Wenn nicht mit der vordefinierten Excel Datei gearbeitet wird, sind die Data Tables im Simio für jedes Produkt von Hand nachzuführen. Von letzterem ist abzuraten.

Ansonsten sind keine funktionalen Änderungen zu Prototyp1 hinzugekommen, die an der Bedienung der Modellbibliothek Änderungen mit sich bringen. Es gibt nach wie vor die gleichen Verpackungsmöglichkeiten (wiederverwendbar und einmalig verwendbar) und die gleichen Aufrufmöglichkeiten für CostCenter.

Eine Supply Chain, die mit Prototyp2 aufgebaut werden kann, kann zum Beispiel so aussehen, wie in Abbildung 31Abbildung 34 (siehe nachfolgende Seite).

Es ist wichtig, dass das Wegesystem so aufgebaut ist, dass von jedem Input und Output jeder andere Input und Output erreicht werden kann, damit die Entities ihren Weg finden.

In den nachfolgenden Kapiteln wird auf die notwendigen Einstellungen der einzelnen Model Entities eingegangen.

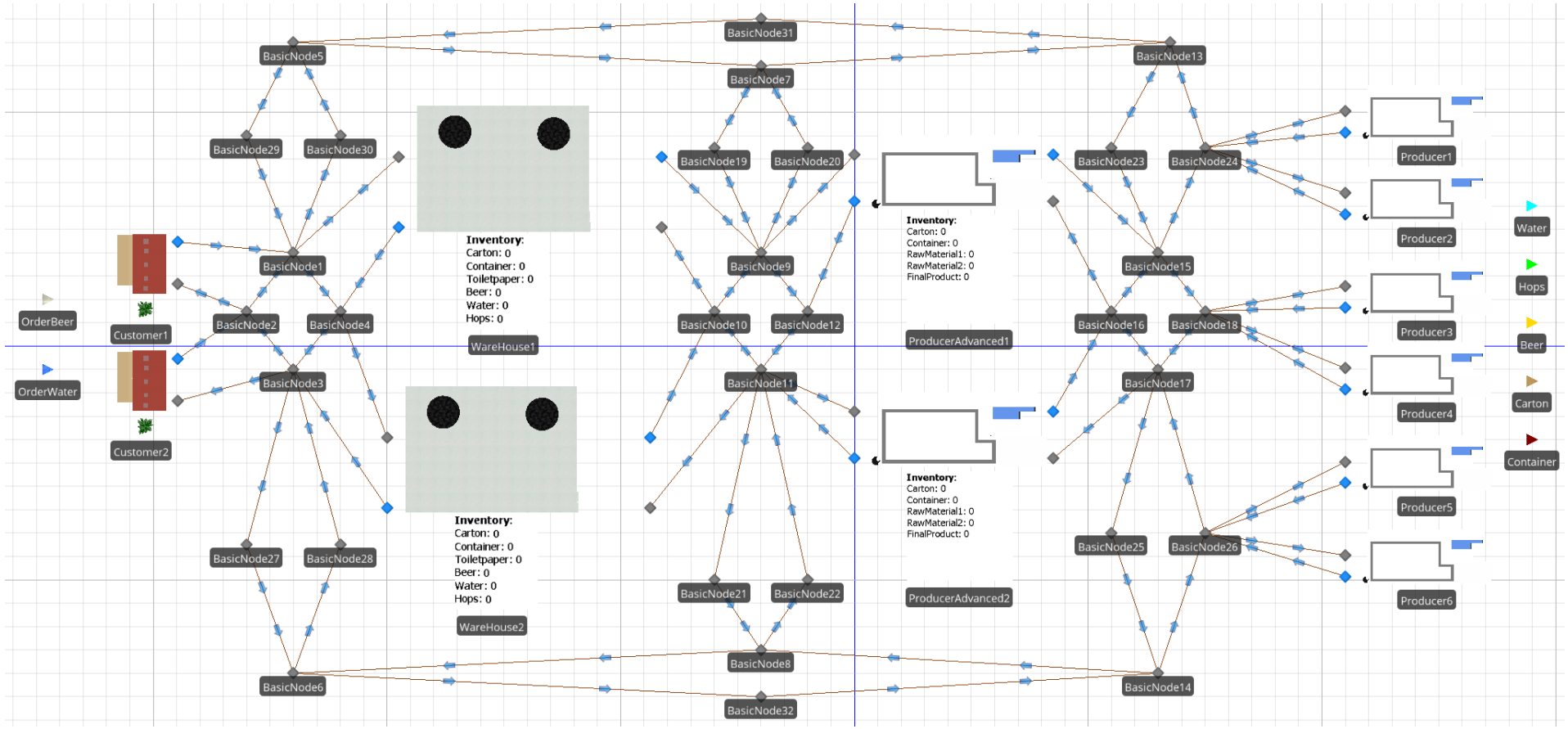
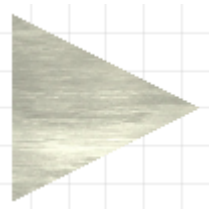


Abbildung 34: Beispiel Supply Chain Prototyp2

11.1 ORDER



General	
Name	OrderBeer
Description	
Public	True
Report Statistics	True
InitialOrderAddressNode	null
InitialDeliveryAddressNode	null
InitialNumberOrdered	1
PackagingType	Carton
OrderedProductType	Beer
Physical Characteristics	

InitialOrderAddress-Node	Defaultwert für die Adresse, bei der bestellt wird. Dieser Wert kann leer gelassen werden.
InitialDeliveryAddress-Node	Defaultwert für die Adresse, zu der die Produkte geliefert werden. Dieser Wert kann leer gelassen werden.
InitialNumberOrdered	Wie viele Produkte mit einer *Order* bestellt werden.
PackagingType	In was die Produkte eingepackt werden sollen.
OrderedProductType	Welches Produkt mit dieser *Order* bestellt wird.

Tabelle 11: Order Einstellungen Prototyp2

11.2 PRODUCT

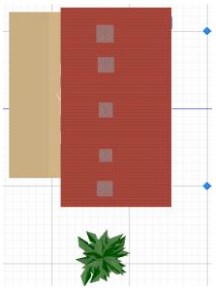
Anmerkung: Produkte und Verpackungen, die im Modell verwendet werden, müssen alle als einzelne ProdcutEntity in das Modell reingezogen und definiert werden.

General	
Name	Beer
Description	
Public	True
Report Statistics	True
PackagingTypeProperty	**NonPackaging**
ProductTypeProperty	**Beer**
Physical Characteristics	

PackagingTypeProperty	Hier wird definiert, ob das Produkt selbst auch Verpackung sein kann. Bsp.: *Beer* soll hier auf *NonPackaging* gesetzt werden, *Container* auf *Container*.
ProductTypeProperty	Hier wird definiert, um was für ein Produkt es sich handelt.

Tabelle 12: Product Einstellungen Prototyp2

11.3 CUSTOMER

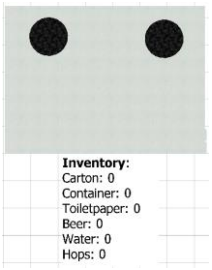


General	
Name	Customer1
Description	
Public	True
OrderEntityType	**OrderBeer**
⊞ OrderInterarrivalTime	**2**
OrderMaximumArrivals	**50**
CustomerAddress	**Input_Customer@Customer1**
OrderAddress	**Input_Management@WareHouse1**
⊞ ProductPrice	**10**
⊞ PackagingWasteCost	**2**
⊞ Physical Characteristics	

OrderEntityType	Welche *Order* versendet werden soll. Über diese *Order* ist das bestellte Produkt, sowie die Menge der bestellten Produkte pro *Order* definiert. **siehe Order*
OrderInterarrivalTime	In welcher Zwischenankunftszeit die *Orders* versendet werden.
OrderMaximumArrivals	Wie viele *Orders* maximal versendet werden im Laufe der Simulation.
CustomerAddress	Zu welcher Adresse die bestellten Produkte oder abgelehnten Bestellungen geschickt werden, also die InputNode des *Customers*. Diese Angabe überschreibt die *InitialDeliveryAdd_Node* auf der *Order*.
OrderAddress	Bei welcher Station die Produkte bestellt werden, also die InputNode eines *WareHouses*. Diese Angabe überschreibt die *InitialDelivery-Add_Node* auf der *Order*.
ProductPrice	Der Verkaufspreis für die bestellten Produkte.
PackagingWasteCost	Wie viel es kostet, die Verpackungen zu entsorgen, in denen die Produkte ankommen.

Tabelle 13: Customer Einstellungen Prototyp2

11.4 WAREHOUSE



General	
Name	WareHouse1
Description	
Public	True
TradeProduct1	**Beer**
TradeProduct2	**Water**
InitialNumberOfProducts	**10**
InitialReorderPoint	**5**
NumberOrdered	**20**
CartonEntity	**Carton**
OrderSizeCarton	50
ContainerEntity	**Container**
ReturnAddress	**Input_Management@WareHouse1**
⊞ PackagingWasteCost	**2**
⊞ CO2PackagingCost	**0.5**
⊞ StorageCostPerProduct	**0.2**
TableProducersProduct1	ProducersProduct1
TableProducersProduct2	ProducersProduct2
TableProducersProduct3	ProducersProduct3
TableProducersProduct4	ProducersProduct4
TableProducersProduct5	ProducersProduct5
TableProducersProduct6	ProducersProduct6
TableProducersProduct7	ProducersProduct7
TableProducersProduct8	ProducersProduct8
TableProducersProduct9	ProducersProduct9
TableProducersProduct10	ProducersProduct10
⊞ Physical Characteristics	

TradeProduct1 **TradeProduct2**	Welche Produkttypen in diesem *Warehouse* zu Beginn der Simulation eingelagert sind.
InitialNumberOfProducts	Der Anfangsbestand der Produkte im Lager, dieser Wert gilt für beide *TradeProducts*.
InitialReorderPoint	Bei welchem Lagerbestand das *Warehouse* Waren nachbestellt.
NumberOrdered	Wie viele Produkte mit einer Bestellung nachbestellt werden.
CartonEntity	Hier soll das *Carton Entity* gewählt werden, damit im Lager die Einlagerung von bestellten Kartons funktioniert.
OrderSizeCarton	Wie viele Kartons mit einer Bestellung nachbestellt werden.
ContainerEntity	Hier soll das *Container Entity* gewählt werden, damit im Lager die Einlagerung von Containern funktioniert.
ReturnAddress	Zu welcher Adresse, abgelehnte *Orders* zurückgesendet werden, dies ist das eigene Input_ManagementNode des *Warehouses*.
PackagingWasteCost	Die Entsorgungskosten für Produkte, die in einem Karton verpackt geliefert wurden.
CO2PackagingCost	Die CO_2 Kosten, die für das Verpacken von Produkten anfallen.
StorageCostPerProduct	Die Lagerkosten, die für ein Produkt über einen gewissen Zeitraum anfallen.
TableProducersProduct1 ff.	Hier wird auf die vordefinierten Tabellen unter Data / Tables verwiesen. Dies ist vordefiniert und muss nicht angepasst werden.

Tabelle 14: Warehouse Einstellungen Prototyp2

11.5 PRODUCERADVANCED

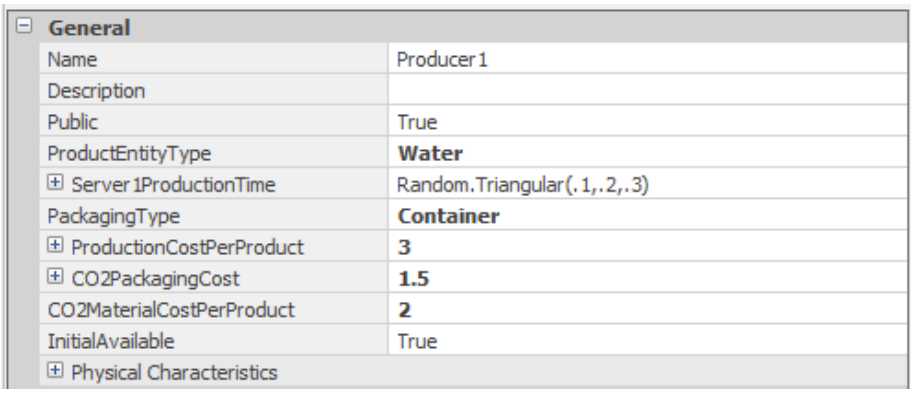
General	
Name	ProducerAdvanced1
Description	
Public	True
InitialNumberOfProducts	**5**
InitialReorderPoint	**3**
NumberOrdered	**3**
OrderSizeCarton	**10**
ReturnAddress	**Input_Management@ProducerAdvanced1**
PackagingWasteCost	**2**
CO2PackagingCost	**0.5**
StorageCostPerProduct	**0.3**
FinalProductType	**Beer**
RawMaterial1	**Water**
RawMaterial2	**Hops**
RawMaterial1OrderSize	**3**
RawMaterial2OrderSize	**5**
RawMaterial1ProductionNumber	**1**
RawMaterial2ProductionNumber	**2**
PackagingType1	**Container**
PackagingType2	**Container**
Packaging1OrderSize	**10**
Packaging2OrderSize	**20**
KeepStock	False
FinalProductTargetStock	**5**
FinalProductProductionBatchSize	**3**
StandardProductionSize	**2**
⊞ ProductionUnit1_ProductionUnit...	**5**
RawMaterial1ProducerTable	**ProducersProduct5**
RawMaterial2ProducerTable	**ProducersProduct6**
PackagingType1ProducerTable	**ProducersProduct1**
PackagingType2ProducerTable	**ProducersProduct2**
InitialAvailability	True
⊞ Physical Characteristics	

InitialNumberOfProducts	Der Anfangsbestand der Produkte im Lager, dieser Wert gilt für alle 3 Produkte: die beiden *RawMaterials* und das *FinalProduct*.
InitialReorderPoint	Bei welchem Lagerbestand der *ProducerAdvanced* Waren nachbestellt.
NumberOrdered	Wie viele Produkte mit einer Bestellung nachbestellt werden.
OrderSizeCarton	Wie viele Kartons mit einer Bestellung nachbestellt werden.
ReturnAddress	Zu welcher Adresse, abgelehnte *Orders* zurückgesendet werden, dies ist das eigene Input_ManagementNode des *ProducerAdvanced*.
PackagingWasteCost	Die Entsorgungskosten für Produkte, die in einem Karton verpackt geliefert wurden.
CO2PackagingCost	Die CO2 Kosten, die für das Verpacken von Produkten anfallen.
StorageCostPerProduct	Die Lagerkosten, die für ein Produkt über einen gewissen Zeitraum anfallen.
FinalProductType	Der Produkttyp, den der *ProducerAdvanced* herstellt.
RawMaterial1 ff.	Gibt die beiden *Rawmaterials* an, die der *ProducerAdvanced* für die Herstellung des *FinalProducts* benötigt.

RawMaterial1OrderSize ff.	Gibt die Nachbestellmenge der beiden *RawMaterials* an, die der *ProducerAdvanced* für die Herstellung des *FinalProducts* benötigt.
RawMaterial1ProductionSize ff.	Gibt je die Anzahl *RawMaterials* an, die für die Herstellung einer *BatchSize* benötigt werden.
PackagingType1 ff.	Gibt dem *ProducerAdvanced* die beiden Verpackungstypen, die im Modell bestehen an. Der erste Wert soll *Carton* sein, der zweite *Container*.
Packaging1OrderSize ff.	Dieser Wert legt die Nachbestellmenge für Karton, resp. Container an.
KeepStock	Dieser Wert soll auf *False* sein und wird in einem Prozess gebraucht.
FinalProductTargetStock	Gibt den Schwellwert an für den eigenen Lagerbestand. Über diesem Wert wird der *ProducerAdvanced* ohne Bestellung keine *FinalProducts* herstellen.
FinalProductProductionBatchSize	Gibt die *BatchSize* eines Produktionsauftrags an. Je der Wert *RawMaterial1ProductionSize* und *RawMaterial2ProductionsSize* ergeben zusammen so viel *FinalProducts* wie hier angegeben.
StandardProductionSize	Sofern kein Bestellauftrag eingegangen ist, der *ProducerAdvanced* aber produzieren muss, um seinen *TargetStock* zu erreichen, gibt dieser Wert die Menge der zu produzierenden Produkte an.
ProductionUnit1_ProductionUnitProcessingTime	Gibt die Produktionszeit an, die der *ProducerAdvanced* benötigt, um eine BatchSize an FinalProducts herzustellen.
RawMaterial1ProducerTable ff.	Hier werden die *ProducerTable* angegeben, in der die Prioritätenliste für *RawMaterial1*, *RawMaterial2* und die beiden Verpackungstypen angegeben sind.
InitialAvailability	Dieser Wert gibt an, ob der *ProducerAdvanced* in diesem Modell verfügbar ist, also Produkte produzieren und liefern kann oder nicht. Wenn dieser Wert auf *False* gesetzt ist, wird er im Bestellvorgang einer anderen Station übersprungen.

Tabelle 15: ProducerAdvanced Einstellungen Prototyp2

11.6 PRODUCER



ProductEntityType	Welches Produkt der *Producer* herstellt.
Server1ProductionTime	Die Produktionszeit, die der *Producer* für ein *Product* benötigt.
PackagingType	In welchen Verpackungen der *Producer* seine Lieferungen verschickt. Für Prototyp1 werden diese einfach erzeugt, man geht von einem unendlichen Lagerbestand für *Container* aus.
ProductionCostPerProduct	Die Produktionskosten, die der *Producer* für ein *Product* hat.
CO2PackagingCost	Die CO_2 Kosten, die für das Verpacken von Produkten anfallen.
CO2MaterialCostPerProduct	Die CO_2 Kosten, die entstehen, für die Produktion eines *Products*.

Tabelle 16: Producer Einstellungen Prototyp2

12 BEDIENUNG: PROTOTYP3

Prototyp3 ermöglicht dem Benutzer eine Supply Chain zu erzeugen mit mehreren nacheinander geschachtelten *Customers*, *Warehouses*, *Products* und Producern. Es werden der Informations- und der Warenfluss abgebildet.

Wie bei Prototyp2, können *Customer* ihre Waren bei einem *Warehouse* bestellen und *Warehouses* und *ProducerAdvanced* wiederum können ihre Waren entweder bei einem anderem *Warehouse*, *Producer* oder *ProducerAdvanced* bestellen.

Wie schon bei Prototyp2 gilt folgende Einschränkung: Sofern mit der «Decision_Matrix.xlsx» gearbeitet wird, sind maximal 20 *Producer* und 10 *Products* umsetzbar, ansonsten die Excel Datei zu erweitern. Wenn nicht mit der vordefinierten Excel Datei gearbeitet wird, sind die Data Tables im Simio für jedes Produkt von Hand nachzuführen. Von letzterem ist abzuraten.

Es gibt nach wie vor die gleichen Verpackungsmöglichkeiten (wiederverwendbar und einmalig verwendbar) und die gleichen Aufrufmöglichkeiten für CostCenter.

Prototyp3 unterscheidet sich vor allem in der erleichterten Bedienung für den Benutzer von seinem Vorgänger. Hinzugekommen oder angepasst worden sind die folgenden Punkte:

- Übersicht über die möglichen CostCenter und Aufrufmöglichkeiten
- Kategorien, Defaultwerte und gekennzeichnete «Required Values» in den einzelnen Stationen
- Optische Anpassung der Stationen, für eine einfachere visuelle Unterscheidung im Modell

Eine Supply Chain, die mit Prototyp3 aufgebaut werden kann, kann zum Beispiel so aussehen, wie in Abbildung 35 (siehe nachfolgende Seite).

Es ist wichtig, dass das Wegesystem so aufgebaut ist, dass von jedem Input und Output jeder andere Input und Output erreicht werden kann, damit die Entities ihren Weg finden.

In den nachfolgenden Kapiteln wird auf die notwendigen Einstellungen der einzelnen Model Entities eingegangen.

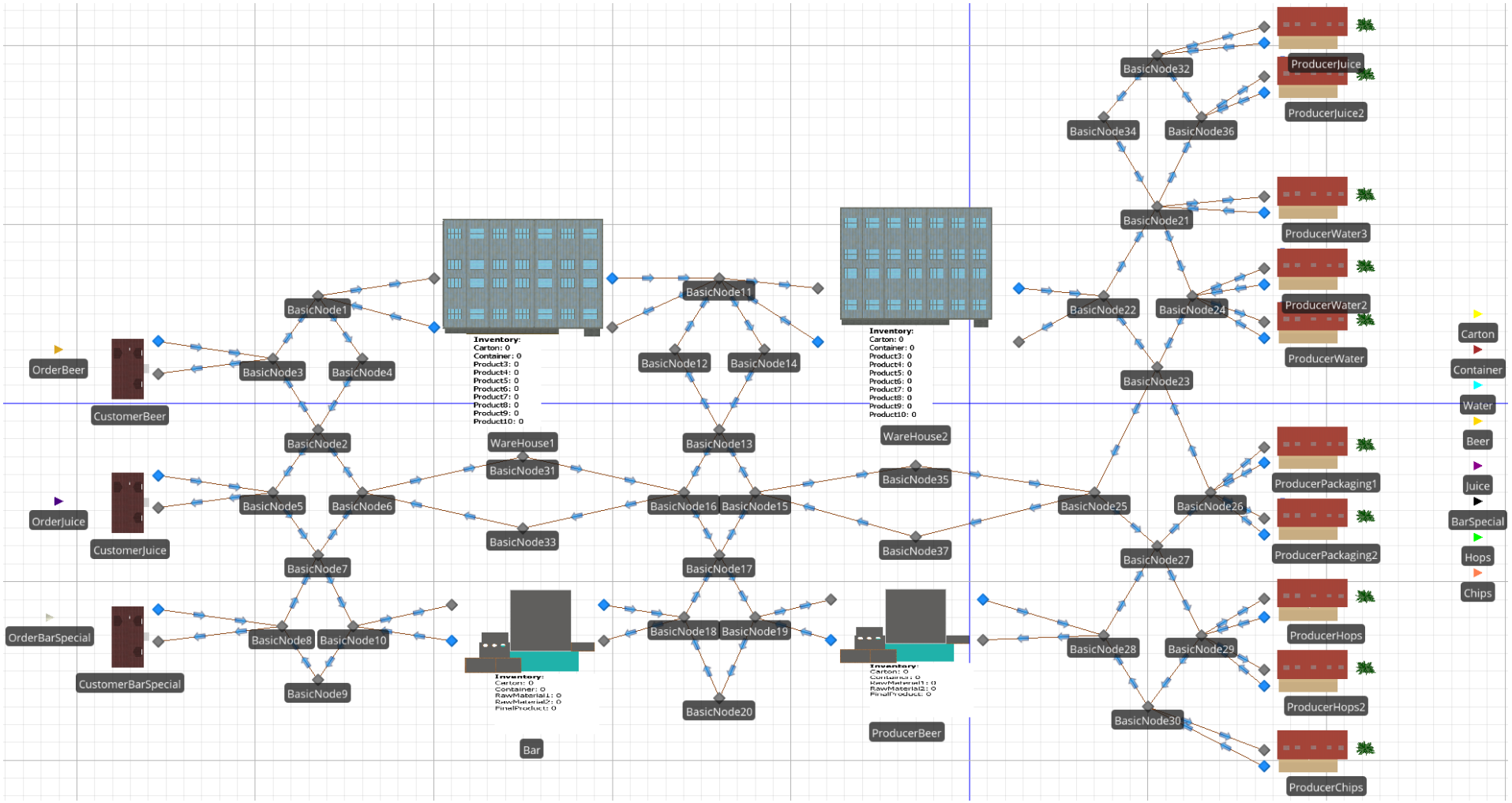
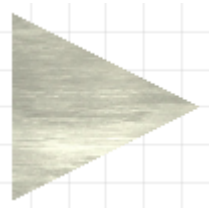


Abbildung 35: Beispiel Supply Chain Prototyp2

12.1 ORDER



Order Details	
InitialNumberOrdered	**5**
PackagingEntityType	**Carton**
OrderedProductEntityType	**BarSpecial**

Order Details	
InitialNumberOrdered	Wie viele Produkte mit einer *Order* bestellt werden.
PackagingEntityType	In was die Produkte eingepackt werden sollen.
OrderedProductEntityType	Welches Produkt mit dieser *Order* bestellt wird.

Tabelle 17: Order Einstellungen Prototyp3

12.2 PRODUCT

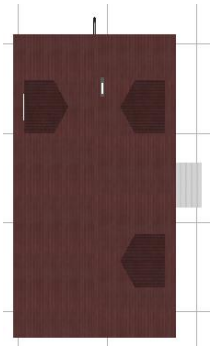
Anmerkung: Produkte und Verpackungen, die im Modell verwendet werden, müssen alle als einzelne ProdcutEntity in das Modell reingezogen und definiert werden.

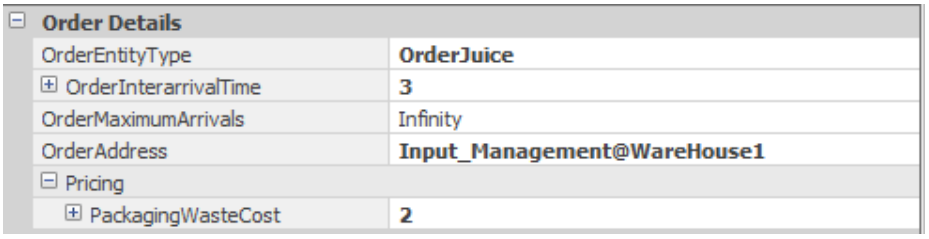
Product Details	
ProductTypeProperty	**Beer**
PackagingTypeProperty	NonPackaging
IsDisposableProperty	False

Product Details	
ProductTypeProperty	Hier wird definiert, um was für ein Produkt es sich handelt.
PackagingTypeProperty	Hier wird definiert, ob das Produkt selbst auch Verpackung sein kann. Bsp.: *Beer* soll hier auf *NonPackaging* gesetzt werden, *Container* auf *Packaging*.
IsDisposableProperty	Hier wird festgelegt, ob das Produkt nach einmaligem Benutzen entsorgt wird. Bsp.: Alle Produkte ausgenommen Karton werden hier auf *False* gesetzt. Karton wird nach einmaligem Benutzen entsorgt.

Tabelle 18: Product Einstellungen Prototyp3

12.3 CUSTOMER

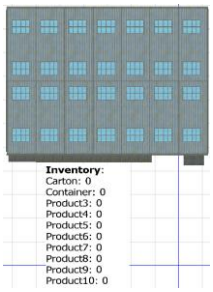




Order Details	
OrderEntityType	Welche *Order* versendet werden soll. Über diese *Order** ist das bestellte Produkt, sowie die Menge der bestellten Produkte pro *Order* definiert. **siehe Order*
OrderInterarrivalTime	In welcher Zwischenankunftszeit die *Orders* versendet werden.
OrderMaximumArrivals	Wie viele *Orders* maximal versendet werden im Laufe der Simulation.
OrderAddress	Bei welcher Station die Produkte bestellt werden, also die InputNode eines Zulieferers. Diese Angabe überschreibt die *InitialDelivery-Add_Node* auf der *Order*.
PackagingWasteCost	Wie viel es kostet, die Verpackungen (Karton) zu entsorgen, in denen die Produkte ankommen.

Tabelle 19: Customer Einstellungen Prototyp3

12.4 WAREHOUSE



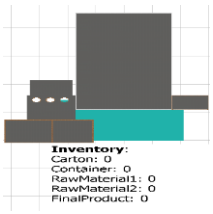
Initialization	
TradeProduct1EntityType	**Juice**
TradeProduct2EntityType	**Chips**
InitialNumberOfProducts	**0**
InitialAvailability	True
InitialNumberOfPackaging1	**50**
InitialNumberOfPackaging2	**10**
Pricing	
PackagingWasteCost	**3**
PackagingCost	**1.2**
StorageCostPerProduct	**0.3**
SellPriceProduct1	0.0
SellPriceProduct2	0.0
SellPriceProduct3	0.0
SellPriceProduct4	0.0
SellPriceProduct5	**6**
SellPriceProduct6	0.0
SellPriceProduct7	0.0
SellPriceProduct8	**3**
SellPriceProduct9	0.0
SellPriceProduct10	0.0
Order Details	
InitialReorderPoint	**50**
NumberOrdered	**150**
Packaging1EntityType	**Carton**
Packaging2EntityType	**Container**
OrderSizePackaging	**50**

Initialization	
TradeProduct1EntityType **TradeProduct2EntityType**	Welche Produkttypen in diesem *Warehouse* zu Beginn der Simulation eingelagert sind.
InitialNumberOfProducts	Der Anfangsbestand der Produkte im Lager, dieser Wert gilt für beide *TradeProductEntityTypes*.
InititalAvailability	Dieser Wert gibt an, ob das *Warehouse* in diesem Modell verfügbar ist, also Produkte liefern kann oder nicht. Wenn dieser Wert auf *False* gesetzt ist, wird es im Bestellvorgang einer anderen Station übersprungen.
InitialNumberOfPackaging1	Der Anfangsbestand von Karton im Lager.
InitialNumberOfPackaging2	Der Anfangsbestand von Containern im Lager.
PackagingWasteCost	Die Entsorgungskosten für Produkte, die in einem Karton verpackt geliefert wurden.
PackagingCost	Die Kosten die für das Verpacken von Produkten anfallen.
StorageCostPerProduct	Die Lagerkosten die für ein Produkt über einen gewissen Zeitraum anfallen. Dies ist in dieser Version für alle Produkte gleich.
SellPriceProduct1 **ff.**	Der Verkaufspreis des *Warehouses* für *Product1*, *Product2*, und so weiter.
Order Details	
InitialReorderPoint	Bei welchem Lagerbestand das *Warehouse* Waren nachbestellt.
NumberOrdered	Wie viele Produkte mit einer Bestellung nachbestellt werden.

Packaging1EntityType	Hier soll das *Carton Entity* gewählt werden, um die Initialisierung des Anfangsbestands zu ermöglichen.
Packaging2EntityType	Hier soll das *Container Entity* gewählt werden, um die Initialisierung des Anfangsbestands zu ermöglichen.
OrderSizePackaging	Wie viele *Packagings* mit einer Bestellung nachbestellt werden.

Tabelle 20: Warehouse Einstellungen Prototyp3

12.5 PRODUCERADVANCED



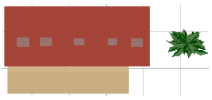
Initialization	
InitialNumberOfProducts	**0**
PackagingType1EntityType	**Carton**
PackagingType2EntityType	**Container**
InitialAvailability	True
StorageManagement	False
Pricing	
SellPriceProduct	**30**
PackagingWasteCost	**2**
PackagingCost	**2**
ProductionCostPerProduct	**1**
StorageCostPerProduct	**0.3**
Order Details	
InitialReorderPoint	**5**
Packaging1OrderSize	**40**
Packaging2OrderSize	**40**
RawMaterial1OrderSize	**70**
RawMaterial2OrderSize	**30**
RawMaterial1ProducerTable	**ProducersProduct8**
RawMaterial2ProducerTable	**ProducersProduct4**
Production Details	
FinalProductType	**BarSpecial**
RawMaterial1	**Chips**
RawMaterial2	**Beer**
RawMaterial1ProductionNumber	1
RawMaterial2ProductionNumber	**5**
FinalProductTargetStock	**5**
FinalProductProductionBatchSize	**5**
StandardProductionSize	**5**
ProductionTime	**1**

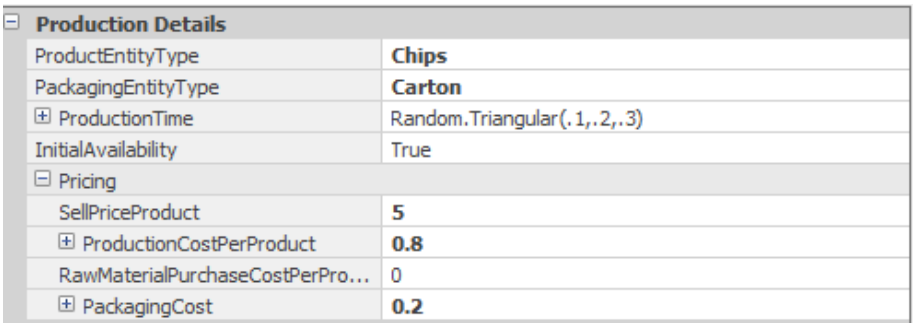
Initialization	
InitialNumberOfProducts	Der Anfangsbestand der Produkte im Lager, dieser Wert gilt für alle 3 Produkte: die beiden *RawMaterials* und das *FinalProduct*.
PackagingType1EntityType ff.	Gibt dem *ProducerAdvanced* die beiden Verpackungstypen, die im Modell bestehen an. Der erste Wert soll *Carton* sein, der zweite *Container*.
InitialAvailability	Dieser Wert gibt an, ob der *ProducerAdvanced* in diesem Modell verfügbar ist, also Produkte produzieren und liefern kann oder nicht. Wenn dieser Wert auf *False* gesetzt ist, wird er im Bestellvorgang einer anderen Station übersprungen.
StorageManagement	Dieser Wert gibt an, ob der *ProducerAdvanced* einen bestimmten Lagerbestand halten will und auf diesen hin produziert (*True*), oder ob er nur für ankommende Bestellung produziert (*False*).
SellPriceProduct	Dieser Wert gibt an, für welchen Preis der *ProducerAdvanced* das hergestellte *FinalProduct* weiterverkauft.

PackagingWasteCost	Die Entsorgungskosten für Produkte, die in einem Karton verpackt geliefert wurden.
PackagingCost	Die Kosten, die für das Verpacken von Produkten anfallen.
ProductionCostPerProduct	Die Kosten, die für das Produzieren eines *FinalProducts* beim *ProducerAdvanced* anfallen.
StorageCostPerProduct	Die Lagerkosten, die für ein Produkt über einen gewissen Zeitraum anfallen.
Order Details	
InitialReorderPoint	Bei welchem Lagerbestand der Verpackungen der *ProducerAdvanced* Verpackungen nachbestellt.
Packaging1OrderSize ff.	Dieser Wert legt die Nachbestellmenge für Karton, resp. Container an.
RawMaterial1OrderSize ff.	Gibt die Nachbestellmenge der beiden *RawMaterials* an, die der *ProducerAdvanced* für die Herstellung des *FinalProducts* benötigt.
RawMaterial1ProducerTable ff.	Hier werden die *ProducerTable* angegeben, in der die Prioritätenliste für *RawMaterial1* und *RawMaterial2* hinterlegt sind.
Production Details	
FinalProductType	Der Produkttyp, den der *ProducerAdvanced* herstellt.
RawMaterial1 ff.	Gibt die beiden *Rawmaterials* an, die der *ProducerAdvanced* für die Herstellung des *FinalProducts* benötigt.
RawMaterial1ProductionNumber ff.	Gibt je die Anzahl *RawMaterials* an, die für die Herstellung einer *BatchSize* benötigt werden.
FinalProductTargetStock	Gibt den Schwellwert an für den eigenen Lagerbestand. Über diesem Wert wird der *ProducerAdvanced* ohne Bestellung keine *FinalProducts* herstellen.
FinalProductProductionBatchSize	Gibt die *BatchSize* eines Produktionsauftrags an. Je der Wert RawMaterial1ProductionSize und RawMaterial2ProductionNumber ergeben zusammen so viel *FinalProducts* wie hier angegeben.
StandardProductionSize	Sofern kein Bestellauftrag eingegangen ist, der *ProducerAdvanced* aber produzieren muss, um seinen *TargetStock* zu erreichen, gibt dieser Wert die Menge an zu produzierenden Produkten an.
ProductionTime	Gibt die Produktionszeit an, die der *ProducerAdvanced* benötigt, um eine *BatchSize* an *FinalProducts* herzustellen.

Tabelle 21: ProducerAdvanced Einstellungen Prototyp3

12.6 PRODUCER





Production Details	
ProductEntityType	Welches Produkt der *Producer* herstellt.
PackagingEntityType	In welchen Verpackungen der *Producer* seine Lieferungen verschickt. Für Prototyp1 werden diese einfach erzeugt, man geht von einem unendlichen Lagerbestand für Container aus.
ProductionTime	Die Produktionszeit, die der *Producer* für ein *Product* benötigt.
InititialAvailability	Dieser Wert gibt an, ob der *Producer* in diesem Modell verfügbar ist, also Produkte produzieren und liefern kann oder nicht. Wenn dieser Wert auf *False* gesetzt ist, wird er im Bestellvorgang einer anderen Station übersprungen.
SellPriceProduct	Dieser Wert gibt an, für welchen Preis der *Producer* das hergestellte *Product* weiterverkauft.
ProductionCostPerProduct	Die Produktionskosten, die der *Producer* für ein *Product* hat.
RawMaterialPurchaseCostPerProduct	Die Einkaufskosten, die der *Producer* für seine Rohmaterialien hat. In dieser Version wird dieser Wert direkt auf dem *Producer* angegeben, da er das Endstück der Supply Chain ist.
PackagingCost	Die Kosten, die für das Verpacken von Produkten anfallen.

Tabelle 22: Producer Einstellungen Prototyp3

13 BEDIENUNG: DECISION MATRIX

Zusätzlich zum Supply Chain Modellbaukasten kann die Excel Datei «Decision_Matrix.xlsx» gebraucht werden, um Ratings anhand Entscheidungsmatrizen ins Simio Modell zu importieren. Anhand der Bewertung werden für die verschiedenen Produkte unterschiedliche *Producer* gewählt, bei denen in der Simulation bestellt wird. In der Excel Datei selbst sind Kommentare hinterlegt, die beim Ausfüllen helfen, der nachfolgende Fliesstext dient als Ergänzung.

13.1 PRODUCER LIST

In diesem Tabellenblatt sind die Listen *Product List* und *Producer List* zu führen. Die Reihenfolge und Namensvergabe muss **exakt gleich** sein, wie sie auch im Simio Modell angegeben wurde.

Für jedes *Product* und für jeden *Producer* ist noch anzuwählen, ob sie in der Simulation auch tatsächlich verwendet werden. Wenn nicht, werden Werte in den nachfolgenden Tabellen als 0 eingetragen, damit sie das Rating nicht durcheinanderbringen.

In der *RatingTable* ist es möglich, Angaben über die verschiedenen *Produkte* und *Producer* nachzuführen und zu bewerten. Der Einfachheit halber empfiehlt es sich in den in Abbildung 36 ausgewählten Spalten, die Filter so zu setzen, dass nur *Products* und *Producer*, die auch in der Simulation verwendet werden, angezeigt werden. Zusätzlich werden diese Bereiche auch gelb markiert, dennoch macht es die Tabelle übersichtlicher, wenn die entsprechenden Filter gesetzt werden.

								RatingTable	
	Product	Producer	Price	RatingPrice	CO2	RatingCO2	Batch Size	RatingBatch	Availa
Product1	Product1	Producer1							
	Product1	Producer2							
	Product1	Producer3							
	Product1	Producer4							
	Product1	Producer5							
	Product1	Producer6		6		1		1	
	Product1	Producer7							
	Product1	Producer8							
	Product1	Producer9							
	Product1	Producer10							
	Product1	Producer11							
	Product1	Producer12							

Abbildung 36: Producer List: Spaltentitel

Für die verschiedenen Produkte sind nur bei den *Producern* Angaben zu machen, die diese Produkte auch tatsächlich anbieten. Sehr viele Zeilen in dieser Tabelle bleiben leer.

13.2 MATRICES

Anzupassende Bereiche werden in diesem Tabellenblatt gelb hervorgehoben. So sind nur Anpassungen nötig bei Produkten, die auch tatsächlich in der Simulation verwendet werden. In jeder Decision Matrix kann die Gewichtung der einzelnen Kriterien separat vergeben werden.

Anhand der Gewichtung und der im Tabellenblatt *Producer* List eingetragenen Ratings wird dann das totale Rating pro *Product* und *Producer* berechnet und in den Tabellenblättern *Intermediate Results* und *Results* weiterverarbeitet.

Die Excel Datei kann nun gespeichert und geschlossen werden.

13.3 In Simio

Für die Beispiel Supply Chain sind die Bezüge im Simio bereits erstellt. Wenn ein neues Modell erzeugt wird, müssen die *Data Tables* ebenfalls neu erstellt werden. Für jedes Produkt wird eine *Data Table* benötigt und via Import Settings auf den Bereich im Tabellenblatt «*Results*» verwiesen, siehe Abbildung 37. Diese Bereiche heissen jeweils *ProductXResults*, wobei X für die Nummerierung des *Products* steht.

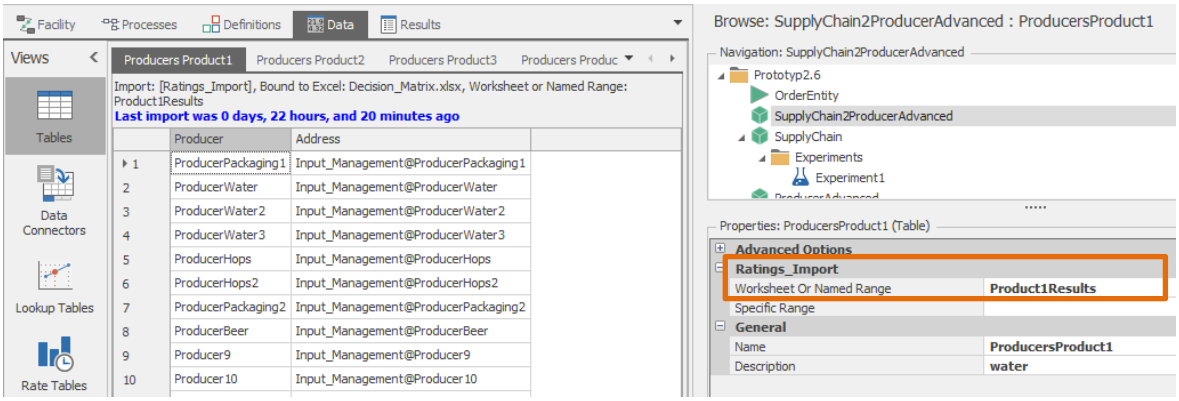


Abbildung 38: Simio Data Tables, Import Settings

14 Anpassung: Simio Modell – Anzahl Produkte

Prototyp3 unterstützt zehn unterschiedliche Produkttypen und der *ProducerAdvanced* kann aus zwei verschiedenen Rohmaterialien ein neues Produkt herstellen. Das nachfolgende Kapitel erläutert, wie das Simio Model angepasst werden muss, um das *Warehouse* und den *ProducerAdvanced* für darauf aufbauende Konfigurationen verwenden zu können.

14.1 Produkte

Wenn der Typ der Produkte angepasst und/oder weitere Produkttypen hinzugefügt werden sollen, so muss dies an folgenden drei Stellen angepasst werden:

1. Auf der *ProductEntity*
2. Auf der *OrderEntity*
3. In den *Data Tables*

ProductEntity und *OrderEntity* verfügen über je eine Liste *Product-Type_List*. In dieser Liste werden alle vom System verwalteten Produkte aufgeführt. Es ist wichtig, dass die Namen der Produkte und die Reihenfolge in beiden Listen identisch sind, da der Vergleich von Listen über den Integer Wert der Position vorgenommen wird.

	String
0	Carton
1	Container
2	Water
3	Beer
4	Juice
5	BarSpecial
6	Hops
7	Chips
*	

Abbildung 39: Product-Type_List

Bei den Data Tables wird eine neue Data Table analog zu den *ProducersProductX* erstellt. Die Tabelle muss über drei Spalten mit den jeweiligen Titeln und Typen von Supplier (*Object*), Address (*Node*) und Expression Supplier (*Expression*) verfügen. Ist dies gegeben, kann ein Binding auf das Excel Dokument, welches als Grundlage für die anderen Tabellen verwendet wird, erstellt werden. Das Binding wird dabei bereits vorgeschlagen (*Ratings_Import*) und muss durch einen Eintrag *ProductXResults* beim *Worksheet Or Named Range* ergänzt werden, wie in Abbildung 9 zu sehen ist.

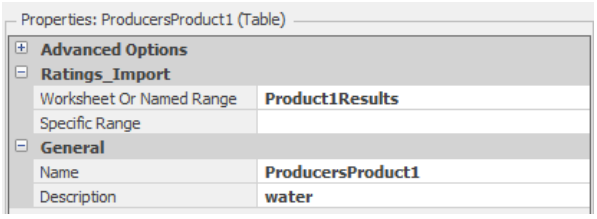


Abbildung 40: Data Tables Properties

Anmerkung: Das Excel Dokument[1] muss ebenfalls um das neue Produkt ergänzt werden.

14.2 Warehouse

Im Prototyp3 kann das *Warehouse* bis zu zehn Produkten verwalten. Wenn weitere Produkte hinzugefügt werden, muss auch angepasst werden, dass das *Warehouse* diese Produkte verwalten kann.

Anmerkung: Idealerweise wird dieser Vorgang in der Zukunft automatisiert. In der verwendeten Simio Version ist dies allerdings nicht unterstützt, und muss mit externen Tools und eigenen Programmerweiterungen gelöst werden. Dies umzusetzen ist nicht Bestandteil der vorliegenden Arbeit.

[1] Excel Dokument: Decision_Matrix.xlsx

Um die Verwaltung in der *Warehouse* Station um ein weiteres Produkt zu erweitern, müssen die folgenden Punkte bearbeitet werden. X steht nachfolgend jeweils für das jeweilige Produkt.

1. Erstellen eines ***Elements*** «*CostCenterProductXIncome*» (CostCenter)
2. Erstellen neuer Properties:
   a. «*TableProducersProductX*» (***Table Property***), hier die im vorangegangenen Schritt erstellte Tabelle als Default Wert angeben
   b. «*SellPriceProductX*» (***Integer Property***)
3. Erstellen eines States «*OpenOrderProduktX*» (***Boolean State Variable***)

14.2.1 StoreWarehouse

In der *StoreWarehouse* Station müssen die nachfolgenden Änderungen vorgenommen werden:

Tipp: Prozessschritte können kopiert und dann angepasst werden.

1. Erstellen eines States «*InventoryProductX*» (***Integer State Variable***)
2. Erstellen eines Events «*OrderReceivedProductX*»
3. Erweitern des Prozesses «*StorageUnit_Entered*». Für ein neues Produkt sind drei zusätzliche Schritte notwendig: *Decide*, *Assign* und *Fire*. Diese sind analog der anderen Schritte im Prozess auszufüllen. ***Wichtig***: Die Indexnummer des Produkts, welches im *Decide* verwendet wird, muss jeweils dem Listindex entsprechen, also X-1.

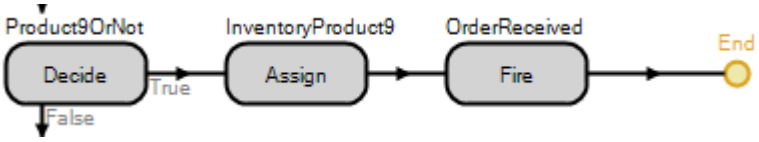


Abbildung 41: StoreWarehouse StorageUnit_Entered Prozess

4. Erweitern des Prozesses «*StorageUnit_Exited*». Für ein neues Produkt sind zwei zusätzliche Schritte notwendig: *Decide* und *Assign.* Diese sind analog der anderen Schritte im Prozess auszufüllen. ***Wichtig***: Die Indexnummer des Produkts, welches im *Decide* verwendet wird, muss jeweils dem Listindex entsprechen, also X-1.

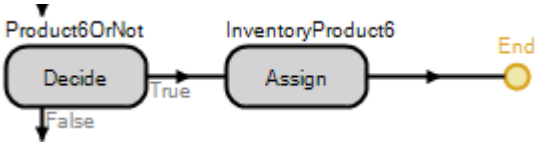


Abbildung 42: StoreWarehouse StorageUnit_Exited Prozess

5. Erweitern des *Labels* «*Inventory*» um das neue Produkt.

14.3 Warehouse

Sind die vorherigen Änderungen soweit umgesetzt, sind weitere Ergänzungen in der *Warehouse* Station notwendig.

1. Erweitern der Prozesse, «*Check_OpenOrder*», «*Set_OpenOrder*», «*OutputShipping_-Exited*» und «*Reset_OpenOrder*». Wie bei *StoreWarehouse* Schritt 4, müssen bei diesen Prozessen lediglich die Schritte, welche pro Produkt ausgeführt werden und der zugehörigen Abfrage auf das Produkt (*Decide*) mit den Schritten für das neue Produkt ergänzt werden.
   Anmerkung: Der Prozess für den *Execute* Schritt von «*Reset_OpenOrder*» wird später erstellt.

2. Erstellen eines Prozesses «*OrderReceived_ProductX*». Hierzu kann einer der vorhandenen «*OrderReceived_ProductX*» Prozesse kopiert und angepasst werden.
   Wichtig: Das auslösende Event ist auf das neue Produkt anzupassen.

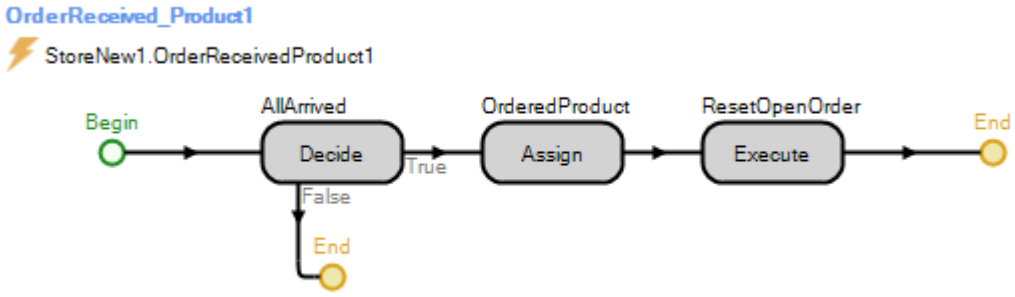


Abbildung 43: Warehouse OrderReceived_ProductX Prozess

3. Erstellen eines Prozesses «*Reorder_ProductX*». Auch hier kann einer der bestehenden «*Reorder_ProductX*» Prozesse kopiert und angepasst werden.

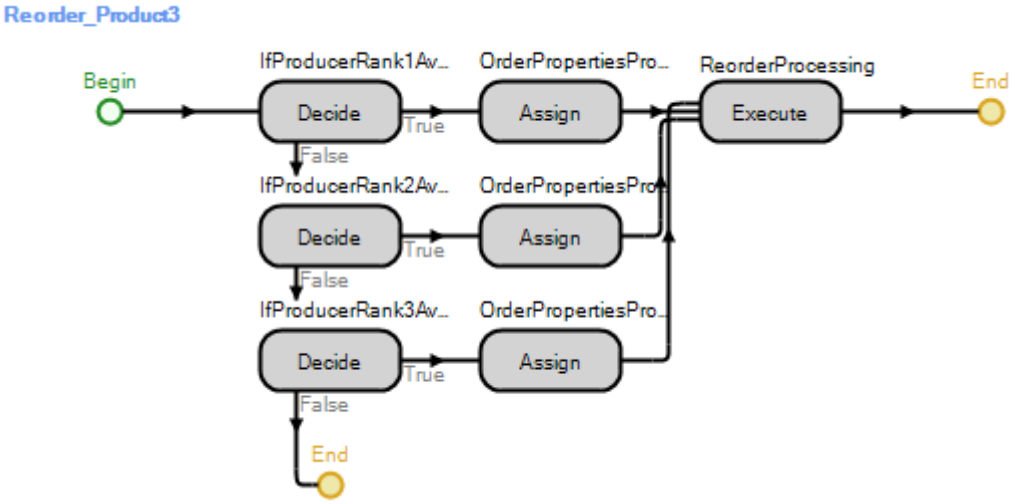


Abbildung 44: Warehouse Reorder_ProductX Prozess

4. Ergänzen des *Execute* Schrittes im Prozess «*Reset_OpenOrder*» mit dem neu erstellen «*Reorder_productX*» Prozess.

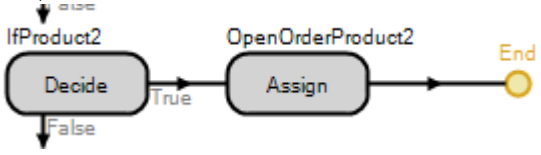


Abbildung 45: Warehouse Reset_OpenOrder Prozess

Alle diese Prozesse werden für die produktspezifische Verwaltung benötigt. Sie verwalten die Auswahl des Herstellers und die Überprüfung, ob für ein Produkt bereits eine offene Bestellung im System ist.

14.4 PRODUCERADVANCED

Der *ProducerAdvanced* stellt aus zwei verschiedenen Produkten ein neues Produkt her. Es muss angegeben werden, wie viele Rohmaterialien welchen Typs verwendet werden, um eine bestimmte Menge Endprodukt herzustellen. Prototyp3 ermöglicht eine Produktion mit jeweils 2 Rohmaterialien. Das folgende Kapitel führt auf, was im Simio Modell angepasst werden muss, um zum Beispiel eine Produktion mit 3 Rohmaterialien zu ermöglichen.

1. Erstellen der zusätzlichen Properties, Events und States auf dem *ProducerAdvanced*, dem *StoreProducer* und der *ProductionUnit*.

Name	Typ	Station
RawMaterialX	Entity Property	ProducerAdvanced
RawMaterialXOrderSize	Expression Property	ProducerAdvanced
RawMaterialXProductionNumber	Integer Property	ProducerAdvanced
RawMaterialXProducerTable	Table Property	ProducerAdvanced
RememberCurrentInventoryRawMaterialX	Integer State Variable	ProducerAdvanced
OpenOrderRawMaterialX	Boolean State Variable	ProducerAdvanced
RawMaterialX	Entity Property	StoreProducer
InventoryRawMaterialX	Integer State Variable	StoreProducer
RawMaterialXDeliveryReceived	Event	StoreProducer
RawMaterialX	Entity Property	ProductionUnit
RequiredRawMaterialX	Integer Property	ProductionUnit
RawMaterialXReadyForProduction	Integer State Variable	ProductionUnit

Tabelle 23: Neu zu erstellende Properties, Events und States auf dem ProducerAdvanced, StoreProducer und der ProductionUnit

2. Erstellen des Prozesses «*Reorder_RawMaterialX*» äquivalent zu den bereits bestehenden «*Reorder_RawMaterial1*» und «*Reorder_RawMaterial2*». Dieser Prozess legt fest, bei welchem Hersteller Rohmaterial nachbestellt wird.

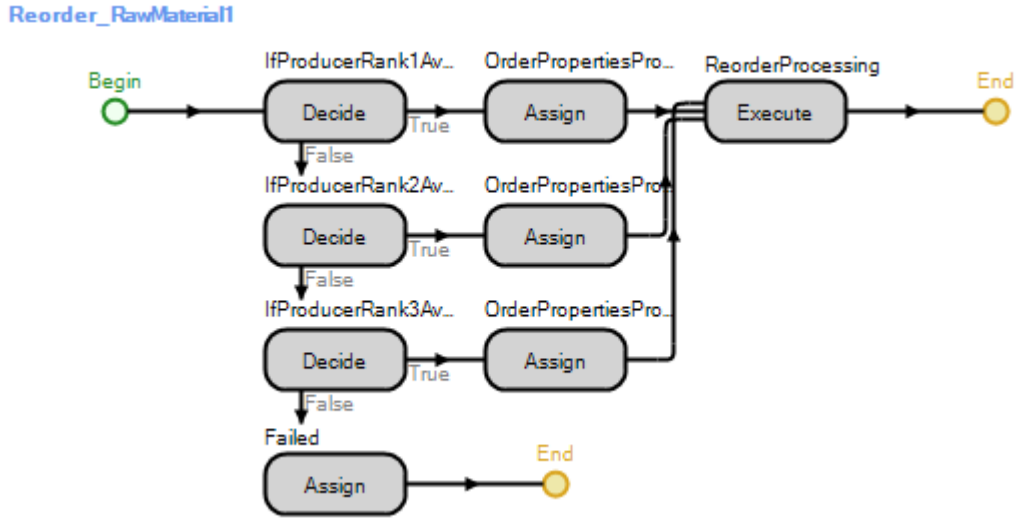


Abbildung 46: ProducerAdvanced Reorder_Rawmaterial1 Prozess

5. Ergänzen der Prozesse «*StorageUnit_Entered*» und «*StorageUnit_Exited*» auf dem *StoreProducer* analog zu den Schritten, welche für *RawMaterial1* und RawMaterial2 vorhanden sind.

3. Ergänzen des «Sink1_Entered» Prozesses auf der *ProductionUnit*. Dieser Prozess zählt die eingehenden Rohmaterialien und stellt daraus neue Produkte her.

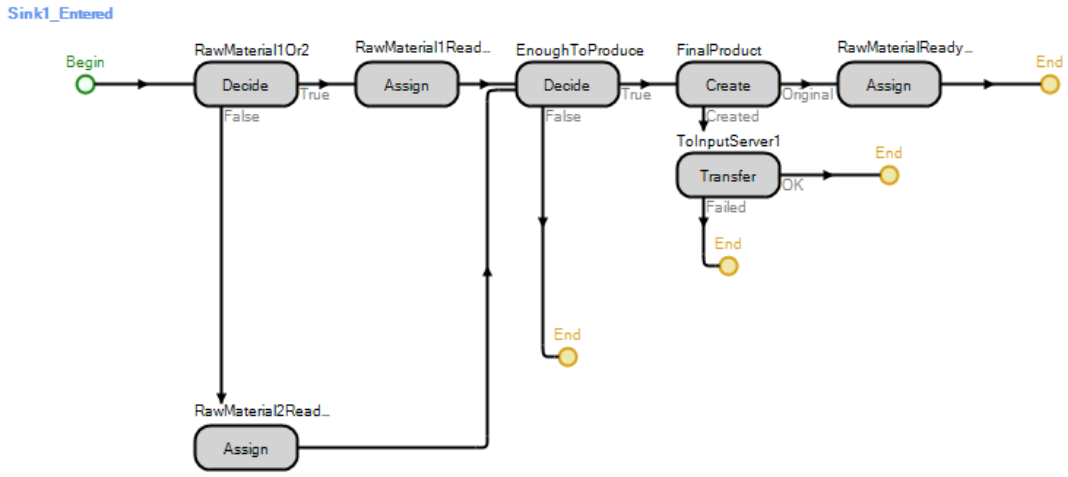


Abbildung 47: ProductionUnit Sink1_Entered Prozess

1. Um den Prozess um ein weiteres Rohmaterial zu ergänzen, muss auf dem *False*-Weg des *Decide* «*RawMaterial1Or2*» ein weiteres *Decide* eingefügt werden. Dieses neue *Decide* überprüft auf «*RawMaterial2Or3*».
2. Der *Assign* «*RawMaterial2ReadyForProduction*» muss auf den *True*-Weg des neuen *Decide* verschoben werden.
3. Auf dem *False*-Weg des neuen *Decide* soll nun ein *Assign* analog des eben verschobenen für *RawMaterialX* erstellt werden.
4. Dieses *Assign* verweist auf *Decide* «*EnoughToProduce*».
5. *Decide* «*EnoughToProduce*» und *Assign* «*RawMaterialReadyForProduction*» ebenfalls um *RawMaterialX* ergänzt werden.

4. Erweiterung des «*Production_Processing*» Prozesses im ***ProducerAdvanced*** um *ProductX*

Es wäre auch möglich den *ProducerAdvanced* so zu erweitern, dass dieser mehr als ein neues Produkt herstellen kann. Dafür müsste in der *ProductionUnit* auf dem «*Sink1_Entered*» Prozess ein zweites *Create* ergänzt werden und die Properties und States dahingehend ergänzt werden, dass ein zweites Produkt verwaltet wird. Für die Handhabung in der Simulation ist es jedoch realistischer, einen solchen Vorgang mit einem zweiten *ProducerAdvanced* darzustellen.

15 ANPASSUNG: SIMIO MODELL - INITIALISIERUNG

Sowohl *Warehouse* als auch *ProducerAdvanced* verfügen über einen Prozess, welcher bei Simulationsstart einen festgelegten Anfangsbestand von Verpackungen und Produkten erzeugt und im Lager bereitstellt. Dieser Initialisierungsprozess dient dem Testen von Abläufen und kann zu Verfälschungen in den berechneten Kosten führen. Das nachfolgende Kapitel erläutert, wie der Initialisierungsprozess des *Warehouses* um zusätzliche Produkte erweitert werden kann. Nachfolgend wird auf den Prozess im *Warehouse* eingegangen. Der Prozess im *ProducerAdvanced* ist äquivalent dazu anpassbar und wird nicht extra aufgeführt.

15.1 WAREHOUSE

Der Initialisierungsprozess erstellt einen Anfangsbestand für zwei Produkttypen und zwei Verpackungstypen und transferiert diese in ein internes Lager. Ebenfalls initialisiert der Prozess die Verfügbarkeit des *Warehouses*.

Um bei der Initialisierung einen Anfangsbestand für mehr als zwei Produkte zu ermöglichen, muss pro Produkt ein zusätzlicher Create Schritt erstellt werden.

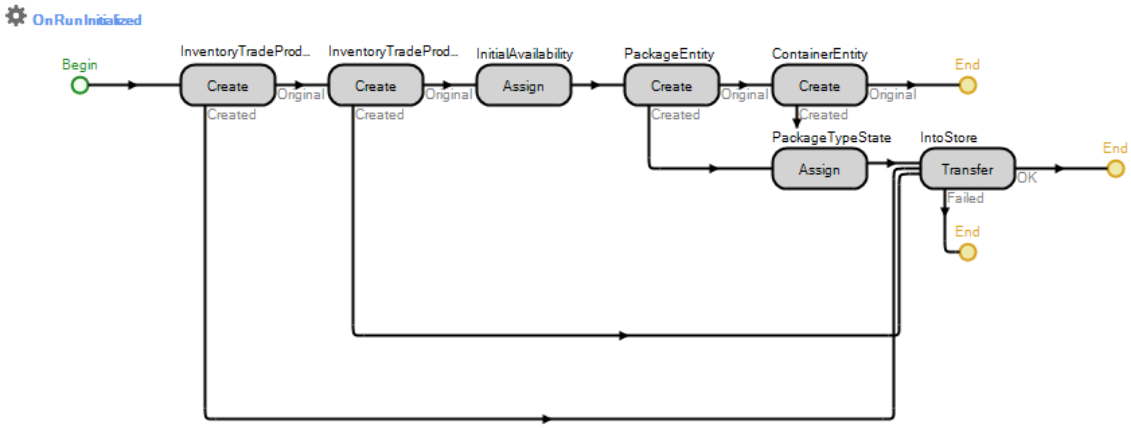


Abbildung 48: Warehouse OnRunInitialized Prozess

Für diesen Schritt werden zwei Angaben benötigt: 1. Welcher Entitytyp soll erstellt werden und 2. Die Menge, die erzeugt werden soll.

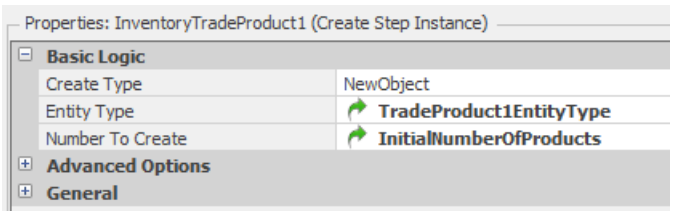


Abbildung 49: Warehouse OnRunInitialized InventoryTradeProduct1 Prozessschritt

Ermöglicht wird dies durch zwei neue Properties, die in Tabelle 2 definiert sind und in Simio unter Properties erstellt werden können.

Ist der zusätzliche Create Schritt erstellt, muss noch der Created Weg zum Transfer Schritt verbunden werden und der Prozess wurde dahin angepasst, den Anfangsbestand für ein weiteres Produkt zu erzeugen.

Name	Typ
TradeProductXEntityType	Entity Property
InitialNumberOfProducts	Integer Property

Tabelle 24: Warehouse OnRunInitialized Properties

16 ANPASSUNG: DECISION MATRIX

Dieser Abschnitt dient als Anleitung, wenn in einer weiterführenden Arbeit der Bedarf an der Menge der *Products* und *Producer* steigt und die Decision_Matrix.xlsx Datei angepasst wird.

Der Kennwortschutz der Blätter lautet: **supplychain**

16.1 PRODUCER LIST

16.1.1 NEW PRODUCT

Das Einfügen von neuen Produkten benötigt folgende Anpassungen:

- In der *ProductList* einen neuen Eintrag erstellen
- Die *RatingTable* um einen neuen Bereich ergänzen: Die Tabelle um den nötigen Bereich vergrössern und dann einen vorhergehenden Produktebereich kopieren und die Verweise anpassen.
- Neue Bereichsnamen* für den neu hinzugefügten Bereich vergeben, wobei X für die entsprechende Nummerierung steht.

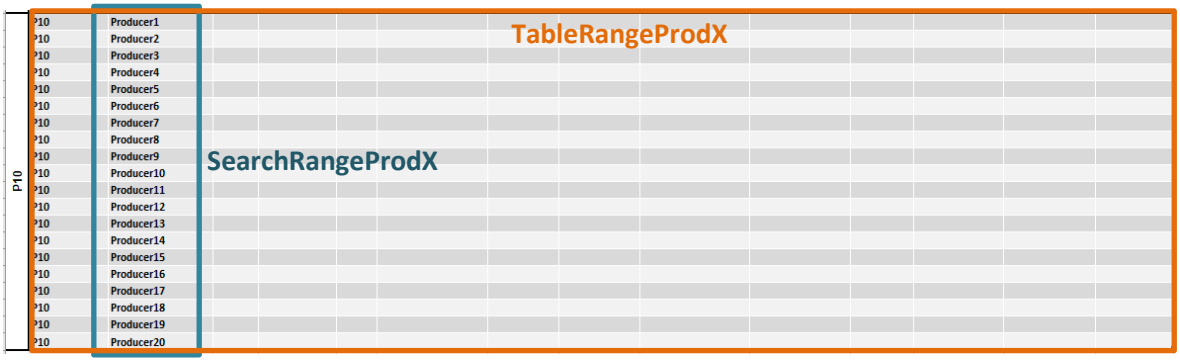


Abbildung 50: RatingTable Bereichsnamen Vergabe

*Diese Bezeichnung ist notwendig, da im Tabellenblatt *Matrices* die Ratings mit einem SVERWEIS() ausgelesen werden. Mit einem SVERWEIS() sind keine doppelten Einträge in einer Excel Tabelle abfragbar, weswegen auf Bereiche verwiesen werden muss.

16.1.2 NEW PRODUCER

Das Einfügen von neuen Producern ist umständlicher und benötigt folgende Anpassungen:

- In der *Producer* List einen neuen Eintrag erstellen
- In jedem Produktbereich eine neue Zeile hinzufügen und die Verweise gemäss den umliegenden Zeilen anpassen.
- Überprüfen ob die Bereichsnamen (siehe Abbildung 19) noch stimmen und gegebenenfalls anpassen. **Tipp:** Wenn die Zeile nicht als erste oder letzte Zeile in einem Bereich hinzugefügt wird, stattdessen irgendwo zwischendrin, wird der Bereich automatisch angepasst.

Sofern die Verweise stimmen sind noch die bedingten Formatierungen anzupassen. **Tipp**: Am besten rauslöschen und neu machen, da das Hinzufügen von Zeilen dazu neigt, die bedingten Formatierungen komplett durcheinander zu bringen. Der Einfachheit halber kann man die bedingten Formatierungen in diesem Bereich auch weglassen, da das Anzeigen der nötigen Produkte und *Producer* auch via der Filteroption erreichbar ist.

16.2 MATRICES

16.2.1 NEW PRODUCT

Für das Einfügen eines neuen Produktes, muss eine neue Decision Matrix hinzugefügt werden. Hierzu am besten eine Matrix aus dem oberen Bereich kopieren und neu einfügen.

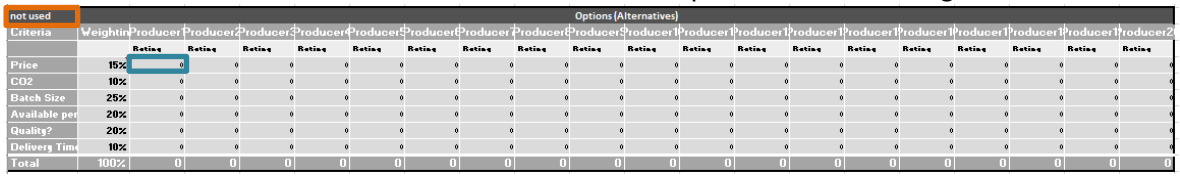


Abbildung 51: Matrices: Decision Matrix, anzupassende Felder

Folgende Anpassungen müssen vorgenommen, respektive überprüft werden:

- In der Produktnamen Zelle (Abbildung 20 orange) soll der Verweis auf die Zellennummer im Tabellenblatt «*Producer List*» *ProductList* stimmen.

```
=WENN('Producer List'!$C$11="YES";'Producer List'!$B$11;"not used")
```

- Die Formel unter Producer1 Rating (Abbildung 20 blau) benötigt einige Anpassungen. Man beachte folgende Vorher/Nachher-Formel von Produkt 9 auf Produkt 10.

```
=WENN($A$89<>"not used";SVERWEIS($A$89;WENN(SearchRangeProd9=C$90;TableRangeProd9;"");'Help Lists'!$A2);0)
```

```
=WENN($A$100<>"not used";SVERWEIS($A$100;WENN(SearchRangeProd10=C$101;TableRangeProd10;"");'Help Lists'!$A2);0)
```

  Der blaue Bereich muss auf die Zelle mit dem Produktnamen verweisen, der rote auf den Producer1 Spaltentitel, die *SearchRangeProd* und *TableRangeProd* Nummerierung muss angepasst werden auf die Nummerierung des entsprechenden Produkts, und der Aufruf der Help Lists muss auf A2 gesetzt werden. Mit diesen Änderungen kann die Formel dann nach rechts und runter gezogen werden und passt die Bezüge automatisch an.
- Die bedingten Formatierungen müssen um den Bereich der neuen Decision Matrix erweitert werden. Hierzu: Bedingte Formatierungen / Regeln verwalten / Aktuelle Auswahl. Durch das Kopieren sind folgende zwei bedingten Formatierungen neu erstellt worden:

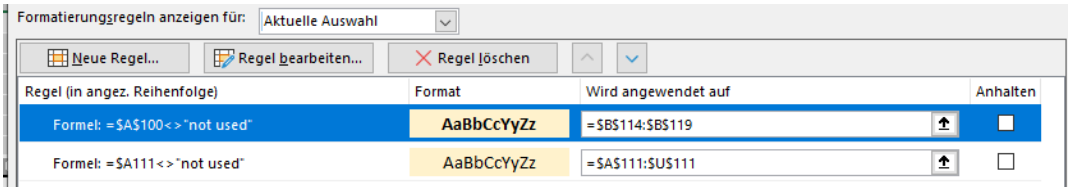


Abbildung 52: Matrices: Bedingte Formatierungen 1

Bei der ersten Regel muss der Zellenverweis in der Formel angepasst werden auf die «not used»/Produktnamen Zelle. Bei der zweiten Regel kann man den «Wird angewendet auf» Bereich kopieren, die Regel löschen, dann die Auswahl auf dieses Arbeitsblatt erweitern und den Bereich in der letzten Regel hinzufügen.

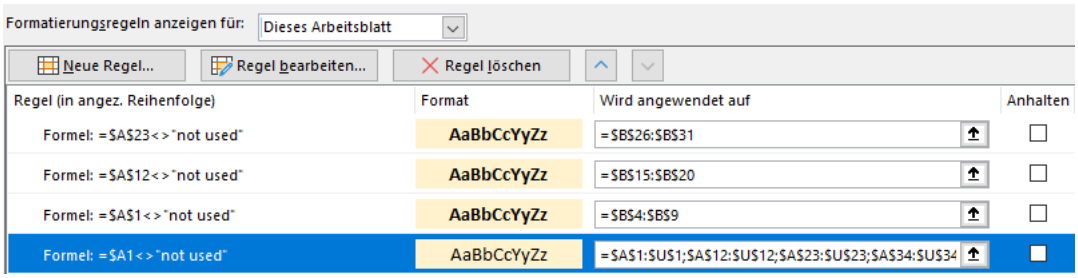


Abbildung 53: Matrices: Bedingte Formatierungen 2

16.2.2 New Producer

Beim Einfügen eines neuen *Producers*, empfiehlt es sich eine Spalte innerhalb des Matrizenbereichs hinzuzufügen, damit die Formatierungen übernommen werden. Also nicht am Ende anschliessend an den letzten *Producer*, sondern zum Beispiel zwischen den letzten beiden Producern eine neue Spalte hinzufügen.

Auf diese Art müssen dann lediglich die folgenden 3 Zeilen/Zellen angepasst werden.

not used													
										Options (Alternatives)			
Criteria	Weighting	Producer1	Producer2	Producer3	Producer4	Producer5	Producer6	Producer7	Producer8	Producer9	Producer10	Producer11	Producer12
		Rating	Rating	Rating	Rating	Rating	Rating	Rating	Rating	Rating	Rating	Rating	Rating
Price	15%	0	0	0	0	0	0	0	0	0	0	0	0
CO2	10%	0	0	0	0	0	0	0	0	0	0	0	0
Batch Size	25%	0	0	0	0	0	0	0	0	0	0	0	0
Available per 24h	20%	0	0	0	0	0	0	0	0	0	0	0	0
Quality?	20%	0	0	0	0	0	0	0	0	0	0	0	0
Delivery Time	10%	0	0	0	0	0	0	0	0	0	0	0	0
Total	100%	0	0	0	0	0	0	0	0	0	0	0	0

Abbildung 54: Matrices: Decision Matrix, anzupassende Bereiche bei einem neuen Producer

Die *Producer*-Zeilenbezüge müssen angepasst werden auf den neuen *Producer* Eintrag. Der Spaltentitel Rating muss übernommen werden. Und die Zelle mit der Formel muss noch nach rechts und nach unten gezogen werden, damit die Bezüge sich anpassen.

16.3 Intermediate Results

16.3.1 New Product

Für das Hinzufügen eines neuen Produkts reicht es, die ersten 3 Zellen einer Matrix (umrahmt) zu kopieren und am Schluss anzufügen.

not used										
NoProducer	NoProducer	NoProducer	NoProducer	NoProducer	NoProducer	NoProducer	NoProducer	NoProducer	NoProducer	NoProdu
0	0	0	0	0	0	0	0	0	0	

Abbildung 55: Intermediate Results: Zu kopierende Zellen

Dann müssen die Bezüge wie folgt angepasst werden:

- Die 1. Zelle soll auf die Producttitel-Zelle im Tabellenblatt *Matrices* verweisen.
- Der blaue Bezug der Formel in der 2. Zelle muss auf die 3. Zelle verweisen, siehe vorige Bezüge.

  `=WENN(A3=0;'Help Lists'!$C$1;Matrices!C$2)`

  Diese Formel kann dann nach rechts gezogen werden, um für alle *Producer* ausgefüllt zu werden.
- Zelle 3 verweist auf das Total der Decision Matrix im Tabellenblatt *Matrices*.

16.3.2 New Producer

Hier muss lediglich der Bereich der 2. Zeile angepasst werden, die Formel muss nach rechts gezogen werden, damit der Bereich für den neuen *Producer* auch ausgefüllt wird. Da Zelle 3 ein Matrixbezug ist, ist hierfür keine weitere Anpassung notwendig, wenn der neue *Producer* unter Matrices so hinzugefügt wurde, wie im vorhergehenden Kapitel beschrieben.

16.4 RESULTS

16.4.1 NEW PRODUCT

Zum Hinzufügen eines neuen Produkts müssen lediglich die in Abbildung 56 ausgewählten Zellen kopiert und am Schluss hinzugefügt werden.

Address	Producer	
Input_Management@ProducerPackaging1	ProducerPack	1.25
Input_Management@NoProducer	NoProducer	0
Input_Management@NoProducer	NoProducer	0
Input_Management@NoProducer	NoProducer	0
Input_Management@NoProducer	NoProducer	0
Input_Management@NoProducer	NoProducer	0
Input_Management@NoProducer	NoProducer	0
Input_Management@NoProducer	NoProducer	0
Input_Management@NoProducer	NoProducer	0
Input_Management@NoProducer	NoProducer	0
Input_Management@NoProducer	NoProducer	0
Input_Management@NoProducer	NoProducer	0
Input_Management@NoProducer	NoProducer	0
Input_Management@NoProducer	NoProducer	0
Input_Management@NoProducer	NoProducer	0
Input_Management@NoProducer	NoProducer	0
Input_Management@NoProducer	NoProducer	0
Input_Management@NoProducer	NoProducer	0
Input_Management@NoProducer	NoProducer	0

Abbildung 56: Results: zu kopierende Zellen

Der Matrixbezug in der Zelle unter dem *Producer* Spaltentitel muss so angepasst werden, dass er auf den richtigen Bereich im Tabellenblatt *Intermediate Results* zeigt.

```
=SORTIEREN(MTRANS('Intermediate Results'!A38:S39);2;-1)
```

Und die Zelle unter dem Spaltentitel Address muss nach unten gezogen werden, damit die Adressengenerierung für alle *Producer* übernommen wird.

Am Schluss muss analog der umrandeten Bereiche der vorhergehenden Produkte, der neue Bereich noch benannt werden mit *ProductXResults*, wobei X für die Nummerierung des *Products* steht, siehe Abbildung 57.

Product3Results

	A	B
43	**Address**	**Producer**
44	Input_Management@ProducerWater2	ProducerWate
45	Input_Management@ProducerWater3	ProducerWate
46	Input_Management@ProducerWater	ProducerWate
47	Input_Management@NoProducer	NoProducer
48	Input_Management@NoProducer	NoProducer
49	Input_Management@NoProducer	NoProducer
50	Input_Management@NoProducer	NoProducer
51	Input_Management@NoProducer	NoProducer
52	Input_Management@NoProducer	NoProducer
53	Input_Management@NoProducer	NoProducer
54	Input_Management@NoProducer	NoProducer
55	Input_Management@NoProducer	NoProducer
56	Input_Management@NoProducer	NoProducer
57	Input_Management@NoProducer	NoProducer
58	Input_Management@NoProducer	NoProducer
59	Input_Management@NoProducer	NoProducer
60	Input_Management@NoProducer	NoProducer
61	Input_Management@NoProducer	NoProducer
62	Input_Management@NoProducer	NoProducer

Abbildung 57: Results: Zu benennender Bereich

16.4.2 NEW PRODUCER

Beim Hinzufügen eines neuen *Producers* muss hier in jeder Tabelle eine neue Zeile hinzufügt werden, damit es keinen Überlauf gibt. Vorzugsweise wird diese Zeile analog vorhergehender Anpassungen innerhalb der bereits vorhandenen Tabellenzeilen hinzugefügt und nicht am Ende oder am Anfang, damit die Bereichsbezeichnung sich anpasst. Dann müssen noch folgende zwei Anpassungen vorgenommen werden:

- Im Matrixaufruf unter dem Spaltentitel *Producer* den Bereich vergrössern, sodass sie auch den neuen *Producer* umfasst.
- Die Formel unterhalb dem Address Spaltentitel nach unten ziehen, damit sich die Bezeichnung für alle *Producer* anpasst.

16.5 HELP LISTS

Die *Row Reference* Liste dient als Zeilennachweis für die übernommenen Ratings aus der *Producer* List in die Matrices. Die *Dropdown Options* ist die Auswahl für die *Products* und *Producer* Aufzählung unter *Producer* List. *NoProducer* dient als Defaultwert für *Producer* in den Resulttables. Wenn ein Rating für einen *Producer* gleich Null ist, dieser also nicht berücksichtigt werden soll für ein *Product*, wird automatisch auf den *NoProducer* verwiesen.

Hier sind **keine Änderungen** notwendig, wenn neue *Products* oder *Producer* hinzugefügt werden.

17 VERZEICHNISSE

17.1 ABBILDUNGSVERZEICHNIS

17.2 TABELLENVERZEICHNIS

Implementierung eines Supply-Chain-Management-Simulationstools

Technische Dokumentation

Studiengang Informatik
OST – Ostschweizer Fachhochschule
Campus Rapperswil-Jona

Herbstsemester 2020

Autoren:	Fabienne Lienhard, Jana Kravarik
Betreuer:	Prof. Dr.-Ing. Andreas Rinkel
Experte:	Knut Schmahl
Gegenleserin:	Prof. Dr. Nathalie Weiler
Abgabe:	15.01.2021

INHALTSVERZEICHNIS

18 EINLEITUNG

Dieses Dokument dient der detaillierten technischen Dokumentation der drei Prototypen, die im Rahmen der Bachelorarbeit «Implementierung eines Supply-Chain-Management-Simulationstools» entstanden sind.

Die Technische Dokumentation ist nach den verschiedenen Stationen der Simulation gegliedert. Sie beschränkt sich ausschliesslich auf die funktionale Beschreibung der Umsetzung in Simio. Die Überlegungen, die zu den jeweiligen Umsetzungen geführt haben, sind im Hauptdokument der Bachelorarbeit näher beschrieben.

Es wird jeweils die Grundfunktionalität für Prototyp1 beschrieben. Sofern in den weiterführenden Prototypen Änderungen an der Grundfunktionalität vorgenommen worden sind, werden diese Unter «Anpassungen» dokumentiert. Sind die Prototypen um neue Funktionalitäten ergänzt worden, wird ein neuer Abschnitt für den jeweiligen Prototypen hinzugefügt.

19 DARSTELLUNG

Um die Leserlichkeit zu erhöhen, sind bestimmte Begriffe speziell hervorgehoben. Diese Hervorhebung soll dem Leser verdeutlichen, in welcher Kategorie der Begriff einzuordnen ist. Unterschieden wird zwischen drei Kategorien:

Simio Begriffe Begriffe, die für Stationen und Kategorien gelten, die von Simio verwendet werden.

Zum Beispiel State, Server, Process, ...

Stationsnamen Während der Arbeit neu definierte und erstellte Simio Objekte.

Zum Beispiel *ProducerAdvanced, Warehouse, Unpack, …*

Eigennamen Während der Arbeit neu definierte Namen für Prozesse, Prozessschritte, Listen, States, Events, Properties und Elemente.

Zum Beispiel *OpenOrder, RememberOpenOrder, InitialAvailability*

20 TECHNISCHE DOKUMENTATION

20.1 PRODUCT

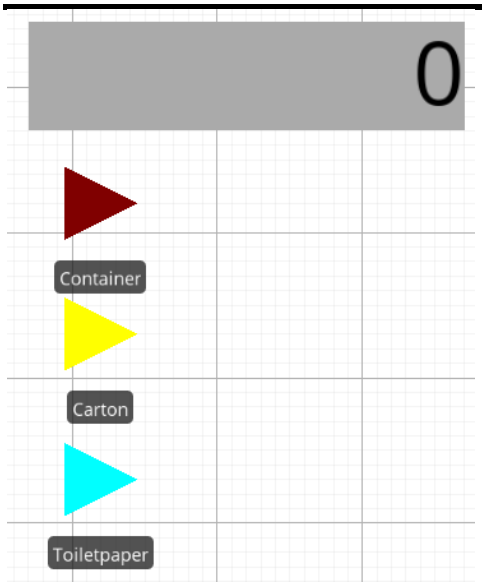


Abbildung 58: Product Types

Prototyp1:
Das Produkt wurde im Prototyp1 so angepasst, dass zwei verschiedene Verpackungstypen abbildbar sind. Die notwendigen zusätzlichen Properties, States und die Logik werden nachfolgend aufgeführt.

Der Verpackungstyp *Container* hat ein angehängtes Label, das anzeigt, wie viele Produkte sich darin befinden. Das Label wird in einer zukünftigen Version entfernt und dient im Prototyp1 der Fehleranalyse.

Prototyp2:
Das Produkt wird um zwei Einstellungen ergänzt, die für die Steuerung der *Product Entities* auf stationsinternen Wegen verwendet werden.

20.1.1 PRODUCT PROPERTIES

Name	Grundfunktionalität: Prototyp1	Anpassungen
PackageType-Property *(List)*	Auf dieser Property lässt sich der Verpackungstyp definieren.	
ProductType-Property *(List)*	Auf dieser Property lässt sich der Produkttyp definieren.	
Name	**Zusätzliche Properties: Prototyp2**	**Anpassungen**
IsDisposable-Property *(Boolean)*	Legt fest, ob es sich um eine Wegwerfverpackung, eine einmalig verwendbare Verpackung, handelt.	
InitialIsInternal-Sending *(Boolean)*	Legt fest, ob das Produkt stationsintern verschickt wird.	

Tabelle 25: Product Properties

20.1.2 PRODUCT STATES

Name	Grundfunktionalität: Prototyp1	Anpassungen
DeliveryAdd-Node *(Node Reference)*	Legt fest, wohin das Produkt versendet wird. Vor dem Verlassen einer Station wird dieser State als nächste Station festgelegt.	
OrderSize *(Integer)*	Legt die Menge der Bestellung/Lieferung fest.	
PackingType-_List *(List)*	Erlaubt das Umkategorisieren von *Packaging* zu *Product* und umgekehrt. Dies ist zum Beispiel dann notwendig, wenn *Carton* nachbestellt wird.	
Name	**Zusätzliche States: Prototyp2**	**Anpassungen**
IsDisposable *(Boolean)*	Wird dafür verwendet, Verpackungen nach einmaligem Gebrauch zu entsorgen.	
IsInternal-Sending *(Boolean)*	Wird dafür verwendet, stationsintern versendete Produkte an den richtigen Ort zu bewegen.	
Name	**Zusätzliche States: Prototyp3**	**Anpassungen**
Price *(Integer)*	Wird verwendet, wenn der Preis einer Sendung auf die Verpackung geschrieben wird.	
Content *(List)*	Gibt an, welche Produkte in einer Verpackung enthalten sind. Sofern die Verpackung über einen Inhalt verfügt.	

Tabelle 26: Product States

20.1.3 PRODUCT LISTS

Name	Grundfunktionalität: Prototyp1	Anpassungen
PackagingType-List *(Strings)*	In dieser Liste werden die möglichen Verpackungstypen aufgelistet. Bei Prototyp1 sind dies *Carton*, Container und *NonPackaging*.	
ProductTypeList *(Strings)*	In dieser Liste werden die möglichen Produkttypen aufgelistet. Bei Prototyp1 sind dies *Carton*, *Container*, *Toiletpaper* und *Beer*.	

Tabelle 27: Product Lists

20.1.4 PROZESSE

Der nachfolgende Prozess initialisiert den Boolean *InternalSendung* auf dem ***Product***.

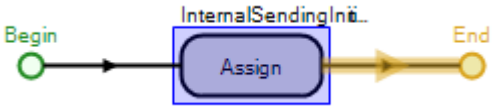
OnCreate (Prototyp3)	
	
Assign.InternalSending-InitialValue	Setzt den initialen Wert des States *InternalSending*, basierend auf der Property *InitialInternalSending*.

Tabelle 28: Product Process

20.2 ORDER

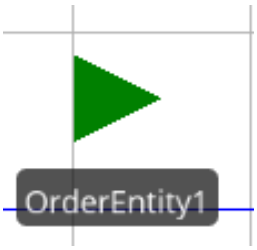


Abbildung 59: Order Entity

Prototyp1:
Die *Order* Entity wird im Prototyp1 nur noch für *Orders*, und nicht mehr für Container verwendet (Änderung zu Prototyp0). Gleichzeitig hat die *Order* den neuen Ordertypen *OrderDecline*. Dieser meldet an den Bestellenden zurück, falls eine *Order* nicht erfüllt werden kann und ermöglicht so, die Simulation ohne Anfangsbestand in den *Warehouses* durchzuführen.

Prototyp2:
Prototyp2 sieht keine funktionalen Änderungen an der *OrderEntity* vor. Property, State und List Namen sind dem Naming Concept[2] angepasst worden.

20.2.1 ORDER PROPERTIES

Name	Grundfunktionalität: Prototyp1	Anpassungen
InitialDelivery-AddressNode *(Node)*	Legt fest, wohin die Produkte geliefert werden. Dies entspricht im Prototyp0 immer dem Wareneingang der Station, welche die Bestellung abgeschickt hat.	**Prototyp2**: Umbenennen von *InitialDeliveryAdd_Node* zu *InitialDeliveryAddressNode*
InitialNumber-Ordered *(Integer)*	Legt fest, wie viele Produkte mit einer Bestellung bestellt werden.	
InitialOrder-AddressNode *(Node)*	Legt fest, wohin die *Order* geschickt wird. Als Initialwert wird der Bestelleingang des *Warehouses* verwendet.	**Prototyp2**: Umbenennen von *InitialOrderAdd_Node* zu *InitialOrderAddressNode*.
Packaging-EntityType *(List)*	Gibt den Verpackungstypen beim Erstellen eines *Products* an, verwendet die Liste *PackagingType_List*.	**Prototyp3**: Umbenennen von *PackagingType* zu *PackagingEntityType*
OrderedProduct EntityType *(List)*	Gibt den Produkttyp beim Erstellen eines *Products* an, verwendet die Liste *ProductType_Liste*.	**Prototyp3**: Umbenennen von *OderedProductType* zu *OrderedProductEntityType*

Tabelle 29: Order Properties

20.2.2 ORDER STATES

Name	Grundfunktionalität: Prototyp1	Anpassungen
Delivery-AddressNode *(Node Reference)*	Legt fest, wohin die Produkte geliefert werden. Dies entspricht im Prototyp0 immer dem Wareneingang der Station, welche die Bestellung abgeschickt hat.	**Prototyp2**: Umbenennen von *DeliveryAdd_Node* zu *DeliveryAddressNode*
Number-Ordered *(Integer)*	Legt fest, wie viele Produkte mit einer Bestellung bestellt werden.	

[2] Zu finden im Dokument: Projektplan_Implementierung eines Supply-Chain-Management-Simulationstools

OrderAddress-Node *(Node Reference)*	Legt fest, wohin die *Order* geschickt wird. Als Initialwert wird der Bestelleingang vom Lager (Storage/SimpleWarehouse) verwendet.	**Prototyp2**: Umbenennen von *OrderAdd_Node* Zu *OrderAddressNode*
PackagingType-List *(List)*	Legt fest, in welchen Verpackungstyp die bestellten Produkte verpackt werden sollen.	**Prototyp2**: Umbenennen von *PackagingType_List* Zu *PackagingTypeList*
ReturnAddress-Node *(Node Reference)*	Legt fest, wohin *OrderDecline* Nachrichten geschickt werden.	
OrderTypeList *(List)*	Legt fest, ob eine Bestellung eine *Order* oder eine *OrderDecline* Nachricht ist.	**Prototyp2**: Umbenennen von *OrderType_List* zu *OrderTypeList*
OrderedProductList *(List)*	Legt fest, welcher Produkttyp bestellt wird.	**Prototyp2**: Umbenennen von *OrderedProduct_List* zu *OrderedProductList*

Tabelle 30: Order States

20.2.3 Order Lists

Name	Grundfunktionalität: Prototyp1	Anpassungen
PackagingType_List	In dieser Liste sind die bestellbaren Verpackungstypen definiert. Im Prototyp1 sind dies *Container* und *Carton*.	**Prototyp2**: Umbenennen von *PackagingTypeList* zu *PackagingType_List*
OrderType_List	In dieser Liste sind die Bestellungsarten aufgeführt. Im Prototyp1 sind dies *Order* und *OrderDecline*.	**Prototyp2**: Umbenennen von *OrderTypeList* zu *OrderType_List*
ProductType_List	In dieser Liste werden die bestellbaren Produkttypen definiert.	**Prototyp2**: Umbenennen von *ProductTypeList* zu *ProductType_List*

Tabelle 31: Order Lists

20.2.4 Prozesse

Der nachfolgende Prozess initialisiert die Einstellungen auf neu erstellten *Order* Entities.

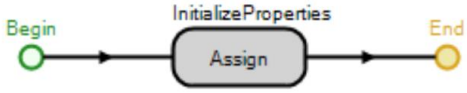
OnCreated *und* **OnRunInitialized (Prototyp1-3)**	
	
InitializeProperties	Schreibt die Lieferadresse, die Bestelladresse, die bestellte Stückzahl sowie das bestellte Produkt auf eine *Order*.

Tabelle 32: Order Process

20.3 WAREHOUSE

Prototyp1:

Das *Warehouse* ist in verschiedene Teilstationen unterteilt und so aufgebaut, dass es beliebig oft hintereinander verwendet werden kann. Es kann *Customer* und andere *Warehouses* beliefern.

Prototyp2:

Im Prototyp2 ist das *Warehouse* um die Möglichkeit, den *Producer* dynamisch auszuwählen, ergänzt. Des Weiteren sind die Prozesse und Teilstationen so erweitert, dass das *Warehouse* verschiedene Produkte parallel verwalten, lagern und nachbestellen kann.

Prototyp3:

Im Protoyp3 sind keine grösseren Änderungen vorgenommen worden. Probleme mit der Vorherigen Version sind gelöst und Abläufe optimiert worden. Des Weiteren wurden die Properties in Kategorien unterteilt, um die Bedienung benutzerfreundlicher zu gestalten.

20.3.1 ABLÄUFE

Das *Warehouse* Management verwaltet die folgenden Abläufe:

- Entgegennehmen und Zerstören von *Orders*
- Lagern von *Products* und *Packagings*
- Überprüfen, ob die bestellte Menge im Lager vorhanden ist und basierend darauf, die Entscheidung, ob Waren geliefert werden können
- Überprüfen, ob der Lagerbestand einen festgelegten Schwellwert erreicht hat, und, falls dieser erreicht wird, das Auslösen einer Bestellung
- Liefern von Produkten aus dem Lager an eine, von der Bestellung vorgegebene, Lieferadresse
- Auszuliefernde Produkte in der von der *Order* festgelegten Verpackung verpacken
- Entgegennehmen und Auspacken von Produkten, welche in *Containern* verpackt ankommen und das Weiterleiten in das interne Lager.
- Aussenden von *OrderDecline* Nachrichten an den Bestellenden, falls eine Bestellung nicht erfüllt werden kann
- Auswählen von passenden *Producern*, anhand einer Rangliste und deren Verfügbarkeit

20.3.2 AUFBAU

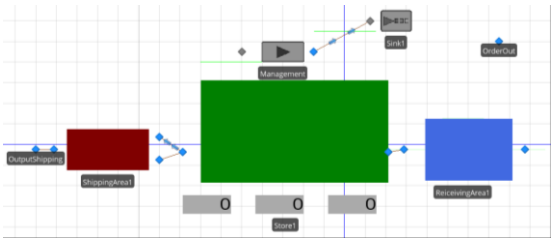
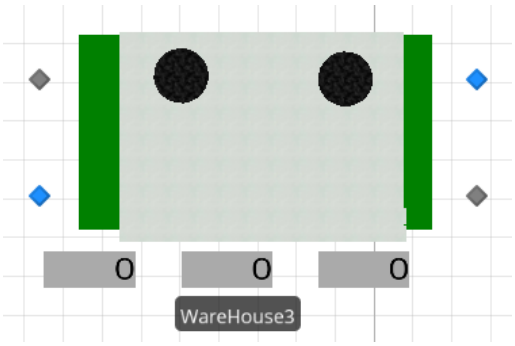
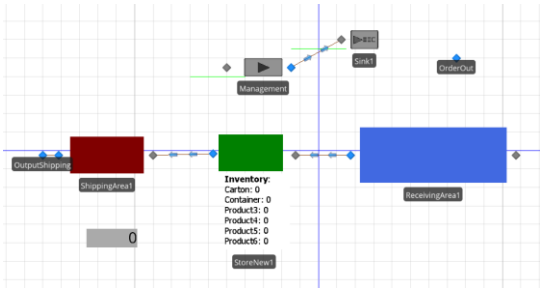
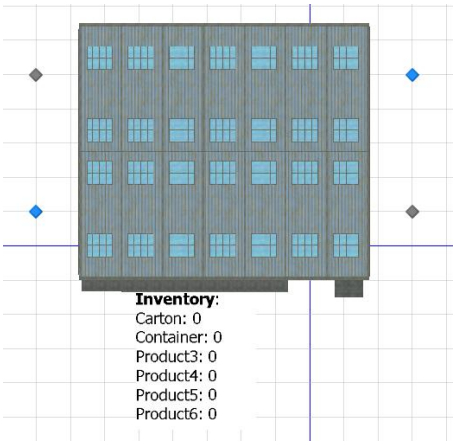
Prototyp1, Prototyp2	
Interner Aufbau	**Externe Modellansicht**
	
Prototyp3	
Interner Aufbau	**Externe Modellansicht**
	
Das *Warehouse* besteht aus 3 Teilstationen und dem Management Server. Das Management nimmt ankommende *Orders* entgegen und löst die bestellten Sendungen oder Nachbestellungen aus. Wenn das *Warehouse* eine Bestellung nicht erfüllen kann, wird eine *OrderDecline* Nachricht zurück an den Sender der *Order* geschickt.	

Tabelle 33: Warehouse Aufbau

20.3.3 WAREHOUSE ELEMENTS

Name	Grundfunktionalität: Prototyp1	Anpassungen
CostCenterCO2-Packaging *(Cost Center)*	Für die Verwaltung der CO_2 Kosten des Verpackungsvorgangs. **Prototyp3**: Wurde zu Verpackungskosten umfunktioniert.	**Prototyp3**: Umbenennen von *CostCenterCO_2Packaging* zu *CostCenterPackaging*.
CostCenter-Waste-Packaging *(Cost Center)*	Für die Verwaltung der Kosten, welche durch das Entsorgen von Einwegverpackungen entstehen.	

CostCenter-Storage *(Cost Center)*	Für die Verwaltung der Kosten, welche durch die Einlagerung von Produkten entstehen.	
Name	**Zusätzliche Elements: Prototyp3**	**Anpassungen**
CostCenter-Product-[1-10]Return *(Cost Center)*	CostCenter für die Verwaltung der Return Cost der einzelnen Produktarten.	
CostCenter-Product-[1-10]Income *(Cost Center)*	CostCenter für die Verwaltung der Income Cost der einzelnen Produktarten.	
CostCenter-Product-[1-10]Purchase *(Cost Center)*	CostCenter für die Verwaltung der Purchase Cost der einzelnen Produktarten.	
CostCenter-ProductReturn-Total *(Cost Center)*	CostCenter für das Return über alle Produkte.	
CostCenter-ProductIncome-Total *(Cost Center)*	CostCenter für das Income über alle Produkte.	
CostCenter-Product-PurchaseTotal *(Cost Center)*	CostCenter für das Purchase Total.	
CostCenter-TotalInvestment *(Cost Center)*	CostCenter über alle Ausgaben.	

Tabelle 34: Warehouse Elements

20.3.4 WAREHOUSE PROPERTIES

Name	Grundfunktionalität: Prototyp1	Anpassungen
TradeProduct1-EntityType *(Entity)*	Legt fest, was für ein Producttyp in diesem Lager gelagert und nachbestellt werden kann. **Prototyp2 und Prototyp3:** Von diesem Produkt wird bei der Initialisierung ein Anfangsbestand erstellt.	**Prototyp2**: Umbenennen auf *ProductType1* und anpassen der Funktionalität. **Protoyp3**: Umbenennen auf *TradeProduct1EntityType*
InitialNumber-ofProducts *(Expression)*	Legt den Anfangsbestand der Produkte im *Warehouse* fest.	
InitialReorder-Point *(Integer)*	Legt den Schwellwert fest, also ab wie vielen Produkten im Lager nachbestellt wird. Der Lagerbestand wird nach dem Versenden von Produkten oder nach Erhalten einer *Order-Decline* Nachricht überprüft.	
ChosenProducer *(Node)*	Gibt die Adresse an, an welche Bestellungen geschickt werden.	**Prototyp2**: Property ist obsolet und wird gelöscht.
Packaging1-EntityType *(Entity)*	Zum Angeben der *Packaging1Entity*	**Prototyp2**: Umbenennen von *PackageType* zu *CartonEntity* **Prototyp3**: Umbenennen von *CartonEntity* zu *Packing1EntityType*
Number-Ordered *(Expression)*	Definiert, wie viele Produkte bei einer ausgehenden Bestellung bestellt werden.	**Prototy2**: Umbenennen von *Number_Ordered* zu *Number-Ordered*
Packaging2-EntityType *(Entity)*	Zum Angeben der *Packaging2Entity*.	**Prototyp3**: Umbenennen von *ContainerEntity* zu *Packaging2EntityType*
ReturnAddress *(Node)*	Adresse der Node, an welcher *OrderDecline* Nachrichten entgegengenommen werden.	
Packaging-WasteCost *(Expression)*	Definiert, wie teuer das Entsorgen einer einzelnen Verpackung ist. In Prototyp1 gilt dieser Wert für die Entsorgung von *Carton*.	
OrderSize-Packaging *(Expression)*	Definiert, wie viel *Carton* nachbestellt werden soll, wenn der Schwellwert erreicht wird. **Prototyp3**: Gilt für Nachbestellungen beider *Packaging* Typen.	**Prototyp3**: Umbenennen von *OrderSizeCarton* zu *OrderSizePackaging*

PackagingCost *(Expression)*	Definiert, wie teuer der Verpackungsvorgang ist.	**Prototyp3**: Umbennenung zu *PackagingCost*
StorageCost-PerProduct *(Expression)*	Definiert, wie teuer die Lagerung eines einzelnen Produktes während einer Stunde ist.	
Name	**Zusätzliche Properties: Prototyp2**	**Anpassungen**
TradeProduct2-EntityType *(Entity)*	Von diesem Produkt wird bei Initialisierung ein Anfangsbestand erstellt.	**Prototyp2**: Umbennenung von *TradeProduct2 zu TradeProduct2EntityType*
TableProducers-Product[1-10] *(Table)*	Verlinkung zu den Data Tables, welche die Rating Lists zu den *Producern* pro Produkt enthalten. Die Standardwerte enthalten die Standardnamen der Lists für die Produkte 1-10.	
Name	**Zusätzliche Properties: Prototyp3**	**Anpassungen**
Initial-Availability *(Boolean)*	Legt die initiale Verfügbarkeit der Station fest.	
InitialNumber-ofPackaging[1-2] *(Integer)*	Legt fest, wie viel Verpackungen beider Typen beim Beginn der Simulation im Lager vorhanden ist.	
SellPrice-Product[1-10] *(Integer)*	Legt den Verkaufspreis für die jeweiligen Produkte 1-10 fest.	

Tabelle 35: Warehouse Properties

20.3.5 WAREHOUSE STATES

Name	Grundfunktionalität: Prototyp1	Anpassungen
ReorderPoint *(Integer)*	Legt den Schwellwert fest, also ab wie vielen Produkten im Lager nachbestellt wird. Der Lagerbestand wird nach dem Versenden von Produkten oder nach Erhalten einer *Order-Decline* Nachricht überprüft.	
Remember-DeliveryAddress *(Node Reference)*	In diesem State wird die Lieferadresse zwischengespeichert. Von diesem Wert wird sie auf die zu liefernden Produkte übertragen.	
OpenOrder *(Integer)*	Mit diesem State kann das *Warehouse* überprüfen, ob es bereits eine Bestellung gesendet hat, die noch nicht erfüllt wurde. Es verhindert mehrere aufeinanderfolgende	**Prototyp2**: Deprecated

	Bestellungen aufgrund der gleichen Überschreitung des Schwellwertes.	
Remember-Number-Ordered *(Integer)*	In diesem State wird die Bestellgrösse zwischengespeichert, um die Bestellmenge beim Versenden in Containern auf ein Container Label schreiben zu schreiben.	
Remember-ReturnAddress *(Node Reference)*	In diesem State wird die Rücksendeadresse zwischengespeichert.	
OpenOrder-Packaging *(Integer)*	Dieser State wird dafür gebraucht, dass das *Warehouse* überprüfen kann, ob es bereits eine Bestellung für *Carton* rausgeschickt hat.	**Prototyp2**: Deprecated
Name	**Zusätzliche States: Prototyp2**	**Anpassungen**
Remember-Current-Inventory *(Integer)*	Zwischenspeicher für den aktuellen Lagerbestand eines Produktes.	
Remember-OrderedProduct *(Integer)*	Zwischenspeicher für den Typ des *Products* der aktuellen Bestellung.	
Remember-Current-Inventory-Packaging *(Integer)*	Zwischenspeicher für den aktuellen Lagerbestand der gewünschten Verpackung einer Bestellung.	
Remember-PackagingType *(Integer)*	Zwischenspeicher für den gewünschten Verpackungstypen einer Bestellung.	
OpenOrder-Product1-10 *(Boolean)*	State Variablen für das Dokumentieren, ob das Produkt eine offene Nachbestellung hat.	
CurrentOrder-Address *(Node)*	Zwischenspeicher für die Adresse des für eine Nachbestellung ausgesuchten *Producers*.	
Name	**Zusätzliche States: Prototyp3**	**Anpassungen**
Availability *(Boolean)*	Gibt die Verfügbarkeit der Station an.	

Tabelle 36: Warehouse States

20.3.6 SCHNITTSTELLEN

Das *Warehouse* verfügt über die folgenden vier Schnittstellen:

Bestelleingang Der Bestelleingang wird beim *Warehouse* auch als Management Eingang bezeichnet. Er nimmt Bestellungen entgegen und leitet diese zur Bearbeitung an das *Warehouse*-Management weiter.

Warenausgang Sendungen (verpackte Produkte) verlassen das *Warehouse* über diesen Knoten.

Wareneingang Sendungen werden über diesen Knoten entgegengenommen.

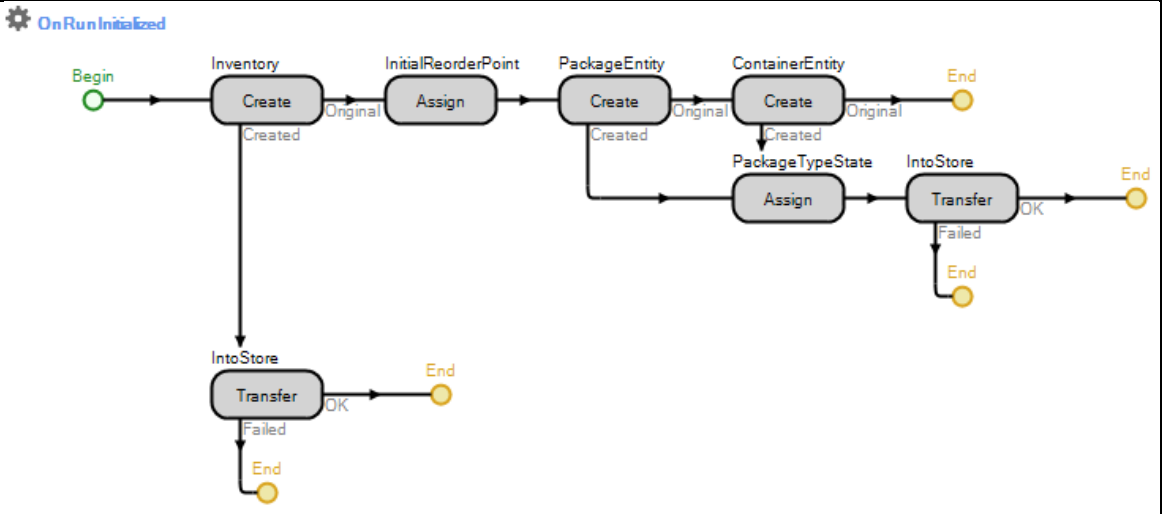
Bestellausgang Lieferungen werden über diesen Knoten verschickt.

20.3.7 PROZESSE

20.3.7.1 Prototyp1

Nachfolgender Prozess wird bei Simulationsstart ausgeführt.

OnRunInitialized (Prototyp1)



Create.Inventory	Erzeugt den Anfangsbestand der Produkte im Lager.
Assign.InitialReorderPoint	Legt den Schwellwert fest.
Transfer.IntoStore	Verschiebt den Anfangsbestand an das InputNode des Lagers.
Create.PackageEntity	Erzeugt den Anfangsbestand von *Carton* im Lager.
Create.ContainerEntity	Erzeugt den Anfangsbestand von *Containern* im Lager.
Assign.PackageTypeState	Legt den PackagingType der generierten Verpackung fest.
Transfer.IntoStore	Verschiebt die erzeugten Verpackungen an das InputNode des Lagers.

Tabelle 37: Warehose Process: OnRunInitialized

Wenn Produkte im Lager ankommen, wird ein Event (*OrderReceived*) ausgelöst. Dieses Event löst nachfolgenden Prozess aus, der den State *OpenOrder* auf 0 setzt. So weiss das *Warehouse*, dass es keine offene Bestellung hat.

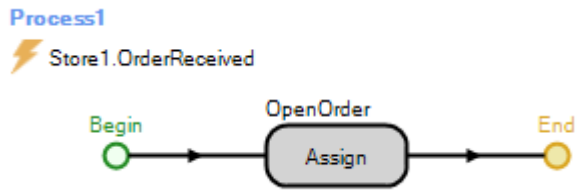
Process1 (Prototyp1)	
	
Assign.OpenOrder	Setzt *OpenOrder* auf 0.

Tabelle 38: Warehouse Process: Process1

Wenn *Carton* im Lager ankommen, wird ein Event ausgelöst. Dieses Event löst folgenden Prozess aus, der die *OpenOrder* für *Carton* auf 0 setzt. So weiss das *Warehouse*, dass es keine offene Bestellung hat.

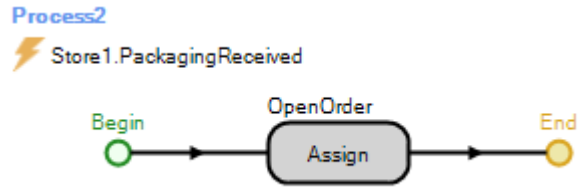
Process2 (Prototyp1)	
	
Assign.OpenOrder	Setzt *OpenOrderPackaging* auf 0.

Tabelle 39: Warehouse Process: Process2

Nachfolgend der Prozess, der für die Verwaltung des *Container Labels* verantwortlich ist.

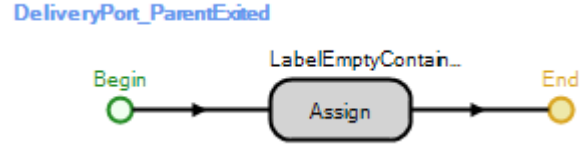
DeliveryPort_ParentExited (Prototyp1)	
	
Assign.LabelEmptyContainer	Sobald eine Warenlieferung das Lager erreicht, wird die Stückzahl auf dem dazugehörigen *Container* auf 0 gesetzt.

Tabelle 40: Warehouse Process: DeliveryPort_ParentExited

Der Management Prozess bearbeitet einkommende Bestellungen, verwaltet das Lager und verschickt Bestellungen und Waren.

Management_Processing (Prototyp1)	
Prozess zu sehen in Abbildung 3.	
Decide.OrderOrOrderDecline	Entscheidet, ob die eingehende *OrderEntity*, eine Bestellung oder eine *OrderDecline* Nachricht ist.
Decide.IfDeclineForProduct	Entscheidet, ob die eingehende *OrderDecline* Nachricht für *Products* oder *Carton* ist.
Assign.OpenOrder	Setzt *OpenOrder* zurück auf 0, damit das *Warehouse* weiss, dass es keine offene Bestellung für Produkte hat.
Assign.OpenOrderPackagging	Setzt *OpenOrderPackaging* zurück auf 0, damit das *Warehouse* weiss, dass es keine offene Bestellung für *Carton* hat.
Assign.RememberDeliveryAddress	Schreibt die Lieferadresse, die Liefermenge und die Rücksendeadresse in die dafür vorgesehenen Zwischenspeicher.
Decide.IfProduct	Überprüft, ob die Bestellung für *Products* oder *Carton* ist.
Decide.EnoughInStore	Überprüft, ob genug *Products* oder *Carton* für die Bestellung im Lager sind.
Create.OrderDecline	Erzeugt eine *OrderEntity*.
Assign.OrderType	Setzt den OrderTyp auf *OrderDecline* und schreibt die Rücksendeadresse aus dem Zwischenspeicher in die Rücksendeadresse der *Order*.
SetNode.ReturnAddress	Setzt die Rücksendeadresse als nächsten Hop.
Search.ContentForTransfer	Holt die bestellte Menge an *Products* oder *Carton* aus dem entsprechenden Store.
Assign.DelieveryAddress	Schreibt die Lieferadresse aus dem Zwischenspeicher auf die im vorhergehenden Search Schritt gefundenen Produkte.
SetNode.DeliveryAddress	Setzt die Lieferadresse als nächsten Hop.
Transfer.IntoShipping	Verschiebt die zuvor gefundenen *Products* oder *Carton* in den Output Buffer.
Decide.ReorderpointCrossed	Überprüft, ob der Lagerbestand noch über dem Reorderpoint liegt.
Decide.CheckIfOpenOrder	Überprüft, ob bereits eine offene Bestellung verschickt worden ist. Verhindert, das mehrmals nachbestellt wird, bevor die Lieferung ankommt.
Create.CreateOrder	Erstellt eine *OrderEntity*.

Assign.OrderProperties	Setzt die Parameter für die *Order*, welche im vorherigen Schritt erstellt wurde. Gesetzt werden Bestelladresse, Lieferadresse, Liefergrösse, Verpackungstyp, Bestelltyp, Rücksendeadresse und welches Produkt bestellt wird.
Assign.SetOpenOrder	Setzt *OpenOrder* auf 1, damit das *Warehouse* weiss, dass es bereits eine offene Bestellung hat.
SetNode.SetOrderAddress	Setzt den nächsten Hop nach dem Verlassen des *Warehouses*.
Transfer.SendOutOrder	Schiebt die *OrderEntity* zum ExitNode des *Warehouses*.
Assign.Role	Wenn *Carton* bestellt wird, wird der Producttyp mit diesem Schritt auf *Nonpackaging* gesetzt. So kann *Carton* in *Container* verpackt werden.

Tabelle 41: Warehouse Process: Management_Processing

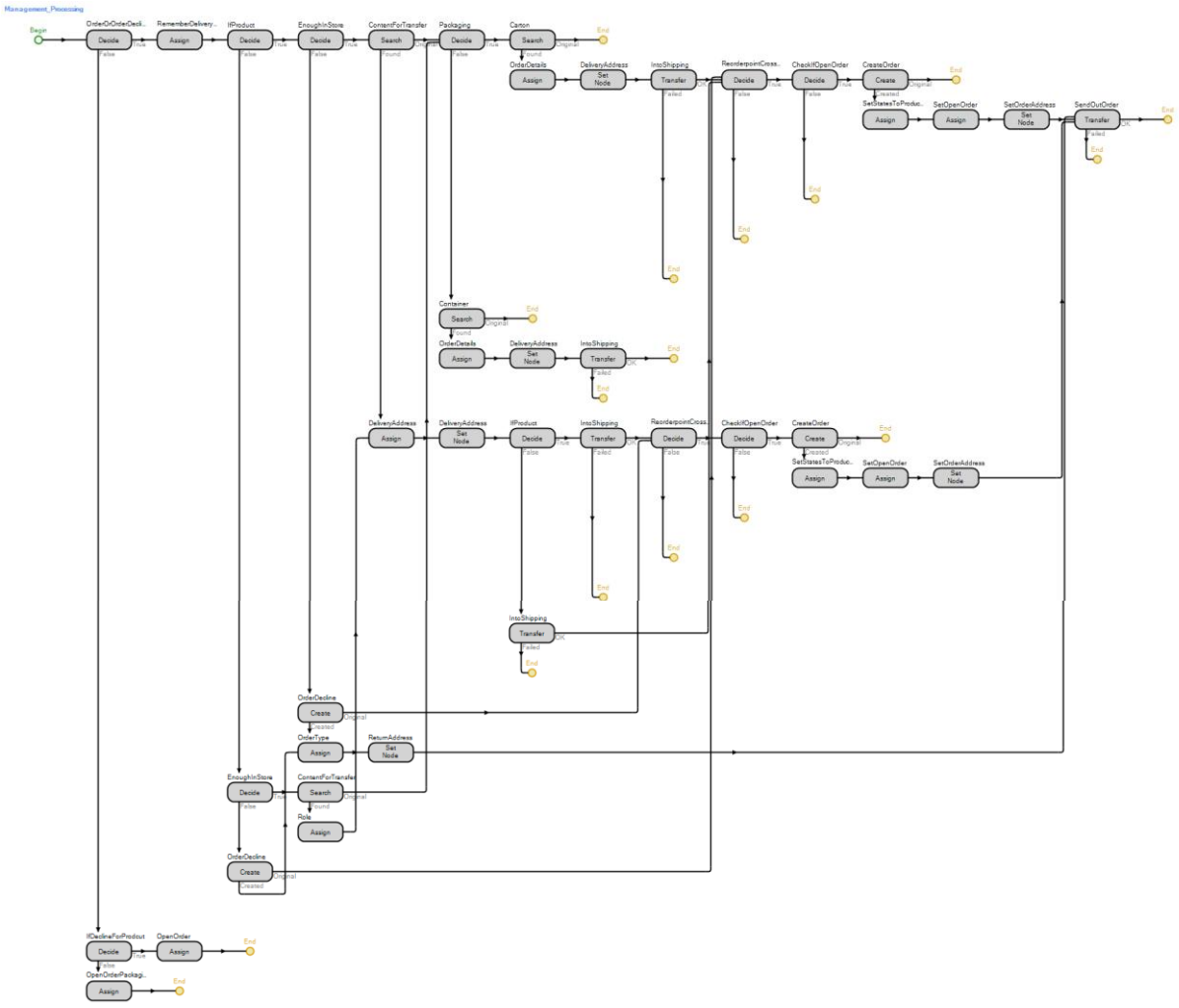


Abbildung 60: Warehouse Management Process

20.3.7.2 Prototyp2

Nachfolgender Prozess setzt den State, welcher speichert, ob für ein Produkt bereits eine Bestellung draussen ist auf True.

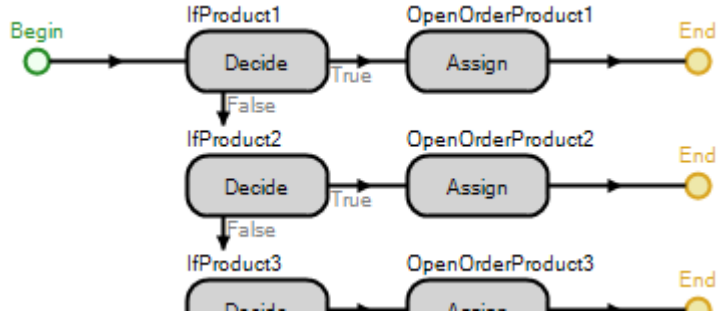
Set_OpenOrder (Prototyp2-3)	
	
Decide.IfProduct1-10	Überprüft, ob das nachbestellte Produkt Produkt1-10 ist.
Assign.OpenOrderProduct1-10	Setzt den produktspezifischen State auf *True*.

Tabelle 42: Warehouse Process: Set_OpenOrder

Dieser Prozess dient der Überprüfung, ob für ein Produkt bereits eine *Order* versendet wurde. Er besteht aus den gleichen drei Schritten für jeden Producttyp.

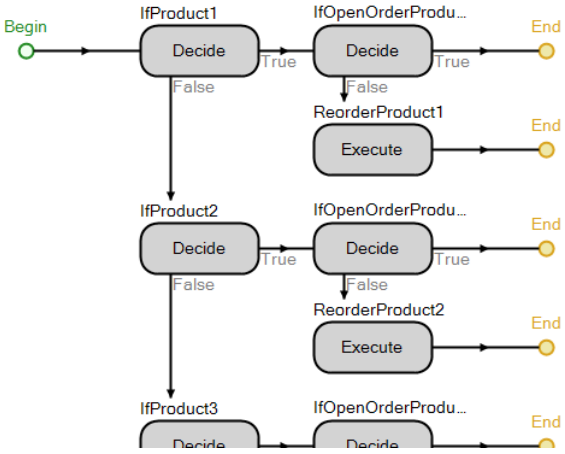
Check_OpenOrder (Prototyp2-3)	
	
Decide.IfProduct1-10	Überprüft, ob das zu nachbestellende Produkt gleich dem Produkt1-10.
Decide.IfOpenOrderProduct1-10	Überprüft, ob *OpenOrder* für das jeweilige Produkt auf *True* gesetzt ist.
Execute.ReorderProduct1	Löst den *Reorder*-Prozess für das jeweilige Produkt aus.

Tabelle 43: Warehouse Process: Check_OpenOrder

Nachfolgend der Prozess, um *OpenOrder* beim Eingehen einer Bestellung wieder auf *False* zu setzen.

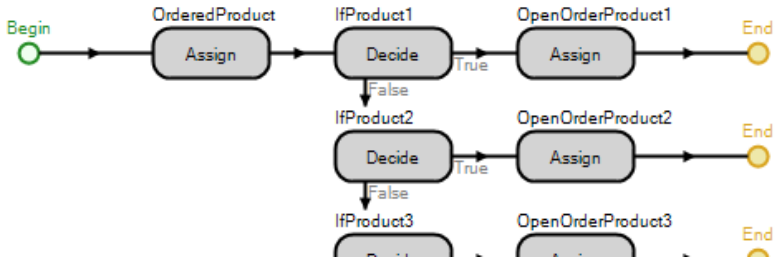
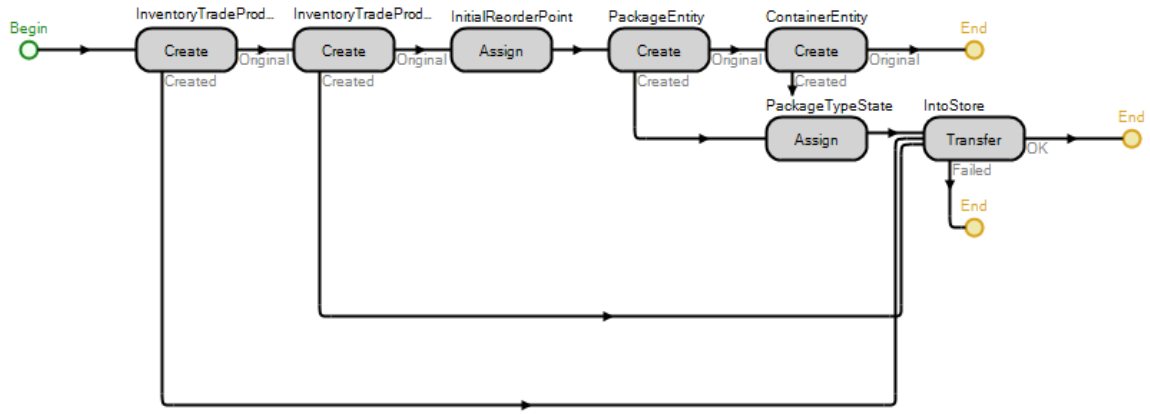
Reset_OpenOrder (Prototyp2-3)	
	
Assign.OrderedProduct	Legt das *Product*, für welches der State zurückgesetzt werden muss, fest.
Decide.IfProduct1-10	Überprüft, ob das angekommene *Product* gleich dem Produkt1-10 ist.
Assign.OpenOrderProduct1-10	Setzt den produktspezifischen State auf *False*.

Tabelle 44: Warehouse Process: Reset_OpenOrder

Der Prozess *OnRunInitialized* ist im Prototyp2 mit einem zweiten Produkt für einen Anfangsbestand ergänzt, sowie um Redundanz verringert worden.

OnRunInitialized (Prototyp2-3 basierend auf Version von Prototyp1)	
	
Create.InventoryTradeProduct1-2	Erzeugen den Anfangsbestand der *TradeProducts* eins und zwei im Lager.
Assign.InitialReorderPoint	Legt den Schwellwert fest.
Transfer.IntoStore	Verschiebt den Anfangsbestand an das InputNode des Lagers.
Create.PackageEntity	Erzeugt den Anfangsbestand für *Carton* im Lager.
Create.ContainerEntity	Erzeugt den Anfangsbestand der *Container* im Lager.
Assign.PackageTypeState	Legt den *PackagingType* der generierten Verpackung fest.

Transfer.IntoStore	Verschiebt die erzeugten Verpackungen an das InputNode des Lagers.

Tabelle 45: Warehouse Process: OnRunInitialized

Der Order_Processing Prozess enhält die Logik für das Erstellen einer neuen *Order*, sowie Konfigurieren und Aussenden dieser. Dieser Ablauf war in Prototyp1 teil des Management Prozesses.

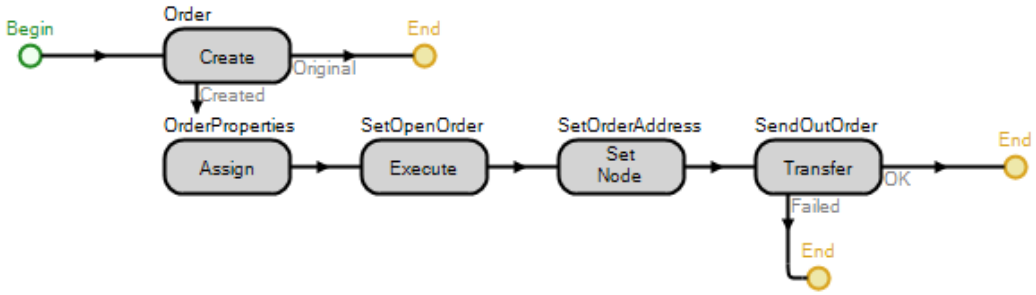
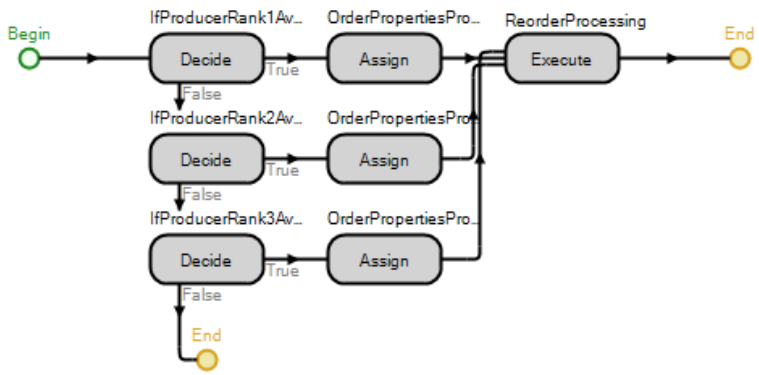
Order_Processing (Prototyp2-3)	
	
Create.Order	Erstellt eine *Order Entity*.
Assign.OrderProperties	Setzt die Parameter für die ausgehende *Order*, welche im vorherigen Schritt erstellt wurde. Gesetzt werden Bestelladdresse, Lieferaddresse, Liefermenge, Verpackungstyp, Bestelltyp, Rücksendeadresse und welches Produkt bestellt wird.
Execute.SetOpenOrder	Ruft den *Set_OpenOrder* Prozess auf.
SetNode.SetOrderAddress	Setzt den nächsten Hop nach dem Verlassen des *Warehouses*.
Transfer.SendOutOrder	Schiebt die *OrderEntity* zum ExitNode des *Warehouses*.

Tabelle 46: Warehouse Process: Order_Processing

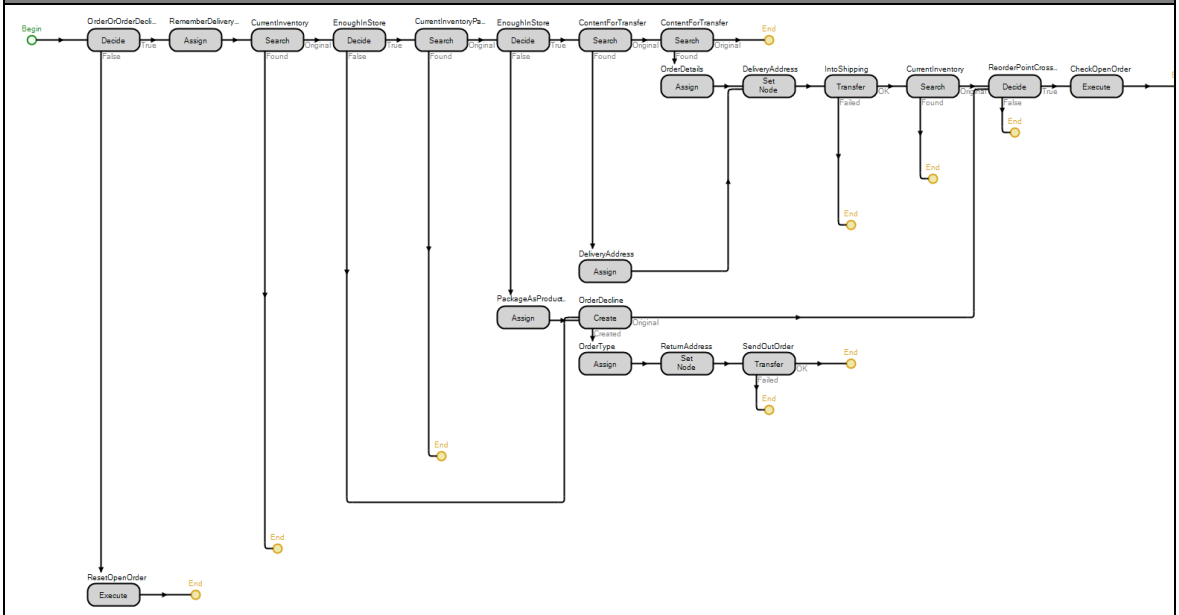
Für jedes Produkt gibt es einen *Reorder* Prozess welcher den *Producer*, bei dem bestellt wird, anhand einer Rangliste und Verfügbarkeit festlegt.

Reorder_Product1-10 (Prototyp2-3)	
	
Decide.IfProducerRank1-3Available	Überprüft die *Availability* des jeweiligen *Producers*.

Assign.OrderPropertiesProducerRank1-3	Schreibt die Adresse des ausgewählten *Producers* in den dazugehörigen Zwischenspeicher.
Execute.ReorderProcessing	Ruft den Prozess *Order_Processing* auf.

Tabelle 47: Warehouse Process: Reorder_Product1-10

Der *Management_Processing* Prozess auf Prototyp1 ist komplett überarbeitet, sowie in verschiedene Prozesse aufgeteilt worden. Er bearbeitet ankommende Bestellungen, überprüft die Lagerbestände, verschickt bestellte Waren und löst Nachbestellungen aus.

Management_Processing (Prototyp2)	
	
Decide.OrderOrOrderDecline	Entscheidet, ob die eingehende *OrderEntity*, eine Bestellung oder eine *OrderDecline* Nachricht ist.
Execute.ResetOpenOrder	Ruft den *Reset_OpenOrder* Prozess auf.
Assign.RememberDeliveryAddress	Schreibt die Lieferaddresse, die Liefermenge, die Rücksendeadresse, den bestellten Produkttypen und die gewünschte Verpackungsart in die dafür vorgesehenen Zwischenspeicher.
Search.CurrentInventory	Überprüft den Lagerbestand des bestellten Produktes und schreibt diesen in einen Zwischenspeicher.
Decide.EnoughInStore	Vergleicht den Lagerbestand mit der für die Bestellung benötigte Menge und entscheidet, ob genug Produkte an Lager sind, um die Bestellung auszuführen.
Search.CurrentInventoryPackaging	Überprüft den Lagerbestand der gewünschten Verpackung für das bestellte Produkt und schreibt diesen in einen Zwischenspeicher.

Assign.PackageAsProduct	Schreibt die Verpackung in den Zwischenspeicher für das nachzubestellende Produkt. Wird verwendet für das Nachbestellen von Verpackungen.
Search.ContentForTransfer	Sucht die Produkte für die Bestellung im Lager und markiert diese für den Transport aus dem Lager.
Assign.DeliveryAddress	Schreibt die Lieferadresse und den Verpackungstypen aus dem Zwischenspeicher auf die im vorherigen Schritt gefundenen Produkte.
Create.OrderDecline	Erzeugt eine *OrderEntity*.
Assign.OrderType	Setzt den OrderTyp auf *OrderDecline* und schreibt die Rücksendeadresse aus dem Zwischenspeicher in die Rücksendeadresse der *Order*
Assign.OrderDetails	Setzt folgende Einstellungen auf der Verpackung: Lieferadresse, Liefermenge und Verpackungstyp.
SetNode.ReturnAddress	Setzt die Rücksendeadresse als nächsten Hop.
SetNode.DeliveryAddress	Setzt die Lieferadresse als nächsten Hop.
Transfer.SendOutOrder	Verschickt die *OrderDecline* Nachricht.
Transfer.IntoShipping	Verschiebt die Produkte und Verpackung in den Output Buffer des Stores.
Decide.ReorderPointCrossed	Überprüft, ob der Lagerbestand unter dem *Reorderpoint* liegt.
Execute.CheckOpenOrder	Ruf den Prozess *Check_OpenOrder* auf.

Tabelle 48: Warehouse Process: Management_Processing

20.3.7.3 Prototyp3

Nachfolgender Prozess überprüft, welches Produkt im Lager eingeht, um das entsprechende *OpenOrder* zurückzusetzen.

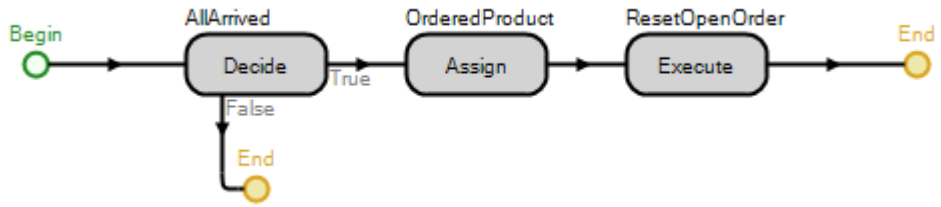
OrderReceived_Product1-10 (Prototyp3)	
	
Decide.AllArrived	Überprüft, ob alle Produkte der Bestellung im Lager eingegangen sind.
Assign.OrderedProduct	Schreibt das eingegangene Produkt in den Zwischenspeicher des bestellten Produktes.
Execute.ResetOpenOrder	Ruft den Prozess Reset_OpenOrder auf.

Tabelle 49: Warehouse Process: OrderReceived_Product1-10

Dieser Prozess weist den Produkten ihren Preis zu.

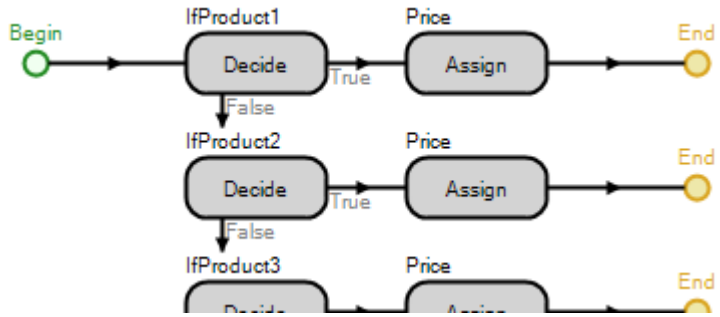
Set_ProductPrice (Prototyp3)	
	
Decide.IfProduct1	Filtert nach Produkttypen.
Assign.Price	Setzt den Preis basierend auf dem Produkttypen.

Tabelle 50: Warehouse Process: Set_ProductPrice

Dieser Prozess dient der Kostenberechnung für ankommende Lieferungen.

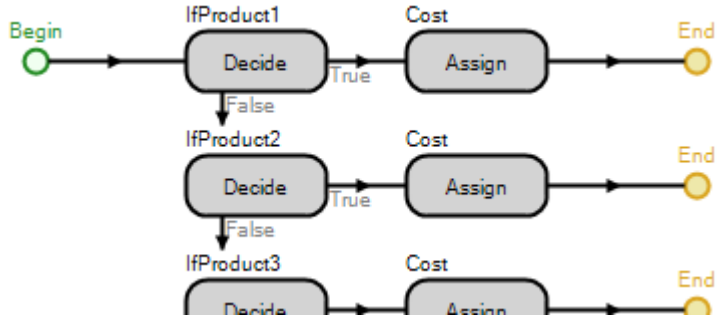
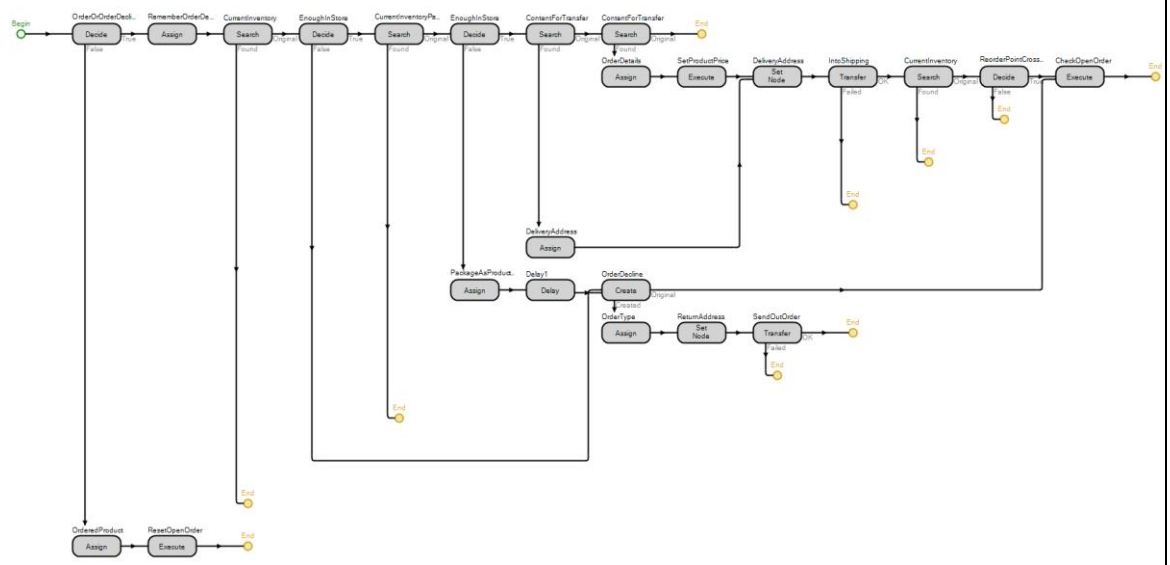
Input_ReceivingArea_ReceivingArea1_Entered (Prototyp3)	
	
Decide.IfProduct1	Filtert nach Produkttypen.
Assign.Cost	Berechnet die Kosten.

Tabelle 51: Warehouse Process: Input_ReceivingArea1_Entered

Der Management Prozess wird im Prototyp3 minimal angepasst. Die neu hinzugekommenen Schritte sind am Ende der Tabelle aufgeführt.

Management_Processing (Prototyp3)	
	
Decide.OrderOrOrderDecline	Entscheidet, ob die eingehende *OrderEntity*, eine Bestellung oder eine *OrderDecline* Nachricht ist.
Execute.ResetOpenOrder	Ruft den *Reset_OpenOrder* Prozess auf.
Assign.RememberDeliveryAddress	Schreibt die Lieferaddresse, die Liefermenge, die Rücksendeadresse, den bestellten Produkttypen und die gewünschte Verpackungsart in die dafür vorgesehenen Zwischenspeicher.

Search.CurrentInventory	Überprüft den Lagerbestand des bestellten Produktes und schreibt diesen in einen Zwischenspeicher.
Decide.EnoughInStore	Vergleicht den Lagerbestand mit der für die Bestellung benötigte Menge und entscheidet, ob genug Produkte an Lager sind, um die Bestellung auszuführen.
Search.CurrentInventoryPackaging	Überprüft den Lagerbestand der gewünschten Verpackung für das bestellte Produkt und schreibt diesen in einen Zwischenspeicher.
Assign.PackageAsProduct	Schreibt die Verpackung in den Zwischenspeicher für das nachzubestellende Produkt. Wird verwendet für das Nachbestellen von Verpackungen.
Search.ContentForTransfer	Sucht die Produkte für die Bestellung im Lager und markiert diese für den Transport aus dem Lager.
Assign.DeliveryAddress	Schreibt die Lieferadresse und den Verpackungstypen aus dem Zwischenspeicher auf die im vorherigen Schritt gefundenen Produkte.
Create.OrderDecline	Erzeugt eine *OrderEntity*.
Assign.OrderType	Setzt den OrderTyp auf *OrderDecline* und schreibt die Rücksendeadresse aus dem Zwischenspeicher in die Rücksendeadresse der *Order*
Assign.OrderDetails	Setzt folgende Einstellungen auf der Verpackung: Lieferadresse, Liefermenge und Verpackungstyp.
SetNode.ReturnAddress	Setzt die Rücksendeadresse als nächsten Hop.
SetNode.DeliveryAddress	Setzt die Lieferadresse als nächsten Hop.
Transfer.SendOutOrder	Verschickt die *OrderDecline* Nachricht.
Transfer.IntoShipping	Verschiebt die Produkte und Verpackung in den Output Buffer des Stores.
Decide.ReorderPointCrossed	Überprüft, ob der Lagerbestand unter dem Reorderpoint liegt.
Execute.CheckOpenOrder	Ruf den Prozess *Check_OpenOrder* auf.
Execute.ResetOpenOrder	Ruft den Prozess *Reset_OpenOrder* auf.
Delay.Delay1	Fügt eine Verzögerung von einer Sekunde ein, damit der Prozess nicht zweimal gleichzeitig beim gleichen Schritt ankommt und sich selbst beendet.

Tabelle 52: Warehouse Process: Management_Processing (Prototyp3)

Nachfolgender Prozess ist für die Kostenverrechnung von Sendungen, welche das Lagerhaus verlassen, verantwortlich.

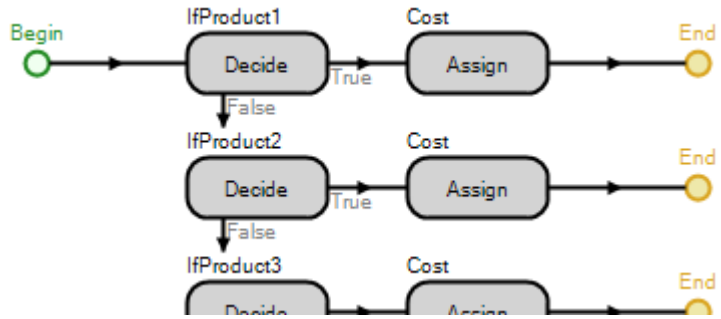
OutputShipping_Exited (Prototyp3)	
	
Decide.IfProduct1	Filtert nach Produkttypen.
Assign.Cost	Berechnet die Kosten.

Tabelle 53: Warehouse Process: OutputShipping_Exited

20.4 RECEIVINGAREA

20.4.1 AUFBAU

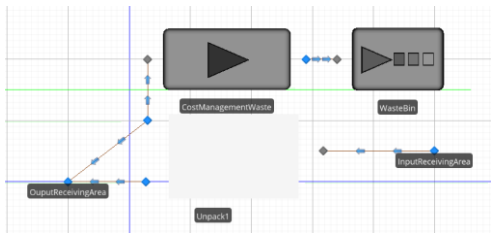
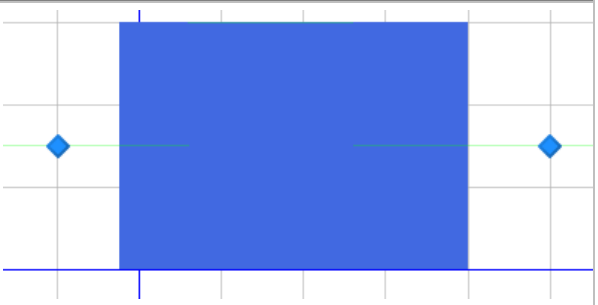
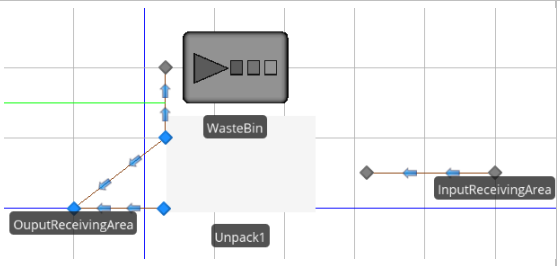
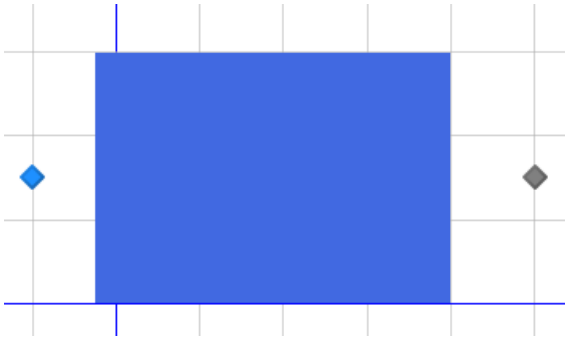
Prototyp1-2	
Interner Aufbau	**Externe Modellansicht**
	
Prototyp3	
Interner Aufbau	Externe modellansicht
	
Die *ReceivingArea* nimmt Pakete entgegen und reicht diese an die *Unpack* Station weiter. Dort werden Pakete ausgepackt und Verpackungen getrennt. *Carton* wird über eine Sink zerstört und Container werden zum Ausgang der *ReceivingArea* weitergeleitet.	

Tabelle 54: ReceivingArea Aufbau

20.4.2 RECEIVINGAREA PROPERTIES

Name	Grundfunktionalität: Prototyp1	Anpassungen
Packaging-WasteCost *(Expression)*	Definiert, wie teuer das Entsorgen einer einzelnen Verpackung ist. Bei Prototyp1 betrifft dies die Entsorgungskosten für *Carton*.	
CostCenter-Packaging-Waste *(Cost Center)*	Definiert, auf welchem CostCenter die Verpackungsentsorgungskosten gebucht werden.	

Tabelle 55: ReceivingArea Properties

20.4.3 SCHNITTSTELLEN

Die *ReceivingArea* hat zwei Schnittstellen. Einen Ein- und einen Ausgang. Beide sind Teil des Warenflusses.

Wareneingang An dieser Schnittstelle werden verpackte Produkte entgegengenommen.

Warenausgang An dieser Schnittstelle verlassen Produkte und wiederverwendbare Verpackungen die *ReceivingArea*.

20.4.4 PROZESSE

20.4.4.1 Prototyp1

Nachfolgender Prozess dient der Kostenverwaltung von Entsorgungskosten, welche durch das Zerstören von Verpackungen entstehen.

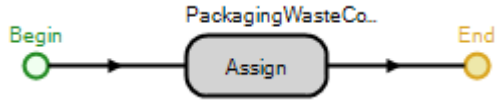
WasteBin_Entered (Prototyp1-3)	
	
Assign.Packaging-WasteCost	Erhöht die Kosten, wenn eine Verpackung entsorgt wird.

Tabelle 56: ReceivingArea Process: WasteBin_Entered

20.5 UNPACK

20.5.1 AUFBAU

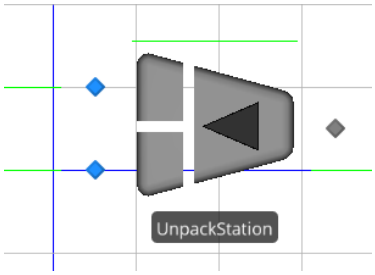
Prototyp1-3	
Interner Aufbau	**Externe Modellansicht**
	
Die *Unpack* Station nimmt in Verpackung eingepackte Produkte entgegen und trennt die Verpackung vom Inhalt und gibt diese über zwei verschiedene Ausgänge weiter.	

Tabelle 57: Unpack Aufbau

20.5.2 SCHNITTSTELLEN

Das *Unpack* Station hat drei Schnittstellen. Einen Ein- und zwei Ausgänge. Alle sind Teil des Warenflusses.

Wareneingang Auf dieser Schnittstelle werden verpackte Produkte entgegengenommen.

Warenausgang Verpackungen Auf dieser Schnittstelle verlassen Verpackungen die *Unpack* Station.

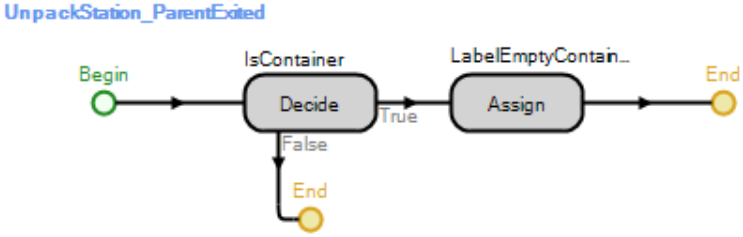
Warenausgang Produkte Auf dieser Schnittstelle verlassen Produkte die *Unpack* Station.

20.5.3 PROZESSE

20.5.3.1 Prototyp1

Im Prototyp1 gibt es auf dem Container ein Label, welches über folgenden Prozess verwaltet wird.

UnpackStation_ParentExited (Prototyp1)



Decide.IsContainer	Filtert die *Container* aus den Verpackungstypen.
Assign.LabelEmptyContainer	Setzt das *Container Label* zurück auf 0.

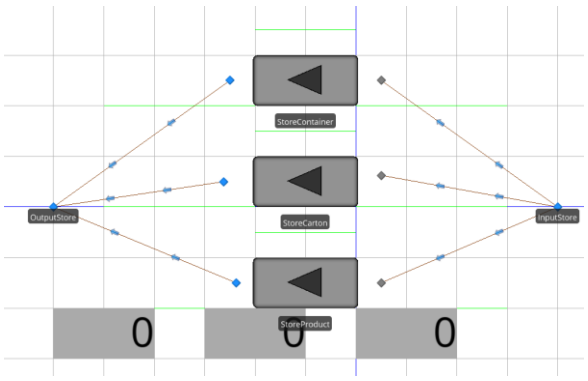
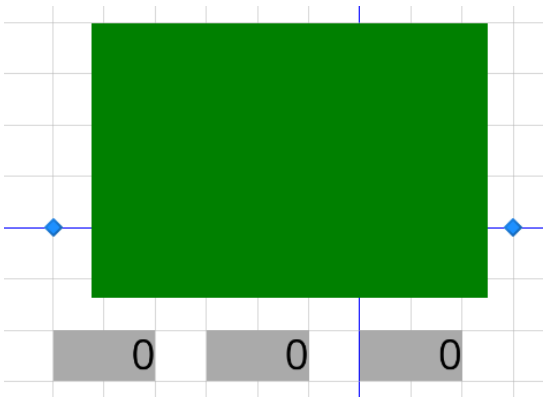
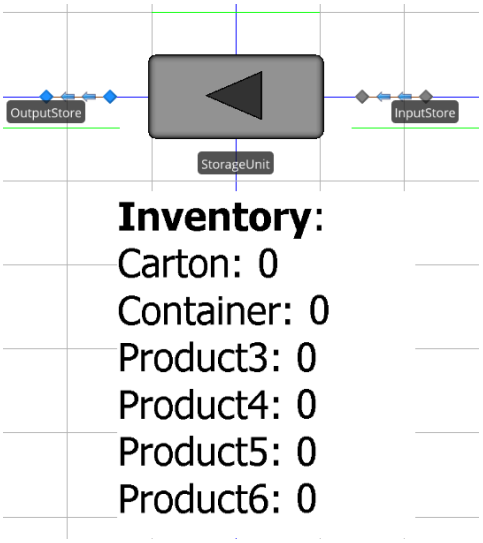
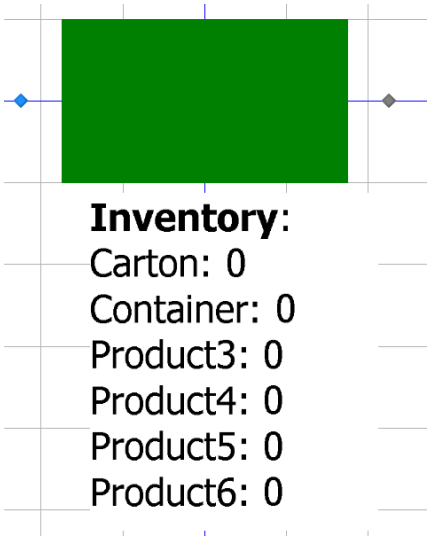
Tabelle 58: Unpack Process: UnpackStation_ParentExited

20.5.3.2 Prototyp2

Im Prototyp2 hat die *Unpack* Station keine Prozesse, das Label am *Container* wurde entfernt.

20.6 StoreWarehouse (Store)

20.6.1 Aufbau

Prototyp1-2	
Interner Aufbau	**Externe Modellansicht**
	
Prototyp3	
Interner Aufbau	**Externe Modelansicht**
	

Prototyp1, Prototyp2:
Der Store hat einen Eingang und einen Ausgang. Die ankommenden Produkte werden aufgeteilt nach *Carton*, *Container* oder *Nonpackaging* (normale Produkte). Anschliessend werden sie getrennt in Servern gelagert. Die drei Label darunter zeigen den Inhalt der jeweiligen Stores an und dienen vor allem der Überprüfung während der Entwicklung von Prototyp1.

Prototyp3:
Ankommende Produkte (unabhängig der Produktart) werden in einem Server gelagert, bis sie von einem Prozess wieder daraus heraus bewegt werden. Das Lager führt ein Inventar darüber, wie viele Produkte von welchem Typen vorhanden sind.

Tabelle 59: StoreWarehouse Aufbau

20.6.2 StoreWarehouse (Store) Properties

Name	Grundfunktionalität: Prototyp1	Anpassungen
StorageCost-PerProduct *(Expression)*	Legt fest, wie teuer das Lagern eines Produktes pro Stunde ist.	
CostCenter-Storage *(Cost Center)*	Legt fest, auf welches CostCenter Lagerkosten gebucht werden.	

Tabelle 60: StoreWarehouse Properties

20.6.3 StoreWarehouse States

Name	Grundfunktionalität: Prototyp1	Anpassungen
ProductInStore *(Integer)*	Wie viele Produkte sich im Lager befinden.	**Prototyp2**: Obsolet und gelöscht.
Name	**Zusätzliche States: Prototyp2**	**Anpassungen**
Inventory-Product1-10 *(Integer)*	Zehn States für die Verwaltung des Lagerbestands.	

Tabelle 61: StoreWarehouse States

20.6.4 StoreWarehouse Events

Name	Grundfunktionalität: Prototyp1	Anpassungen
OrderReceived	Wird ausgelöst, wenn Produkte im Lager ankommen.	**Prototyp3**: Obsolet und gelöscht.
Packaging-Received	Wird ausgelöst, wenn *Carton* im Lager ankommt.	**Prototyp2**: Obsolet und gelöscht.
Name	**Zusätzliche Events: Prototyp3**	**Anpassungen**
OrderReceived-Product1-10	Wird ausgelöst, wenn das betreffende Produkt im Lager ankommt.	

Tabelle 62: StoreWarehouse Events

20.6.5 Schnittstellen

Die *StoreWarehouse* Station verfügt über zwei Schnittstellen. Einen Ein- und einen Ausgang. Beide sind Teil des Warenflusses.

Wareneingang An dieser Schnittstelle werden Produkte und Verpackungen entgegengenommen.

Warenausgang Über diese Schnittstelle verlassen Produkte und Verpackungen den Store.

20.6.6 PROZESSE

20.6.6.1 Prototyp1

Nachfolgender Prozess dient der Verwaltung für den Lagerbestand von *Carton*.

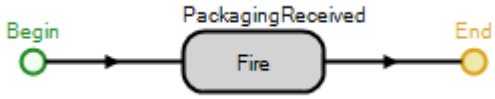
StoreCarton_Entered (Prototyp1)	
	
Fire.PackagingReceived	Löst das Event *PackagingReceived* aus.

Tabelle 63: StoreWarehouse Process: StoreCarton_Entered

Dieser Prozess dient der Verwaltung für den Lagerbestand von Produkten.

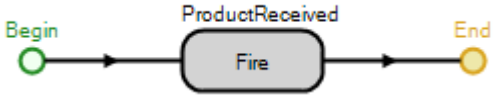
StoreProduct_Entered (Prototyp1)	
	
Fire.ProductReceived	Löst das Event *OrderReceived* aus.

Tabelle 64: StoreWarehouse Process: StoreProduct_Entered

20.6.6.2 Prototyp2

Dieser Prozess dient der allgemeinen Lagerverwaltung. Der Prozess ist für 10 ausgelegt, sofern er nicht erweitert wird.

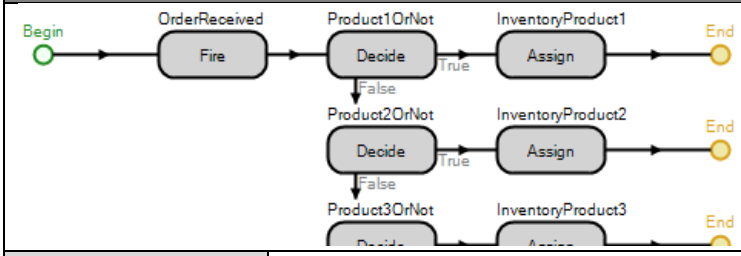
StorageUnit_Entered (Prototyp2)	
	
Fire.OrderReceived	Löst das Event *OrderReceived* aus.
Decide.Product1-10OrNot	Entscheidet, um welches Produkt es sich handelt.
Assign.InventoryProduct1-10	Zählt das Inventar für das jeweilige Produkt hoch.

Tabelle 65: StoreWarehouse Process: StorageUnit_Entered

Nachfolgender Prozess ist das Gegenstück zum *StorageUnit_Entered* Prozess. Er wird für einen abnehmenden Lagerbestand verwendet.

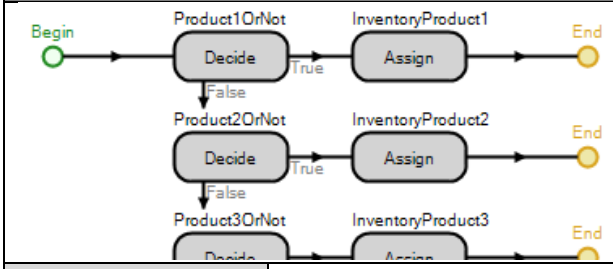
StorageUnit_Exited (Prototyp2-3)	
	
Decide.Product1-10OrNot	Entscheidet, um welches Produkt es sich handelt.
Assign.InventoryProduct1	Zählt das Inventar für das jeweilige Produkt runter.

Tabelle 66: StoreWarehouse Process: StorageUnit_Entered

20.6.6.3 Prototyp3

Für Prototyp3 ist der *StorageUnit_Entered* Prozess dahingehend angepasst worden, dass ein *OrderReceived* Event ausgelöst wird.

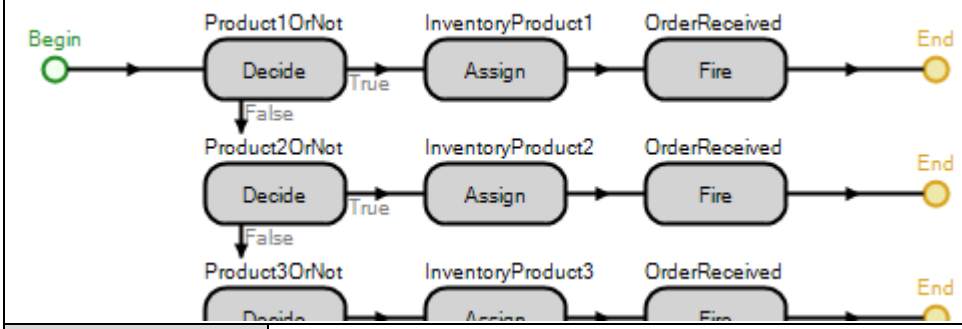
StorageUnit_Entered (Prototyp3)	
	
Fire.OrderReceived	Löst die Events *OrderReceivedProduct1-10* aus.
Decide.Product1-10OrNot	Entscheidet, um welches Produkt es sich handelt.
Assign.InventoryProduct1-10	Zählt das Inventar für das jeweilige Produkt hoch.

Tabelle 67: StoreWarehouse Process: StorageUnit_Entered (Prototyp3)

20.7 SHIPPINGAREA

20.7.1 AUFBAU

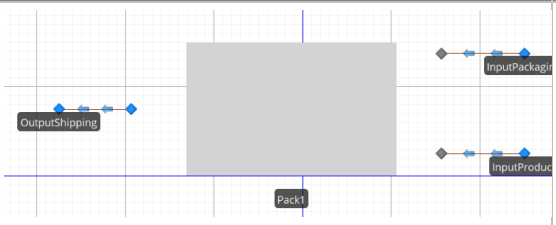
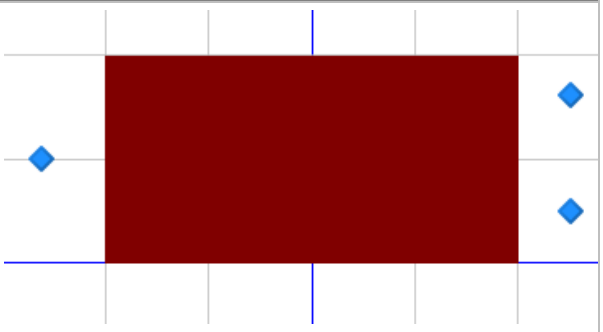
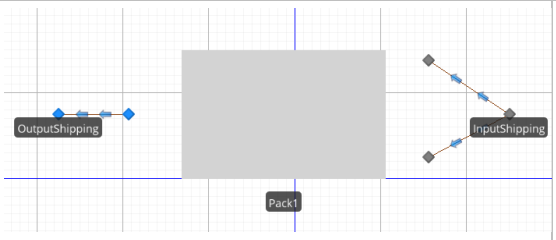
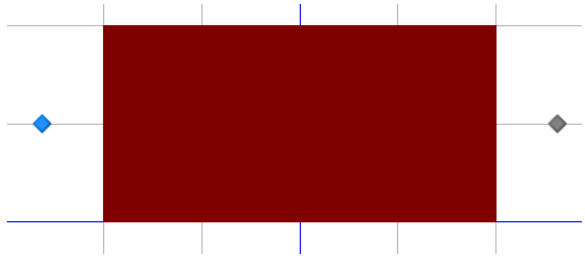
Prototyp1	
Interner Aufbau	**Externe Modellansicht**
	
Die *ShippingArea* nimmt Verpackungen und Produkte entgegen und gibt diese an die *Pack* Station zum Einpacken weiter. Sie nimmt die verpackten Produkte dann wieder zurück und sendet diese an andere *Warehouses* oder *Customer* weiter.	
Prototyp2	
Interner Aufbau	**Externe Modellansicht**
	
Die Funktionalität hat sich von Prototyp1 zu Prototyp2 nicht verändert. Angepasst wurde, wie die Verpackungen und Produkte entgegengenommen und zur *Pack* Station geleitet werden. Anstatt über zwei verschiedene InputNodes, wird ab Prototyp2 nur eine verwendet und stationsintern aufgelöst, um welche Art Produkt es sich handelt.	

Tabelle 68: ShippingArea Aufbau

20.7.2 SHIPPINGAREA PROPERTIES

Name	Grundfunktionalität: Prototyp1	Anpassungen
CostCenter-Packaging *(Cost Center)*	Definiert das CostCenter, auf welchem die Verpackungskosten verbucht werden.	**Prototyp3**: Umbenennen und Umfunktionierung von CO_2 auf allgemeine *PackagingCost*.
PackagingCost *(Expression)*	Definiert die Verpackungskosten in Bezug auf CO_2. **Prototyp3**: Unterscheidung nach CO_2 wurde entfernt.	**Prototyp3**: Umbenennen und Umfunktionierung von CO_2 auf allgemeine *PackagingCost*.

Tabelle 69: ShippingArea Properties

20.7.3 SHIPPINGAREA STATES

Name	Grundfunktionalität: Prototyp1	Anpassungen
NumberToShip *(Integer)*	Definiert, wie viele Produkte in eine Verpackung verpackt werden.	

Tabelle 70: ShippingArea States

20.7.4 SCHNITTSTELLEN

20.7.4.1 Prototyp1

Die *ShippingArea* verfügt über drei Schnittstellen. Zwei Eingänge und einen Ausgang. Alle sind Bestandteil des Wareflusses.

Wareneingang Verpackung An dieser Schnittstelle werden Verpackungen entgegengenommen.

Wareneingang Produkte An dieser Schnittstelle werden Produkte entgegengenommen.

Warenausgang Über diese Schnittstelle verlassen verpackte Produkte die *ShippingArea*.

20.7.4.2 Prototyp2, Prototyp3

Im Prototyp2 ist angepasst worden, wie Waren und Verpackungen entgegengenommen werden. Es gibt nur noch zwei Schnittstellen: Einen Eingang und einen Ausgang.

Wareneingang Auf dieser Schnittstelle werden Produkte und Verpackungen entgegengenommen.

Warenausgang Auf dieser Schnittstelle verlassen verpackte Produkte die *ShippingArea*.

20.7.5 PROZESSE

Die *ShippingArea* hat keine stationspezifischen Prozesse.

20.8 PACK

20.8.1 AUFBAU

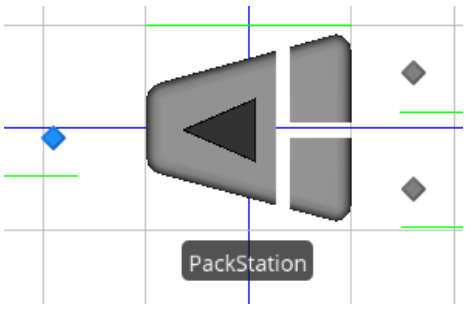
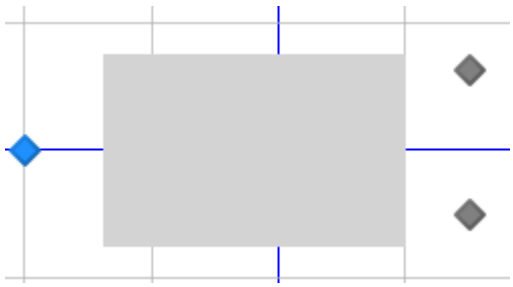
Prototyp1-3	
Interner Aufbau	**Externe Modellansicht**
	
Die *Pack* Station nimmt Verpackungen und Produkte entgegen und verpackt die Produkte in die Verpackungen. Die fertigen Pakete werden dann an die drüberliegende Station zurückgegeben.	

Tabelle 71: Pack Aufbau

20.8.2 PACK PROPERTIES

Name	Grundfunktionalität: Prototyp1	Anpassungen
CostCenter-Packaging *(Cost Center)*	Definiert das CostCenter, auf welchem die Verpackungskosten verbucht werden.	**Prototyp3**: Umbenennen und Umfunktionierung von CO_2 auf allgemeine *PackagingCost*.
PackagingCost *(Expression)*	Definiert die Verpackungskosten in Bezug auf die CO_2 Abgaben.	**Prototyp3**: Umbenennen und Umfunktionierung von CO_2 auf allgemeine *PackagingCost*.

Tabelle 72: Pack Properties

20.8.3 PACK STATES

Name	Grundfunktionalität: Prototyp1	Anpassungen
OrderSize *(Integer)*	Definiert, wie viele Produkte in eine Verpackung verpackt werden.	

Tabelle 73: Pack States

20.8.4 SCHNITTSTELLEN

Die *Pack* Station verfügt über drei Schnittstellen. Zwei Eingänge und einen Ausgang. Alle sind Bestandteil des Wareflusses.

Wareneingang Verpackung An dieser Schnittstelle werden Verpackungen entgegengenommen.

Wareneingang Produkte An dieser Schnittstelle werden Produkte entgegengenommen.

Warenausgang Über diese Schnittstelle verlassen verpackte Produkte die *Pack* Station.

20.8.5 PROZESSE

20.8.6 PROTOTYP1

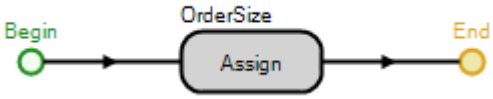
ParentInput_PackStation_Entered (Prototyp1-3)	
	
Assign.OrderSize	Legt anhand der Bestellgrösse fest, wie viele Produkte in die Verpackung eingepackt werden.

Tabelle 74: Pack Process: ParentInput_PackStation_Entered

20.9 CUSTOMER

Prototyp1:

Die Warenannahme hat sich von Prototyp0 dahingehend verändert, dass ankommende Pakete via einer *Unpack* Station ausgepackt und deren Inhalt an getrennte Sinks weitergeleitet wird. Des Weiteren findet eine Trennung der ankommenden Informationen (*OrderDecline* Messages) und der bestellten Produkte statt. Die Trennung erfolgt, um eine saubere Zählung der Kosten zu ermöglichen.

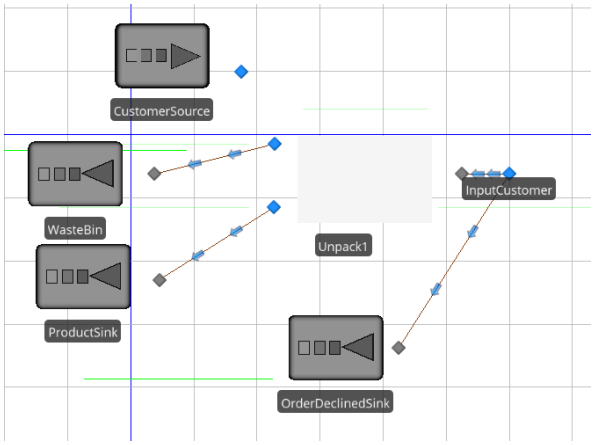
20.9.1 ABLÄUFE

Beim *Customer* finden folgende Abläufe statt:

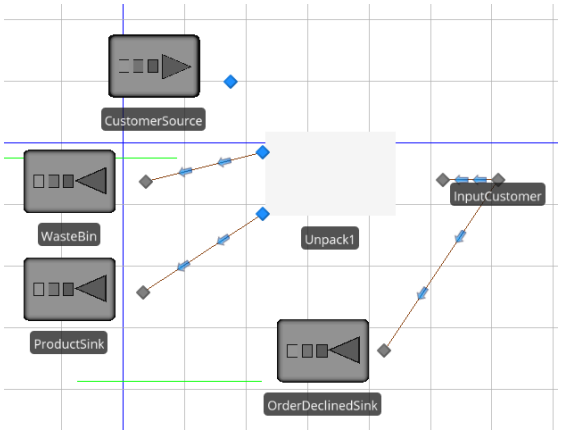
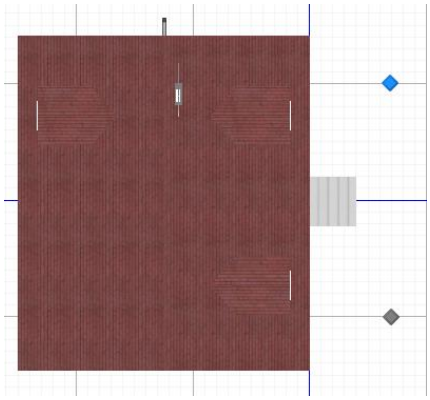
- Erstellen von OrderEntities
- Setzen der initialen Bestelladresse als nächste Station, wenn Bestellungen das System verlassen
- Entgegennehmen, Auspacken und Zerstören von empfangenen Bestellungen

20.9.2 AUFBAU

Prototyp1-2

Interner Aufbau	Externe Modellansicht
	

Prototyp3

Interner Aufbau	Externe Modelansicht
	

Der *Customer* besteht aus einer *CustomerSource*, welche neue *Orders* generiert und verschickt und einem eingehenden Warenfluss. Über das Node *InputCustomer* empfängt der *Customer* sowohl *OrderDecline* Nachrichten als auch bestellte Lieferungen. Die *OrderDecline* Nachrichten werden in eine Sink geleitet. Die empfangenen Produkte werden zuerst von einer *Unpack* Station ausgepackt; die Verpackungen dann an eine Sink für Verpackungen und die Produkte an eine Sink für Produkte weitergeleitet.

Tabelle 75: Customer Aufbau

20.9.3 CUSTOMER ELEMENTS

Name	Grundfunktionalität: Prototyp1	Anpassungen
CostCenter-Waste-Packaging *(Cost Center)*	CostCenter für die Verbuchung von Entsorgungskosten von Verpackungen.	
CostCenter-Purchase *(Cost Center)*	CostCenter für die Verbuchung von Ausgaben für den Einkauf von Produkten.	**Prototyp3**: Umbenennen von *CostCenterProductionIncome* zu *CostCenterPurchase*

Tabelle 76: Customer Elements

20.9.4 CUSTOMER PROPERTIES

Auf dem *Customer* kann eingestellt werden, wie viel und wie oft *Orders* verschickt werden und wo bestellt werden soll.

Name	Grundfunktionalität: Prototyp1	Anpassungen
OrderEntity-Type *(Entity)*	Legt den Typ der *OrderEntity* fest.	**Prototyp2**: Umbenennen von *Order_EntityType* zu *OrderEntityType*
Order_InterarrivalTime *(Expression)*	Legt fest, wie viele OrderEntities erstellt werden.	**Prototyp2**: Umbenennen von Order_ *InterarrivalTime* zu *OrderInterarrivalTime*
Order_MaximumArrivals *(Expression)*	Definiert eine Obergrenze, wie viele Bestellungen für den Durchlauf einer Simulation erstellt werden können.	**Prototyp2**: Umbenennen von *Order_MaximumArrivals* zu *OrderMaximumArrivals*
Customer-Address *(Expression)*	Definiert die Adresse, an welcher der Kunde die Waren entgegennimmt. Normalerweise ist das der eigene Wareneingang, es kann aber auch der Wareneingang eines anderen *Customers* angegeben werden.	
OrderAddress *(Node)*	Definiert die Adresse, bei der bestellt wird.	
ProductPrice *(Integer)*	Definiert, wie viel der *Customer* pro Produkt, das bei ihm ankommt, bezahlt.	**Prototyp3**: Obsolet und gelöscht.

Packaging-WasteCost *(Expression)*	Definiert, wie teuer die Entsorgung einer Verpackung ist.	

Tabelle 77: Customer Properties

20.9.5 SCHNITTSTELLEN

Der *Customer* verfügt über zwei Schnittstellen, welche als Verbindungspunkte zur restlichen Supply Chain verwendet werden.

Bestellausgang Über diesen Knoten verlassen die *OrderEntities* den Kunden.

Wareneingang Über diesen Knoten werden Produkte und *OrderDecline* Nachrichten empfangen.

20.9.6 PROZESSE

Der *Customer* hat drei Prozesse, zwei für das Costmanagement und einen, um auf den neu generierten *Orders* Angaben über die dazugehörige Bestellung zu machen.

20.9.6.1 Prototyp1

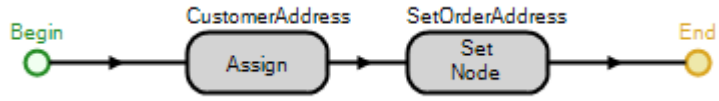
Output_Source1_Entered (Prototyp1-3)	
	
Assign.CustomerAddress	Füllt die Bestelladresse und die Returnadresse auf der *OrderEntity* aus.
SetNode.SetOrderAddress	Übergibt den erzeugten *OrderEntities* die initiale Bestelladresse.

Tabelle 78: Customer Process: Output_Source1_Entered

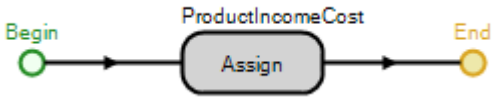
ProductSink_Entered (Prototyp1-3)	
	
Assign.ProductIncomeCost	Verbucht die Kosten, welche auf den empfangenen Produkten steht, auf dem CostCenterPurchase .

Tabelle 79: Customer Process: ProductSink_Entered

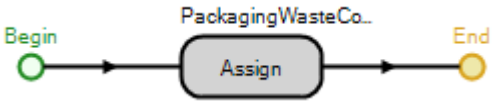
WasteBin_Entered (Prototyp1-3)	
	
Assign.Packaging-WasteCost	Verbucht die *PackagingWasteCost* auf dem *PackagingWasteCostcenter*.

Tabelle 80: Customer Process: WasteBin_Entered

20.10 PRODUCER

Prototyp1:

Der *Producer* ist im Prototyp1 noch sehr ähnlich zu dem im Prototyp0. Er erstellt *ProductEntities*, verpackt diese in *Container* und verschickt diese. Die Produkte werden erst erstellt, wenn eine *OrderEntity* mit einer Bestellung ankommt. Diese löst dann das Erstellen der bestellten Menge von Produkten aus. Der *Producer* entspricht dem äussersten Lieferanten in der abgebildeten Supply Chain, welcher für die abgebildete Kette keine Bestellungen absetzt, sondern nur Produkte liefert.

Die Änderung zu Prototyp0 ist das Hinzufügen der *Pack* Station, um Produkte in *Container* zu verpacken.

Prototyp2:

Im Prototyp2 wurde am *Producer* nur eine Änderung vorgenommen: Das Hinzufügen eines *Availability* Status.

20.10.1 ABLÄUFE

Beim Hersteller finden die folgenden Abläufe statt:

- Entgegennehmen und Zerstören von *OrderEntities*
- Erstellen von *Container*- und *ProductEntities*, wobei die *ProductEntities* in der von einer *Order* vorgegebenen Menge hergestellt werden
- Verpacken von *ProductEntities* in *ContainerEntities*
- Verschicken von mit Produkten beladenen *Containern* an die Lieferadresse, welche bei der *Order* angegeben ist.

Damit die *Container* am Schluss an die Lieferadresse geschickt werden können, muss die Lieferadresse beim Erhalten der Bestellung zwischengespeichert werden. Das gleiche passiert mit der Anzahl der bestellten Produkte, beide Angaben werden zwischengespeichert und dann auf den *Container* geschrieben.

20.10.2 AUFBAU

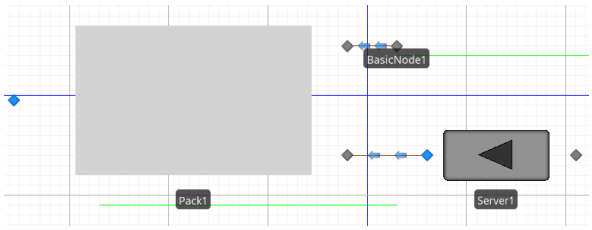
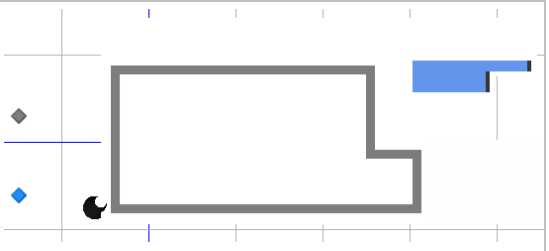
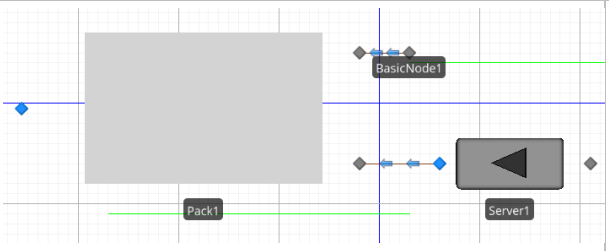
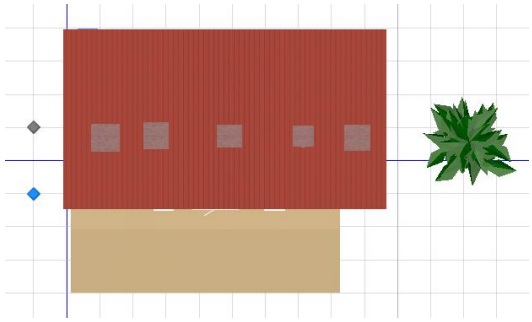
Prototyp1-2	
Interner Aufbau	Externe Modellansicht
	
Prototyp3	
Interner Aufbau	**Externe Modellansicht**
	
Der *Producer* besteht aus einem Server, welcher die *ProductEntities* erstellt und einer Packstation, welcher die *ProductEntities* in Container verpackt. Kommt eine *Order* an, wird ein Container erstellt, mit der Lieferadresse und der Anzahl bestellter Produkte beschrieben und an die Packstation weitergegeben. Gleichzeitig wird beim Server der Auftrag erteilt, die bestellte Anzahl Produkte herzustellen und an an die Packstation zu schicken. Bei der Packstation werden die Container mit der benötigten Anzahl *ProductEntities* beladen und an die Lieferadresse geschickt.	

Tabelle 81: Producer Aufbau

20.10.3 PRODUCER ELEMENTS

Name	Grundfunktionalität: Prototyp3	Anpassungen
CostCenter-Production *(Cost Center)*	CostCenter für die Produktionskosten.	
CostCenter-Packaging *(Cost Center)*	CostCenter für die Verpackungskosten.	
CostCenterTotal Investment *(Cost Center)*	CostCenter über alle Ausgaben.	

CostCenter-ProductIncome *(Cost Center)*	CostCenter für das Einkommen durch verkaufte Produkte.	
CostCenter-ProductReturn *(Cost Center)*	CostCenter für das *Product* Return.	
CostCenter-Purchase *(Cost Center)*	CostCenter für die Ausgaben, welche für Rohmaterial getätigt werden.	

Tabelle 82: Producer Elements

20.10.4 PRODUCER PROPERTIES

Auf dem *Producer* wird eingestellt, welche *ProductEntities* und mit welcher Produktionszeit diese hergestellt werden. Bei der Herstellung handelt es sich um einen «Pull», der durch eine *OrderEntity* ausgelöst wird.

Hinzugekommen sind die nötigen Properties, um entstehende Kosten auf dem *Producer* einzutragen, sowie die Möglichkeit, die Verpackungsart für die versendeten Produkte auszuwählen.

Name	Grundfunktionalität: Prototyp1	Anpassungen
ProductEntityType *(Entity)*	Legt fest, welches *ProductEntity* der *Producer* herstellt.	
ProductionTime *(Expression)*	Legt fest, wie schnell *ProductEntities* hergestellt werden.	**Prototyp3**: Umbenennen auf *ProductionTime*.
PackagingEntityType *(Entity)*	Definiert den Verpackungstypen, welcher für die Verpackung von Sendungen generiert wird.	
ProductionCostPerProduct *(Expression)*	Gibt die Produktionskosten pro Produkt an.	
PackagingCost *(Expression)*	Definiert die Kosten für den Verpackungsvorgang.	**Prototyp3**: Entfernen von CO_2 aus dem Namen und der Funktionalität.
RawMaterialPurchaseCostPerProduct *(Integer)*	Kosten für die Herstellung eines Produktes, anhand der Ausgaben für die benötigten Rohmaterialien.	
Name	**Zusätzliche Properties: Prototyp2**	**Anpassungen**
InitialAvailability *(Boolean)*	Dient dem Einlesen des *Availability* Status des *Producers* bei Simulationsstart.	

Name	Zusätzliche Properties: Prototyp3	Anpassungen
SellPriceProduct *(Integer)*	Legt den Verkaufspreis pro Produkteinheit fest.	

Tabelle 83: Producer Properties

20.10.5 PRODUCER STATES

Name	Grundfunktionalität: Prototyp1	Anpassungen
Remember-DeliverAddress *(Node Reference)*	In diesem State wird die Lieferadresse zwischengespeichert, nachdem eine *Order* erhalten wurde. Nach dem Erstellen des Containers wird die Lieferadresse aus diesem State heraus auf den *Container* geschrieben.	
Remember-OrderSize *(Integer)*	Dient als Zwischenspeicher für die Bestellgrösse. Nach dem Eingang einer *Order* wird die Anzahl der bestellten Produkte hier zwischengespeichert. Nach dem Erstellen des Containers wird dieser Wert auf den Container geschrieben.	
ProductCost *(Integer)*	Gibt die Produktkosten für ein Produkt an. Anhand dieses Wertes werden die Einnahmen berechnet.	
Name	**Zusätzliche States: Prototyp2**	**Anpassungen**
Availability *(Boolean)*	State zum Abspeichern des aktuellen Availability-Status des *Producers*.	
Name	**Zusätzliche States: Prototyp3**	**Anpassungen**
Remember-ProductOrdered *(Integer)*	Zwischenspeicher für den Typen des bestellten Produktes	

Tabelle 84: Producer States

20.10.6 SCHNITTSTELLEN

Der Hersteller hat zwei Schnittstellen, welche für das Annehmen von Bestellungen und das Ausschicken von beladenen Containern verantwortlich sind. Dabei ist jede Schnittstelle nur für eine der beiden Funktionen nutzbar.

Bestelleingang *OrderEntities* werden hier entgegengenommen.

Warenausgang *Container* verlassen den *Producer* über diesen Knoten.

20.10.7 PROZESSE

20.10.7.1 Prototyp1

Schreibt die Lieferadresse auf die Container, welche im Combiner mit den bestellten *ProductEntities* beladen werden.

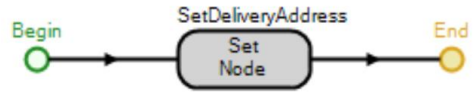
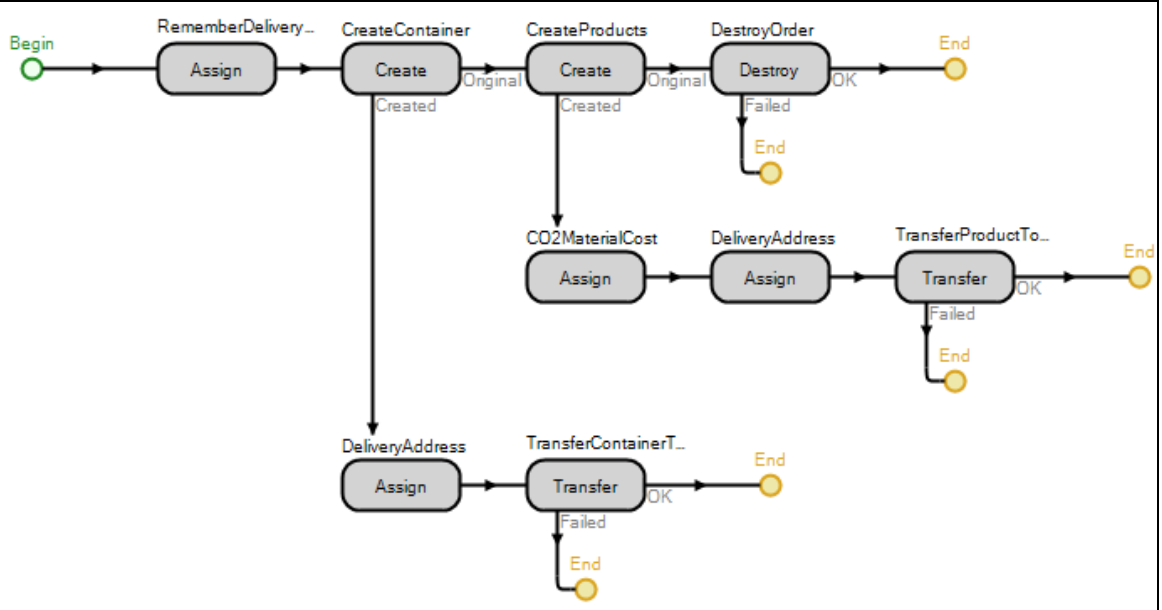
Output_Combiner1_Entered (Prototyp1-3)	
	
SetNode.Deliv-eryAddress	Schreibt die Lieferadresse auf die *Container*.

Tabelle 85: Producer Process: Output_Combiner1_Entered

Nachfolgend ist der Produktionsprozess aufgeführt, welcher Produkte und Verpackungen erstellt, konfiguriert und and den Combiner sendet.

ProductionProcess (Prototyp1-3)	
	
Assign.RememberDeliveryAddress	Dient dazu, die Lieferadresse und die Bestellmenge zwi-schenzuspeichern.
Create.Container	Erzeugt für jede Bestellung einen *Container*.
Create.Products	Erstellt die bestellte Menge an *ProductEntities*
Destroy.Order	Zerstört die zu bearbeitende *OrderEntity*.
Assign.CO2MaterialCost	Ordnet die *CO_2MaterialCost* dem *CO_2 Material* CostCenter zu.
Assign.DeliveryAddress	Schreibt die zuvor zwischengespeicherte Lieferadresse auf den Container und das Produkt.

Transfer.ProductToCombiner	Verschickt die erzeugten Produkte an den Combiner.
Transfer.ContainerToCombiner	Verschickt die erzeugten Container an den Combiner.

Tabelle 86: Producer Process: ProductionProcess

20.10.7.2 Prototyp2

Dieser Prozess wird beim Starten der Simulation ausgeführt und überträgt den Initialwert des Availability-Status.

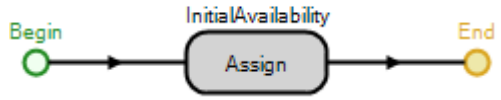
OnRunInitialized (Prototyp2-3)	
	
Assign.InitialAvailability	Schreibt den Wert der Properties *InititalAvailability* in den State Availability.

Tabelle 87: Producer Process: OnRunInitialized

20.10.7.3 Prototyp3

Berechnet *Product* Income and Return auf dem *Producer*.

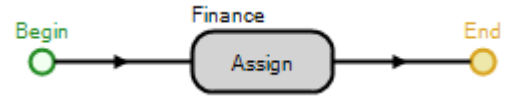
Output_Pack_Pack1_Exited (Prototyp3)	
	
Assign.Finance	Verbucht die Kosten auf dem *Income* und *Return* CostCenter.

Tabelle 88: Producer Process: Output_Pack_Pack1_Exited

20.11 PRODUCERADVANCED

Prototyp2:

Der *ProducerAdvanced* ist während der Arbeit zu Prototyp2 entwickelt und dem Modellbaukasten hinzugefügt worden. Die Hauptaufgabe des ProducerAdvanceds ist es, aus zwei Rohmaterialien ein Endprodukt herzustellen. Dazu gehört das Nachbestellen von Rohmaterialien und das Aussenden von Bestellungen.

20.11.1 ABLÄUFE

Beim *ProducerAdvanced* finden folgende Abläufe statt:

- Entgegennehmen und Erfüllen von Bestelllungen
- Produzieren eines Endproduktes aus zwei Rohmaterialien
- Nachbestellen von Rohmaterialien und Verpackungsmaterial

20.11.2 AUFBAU

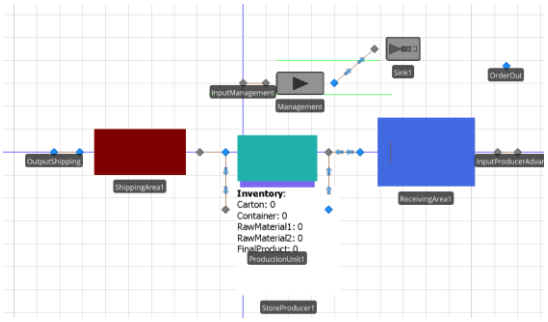
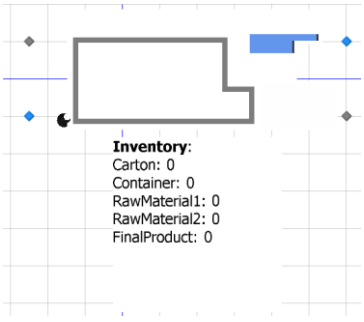
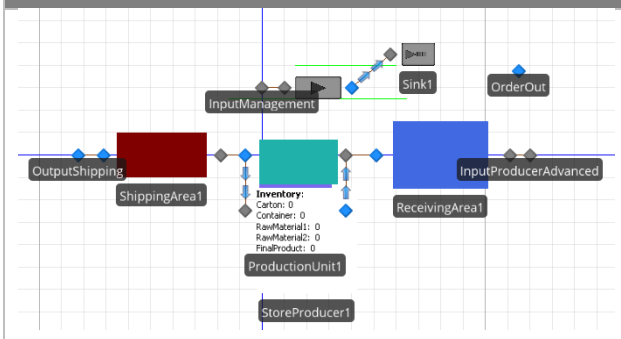
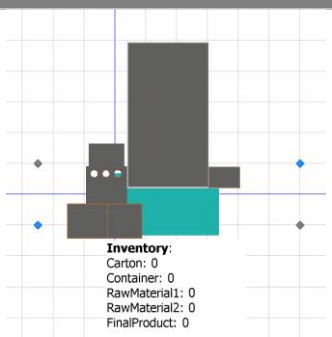
Prototyp2	
Interner Aufbau	**Externe Modellansicht**
	
Prototyp3	
Interner Aufbau	**Externe Modellansicht**
	
Der *ProducerAdvanced* besteht aus dem Management, einer *ReceivingArea*, einer *ShippingArea*, einer eigenen *Store*-Einheit sowie einer *ProductionUnit*. Ankommende Bestellungen werden vom Management entgegengenommen. Das Management löst dann entweder das Aussenden von Produkten oder die Produktion aus. Die *ReceivingArea* handelt unabhängig vom Management und nimmt erhaltene sendungen entgegen und gibt diese an die *StoreProducer* Station weiter.	

Tabelle 89: ProducerAdvanced Aufbau

20.11.3 PRODUCERADVANCED ELEMENTS

Name	Grundfunktionalität: Prototyp3	Anpassungen
CostCenter-FinalProduct-Return *(Cost Center)*	CostCenter für das *Product* Return über das herzustellende Produkt.	
CostCenter-TotalInvest-ment *(Cost Center)*	CostCenter über alle Ausgaben.	
CostCenter-FinalProduct-Income *(Cost Center)*	CostCenter über alle Einnahmen.	
CostCenter-RawMaterial-Pruchase *(Cost Center)*	CostCenter über alle Rohmaterial Einkäufe.	
CostCenter-Packaging *(Cost Center)*	CostCenter für die Verpackungskosten.	
CostCenter-FinalProduct-Production *(Cost Center)*	CostCenter für die Produktionskosten.	
CostCenter-Packaging-Waste *(Cost Center)*	CostCenter für die Entsorgungskosten von Verpackungen.	

Tabelle 90: ProducerAdvanced Properties

20.11.4 PRODUCERADVANCED PROPERTIES

Name	Grundfunktionalität: Prototyp2	Anpassungen
InitialNumber-OfProducts *(Expression)*	Anzahl Produkte, welche beim Initialisieren der Simulation im Lager des *ProducerAdvan-ced* vorhanden sind.	
PackagingType-1-2EntityType *(Entity)*	EntityType der Packaging Typen.	
Initial-Availability *(Boolean)*	Gibt die Verfügbarkeit bei Simulationsstart an.	

StorageManagement *(Boolean)*	Gibt an, ob die Lagerverwaltung verwendet werden soll.	
SellPriceProduct *(Integer)*	Verkaufspreis des hergestellten Produkts.	
PackagingWasteCost *(Expression)*	Kosten, die durch die Entsorgung von Verpackung entstehen.	
PackagingCost *(Integer)*	Kosten, welche durch den Verpackungsvorgang entstehen.	
ProductionCostPerProduct *(Expression)*	Herstellungskosten pro Produkt.	
StorageCostPerProduct *(Expression)*	Lagerkosten, pro Produkt pro Stunde im Lager.	
InitialReorderPoint *(Integer)*	Ab welchem Lagerbestand nachbestellt wird.	
Packaging1-2OrderSize *(Expression)*	Bestellgrössen der Verpackungen.	
PackagingType1-2ProducerTable *(Table)*	Tabellen, in welchen die Lieferanten für die Verpackungen aufgeführt werden.	
RawMaterial1-2OrderSize *(Expression)*	Wie viel Rohmaterial bei einer Nachbestellung bestellt wird.	
RawMaterial1-2ProducerTable *(Table)*	Tabellen, in welchen die Lieferanten für die Rohmaterialien aufgeführt werden.	
ReturnAddress *(Node)*	Annahmestelle für *OrderDecline* Nachrichten.	
FinalProductType *(Entity)*	Produkttyp des hergestellten Produkts.	
RawMaterial1-2 *(Entity)*	Produkttypen der verwendeten Rohmaterialien.	

RawMaterial1-2Production-Number *(Integer)*	Wie viel Rohmaterial für die Herstellung eines Batchs benötigt wird.	
FinalProductTargetStock *(Integer)*	Welchen Lagerstand die Lagerverwaltung halten soll. Wird nur verwendet, wenn Lagerverwaltung aktiviert wurde.	
FinalProduct-Production-BatchSize *(Integer)*	Batch Size der hergestellten Produkte. Spezifiziert wie viele Produkte in einem Produktionsvorgang hergestellt werden.	
StandardProductionSize *(Integer)*	Legt fest, wie viele Produkte von der *ProductionUnit* hergestellt werden, wenn der TargetStock unterschritten wird.	
ProductionTime *(Expression)*	Produktionszeit pro Produkt.	

Tabelle 91: ProducerAdvanced Properties

20.11.5 PRODUCERADVANCED STATES

Name	Grundfunktionalität: Prototyp2	Anpassungen
RememberDeliverAddress *(Node Reference)*	Zwischenspeicher für die Lieferadresse.	
ReorderPoint *(Integer)*	Schwellwert, der bei Simulationsstart initialisiert wird.	
RememberNumberOrdered *(Integer)*	Zwischenspeicher für die Anzahl der bestellten Produkte.	
RememberReturnAddress *(Node Reference)*	Zwischenspeicher für die Adresse, an welche *OrderDecline* Nachrichten für die Bestellung gesendet werden.	
OpenOrderPackaging *(Integer)*	Speichert, ob es eine offene Bestellung für Verpackungen gibt.	
RememberCurrentInventory *(Integer)*	Zwischenspeicher für den momentanen Lagerbestand.	

RememberOrderedProduct *(Integer)*	Zwischenspeicher für den Produkttypen, der bestellt wurde	
RememberCurrentInventoryPackaging *(Integer)*	Zwischenspeicher für die Anzahl Verpackungen im Lager.	
RememberPackagingType *(Integer)*	Zwischenspeicher für den in der *Order* angegebenen Verpackungstypen.	
RememberCurrentInventoryRawMaterial1-2 *(Integer)*	Zwischenspeicher für den Lagerbestand der Rohmaterialien.	
CurrentOrderAddress *(Node Reference)*	Zwischenspeicher für die ausgewählte Bestelladresse.	
CurrentReorderAmount *(Integer)*	Zwischenspeicher für die Anzahl Produkte, welche nachbestellt werden.	
OpenOrderRawmaterial1-2 *(Boolean)*	Speichert, ob bereits eine offene Bestellung vorhanden ist.	
RememberProductionSize *(Integer)*	Zwischenspeicher für die Produktionsgrösse	
ProductionCounter *(Integer)*	Zähler, welcher für das Zählen der im Lager ankommenden Produkte verwendet wird.	
Availability *(Boolean)*	Verfügbarkeit des *ProducerAdvanced* zum Simulationsstart.	
Name	**Zusätzliche States: Prototyp3**	**Anpassungen**
ProductionSuccessful *(Boolean)*	Gibt an, ob die Produktion erfolgreich durchgelaufen ist.	

Tabelle 92: ProducerAdvanced States

20.11.6 SCHNITTSTELLEN

Der *ProducerAdvanced* hat die gleichen vier Schnittstellen wie das *Warehouse*.

Bestelleingang *OrderEntities* werden hier entgegengenommen.

Bestellausgang *OrderEntities* werden über diesen Ausgang versendet.

Wareeingang Waren werden über diesen Eingang entgegengenommen.

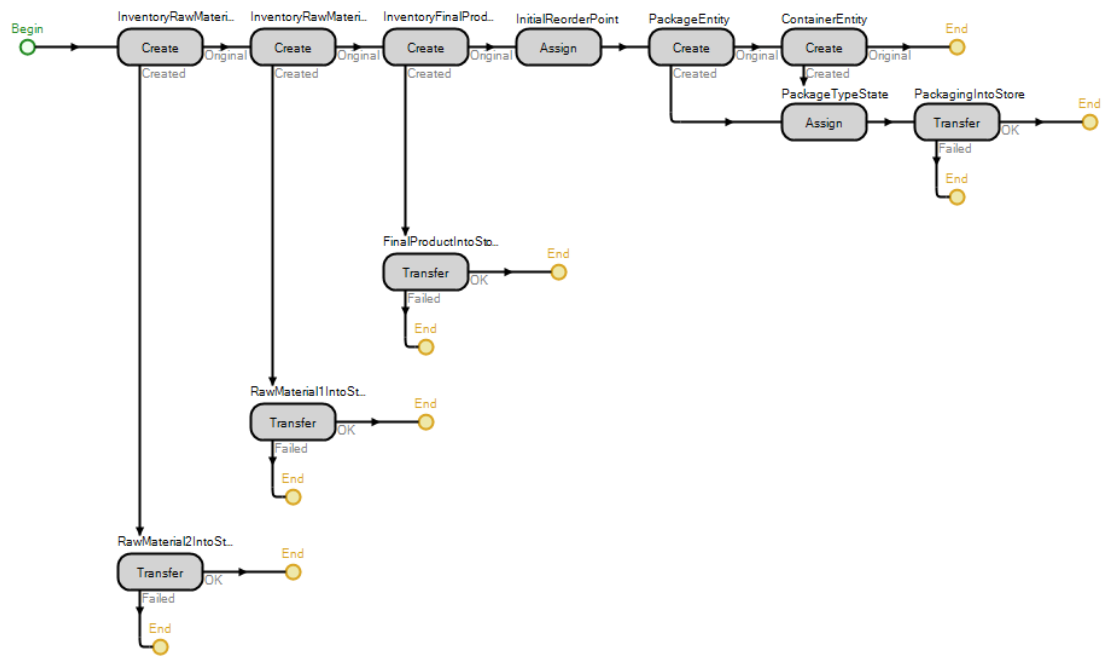
Warenausgang Waren werden über diesen Ausgang versendet.

20.11.7 PROZESSE

20.11.7.1 Prototyp2

Nachfolgend der Prozess, welcher bei Simulationsstart einen Anfangsbestand für das interne Lager erstellt und die *Availability* initialisiert.

OnRunInitialized (Prototyp2-3)

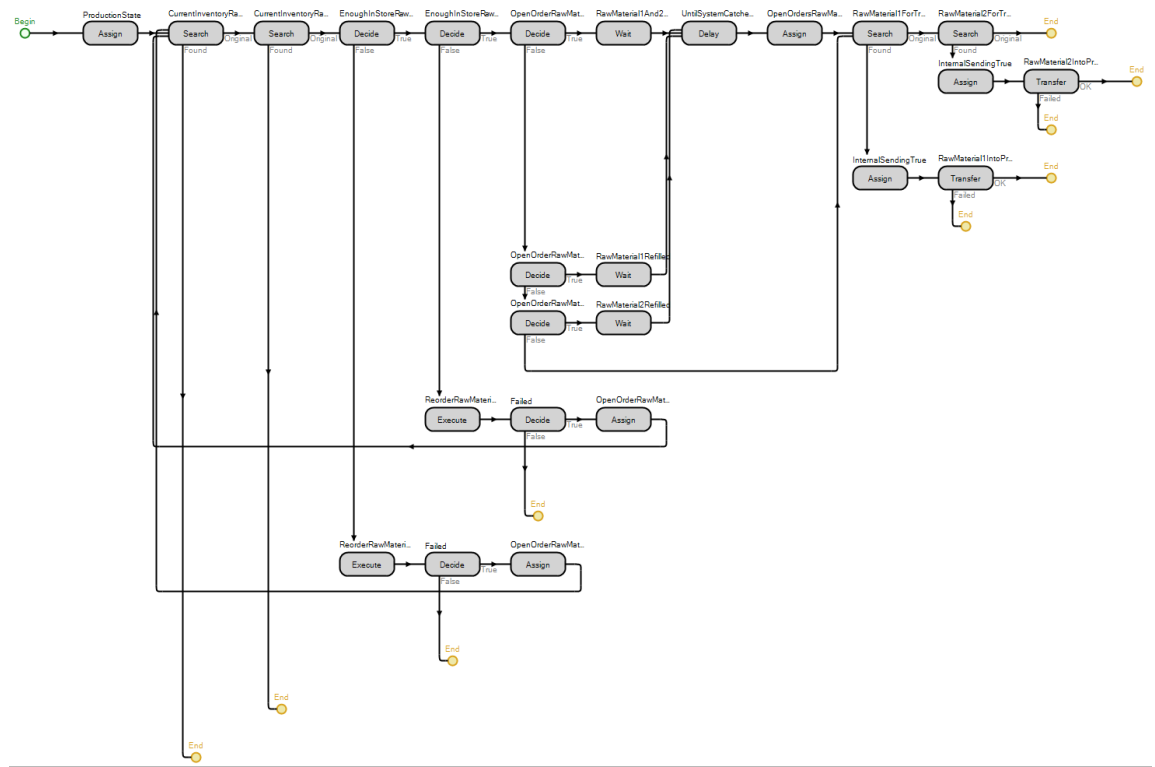


Create.InventoryRaw-Material1-2	Erstellt den Anfangsbestand für *RawMaterial1* und *RawMaterial2*
Transfer.X	Verschiebt die neu erstellten Objekte in den Input Buffer des stations-internen Lagers.
Create.InventoryFi-nalProduct	Erstellt den Anfangsbestand für das *FinalProduct*.
Assign.InitialAvailabil-ity	Setzt die initiale Verfügbarkeit des *ProducerAdvanced* fest.
Create.PackageEntity	Erstellt den Anfangsbestand der Verpackungen.
Assign.Pack-ageTypeStates	Schreibt den Produkttypen der erstellten Verpackungen in die *Produc-tEntity*.

Tabelle 93: ProducerAdvanced: OnRunInitialized

Dieser Prozess stellt dar, wie die Produktion verwaltet wird.

Production_Processing (Prototyp2-3)



Assign.Production-State	Setzt den Production State auf 0.
Search.CurrentInventoryRawmaterial1-2	Überprüft den Lagerbestand für die Rohmaterialien.
Decide.EnoughInStoreRawMaterial1-2	Entscheidet, ob der Lagerbestand für die Produktion ausreicht.
Execute.ReorderRawMaterial1-2	Führt den Prozess *Reorder_RawMaterial1*-2 aus.
Decide.Failed	Überprüft, ob die Bestellung fehlgeschlagen ist und beendet in diesem Fall den Produktionsprozess.
Assign.OpenOrderRawMaterial1-2	Setzt *OpenOrder* auf 1, falls eine Bestellung ausgelöst wird.
Decide.OpenOrderRawmaterial1-2	Entscheidet, welche *OpenOrder* gesetzt werden.
Wait.RawMaterial1-2(And2)Refilled	Wartet darauf, dass Bestellungen im Lager ankommen.
Delay.UntilSystemCatchesUp	Wartet darauf, bis die benötigten Produkte im Lager angekommen sind, da diese gleichzeitig aber nicht «ganz» gleichzeitig eintreffen.

Assign.OpenOrders-RawMaterials	Setzt die *OpenOrders* auf 0.
Search.RawMaterial1-2ForTransfer	Sucht und markiert die Rohmaterialien für den Transfer in die Produktion.
Assign.InternalSendingTrue	Setzt das *InternalSending* Zeichen, welches signalisiert, dass Produkte in die Produktion transportiert werden sollen.
Transfer.RawMaterial1-2IntoProduction	Verschiebt die Produkte aus dem Store.

Tabelle 94: ProducerAdvanced Process: Production_Processing

Nachfolgender Prozess zeigt, wie die Nachbestellungen von Verpackungen stattfinden.

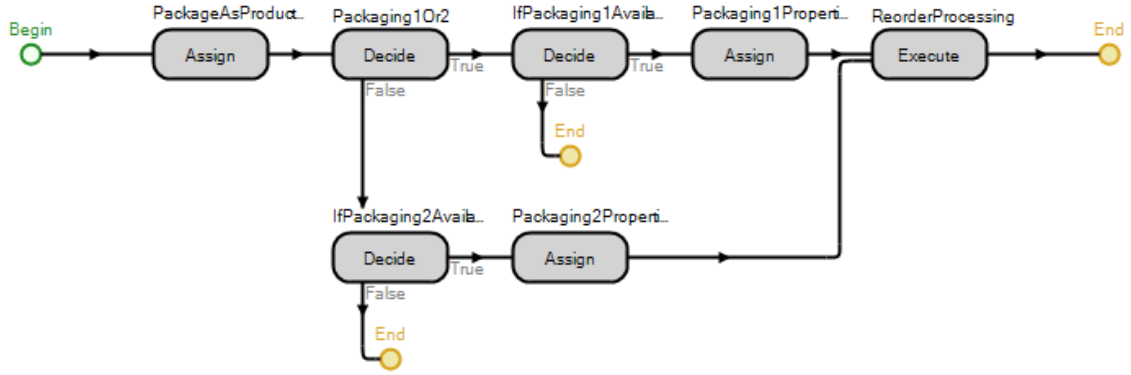
Reorder_Packaging (Prototyp2-3)	
	
Assign.PackageAsProductToOrder	Setzt den Produkttyp des zu bestellenden Produkts auf *Packaging*.
Decide.Packaging1Or2	Entscheidet, ob *Packaging* 1 oder 2 nachbestellt werden soll.
Decide.IfPackaging1-2Available	Überprüft die Verfügbarkeit des Verpackungslieferanten.
Assign.Packaging1-2Properties	Schreibt die Information des ausgewählten Lieferanten in die allgemeinen States.
Execute.ReorderProcessing	Führt den Prozess *Reorder_Processing* aus.

Tabelle 95: ProducerAdvanced Process: Reorder_Packaging

Dieser Prozess die allgemeinen Schritte aller Nachbestellungen aus.

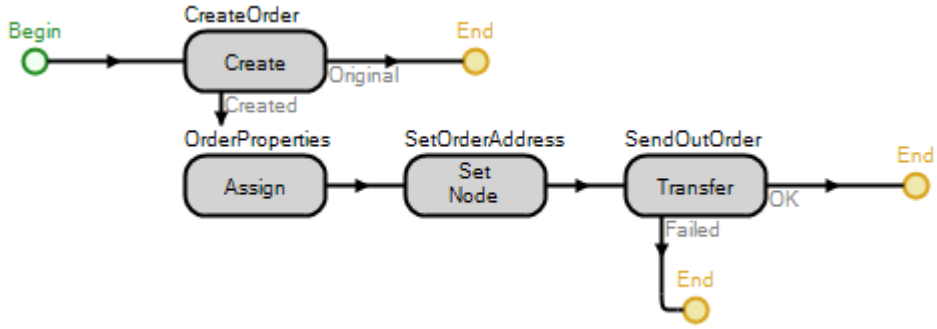
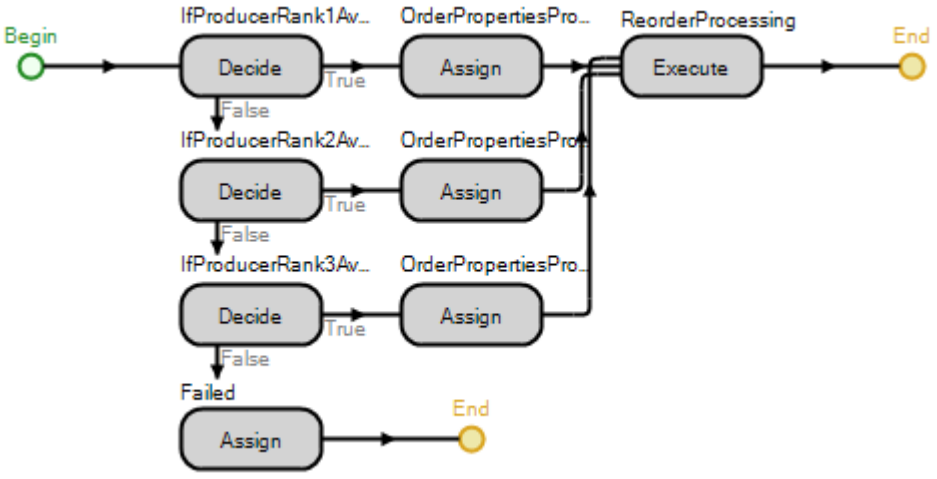
Reorder_Processing (Prototyp2-3)	
	
Create.CreateOrder	Erstellt eine *Order* Entity.
Assign.OrderProperties	Füllt die Bestellangaben auf der *Order* aus.
SetNode.SetOrderAddress	Setzt die Bestelladresse als nächste Station für die Bestellung.
Transfer.SendOutOrder	Verschickt die *Order*.

Tabelle 96: ProducerAdvanced Process: Reorder_Processing

Prozess für die Lieferantenwahl bei Nachbestellungen von Rohmaterialien.

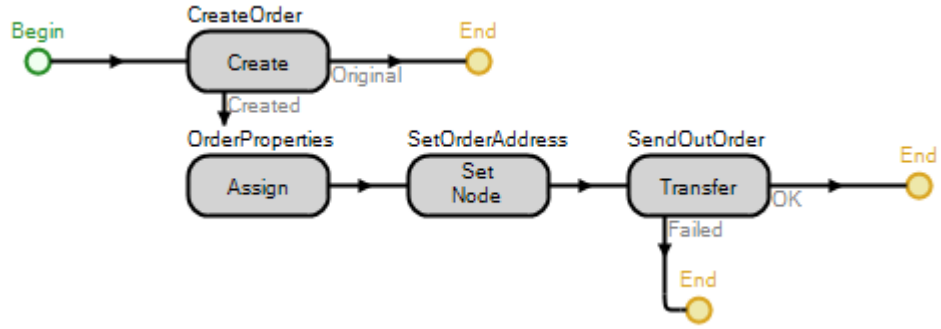
Reorder_RawMaterial1-2 (Prototyp2-3)	
	
Decide.IfProducerRank1-3Available	Überprüft, ob die Lieferanten verfügbar sind.
Assign.OrderPropertiesProducer1-3	Schreibt die Angaben des ausgewählen Lieferanten in die allgemeinen States.
Assign.Failed	Setzt das failed Zeichen.

Execute.ReorderProcessing	Führt den Prozess *Reorder_Processing* aus.

Tabelle 97: ProducerAdvanced Process: Reorder_RawMaterial1-2

Prozess, welcher für das Versenden von *OrderDecline* Nachrichten verantwortlich ist.

SendOrderDecline (Prototyp2-3)

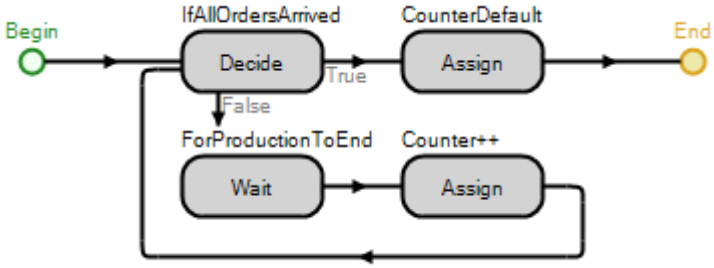


Create.CreateOrder	Erstellt eine *OrderEntity*.
Assign.OrderProperties	Setzt die *OrderDecline* Properties.
SetNode.SetOrderAddress	Setzt den nächsten Hop.
Transfer.SendOutOrder	Verschickt die *OrderDecline* Nachricht.

Tabelle 98: ProducerAdvanced Process: SendOrderDecline

Prozess, welcher die Anzahl der hergestellten Produkte mit der benötigten Auftragsmenge abgleicht und die Produkte nur weitergibt, die Bestellung erfüllt werden kann.

Wait_ForProductionToEnd (Prototyp2-3)



Decide.IfAllOrderArrived	Überprüft, ob alle Produkte im Lager angekommen sind.
Wait.ForProductionToEnd	Wartet auf das Event, dass verschickt wird, wenn etwas im Lager angekommen ist.
Assign.CounterDefault	Setzt den Counter zurück auf 0.
Assign.Coutner++	Zählt den Counter hoch.

Tabelle 99: ProducerAdvanced Process: Wait_ForProductionToEnd

Prozess, der für die Verwaltung der Kosten verantwortlich ist.

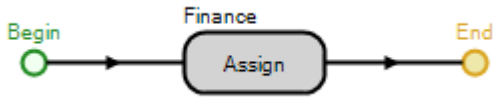
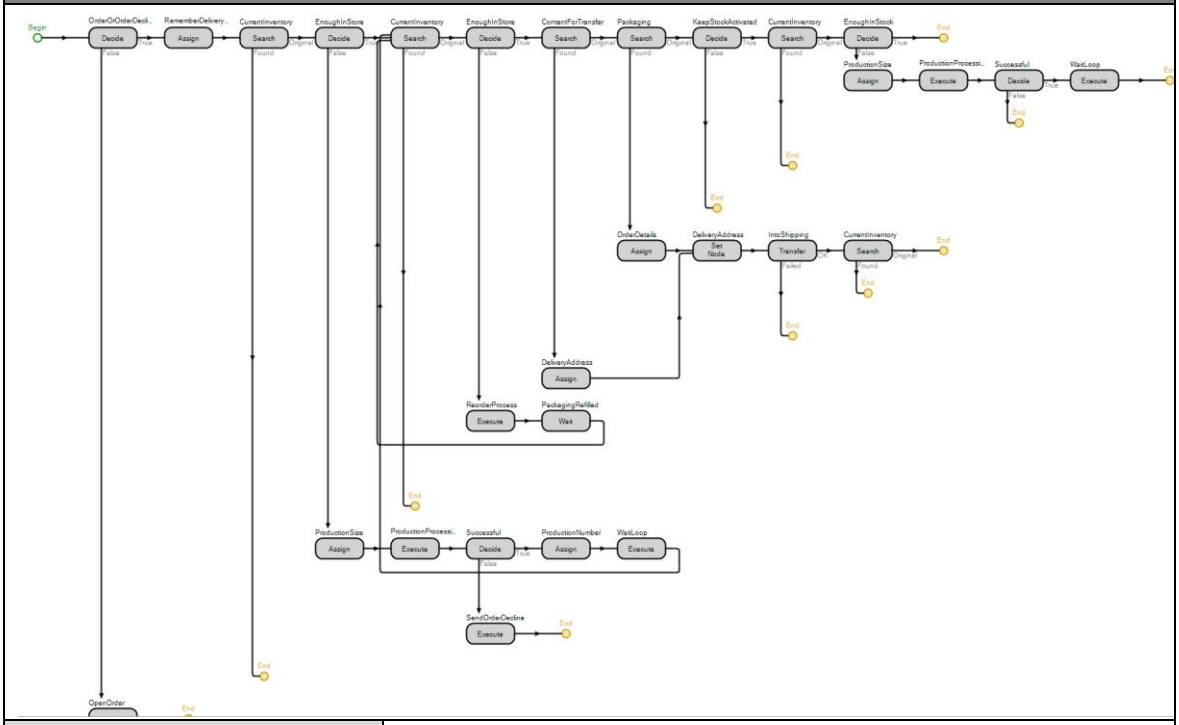
InputProducerAdvanced_Entered (Prototyp2-3)	
	
Assign.Finance	Zählt die Kosten hoch.

Tabelle 100: ProducerAdvanced Process: InputProducerAdvanced_Entered

Prozess, welcher die allgemeinen Bestellschritte aufführt.

Management_Processing (Prototyp2-3)	
	
Decide.OrderOrOrderDecline	Überprüft, ob es sich bei der *OrderEntity* um eine Bestellung oder eine *OrderDecline* Nachricht handelt.
Assign.OpenOrder	Setzt *OpenOrder* auf 0.
Assign.RememberDeliveryAddress	Schreibt die Bestellangaben in die Zwischenspeicher.
Search.CurrentInventory	Überprüft den momentanen Lagerstand.
Decide.EnoughInStore	Entscheidet, ob der Lagerbestand ausreicht, um die Bestellung zu erfüllen oder ob in der Produktion nachbestellt werden muss
Assign.ProductionSize	Berechnet, wie viele Batches hergestellt werden müssen und setzt die *ProductionSize* dementsprechend.
Execute.ProductionProcessing	Führt den Prozess *Production_Processing* aus.

Decide.Successful	Überprüft, ob die Produktion erfolgreich abgeschlossen ist.
Execute.SendOrderDecline	Führt den Prozess *SendOrderDecline* aus.
Execute.ReorderProcess	Führt den Prozess *Reorder_Packaging* aus.
Wait.PackagingRefilled	Wartet darauf, dass die bestellten Verpackungen beim *ProducerAdvanced* eintreffen.
Search.ContentForTransfer	Sucht und markiert Produkte im Lager für das Verschicken von Lieferungen.
Assign.ProductionNumber	Rechnet aus, wie viele *Batches* hergestellt werden müssen und setzt die *ProductionSize* dementsprechend.
Search.Packaging	Sucht und markiert Verpackungen im Lager für das Verschicken von Lieferungen.
Assign.OrderDetails	Füllt die Bestellangaben aus.
SetNode.DeliveryAddress	Setzt den nächsten Hop auf die Lieferaddresse.
Assign.DeliveryAddress	Füllt die Lieferadresse aus.
Execute.WaitLoop	Führt den Prozess *Wait_ForProductionToEnd* aus.
Decide.KeepStockActivated	Überprüft, ob die Lagerverwaltung aktiviert ist.
Transfer.IntoShipping	Verschiebt die Produkte und Verpackung in den Warenausgang.

Tabelle 101: ProducerAdvanced Process: Management_Processing

Nachfolgender Prozess dient der Kostenverrechnung an die entsprechenden CostCenter, wenn eine Lieferung versendet wird.

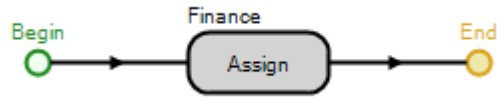
OutputShipping_Exited (Prototyp2-3)	
	
Assign.Finance	Rechnet die Kosten hoch.

Tabelle 102: ProducerAdvanced: OutputShipping_Exited

20.12 STOREPRODUCER

20.12.1 AUFBAU

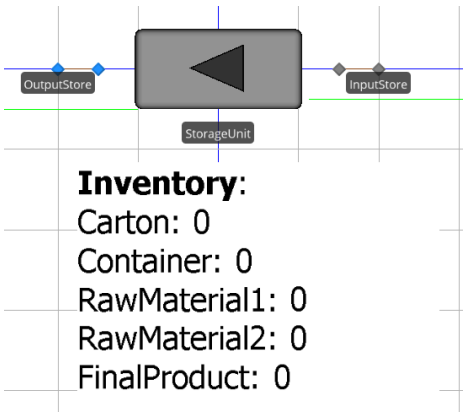
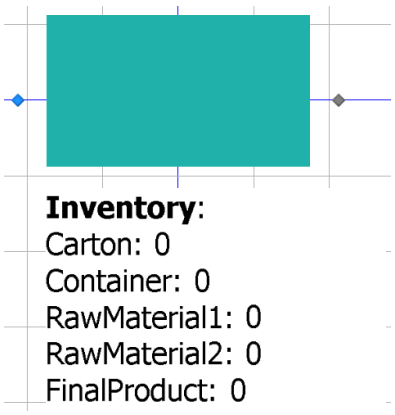
Prototyp2	
Interner Aufbau	**Externe Modellansicht**
	
Die *StoreProducer* Station nimmt Verpackungen und Produkte entgegen, lagert diese in einem Server ein, respektive aus und führt den Lagerbestand dazu.	

Tabelle 103: StoreProducer Aufbau

20.12.2 STOREPRODUCER PROPERTIES

Name	Grundfunktionalität: Prototyp2	Anpassungen
RawMaterial1 *(Entity)*	Gibt den Produkttyp für *RawMaterial1* an.	
RawMaterial2 *(Entity)*	Gibt den Produkttyp für *RawMaterial2* an.	
FinalProduct *(Entity)*	Gibt den Produkttyp für *FinalProduct* an.	

Tabelle 104: StoreProducer Properties

20.12.3 STOREPRODUCER STATES

Name	Grundfunktionalität: Prototyp2	Anpassungen
Inventory-Carton *(Integer)*	In diesem State wird der Lagerbestand für *Carton* nachgeführt.	
Inventory-Container *(Integer)*	In diesem State wird der Lagerbestand für *Container* nachgeführt.	
Inventory-RawMaterial1 *(Integer)*	In diesem State wird der Lagerbestand für *RawMaterial1* nachgeführt.	

Inventory-RawMaterial2 *(Integer)*	In diesem State wird der Lagerbestand für *RawMaterial2* nachgeführt.	
InventoryFinal-Product *(Integer)*	In diesem State wird der Lagerbestand für das *FinalProduct* nachgeführt.	

Tabelle 105: StoreProducer States

20.12.4 STOREPRODUCER EVENTS

Name	Grundfunktionalität: Prototyp1	Anpassungen
OrderReceived	Wird ausgelöst, wenn Produkte im Lager ankommen.	
RawMaterial1-Delivery-Received	Wird ausgelöst, wenn *RawMaterial1* im Lager ankommen.	
RawMaterial2-Delivery-Received	Wird ausgelöst, wenn *RawMaterial2* im Lager ankommen.	
FinalProduct-Instore	Wird ausgelöst, wenn *FinalProduct* im Lager ankommen.	
Packaging-Delivery-Received	Wird ausgelöst, wenn bestellte Verpackungen im Lager ankommen.	

Tabelle 106: StoreProducer Events

20.12.5 SCHNITTSTELLEN

Die *StoreProducer*-Station hat zwei Schnittstellen, welche für das Entgegennehmen von Produkten und Verpackungen und das Ausschicken dieser verantwortlich sind. Dabei ist jede Schnittstelle nur für eine der beiden Funktionen nutzbar.

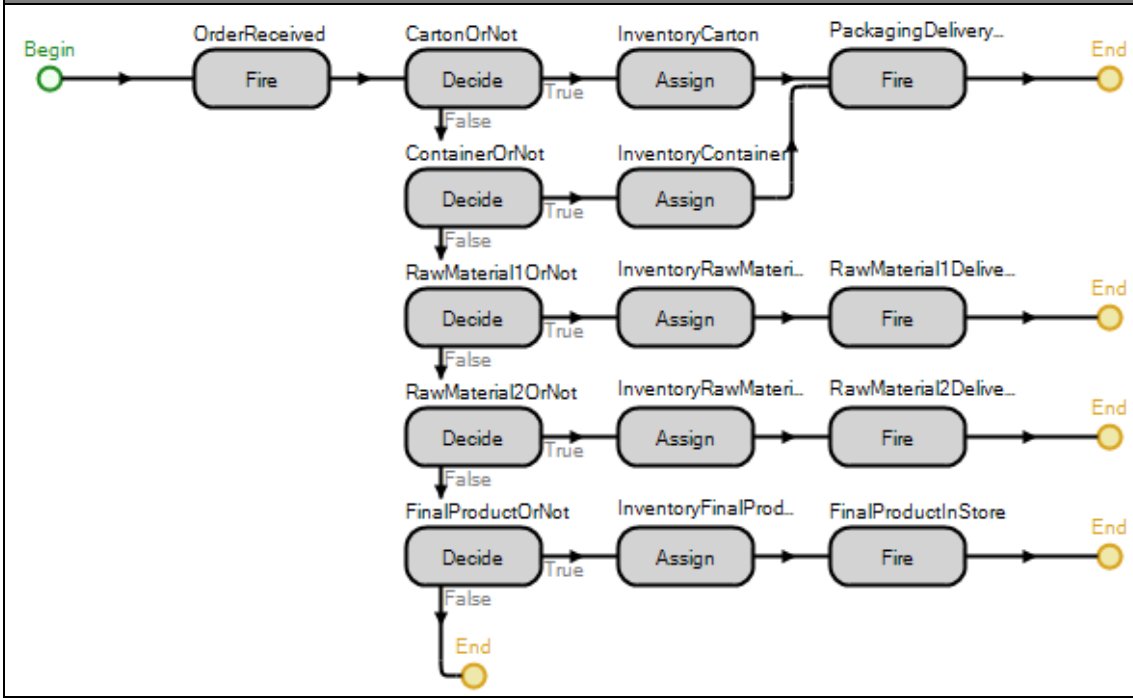
Wareneingang *ProductEntities* werden hier entgegengenommen.

Warenausgang Über diesen Knoten verlassen *ProductEntities* die *StoreProducer*-Station.

20.12.6 PROZESSE

Dieser Prozess dient dazu, den Lagerbestand hochzuzählen, wenn Produkte oder Verpackungen in den Server, also das Lager, eingelagert werden.

StorageUnit_Entered (Prototyp2-3)



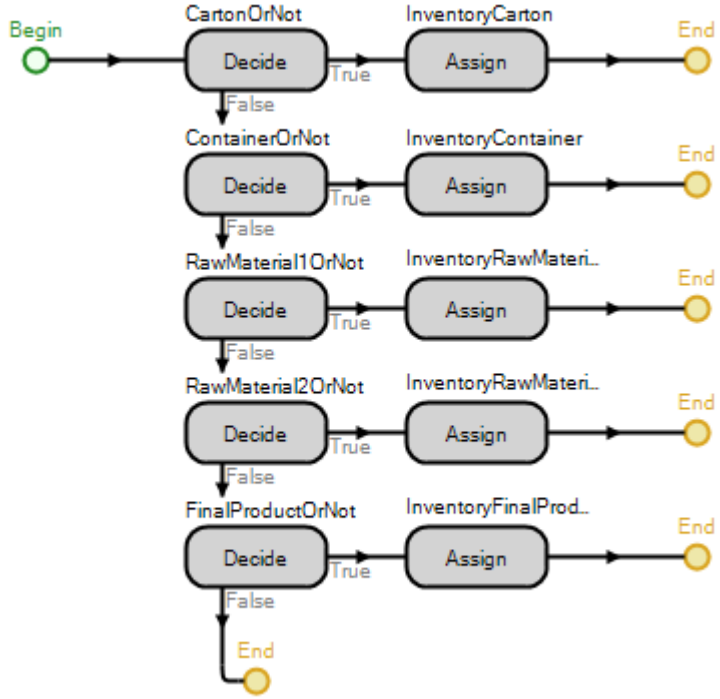
Fire.OrderReceived	Löst das Event *OrderReceived* aus.
Decide.CartonOrNot	Überprüft, ob es sich beim *Product* um einen *Carton* handelt.
Assign.InventoryCarton	Zählt das Inventar für *Carton* um 1 hoch, wenn ein *Carton* das Lager betritt.
Fire.PackagingDeliveryReceived	Löst das Event *PackagingDeliveryReceived* aus.
Decide.ContainerOrNot	Überprüft, ob es sich beim *Product* um einen *Container* handelt.
Assign.InventoryContainer	Zählt das Inventar für *Container* um 1 hoch, wenn ein *Container* das Lager betritt.
Decide.RawMaterial1OrNot	Überprüft, ob es sich beim *Product* um ein *RawMaterial1* handelt.
Assign.InventoryRawMaterial1	Zählt das Inventar für *RawMaterial1* um 1 hoch, wenn ein *RawMaterial1* das Lager betritt.
Fire.RawMaterial1DeliveryReceived	Löst das Event *RawMaterial1DeliveryReceived* aus.
Decide.RawMaterial2OrNot	Überprüft, ob es sich beim *Product* um ein *RawMaterial2* handelt.
Assign.InventoryRawMaterial2	Zählt das Inventar für *RawMaterial2* um 1 hoch, wenn ein *RawMaterial2* das Lager betritt.

Fire.RawMaterial2DeliveryReceived	Löst das Event *RawMaterial2DeliveryReceived* aus.
Decide.FinalProductOrNot	Überprüft, ob es sich beim *Product* um ein *FinalProduct* handelt.
Assign.InventoryFinalProduct	Zählt das Inventar für *FinalProduct* um 1 hoch, wenn ein *FinalProduct* das Lager betritt.
Fire.FinalProductInStore	Löst das Event *FinalProduct* aus.

Tabelle 107: StoreProducer Process: StorageUnit_Entered

Dieser Prozess dient dazu, den Lagerbestand runterzuzählen, wenn Produkte oder Verpackungen den Server, also das Lager, verlassen.

StorageUnit_Exited (Prototyp2-3)



Decide.CartonOrNot	Überprüft, ob es sich beim *Product* um einen *Carton* handelt.
Assign.InventoryCarton	Zählt das Inventar für *Carton* um 1 runter, wenn ein *Carton* das Lager verlässt.
Decide.ContainerOrNot	Überprüft, ob es sich beim *Product* um einen *Container* handelt.
Assign.InventoryContainer	Zählt das Inventar für *Container* um 1 runter, wenn ein *Container* das Lager verlässt.
Decide.RawMaterial1OrNot	Überprüft, ob es sich beim *Product* um ein *RawMaterial2* handelt.
Assign.InventoryRawMaterial1	Zählt das Inventar für *RawMaterial1* um 1 runter, wenn ein *RawMaterial1* das Lager verlässt.

Decide.RawMaterial2OrNot	Überprüft, ob es sich beim *Product* um ein *RawMaterial1* handelt.
Assign.InventoryRawMaterial2	Zählt das Inventar für *RawMaterial2* um 1 runter, wenn ein *RawMaterial2* das Lager verlässt.
Decide.FinalProductOrNot	Überprüft, ob es sich beim *Product* um ein FinalProduct handelt.
Assign.InventoryFinalProduct	Zählt das Inventar für *FinalProduct* um 1 runter, wenn ein *FinalProduct* das Lager verlässt.

Tabelle 108: StoreProducer Process: StorageUnit_Exited

20.13 PRODUCTIONUNIT

20.13.1 AUFBAU

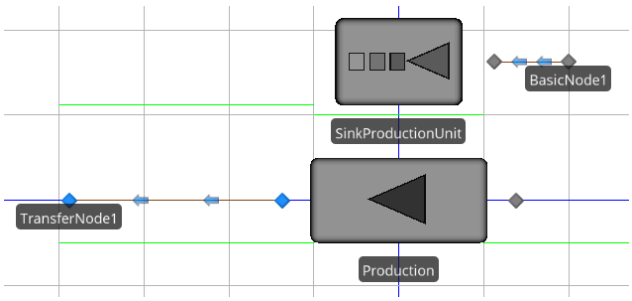
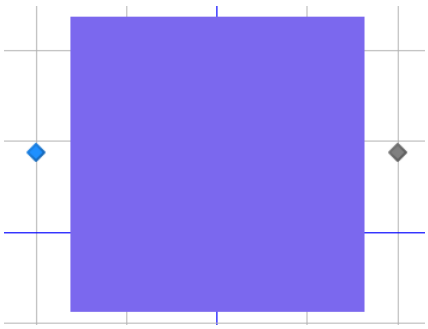
Prototyp2	
Interner Aufbau	**Externe Modellansicht**
	
Die *ProductionUnit* Station nimmt Produkte entgegen und stellt neue Produkte her. Standardmässig nimmt es zwei *RawMaterials* entgegen und produziert ein *FinalProduct*.	

Tabelle 109: ProductionUnit Aufbau

20.13.2 PRODUCTIONUNIT PROPERTIES

Name	Grundfunktionalität: Prototyp2	Anpassungen
RawMaterial1 *(Entity)*	Gibt den benötigten Produkttyp für *RawMaterial1* an.	
RawMaterial2 *(Entity)*	Gibt den benötigten Produkttyp für *RawMaterial2* an.	
Required-RawMaterial1 *(Integer)*	Gibt die benötigte Menge an *RawMaterial1* an, die für die Herstellung einer *BatchSize* für das *FinalProduct* benötigt werden.	
Required-RawMaterial2 *(Integer)*	Gibt die benötigte Menge an *RawMaterial2* an, die für die Herstellung einer *BatchSize* für das *FinalProduct* benötigt werden.	
FinalProduct *(Entity)*	Gibt den Produkttyp an, der aus den zwei *RawMaterials* hergestellt wird.	
BatchSizeFinal-Product *(Integer)*	Gibt die Produktionsmenge von *FinalProduct* an, die in einem Schritt produziert wird.	
ProductionUnit-ProcessingTime *(Expression)*	Gibt die benötigte Produktionszeit an, die für die Herstellung einer *BatchSize* für das *FinalProduct* benötigt wird.	
Name	**Zusätzliche Properties: Prototyp3**	**Anpassungen**
CostCenter-Production *(Cost Center)*	Gibt das CostCenter an, auf welchem die Production kosten gebucht werden.	

ProductionCost-PerProduct *(Expression)*	Gibt die Produktionskosten pro Produkt an.	

Tabelle 110: ProductionUnit Properties

20.13.3 PRODUCTIONUNIT STATES

Name	Grundfunktionalität: Prototyp2	Anpassungen
RawMaterial1-ReadyForPro-duction *(Integer)*	Hier wird gezählt, wie viele *RawMaterial1* die *ProductionUnit* bereits zur Verfügung hat.	
RawMaterial2-ReadyForPro-duction *(Integer)*	Hier wird gezählt, wie viele *RawMaterial2* die *ProductionUnit* bereits zur Verfügung hat.	
CurrentProduc-tionNumber *(Integer)*	Hier wird gezählt, wie viele *FinalProducts* hergestellt wurden.	
Production-Number *(Integer)*	Dieser Wert gibt an, wie viele *FinalProducts* produziert werden sollen.	

20.13.4 SCHNITTSTELLEN

Die *StoreProducer*-Station hat zwei Schnittstellen, welche für das Entgegennehmen von Produkten (*RawMaterials*) und das Ausschicken von *FinalProducts* verantwortlich sind. Dabei ist jede Schnittstelle nur für eine der beiden Funktionen nutzbar.

Wareneingang *RawMaterials* werden hier entgegengenommen, gezählt und dann zerstört.

Warenausgang Über diesen Knoten verlassen *ProductEntities*, also die hergestellten *FinalProducts* die *ProductionUnit* Station.

20.13.5 PROZESSE

Dieser Prozess dient dazu, den Lagerbestand hochzuzählen, wenn Produkte oder Verpackungen in den Server, also das Lager, eingelagert werden.

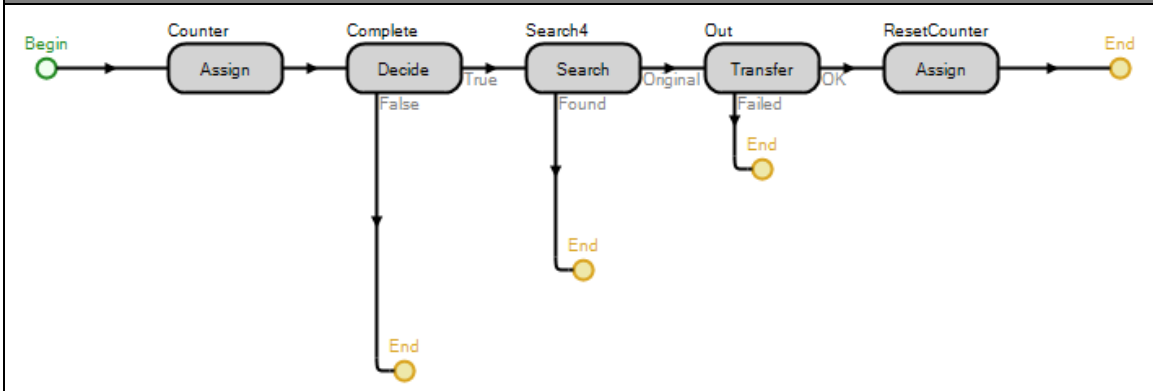
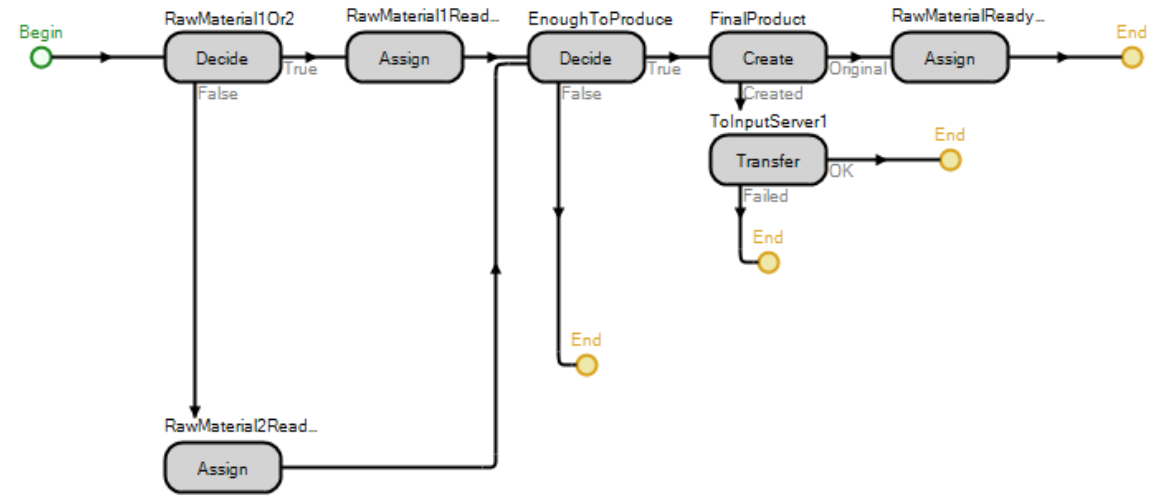
SendOut_BeforeProcessing (Prototyp2)	
	
Assign.Counter	Zählt den State *CurrentProductionNumber* um 1 hoch, wenn ein *FinalProduct* hergestellt wurde.
Decide.Complete	Überprüft, ob die bereits hergestellte Menge an *FinalProducts* der benötigten Menge entspricht.
Search.Search4	Sucht die benötigte Menge der produzierten *FinalProducts*.
Transfer.Out	Transferiert die *FinalProducts* zum Aussenden aus der ***ProductionUnit***.
Assign.ResetCounter	Setzt den Zähler für *CurrentProductionNumber* zurück, also auf 0.

Tabelle 111: ProductionUnit Process: SendOut_BeforeProcessing

Dieser Prozess dient dazu, den Lagerbestand runterzuzählen, wenn Produkte oder Verpackungen den Server, also das Lager, verlassen.

Sink1_Entered (Prototyp2-3)	
	
Decide.RawMaterial1Or2	Überprüft, ob es sich beim ankommenden ***Product*** um ein *RawMaterial1* handelt.

Assign.RawMaterial1ReadyForProduction	Zählt den Counter der zur Verfügung stehenden *RawMaterial1* um eins hoch.
Decide.EnoughToProduce	Überprüft, ob die zur Verfügung stehenden *RawMaterial1* und *RawMaterial2* der benötigten Menge für die Produktion ausreichend ist.
Create.FinalProduct	Produziert die in der *BatchSize* festgelegten Menge an *FinalProducts*.
Assign.RawMaterialReadyForProduction	Zählt die Menge an *RawMaterial1* und *RawMaterial2*, die für die Produktion benötigt wurden, ab von der zur Verfügung stehenden Menge.
Assign.RawMaterial2ReadyForProduction	Zählt den Counter der zur Verfügung stehenden *RawMaterial2* um eins hoch.
Transfer.ToInputServer1	Transferiert die produzierten FinalProducts in das InputNode des Servers.

Tabelle 112: ProductionUnit Process: Sink1_Entered

21 VERZEICHNISSE

21.1 ABBILDUNGSVERZEICHNIS

21.2 TABELLENVERZEICHNIS

IV. EIGENSTÄNDIGKEITSERKLÄRUNG

Erklärung

Wir erklären hiermit,

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

Ort, Datum: Rapperswil, 12.01.2021

Name, Unterschrift: Fabienne Lienhard F. Lienhard

Jana Kravarik J. Kravarik