Hochschule für Technik und Wirtschaft Dresden
- Fakultät Informatik/Mathematik -

Entwicklung einer performanten zentralen Produktionsplanung zu Vergleichs- und Integrationszwecken

Modul
- Diplomarbeit I193 -
im Studiengang Informatik Diplom
Jahrgang 2015

eingereicht von:
Pascal Schumann
Matrikelnummer: 39537

Betreuer:	Prof. Dr. rer. pol. Torsten Munkelt
Zweitkorrektor:	Martin Krockert
eingereicht am:	2. Dezember 2019
Thema:	Entwicklung einer performanten zentralen Produktionsplanung zu Vergleichs- und Integrationszwecken

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

Im Rahmen zweier aktueller Forschungsprojekte entwickeln zwei wissenschaftliche Mitarbeiter ein Konzept und einen Prototyp für eine sich selbst organisierende Produktion (SSOP). Die Untersuchung der SSOP hinsichtlich ihrer Qualität und ihrer Performanz erfolgt anhand eines parametrisierbaren ereignisdiskreten Simulationsmodelles einer Produktion. Um die SSOP mit einer zentral geplanten Produktion zu vergleichen und die zentral geplante Produktion mit der SSOP zu kombinieren, ist eine performante zentrale Produktionsplanung (ZPP) zu entwickeln, die auf dem oben erwähnten Simulationsmodell aufsetzt, auf denselben Datenstrukturen wie die SSOP basiert und in die SSOP integriert werden kann.

1.1. Motivation

Der bereits entwickelte Prototyp einer zentralen Produktionsplanung im Rahmen der Masterarbeit von Krockert und Seifert war nicht leistungsfähig genug im Vergleich zu anderen Planungssystemen, um kontinuierliche Ergebnisse zu liefern. Auch fehlte eine einheitlich Simulationsplattform, um ggf. hybride Lösungen bauen zu können. Daraus entwickelte sich der Wunsch nach einer neuen Umsetzung mit expliziten Anforderungen an die Performance. Die weitere Motivation für die SSOP sind hinreichend dokumentiert.[1]

1.2. Ziele

Das wesentliche Ziel ist es also, eine performante zentrale Produktionsplanung zu entwickeln. Ein weiteres Ziel ist es, die SSOP mit der zu entwickelnden ZPP vergleichen zu können. Dafür muss die ZPP eine hinreichend kurze Laufzeit aufweisen, funktional korrekt und in die SSOP integrierbar sein. Das wesentliche Nebenziel ist eine gute Struktur für die Erweiterbarkeit bzw. Ersetzbarkeit von Komponenten.

1.3. Vorgehensweise

Zunächst ist eine Einarbeitung in die Theorie der Produktionsplanung nötig. Dafür muss der Autor zunächst den Planungslauf analysieren und beschreiben. Dann analysiert und beschreibt der Autor die vorhandenen Datenstrukturen. Die Datenstrukturen bilden bereits durch ihre Struktur die Ergebnisse des Planungslaufs ab. Nach der Analyse und Beschreibung der Datenstrukturen entwickelt der Autor einen ersten Prototypen, der eine rudimentäre Materialbedarfsplanung durchführt.

Die Entwicklung des Prototypen erfolgt im Wechsel mit der Aufnahme der Anforderungen. Schritt für Schritt erweitert der Autor den Prototypen systematisch um die Teile der Kapazitätsplanung und überführt den Prototypen in ein Produktivsystem, dass den Anforderungen möglichst vollständig genügt. Dann beschreibt der Autor die Architektur und algorithmischen Form des entwickelten Systems. Die Verifikation erfolgt durch Unit- und Integrationstests. Abschließend führt der Autor eine Studie zur Überprüfung der Performance durch.

[1] Vgl. Krockert und Seifert, „Selbstorganisation oder zentrale Planung im Kontext von Industrie 4.0“, S. 1-2.

2. Beschreibung der zentralen Produktionsplanung

2.1. Beschreibung des MRP-Laufs

Die zentrale Produktionsplanung besteht im Wesentlichen aus der Materialbedarfsplanung und der Kapazitätsplanung. Die Materialbedarfsplanung und die Kapazitätsplanung sind ein Bestandteil des MRP II. Die Abkürzung MRP II steht für Manufacturing Resource Planning. Nach[1] wird MRP II wie folgt definiert:

> MRP II ist eine Methode zur effektiven Planung aller Ressourcen eines Fertigungsunternehmens. Im Idealfall unterstützt es die operative Planung in Mengeneinheiten sowie die finanzielle Planung in Geldeinheiten und besitzt eine Simulationskomponente zur Beantwortung von What-if-Fragen.

Das MRP II hat mehrere Voraussetzungen. Wesentliche Planungsparameter wie Kapazitäten, Durchlaufzeiten der Aufträge und Bearbeitungszeiten müssen mit hoher Sicherheit prognostizierbar sein.[2] Die Abb. 2.1 zeigt die allgemeine Planungs- und Steuerungslogik des MRP II und ist in[3] hinreichend beschrieben, der Autor betrachtet in diesem Kapitel nur die relevanten Schritte des MRP-Laufs.

Ein Primärbedarf beschreibt die Menge aller verkaufsfähigen Produkte und Erzeugnisse (Absatzprodukte), die zu einem bestimmten Zeitpunkt zur Verfügung stehen müssen, kurz geplante Produktionsmenge.[4] Die Primärbedarfsplanung, auch Produktionsprogrammsplanung, ermittelt aus Absatzprognosen bzw. bereits angenommenen Kundenaufträgen den Bedarf an Absatzprodukten für einen Planungszeitraum. Aus den ermittelten Bedarfen ergibt sich ein Absatzplan.[5] Eine Ressourcengrobplanung ist eine Prüfung, ob die in der Primärbedarfsplanung ermittelten Bedarfe (Absatzplan) mit den vorhandenen Ressourcen wie Betriebsmittel, Material und Personal in der Menge und zu den festgelegten Terminen realisierbar sind.[6]

Die Primärbedarfsplanung und die Ressourcengrobplanung des MRP-Laufs entfallen und sind nicht Gegenstand dieser Arbeit. Somit entfallen die Produktionsprogrammsplanung und ihre vorherigen Schritte aus Abb. 2.1 außer dem Bedarfsmanagement. Stattdessen gibt nur noch das Bedarfsmanagement, das die nachfolgenden Schritte auslöst. Das Bedarfsmanagement umfasst in dieser Arbeit ausschließlich die Kundenauftragserstellung. Die Fertigungssteuerung entfällt ebenfalls. Außerdem gibt es keine Rückkopplung zwischen den Komponenten, da auf Anhieb ein gültiger Plan erstellt werden soll.

Zusammengefasst zeigt die Abb. 2.2 den MRP-Lauf der ZPP als Aktivitätsdiagramm der UML 2.0 Zuerst muss die ZPP die Kundenaufträge erstellen, dann muss die ZPP die Materialbedarfsplanung durchführen und abschließend muss die ZPP die Kapazitätsplanung durchführen.

---

[1] Kurbel, *MRP II*.
[2] Vgl. Kurbel, *Enterprise Resource Planning und Supply Chain Management in der Industrie*, S. 174-175.
[3] Vgl. ebd., S. 99-101.
[4] Vgl. Schuh, *Produktionsplanung und -steuerung*, S. 37.
[5] Vgl. Scheer, *Wirtschaftsinformatik*, S. 98.
[6] Vgl. Schuh, *Produktionsplanung und -steuerung*, S. 98.

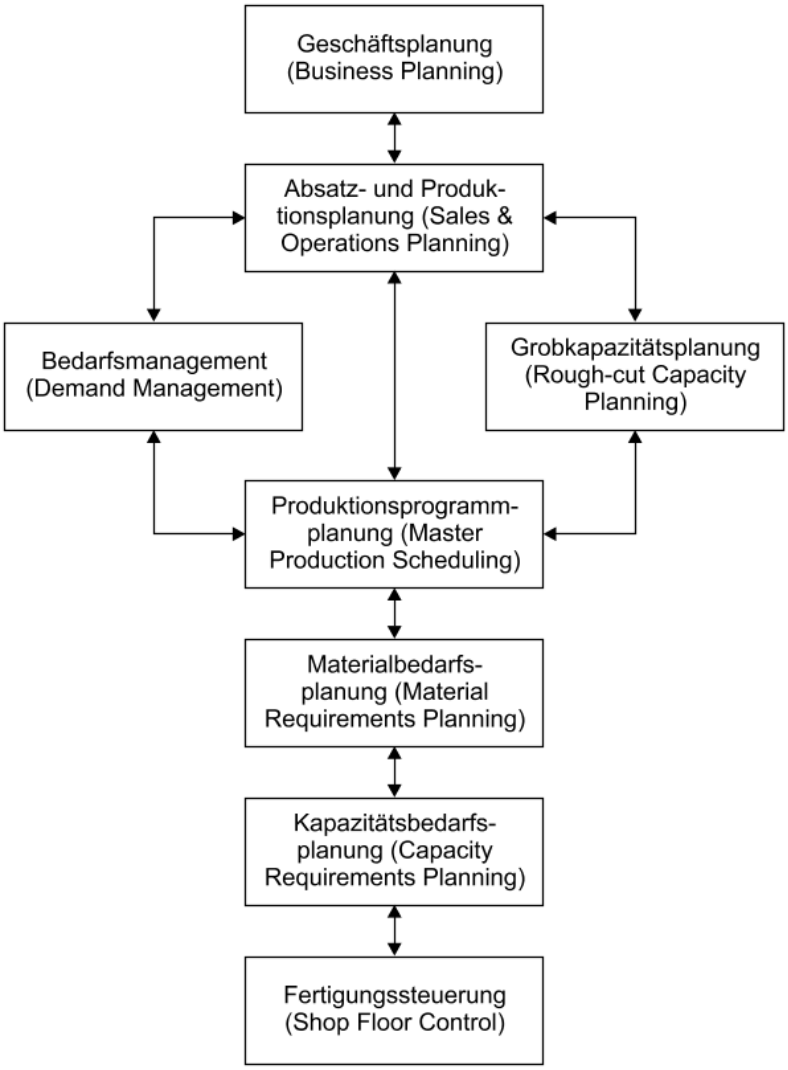

Abbildung 2.1.: Planungs- und Steuerungslogik im MRP II Quelle: Kurbel, *Enterprise Resource Planning und Supply Chain Management in der Industrie*, S. 100

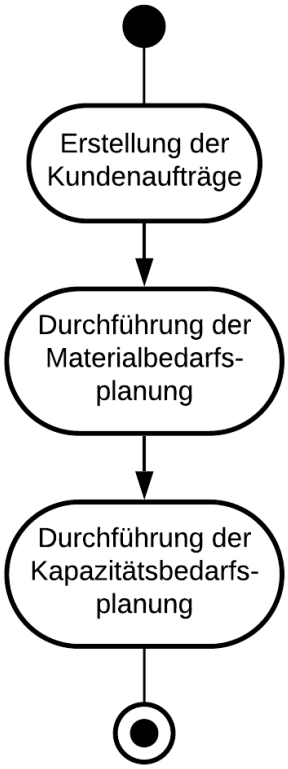

Abbildung 2.2.: MRP-Lauf der ZPP

2.2. Definitionen im Kontext eines gerichteten Graphen

In dieser Arbeit nutzt der Autor immer wieder gerichtete Graphen, um Problemstellungen zu lösen und Sachverhalte zu veranschaulichen. Diese Arbeit verwendet die Begriffe im Kontext eines gerichteten Graphen wie folgt. Ein Knoten a zeigt auf einen Knoten b, wenn es eine gerichtete Kante von a nach b gibt. Ein Pfeil ist eine gerichtete Kante, die von einem Knoten auf einen anderen Knoten oder sich selbst zeigt. Ein Blattknoten (Blatt) ist ein Knoten, auf den andere Knoten zeigen, er selbst zeigt auf keinen Knoten. Eine Wurzelknoten (Wurzel) ist ein Knoten, der auf andere Knoten zeigt, kein Knoten zeigt auf die Wurzel. Ein Knoten x ist ein Vorgänger eines Knoten y, wenn es einen Pfeil von x nach y gibt. Ein Knoten y ist ein Nachfolger eines Knoten x, wenn es einen Pfeil von x nach y gibt. Ein Wurzelbaum ist ein gerichteter zusammenhängender kreisfreier Graph, bei dem die Richtungen einheitlich weg vom Wurzelknoten gehen und jeder Knoten außer der Wurzel genau ein Vorgänger hat.[7] Ein Elter ist ein Elternteil und bezieht sich auf Knoten, die nur einen Vorgänger haben.[8] Eltern sind Vorgänger eines Knoten. Ein Kind ist ein Nachfolger eines Knoten.

2.3. Beschreibung der Stückliste

Jedes Erzeugnis eines Stückprozesses hat eine Stückliste. In einem Stückprozess (auch diskrete Fertigung genannt) werden die Mengeneinheiten (hauptsächlich) in Stück gemessen. Eine Stückliste ist die listenförmige Darstellung der Zusammensetzung eines Erzeugnis aus seinen Bestandteilen mit den relevanten Daten wie Teilenummern, Mengenangabe etc.[9]

In den folgenden Kapiteln benötigt der Autor eine zweistufige Stückliste, um die Algorithmen und deren Ergebnisse zu veranschaulichen. Der Autor wählt als übersichtliches Beispiel einen einfachen Tisch mit vier Beinen, das Beispiel referenziert der Autor in der übrigen Arbeit als Tischbeispiel. Die Tabelle 2.1 zeigt die zweistufige Stückliste des Tischbeispiels. Es gibt zwei Ebenen. Die erste Ebene ist der Tisch selbst. Ein Tisch besteht aus einer Tischplatte, 16 Schrauben und 4 Tischbeinen. Ein Tischbein besteht aus einer Anschraubplatte, einem Stahlrohr und einem Filzgleiter. Es gibt drei Operationen für die Stückliste. Ein Tischbein entsteht durch die erste Operation „Anschraubplatte anschweißen“ und die zweite sich anschließende Operation „Filzgleiter anstecken“. Die Operation „Anschraubplatte anschweißen“ schweißt eine Anschraubplatte an an ein Stahlrohr. Die Operation „Filzgleiter anstecken“ steckt einen Filzgleiter an ein Stahlrohr. Ein Tisch entsteht durch die Operation „Tisch zusammenbauen“. Die Operation „Tisch zusammenbauen“ schraubt mit 16 Schrauben vier Tischbeine an eine Tischplatte.

ArtikelName	ElterName	Menge	BomLevel
Tischplatte	Tisch	1.00	0
Tischbein	Tisch	4.00	0
Schrauben	Tisch	16.00	0
Anschraubplatte	Tischbein	1.00	1
Stahlrohr	Tischbein	1.00	1
Filzgleiter	Tischbein	1.00	1

Tabelle 2.1.: Hierarchische Stückliste des Tischbeispiels

[7] Vgl. Hollas, *Grundkurs Theoretische Informatik*, S. 35-41.
[8] Dudenverlag, *Elter, das oder der*.
[9] Vgl. Kurbel, *Enterprise Resource Planning und Supply Chain Management in der Industrie*, S. 49.

Das Tischbeispiel ist nicht komplex genug, um die zu entwickelnde ZPP in all ihren Funktionen zu testen. Daher nutzt der Autor die von Martin Krockert eingeführten Stücklisten des Dump-Trucks und Race-Trucks und deren übrige Masterdaten aus seiner Masterarbeit.[10] Die Stücklisten der Trucks referenziert der Autor als Trucks.

Ein gerichteter Wurzelbaum kann eine Stückliste hierarchisch darstellen. Jeder Knoten ist ein Artikel und die Kanten enthalten die Menge des Kinderknoten, die ein Elternknoten benötigt. Abb. 2.3 veranschaulicht die Stückliste des Tisch hierarchisch als gerichteten Wurzelbaum.

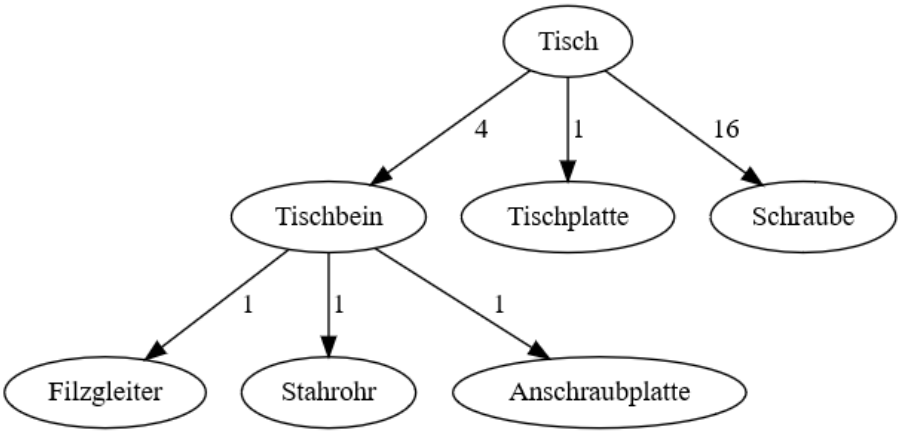

Abbildung 2.3.: Hierarchische Stückliste des Tischbeispiels als gerichteter Wurzelbaum

2.4. Beschreibung der Losgrößenbildung

Im Allgemeinen versucht man Bestellmengen und Fertigungsauftragsgrößen so zu berechnen, dass die beeinflussbaren Kosten minimiert werden (Optimierungsproblem). Eine Bestellmenge ist eine Losgröße der Beschaffung. Bei Einkaufsartikeln spricht man von „optimalen Bestellmengen“ , bei Produktionsartikeln von „optimalen Losgrößen“. Das Los (oder Fertigungslos) bezeichnet die Menge eines Artikels, die zusammenhängend produziert wird.[11] Der Autor verwendet ausschließlich den Begriff Losgröße für „optimale Bestellmengen“ und „optimale Losgrößen“. Diese Arbeit wendet die Losgröße ausschließlich auf die Erstellung von Lagereingängen an. Die Losgrößenbildung durch Berechnung der Losgröße wird hier nicht näher behandelt und soll gemäß Aufgabenstellung als konstant angenommen werden. Die konstanten Losgrößen stehen am Material.

2.5. Beschreibung der Materialbedarfsplanung

Die Materialbedarfsplanung (auch MRP I) ermittelt für einen bestimmten Primärbedarf den entstehenden Materialbedarf (= Sekundärbedarf).[12]

Aus dem Primärbedarf leitet sich der Sekundärbedarf ab. Der Sekundärbedarf beschreibt die Menge aller unmittelbar in das Erzeugnis bzw. in die Produkte des Primärbedarfs eingehenden Rohstoffe und Produkte. Der Tertiärbedarf beschreibt die Menge aller Werkstoffen wie Betriebsstoffe, Hilfsstoffe und Verschleißteilen, die indirekt bei der Produktion

[10] Krockert und Seifert, „Selbstorganisation oder zentrale Planung im Kontext von Industrie 4.0“.
[11] Vgl. Kurbel, *Enterprise Resource Planning und Supply Chain Management in der Industrie*, S. 79.
[12] Vgl. ebd., S. 2.

verbraucht werden.[13] Die Materialbedarfsplanung betrachtet ausschließlich Artikel, die in der Stückliste definiert sind, daher geht ein Primär-/Sekundär-/Tertiärbedarf nur in die Planung ein, sofern der Bedarf in der Stückliste definiert ist.

Die Bedarfsermittlung durch Traversierung von Stücklisten heißt Stücklistenauflösung.[14] Die Stücklistenauflösung des Primärbedarfs berechnet alle benötigten Mengen der Sekundär-/Tertiärbedarfe unter Beachtung der Losgröße. Außerdem muss die Materialbedarfsplanung die entsprechenden Lageraufträge, Produktionsaufträge und Bestellungen mit der ermittelten Menge erstellen. Durch das Anlegen von neuen Bedarfen (Lagereingang, Produktionsauftrag) und ihren Bedarfsdeckern (Bestellposition, Produktionsauftragsposition, Lagerausgang) entsteht ein gerichteter Bedarf-Bedarfsdecker-Graph. Der gerichtete Bedarf-Bedarfsdecker-Graphen ordnet jedem Bedarf einen Bedarfsdecker und ggf. bei Erstellung neuer Bedarfe jedem Bedarfsdecker Bedarfe zu. Auf einem Pfeil des Bedarf-Bedarfsdecker-Graphen steht die Menge. Die Wurzelknoten des Bedarf-Bedarfsdecker-Graphen sind Kundenauftragspositionen.

Abb. 2.4 zeigt ein Beispiel für einen allgemeinen Bedarf-Bedarfsdecker-Graphen und ein mögliches Ergebnis einer Materialbedarfsplanung ohne Mengendarstellung auf den Pfeilen. Die Abkürzungen D steht für Bedarf (engl. Demand) und P steht für Bedarfsdecker (engl. Provider). Der Betrachter sieht, dass D1 drei Provider zugeordnet sind (P1, P2 und P3). Jeder der Provider hat abhängige Demands zugeordnet (D2, D3 und D4). P4 stillt D2 und D3, P5 stillt D4.

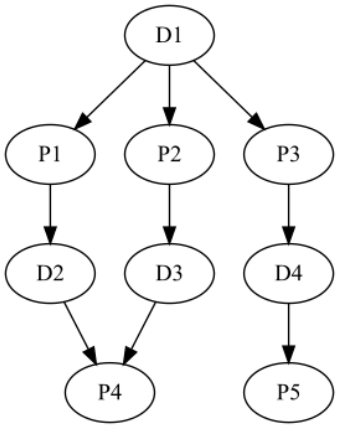

Abbildung 2.4.: Das Ergbnis einer Materialbedarfsplanung allgemein anhand von Bedarfen(D) und Bedarfsdeckern (P)

Nach der Durchführung der Materialbedarfsplanung sind die Mengen aller Bedarfe bekannt, alle Lageraufträge, Produktionsaufträge und Bestellungen erstellt. Ebenso ist der Bedarf-Bedarfsdecker-Graph erstellt.

[13] Vgl. Dangelmaier, *Theorie der Produktionsplanung und -steuerung*, S. 366.
[14] Vgl. Kurbel, *Enterprise Resource Planning und Supply Chain Management in der Industrie*, S. 84.

2.6. Beschreibung der Kapazitätsplanung gegen unbegrenzte Kapazität

2.6.1. Einführung

Nachdem die Materialbedarfsplanung die Mengen berechnet und Aufträge bzw. Operationen erstellt hat, müssen die erstellten Aufträge und im Fall von Produktionsaufträgen die angehängten Operationen terminiert, d.h. zeitlich festgelegt werden. Die Kapazitätsplanung gegen unbegrenzte Kapazität besteht im Wesentlichen aus einer Durchlaufterminierung. Die Durchlaufterminierung versieht Bestellungen, Operationen der Fertigungsaufträge und die Fertigungsaufträge selbst mit Start- und Endzeiten. Diese Arbeit abstrahiert ein Auftrag bzw. eine Operation als Knoten, daher haben Aufträge bzw. Operationen immer Start- und Endzeiten, auch wenn einige Auftragsarten wie ein Produktionsauftrag nur einen Endtermin haben, die Dauer des Auftrags ist dann null und Start- und Endzeit stimmen überein. Denn die Durchlaufterminierung behandelt alle Aufträge bzw. Operationen gleich und terminiert Start- und Endzeiten.

Die Kapazitätsplanung gegen unbegrenzte Kapazität berücksichtigt für die Durchlaufterminierung der Aufträge keine Kapazitäten, sie geht stattdessen von unbegrenzt verfügbaren Kapazitäten aus. Kapazitätsüberlastungen sollten dennoch selten auftreten, da die Durchlaufterminierung Übergangszeiten zwischen Operationen vorsieht. Die Übergangszeiten werden entweder geschätzt oder aus der Erfahrung vergangener Aufträge berechnet.[15] In dieser Arbeit sind die Übergangszeiten als konstant festgelegt und betragen das Dreifache der Operationsdauer.

Kapazitätsüberlast bedeutet, dass eine Maschine in einem Zeitabschnitt mehr Aufträge bekommt, als sie bearbeiten kann. Der Kapazitätsabgleich bezeichnet die Anpassung des Kapazitätsbedarfs an die gegebenen Kapazitätsgrenzen oder umgekehrt.[16] Der Kapazitätsabgleich ist unter den Bedingungen mehrstufiger Fertigung in der Praxis zu komplex; automatisierte Verfahren können nur unbefriedigende Ergebnisse liefern.[17] Daher behandelt diese Arbeit den Kapazitätsabgleich nicht, die Berücksichtigung von Übergangszeiten genügt für diese Arbeit.

2.6.2. Auftragsoperationsgraph

Die Durchlaufterminierung benötigt einen Auftragsoperationsgraphen, der alle Aufträge und Operationen des Bedarf-Bedarfsdecker-Graphen unter Berücksichtigung der Fertigungsreihenfolge enthält. Zur effizienteren Umsetzung ist der Auftragsoperationsgraph vollständig, d.h., er enthält sämtliche Kundenauftragspositionen und damit auch mehrere Wurzeln. Die Durchlaufterminierung legt die Start- und Endzeiten für die Aufträge und Operationen während der Traversierung des Auftragsoperationsgraphen fest.

Bei der ersten Nutzung des Auftragsoperationsgraphen müssen zunächst alle Start- und Endzeiten der Knoten des Auftragsoperationsgraphen gelöscht werden. Die Löschung der Start- und Endzeiten bei der Initialisierung ist wichtig, denn es kann Knoten geben, die mehrfach traversiert werden, z. B. ein Bedarfsdecker, der mehrere Bedarfe decken bzw. ein Bedarf, der mehrere Bedarfsdecker deckt.

[15] Vgl. Kurbel, *Enterprise Resource Planning und Supply Chain Management in der Industrie*, S. 113.
[16] Vgl. ebd., S. 128.
[17] Vgl. ebd., S. 131.

2.6.3. Rückwärtsterminierung

Ein Ziel in der Fertigung ist, dass alle Aufträge und Operationen so spät wie möglich gestartet werden, was die Kapitalbindung minimiert. Das Ziel, alle Aufträge und Operationen so spät wie möglich zu starten, erreicht man durch die Rückwärtsterminierung.[18] Die Rückwärtsterminierung startet vom Endzeitzeitpunkt des Auftrags auf der höchsten Fertigungsstufe, in unserem Fall dem Bedarfsdecker der Kundenauftragsposition (einer Wurzel des Auftragsoperationsgraphen). Von einer der Wurzel aus werden rückwärts durch Top-Down-Traversierung (Tiefensuche) des Auftragsoperationsgraphen alle Aufträge und Operationen zeitlich festgelegt (Start- und Endzeit).[19] Die Durchlaufterminierung muss die Rückwärtsterminierung ab allen Kundenauftragspositionen (Wurzeln) des Auftragsoperationsgraphen durchführen.

Damit Aufträge bzw. Operationen nicht zu früh starten, gilt: Setze die Endzeit eines Auftrags bzw. einer Operation auf den minimalen Startzeitpunkt der Elternknoten im Auftragsoperationsgraphen. Setze die Startzeit gleich der Endzeit minus der Dauer.

2.6.4. Vorwärtsterminierung

Die Rückwärtsterminierung kann negative Start- und Endzeiten verursachen. Aufträge bzw. Operationen dürfen niemals in der Vergangenheit starten, daher muss eine Vorwärtsterminierung vom Startzeitpunkt null erfolgen. Die Vorwärtsterminierung startet von den Blättern des Auftragsoperationsgraphen, z. B. Bestellpositionen oder Lagerausgängen. Von einem Blatt aus mit dem Zeitpunkt null legt die Durchlaufterminierung vorwärts durch Buttom-Up-Traversierung (Tiefensuche) des Auftragsoperationsgraphen alle Aufträge und Operationen zeitlich festgelegt (Start- und Endzeit).[20] Die Traversierung startet von allen Blätter des Auftragsoperationsgraphen, deren Startzeitpunkt negativ ist.

2.6.5. Zusammenspiel von Vorwärts- und Rückwärtsterminierung

Die Terminierung der Aufträge erfolgt durch das Schema Rückwärts-Vorwärts-Rückwärts, d.h., zunächst führt die Terminierung eine Rückwärtsterminierung, dann eine Vorwärtsterminierung und abschließend erneut eine Rückwärtsterminierung durch. So vergibt die erste Rückwärtsterminierung allen Aufträgen bzw. Operationen eine Start- und Endzeit. Die Startzeiten können in der Vergangenheit liegen, daher wird nach der ersten Rückwärtsterminierung vorwärts terminiert.

Nun sind die Startzeitpunkte der Bedarfsdecker der Kundenauftragsposition bekannt, d.h., wann frühestens die Kundenauftragspositionen bedient werden können. Eine zweite Rückwärtsterminierung startet bei den durch die Vorwärtsterminierung ermittelten Startterminen der Bedarfsdecker der Kundenauftragspositionen und maximiert so die Startzeitpunkte aller Aufträge. Nach der zweiten Rückwärtsterminierung startet kein Auftrag bzw. Operation in der Vergangenheit und alle Aufträge bzw. Operationen starten so spät wie möglich.

[18] Vgl. Kurbel, *Enterprise Resource Planning und Supply Chain Management in der Industrie*, S. 115.
[19] Vgl. ebd., S. 113-114.
[20] Vgl. ebd., S. 115.

2.7. Beschreibung der Kapazitätsplanung gegen begrenzte Kapazität

Nachdem die Kapazitätsplanung gegen unbegrenzte Kapazität die Aufträge und Operationen terminiert hat, muss die ZPP den Operationen Start- und Endzeiten unter Berücksichtigung der Maschinenkapazität zuweisen. Die Kapazitätsplanung gegen begrenzte Kapazität, auch Kapazitätsterminierung, Maschinenbelegung oder Job shop scheduling genannt, weist den Operationen Maschinen, Start- und Endzeiten zu und bestimmt so die Reihenfolge, in der die Operationen auf einer Maschine nacheinander abgearbeitet werden.[21] Diese Arbeit beschränkt sich auf den Fall der Produktion mit Werkstattfertigung (Job shop). Der Job-Shop-Fall ist die Planung der Reihenfolge von n Fertigungsaufträgen auf m Maschinen. Für jeden Fertigungsauftrag sind o Operationen in einer bestimmten Reifenfolge durchzuführen. Jede Operation eines Fertigungsauftrags muss auf einer anderen Maschine ausgeführt werden. Jede Maschine kann zu einem Zeitpunkt nur eine Operation durchführen und nach Start einer Operation nicht unterbrochen werden. Jede Operation hat eine Bearbeitungszeit.[22]

Etwaige Kapazitätsüberlastungen können nicht mehr auftreten. Die in den Durchlaufterminierungen festgelegten Zeiten der Operationen werden durch die Kapazitätsterminierung entsprechend zeitlich in die Zukunft verschoben. Eingeplante Übergangszeiten werden ignoriert, denn Übergangszeiten dienen zur Vermeidung von Kapazitätsüberlast und sind somit nur Pufferzeiten.

Die Maschinenbelegung ist ein NP-schweres Optimierungsproblem.[23] Folglich sind selbst die besten Algorithmen bei mehr als 20 Maschinen und 20 Jobs sehr schnell überfordert, daher verwendet man heuristische Verfahren. Eine Heuristik ist eine eindeutig definierte Folge von Rechenoperationen, die eine gute Lösung einer komplexen Aufgabe ermittelt. Das Ergebnis der Heuristik muss kein Optimum sein und man kann keine Aussage darüber treffen, wie weit das Ergebnis vom Optimum entfernt ist.[24]

In dieser Arbeit beschränkt sich der Autor für die Kapazitätsterminierung auf den Giffler-Thompson-Algorithmus als heuristisches Verfahren. Der Giffler-Thompson-Algorithmus ist ein Eröffnungsverfahren, das einen aktiven Ablaufplan ermittelt. Ein Ablaufplan heißt aktiv, wenn man den Start einer beliebigen Operation nicht vorverlegen kann, ohne eine andere Operation zu verzögern.[25] Ein Eröffnungsverfahren erzeugt eine zulässige Lösung auf direktem Weg ohne Umwege über andere zulässige Lösungen zu gehen und dient oft als Ausgangspunkt für (sub-)optimierende Iterationsverfahren, die in dieser Arbeit nicht behandelt werden.[26] Der Giffler-Thompson-Algorithmus ist ein allgemeines Schema für ein Prioritätsregelverfahren der Maschinenbelegung. Der Algorithmus erstellt eine Konfliktmenge mit Operationen auf der gleichen Maschine innerhalb eines Zeitintervalls. Die Prioritätsregel ermittelt die nächste auszuführende Operation aus der Konfliktmenge anhand der höchsten Priorität. In dieser Arbeit beschränkt sich der Autor auf die Prioritätsregel „Kürzeste-Schlupfzeit-Regel“. Die Kürzeste-Schlupfzeit-Regel vergibt die höchste Priorität an eine Operation aus einer Menge von Operationen, bei der die Differenz zwischen dem Liefertermin und der verbleibenden Bearbeitungszeit (Schlupf) am kleinsten ist.[27]

[21] Vgl. Kurbel, *Enterprise Resource Planning und Supply Chain Management in der Industrie*, S. 137.
[22] Vgl. Zäpfel, *Moderne Heuristiken der Produktionsplanung*, S. 5.
[23] Vgl. ebd., S. 21.
[24] Vgl. ebd., S. 22.
[25] Vgl. ebd., S. 29.
[26] Vgl. ebd., S. 23.
[27] Vgl. ebd., S. 36.

Für die Anwendung des Giffler-Thompson Algorithmus braucht man einen gerichteten Graphen, der alle Produktionsaufträge und deren Operationen entsprechend ihrer Fertigungsreihenfolge ohne die Produktionsaufträge selbst enthält, der Graph soll in der Arbeit Operationsgraph heißen. Den Operationsgraphen kann man aus dem Auftragsoperationsgraphen der Durchlaufterminierungen wie folgt erstellen. Alle Knoten, die keine Operationen sind, löschen, bei jeder Löschoperation verbindet man die Eltern des Löschknoten mit den Kindern des Löschknoten. Die Blätter des Operationsgraphen sind die Operationen, die einplanbar sind, d.h., die zuerst bearbeitet werden können.

Ein Gantt-Diagramm kann das Ergebnis der Maschinenbelegung darstellen. Ein Gantt-Diagramm visualisiert für jede Maschine die zugeordneten Operationen mit ihren Bearbeitungszeiten.[28]

2.8. Beschreibung der Rückmeldungen und Neuplanung

Der MRP-Lauf ist letztendlich ein großer Algorithmus, der die Realisierung von Kundenaufträgen in Form von Produktionsaufträgen und Bestellungen plant. Um den Algorithmus ist eine Simulation gebettet. Ein Simulationslauf umfasst zunächst die Erstellung von Kundenaufträgen, das Starten des MRP-Laufs und die Erstellung und Anwendung von Rückmeldungen. Die Simulation führt den Simulationslauf entsprechend häufig aus. Rückmeldungen erfolgen für ein bestimmtes in der Vergangenheit liegendes Zeitintervall. Eine Rückmeldung ist eine Meldung aus der Realität zurück an das Softwaresystem, wann der Start bzw. das Ende eines Auftrags bzw. Operation war.[29] Außerdem muss der Status der Aufträge bzw. Operationen angepasst werden. Ein Status eines Auftrags bzw. einer Operation kann „Erstellt“, „In Bearbeitung“ oder „Beendet“ sein.

Im Rahmen dieser Arbeit werden die Rückmeldungen simuliert. Nachdem die Rückmeldungen in Form von Start- und Endzeiten eingespielt und die Status der Entitäten angepasst sind, müssen die erstellten Rückmeldungen auf die Transaktionsdaten der Planung angewandt werden. Am Ende müssen die Transaktionsdaten persistiert werden. Der Simulationslauf endet. Danach findet eine Neuplanung im Rahmen des nächsten Simulationslaufs statt. Die Neuplanung ist die erneute Ausführung eines MRP-Laufs (MRP II).

2.9. Beschreibung der Objektgymnastik

Im Software-Engineering gibt es u.a. folgende theoretische Konzepte der objektorientierten Programmierung, die einen guten Softwareentwurf beschreiben:

- Kohäsion,
- Lose Kopplung,
- Keine Codeduplikate,
- Datenkapselung,
- Testbarkeit,
- und Lesbarkeit.

[28] Vgl. Zäpfel, *Moderne Heuristiken der Produktionsplanung*, S. 7.
[29] Vgl. Kurbel, *Enterprise Resource Planning und Supply Chain Management in der Industrie*, S. 145.

Die Umsetzung der Konzepte in der Praxis ist im Allgemeinen schwierig. Jeff Bay hat daher das Konzept Objektgymnastik entwickelt, das die folgenden neun Regeln umfasst.[30] Das englischsprachige Original befindet sich in Klammern.

1. Verwende höchstens eine Einrückungsebene pro Methode. (Use only one level of indentation per method.)

   Stattdessen sollen weitere Ebenen in neue Methoden ausgelagert werden. Die innere „For"-Schleife einer verkettete „For"-Schleife ersetzt Bay durch eine andere Methode mit aussagekräftigen Namen und ruft die Methode in der „For"-Schleife auf. Der aussagekräftige Methodenname ersetzt den Beschreibungskommentar für die innere Schleife. Die Regel vermeidet komplexe Schleifen innerhalb einer Methode.

2. Nutze nicht das „Else"-Schlüsselwort. (Don't use the else keyword.)

   Das Problem ist, dass eine bedingte Anweisung und Verzweigung (conditional) beliebig durch weitere Zweige erweitert werden kann und die weiteren Zweige enthalten häufig Codeduplikate und Statusflags. Statt dem „Else" kommt ein frühes „Return". Das Strategiemuster kann die Anzahl der Fälle reduzieren. Beim Strategiemuster siehe Abb. 2.5 implementiert der Codeabschnitt den einzelnen Algorithmus nicht direkt, sondern nutzt ein entsprechendes Interface und führt zur Laufzeit die entsprechende Strategie aus. Die Regel der Objektgymnastik verhindert komplexe Conditionals innerhalb einer Methode.

3. Alle Primitive und Zeichenketten kapseln. (Wrap all primitives and strings.)

   Wenn eine Methode ein Primitiv wie ein Integer (meist „int") als Parameter hat, muss der Name des Parameters vollständig beschreiben, was er beinhalten soll. Das kann mitunter zu langen Variablennamen führen. Außerdem können falsche Werte übergeben werden z. B. Stunde statt Minute. Die Kapselung in einer Klasse beseitigt die Probleme und ermöglicht Methoden auf dem Typ.

4. Nutze nur einen Punkt pro Anweisung (statement). (Use only one dot per line.)

   Mit Punkt ist der Zugriffsoperator in der objektorientierten Programmierung gemeint. Mehrere Punkte weisen daraufhin, dass die Anweisung zu tief in das Objekt bzw. seine verbundenen Objekte greift, was gegen das Konzept der Datenkapselung verstößt. Wenn die Klasse ein Wrapper ist, dann sind zwei Punkte nötig, da der erste Punkt auf die Instanzvariable der gewrappten Klasse zugreift.

5. Bezeichner nicht abkürzen. (Don't abbreviate)

   Abkürzungen erleichterten früher das Tippen, in Zeiten von integrierten Entwicklungsumgebungen mit Autovervollständigung ist das kein Grund für Abkürzungen mehr. Jede Abkürzung muss definiert werden und von Erstlesern oft nachgeschlagen werden, was die Produktivität beeinträchtigt.

6. Alle Entitäten kleinhalten. (Keep all entities small.)

   Kleinhalten bedeutet, dass ein Paket nicht mehr als 10 Dateien und eine Klasse nicht mehr als 50 Zeilen enthalten sollte. Statt alle zusammengehörigen Methoden in einer großen Klasse zu implementieren, sollte ein Paket die zusammenpassende Logik über Klassen verteilt enthalten.

[30] Vgl. ThoughtWorks, *ThoughtWorks Anthology: Essays on Software Technology and Innovation*, S. 71.

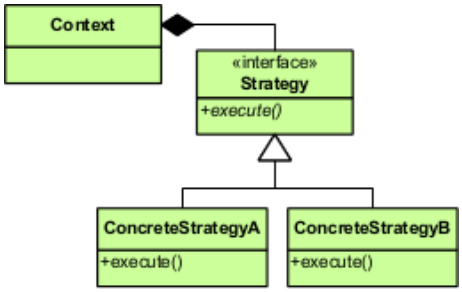

Abbildung 2.5.: Strategiemuster als Klassendiagramm dargestellt Quelle: McDonald, *Design Patterns Quick Reference*, Seite 1

7. Eine Klasse soll höchstens zwei Instanzvariablen verwalten. (Don't use any classes with more than two instance variables.)

   Im Allgemeinen sollte ein Objekt für die Handhabung eines Zustandes zuständig sein. Weitere Instanzen würden die Kohäsion verringern. Es gibt vor allem zwei Arten von Klassen, die eine Art verwaltet einen Objektzustand und die andere koordiniert zwei Objektinstanzen.

8. Container (Collection) als Instanzvariable in eine eigene Klasse auslagern. (Use first-class collections.)

   Jede Klasse, die einen Container als Instanzvariable enthält, sollte keine anderen Instanzvariablen haben. Wenn jeder Container seine eigene Klasse hat, kann Container bezogenes Verhalten dort seinen Platz haben.

9. Nutze keine „Getters", „Setters" und Instanzvariable außerhalb der Klasse. (Don't use any getters/setters/properties.)

   Ansonsten verstößt die Nutzung von „Gettern", „Settern" gegen die Datenkapselung. Das Verhalten für Instanzvariablen sollte an genau einem Ort, nämlich die Klasse der Instanzvariable, implementiert werden. Die Regel vermeidet Codeduplikate und Fehler müssen nur noch an genau einem Ort gesucht werden. Neue Funktionen und Änderungen werden ebenfalls genau dort vorgenommen. Sobald ein Attribut zweier Objekte miteinander verglichen werden soll, muss zumindest ein Getter in einer der beiden Klassen zu dem Attribut vorhanden sein. In dem Fall lässt sich die Regel folglich nicht anwenden.

Jeff Bay weißt ausdrücklich daraufhin, dass einige der Regeln nicht immer einsetzbar seien, es sich lohnen würde, darüber nachzudenken und es zu versuchen. Die Regeln sind als Faustregel zu verstehen.[31]

[31] Vgl. ThoughtWorks, *ThoughtWorks Anthology: Essays on Software Technology and Innovation*, S. 72.

3. Analyse der funktionalen Anforderungen an ein Softwaresystem zur zentralen Produktionsplanung

3.1. Anforderung an die Materialbedarfsplanung

Die zentrale Produktionsplanung muss die in der Datenbank hinterlegte Stückliste auflösen können. Die ZPP muss eine Produktionsauftragsanlage besitzen, die es ermöglicht, Produktionsaufträge aufzugeben. Die ZPP muss eine Bestellungsanlage haben, die es ermöglicht, Bestellungen aufzugeben.

Die Losgröße muss konstant sein und am Material in der Datenbank hinterlegt sein. Eine Schnittstelle für die Offline-Losgrößenbildung muss vorgesehen sein. Offline bedeutet, dass die Losgrößenbildung nach der Erstellung des Produktionsauftrags- und Bestellnetzes ermöglicht werden muss. Eine Schnittstelle für die Online-Losgrößenbildung muss vorgesehen sein. Online bedeutet, dass die Losgrößenbildung während der Stücklistenauflösung und der Rückwärtsterminierung ermöglicht werden muss.

3.2. Anforderung an die Kapazitätsplanung gegen unbegrenzte Kapazität

Die ZPP muss eine Rückwärtsterminierung durchführen. Die Rückwärtsterminierung kann während der Stücklistenauflösung (Online) stattfinden. Die ZPP muss eine Vorwärtsterminierung durchführen, sofern die Rückwärtsterminierung den aktuellen Zeitpunkt unterschreitet. Für die Vorwärtsterminierung muss die ZPP den Start aller Produktionsaufträge und Bestellungen so spät wie möglich festlegen. Es muss zuerst rückwärts, danach vorwärts und abschließend erneut rückwärts terminiert werden. Die Angabe der Start- und Endzeiten der zentralen Produktionsplanung ist in Minuten. Die Übergangszeit einer Operation muss vor dem Start einer Operation liegen und das Dreifache der Operationsdauer betragen. Die ZPP kann einen automatischen Kapazitätsabgleich aufgrund des „Abstreifermodells“ durchführen.

3.3. Anforderung an die Kapazitätsplanung gegen begrenzte Kapazität

Die ZPP muss eine Kapazitätsterminierung mittels des Giffler-Thompson-Algorithmus und Prioritätsregeln durchführen. Die Kapazitätsterminierung ist eine Kapazitätsplanung gegen begrenzte Kapazität. Kapazitätsterminierung muss für mehrere Maschinen pro Maschinentyp ausgelegt sein. Für die Kapazitätsterminierung muss als Standardprioritätsregel die Schlupfzeitregel genutzt werden.

3.4. Anforderung an die Neuplanung unter Verrechnung von Rückmeldungen

Die ZPP muss eine Neuplanung unter Verrechnung von Rückmeldungen beherrschen. Die Simulation muss die Rückmeldungen erstellen und anwenden. Der MRP-Lauf muss unab-

hängig von den Rückmeldungen sein.

4. Analyse der nicht funktionalen Anforderungen an ein Softwaresystem zur zentralen Produktionsplanung

4.1. Anforderung an die Datenstrukturen

Im Zuge der Masterarbeit „Selbstorganisation oder zentrale Planung im Kontext von Industrie 4.0“ ist ein umfangreiches Datenmodell der SSOP entstanden und dort in Anforderung und Umsetzung ausführlich beschrieben.[1] Die zentrale Produktionsplanung muss auf denselben Datenstrukturen wie die SSOP basieren.

4.2. Anforderung an die Integration in das vorhandene Projekt der selbstorganisierenden Produktion

Um die zentrale Produktionsplanung mit der SSOP vergleichen und kombinieren zu können, muss die zentrale Produktionsplanung in das vorhandene Projekt der SSOP integriert werden können.

4.3. Anforderung an das Simulationsmodell

Die Untersuchung der SSOP hinsichtlich ihrer Qualität und ihrer Performanz erfolgt anhand eines parametrisierbaren ereignisdiskreten Simulationsmodells einer Produktion. Damit die SSOP mit der zentralen Produktionsplanung verglichen werden kann, muss die zentrale Produktionsplanung dem genannten Simulationsmodell aufsetzen. Das Simulationsmodell ist ausführlich beschrieben.[2]

4.4. Anforderung an die Performanz

Die ZPP muss performant sein. Die Performance eines Programms umfasst die Laufzeit und den Speicherbedarf. Die zentrale Produktionsplanung muss für empirische Studien viele hunderte oder sogar tausende Male durchgeführt werden. Daher muss die zentrale Produktionsplanung eine Laufzeit aufweisen, sodass die ZPP mindestens 500 Kundenaufträge innerhalb von zwei Stunden planen kann. Das Ziel ist, dass die ZPP ein lineares Verhalten sowohl für die Laufzeit als auch für den Speicherbedarf bezogen auf die Anzahl an Kundenaufträgen aufweist. Das bedeutet, dass mit wachsender Anzahl an Kundenaufträgen die Laufzeit und der Speicherbedarf höchstens linear wächst.

[1] Vgl. Krockert und Seifert, „Selbstorganisation oder zentrale Planung im Kontext von Industrie 4.0“, S. 41-52.

[2] Vgl. ebd., S. 51-55.

4.5. Anforderung an die Struktur des Quelltextes: Objektgymnastik (Object Calisthenics)

Ein wesentlicher Grund für die erneute Implementierung der ZPP ist, abgesehen von der Performance, die Struktur. Der Quelltext muss sehr klar strukturiert, damit er gut wartbar, wiederverwendbar und erweiterbar ist. Ein externer Unbeteiligter, der mit dem MRP-Lauf vertraut ist, muss sich umgehend zurecht finden können. Um die Anforderung umzusetzen, muss im Rahmen des Entwurfs und der Implementierung den Regeln der Object Calisthenics (Objektgymnastik) von Jeff Bay gefolgt werden.

4.6. Anforderung an die Tests

Der Test der ZPP muss die korrekte Funktionsweise der ZPP sicherstellen. Der Test auf die Performance der ZPP muss als empirische Studie durchgeführt werden. Beide Tests müssen innerhalb des Zeitintervalls Null und 20160 erfolgen, das muss auf zwei Arten geschehen. Im ersten Szenario muss die ZPP jeden Tag einzeln planen. Im zweiten Szenario muss die ZPP den gesamten Zeitraum planen.

5. Umsetzung der zentralen Produktionsplanung

5.1. Datenstrukturen der zentralen Produktionsplanung

Die SSOP gibt grundsätzlich das Datenmodell vor und beschreibt das Datenmodell bereits umfangreich.[1] Die Transaktionsdaten des Datenmodells beinhalten die Ergebnisse der Planung. Die Masterdaten des Datenmodells umfassen die Stückliste, die Artikel, Einheiten, Maschinen, Kunden und Beziehungen zwischen Kunden und Artikeln. Jedoch ändert bzw. erweitert sich das Datenmodell teilweise im Zuge der Entwicklung der zentralen Produktionsplanung.

Die folgende Auflistung führt die Entitäten der Transaktionsdaten ein.

- Ein CustomerOrder ist ein Kundenauftrag.
- Ein CustomerOrderPart ist eine Kundenauftragsposition.
- Eine ProductionOrder ist ein Produktionsauftrag.
- Eine ProductionOrderBom ist eine Produktionsauftragsstücklistenposition.
- Eine ProductionOrderOperation ist eine Auftragsoperation (konkrete Ausprägung einer M_Operation), die genau einer ProductionOrder und mindestens einer ProductionOrderBom zugeordnet ist.
- Eine PurchaseOrder ist ein Bestellung.
- Ein PurchaseOrderPart ist eine Bestellposition.
- Ein StockExchange ist eine Lagertransaktion, eine Lagertransaktion kann sowohl eine Lagerabgang, als auch ein Lagerzugang sein.
- Ein Demand ist ein Bedarf. Ein Demand kann ein CustomerOrderPart (COP), eine ProductionOrderBom (PrOB) oder StockExchange (SE) sein.
- Ein Provider ist ein Bedarfsdecker. Ein Provider kann ein PurchaseOrderPart (PuOP), eine ProductionOrder (PrO) oder ein StockExchange (SE) sein.

Jeder der eingeführten Begriffe beschreibt eine Entität in der Datenbank und hat somit eine Tabelle. Da alle genannten Entitäten zu den Bewegungsdaten gehören, haben deren Tabellennamen den Präfix „T_“. Die T_*-Tabellen bezeichnen also die Transaktionsdaten der ZPP. Die Schreibung der genannten Entitäten erfolgt in dieser Arbeit in Binnenmajuskel (eng. camel case). Um den Typ eines StockExchanges im folgenden klar unterscheiden zu können, führt der Autor den Begriff StockExchangeDemand und StockExchangeProvider ein. Ein StockExchangeDemand ist ein StockExchange vom Typ Lagerzugang, ein StockExchangeProvider ist ein StockExchange vom Typ Lagerabgang.

Im folgenden werden die Änderungen bzw. Erweiterungen an den Tabellen der Transaktionsdaten beschrieben:

[1] Vgl. Krockert und Seifert, „Selbstorganisation oder zentrale Planung im Kontext von Industrie 4.0“, S. 38-51.

- Die Tabelle DemandToProvider erweitert das Datenmodell um die Verknüpfung zwischen Demands und Providern. Ein Demand kann auf n Provider zeigen. In einem gerichteten Graphen ist eine Tabellenzeile die Kante von einem Demand-Knoten zu einem Provider-Knoten. Die Tabelle DemandToProvider ersetzt die Tabelle Demands.
- Die Tabelle ProviderToDemand erweitert das Datenmodell um die Verknüpfung zwischen Providern und Demands. Ein Provider kann auf n Demands zeigen. In einem gerichteten Graphen ist eine Tabellenzeile die Kante von einem Provider-Knoten zu einem Demand-Knoten.
- Jede ProductionOrderBom hat genau eine Operation. Eine ProductionOrderOperation kann mehreren ProductionOrderBoms zugeordnet werden.
- Sämtliche Tabellennamen bekommen einen Präfix „T_“ (engl. für transaction data) bzw. „M_“ (engl. für master data) um die dazugehörigen Klassen einfach nach Stamm und Bewegungsdaten unterscheiden zu können.
- Alle Tabellennamen im Plural sind jetzt im Singular, z. B. die Tabelle Stockexchanges heißt jetzt StockExchange.
- Die Tabelle Purchases heißt jetzt T_PurchaseOrder.
- Die Tabelle WorkSchedules heißt jetzt M_Operation.
- Die Tabelle PurchaseParts heißt jetzt T_PurchaseOrderPart.
- Die Tabelle ProductionOrderWorkSchedules heißt jetzt T_ProductionOrderOperation.
- Die Tabelle Orders heißt jetzt CustomerOrder.
- Die Tabelle OrderParts heißt jetzt CustomerOrderPart.
- Die Simulationstabellen Kpis, SimulationConfigurations, SimulationOrders und SimulationWorkschedules sind jetzt in einer gesonderten Datenbank für die Simulation.

Der Autor entfernte im Laufe der Umsetzung die Tabellen T_Demand und T_Provider aufgrund des massiven Verwaltungsaufwandes, denn für jeden Demand (CustomerOrderPart, ProductionOrderBom, StockExchangeDemand) und Provider (PurchaseOrderPart, ProductionOrder, StockExchangeProvider) muss zusätzlich ein zu verwaltender T_Demand bzw. T_Provider der Datenbank erstellt werden. Je nach Kontext muss zwischen einem T_Demand bzw. T_Provider und seiner Ausprägung konvertiert werden. Ein weiteres Problem ist, dass jeder Provider bzw. Demand zwei Ids hat (T_Demand bzw. T_Provider und seine spezielle Ausprägung), somit bietet eine Demand- bzw. Provider-Id nur noch eine kontextabhängige eindeutige Identifizierung.

Eine aktualisierte Variante des Datenbankschemas befindet sich im Anhang A. Im übrigen Teil der Arbeit nutzt der Autor die Begriffe des Datenmodells, um sämtliche Sachverhalte übersichtlich und exakt beschreiben zu können.

5.2. Eingangs- und Ausgangsdaten der zentralen Produktionsplanung

Dieses Kapitel beschreibt die Eingangsdaten, die notwendig sind, um die ZPP auszuführen und die Ausgangsdaten, die das Resultat der Ausführung sind. Die Eingangsdaten sind in einer Konfigurationsdatei in JSON-Notation hinterlegt. Die JavaScript-Object-Notation (JSON) ist ein Textformat für die Serialisierung von strukturierten Daten.[2] Der Anwender muss beim Start der ZPP seine Konfigurationsdatei angegeben. Der Anwender kann jedes Szenario in einer Konfigurationsdatei abbilden. Die Eingangsdaten (ein Szenario) umfassen folgende Parameter:

- CustomerOrderQuantity: Der Parameter legt die Anzahl an Kundenaufträgen fest, die die ZPP in einem Intervall angelegt soll.
- LotSize: Der Parameter legt die Losgröße für die Produkte fest, deren Losgröße nicht am Material steht. Artikel ohne Losgröße sind im Allgemeinen Produktionsartikel.
- Name: Der Zeichenkettenparameter legt den Namen fest, den die ZPP für die Erstellung von Logdateien oder szenariospezifischen Konsolenausgaben nutzt.
- DbSetInitializer: Der Parameter legt die Klasse und dessen Assembly fest, die die Initialisierung der Datenbank umsetzt. Dazu gehört das Anlegen der Masterdaten.
- SimulationMaximumDuration: Der Parameter legt die maximale Dauer der Simulation in Minuten fest, z. B. 20160 für 14 24-Stunden-Tage.
- SimulationInterval: Der Parameter legt die Dauer eines Intervalls in Minuten fest, z. B. 1440 für einen 24-Stunden-Tag.

Die Ausgangsdaten sind die Transaktionsdaten der ZPP, die sich in den T_*-Tabellen befinden. Da es zwei Varianten der Datenbank gibt (mit und ohne „_archive“), sind alle abgeschlossenen Entitäten im Transaktionsdatenbankarchiv zu finden, alle noch offenen Entitäten befinden sich in den Transaktionsdatenbank.

5.3. Umsetzung des MRP-Laufs

Der MRP-Lauf ist letztendlich ein großer Algorithmus, der die Realisierung von Kundenaufträgen in Form von Produktionsaufträgen und Bestellungen plant. Um den Algorithmus ist eine Simulation gebettet. Ein Simulationslauf umfasst zunächst die Erstellung von Kundenaufträgen, das Starten des MRP-Laufs und die abschließende Simulation und Anwendung von Rückmeldungen. Die Simulation führt den Simulationslauf entsprechend häufig aus. Ein Simulationslauf ist im folgenden ein Zyklus. Die Erstellung der Kundenaufträge erfolgt analog dem Simulationsmodell der SSOP durch Nutzung einer Funktion des Pakets „SimulationCore“ der SSOP.

Die Abb. 5.3 zeigt den Algorithmus der ZPP. Bei der Beschreibung von Algorithmen verzichtet der Autor darauf, Schlüsselwörter wie „While“ und Variablennamen in Anführungszeichen zu schreiben. Zunächst berechnet der Algorithmus die Anzahl zu simulierender Zyklen n. Die Variable n berechnet sich aus der Division von SimulationMaximumDuration durch SimulationInterval. Dann führt der Algorithmus n Zyklen über eine Schleife aus. In

[2] Vgl. IETF, *The JavaScript Object Notation (JSON) Data Interchange Format*, S. 3.

jedem Zyklus erstellt der Algorithmus zunächst die konfigurierte Anzahl (CustomerOrder-Quantity) an CustomerOrders. Dann führt der Algorithmus den MRP-Lauf aus (MRP2). Dann simuliert der Algorithmus die Rückmeldungen. Zum Abschluss des Zyklus wendet der Algorithmus die Rückmeldungen an. Der Zyklus endet.

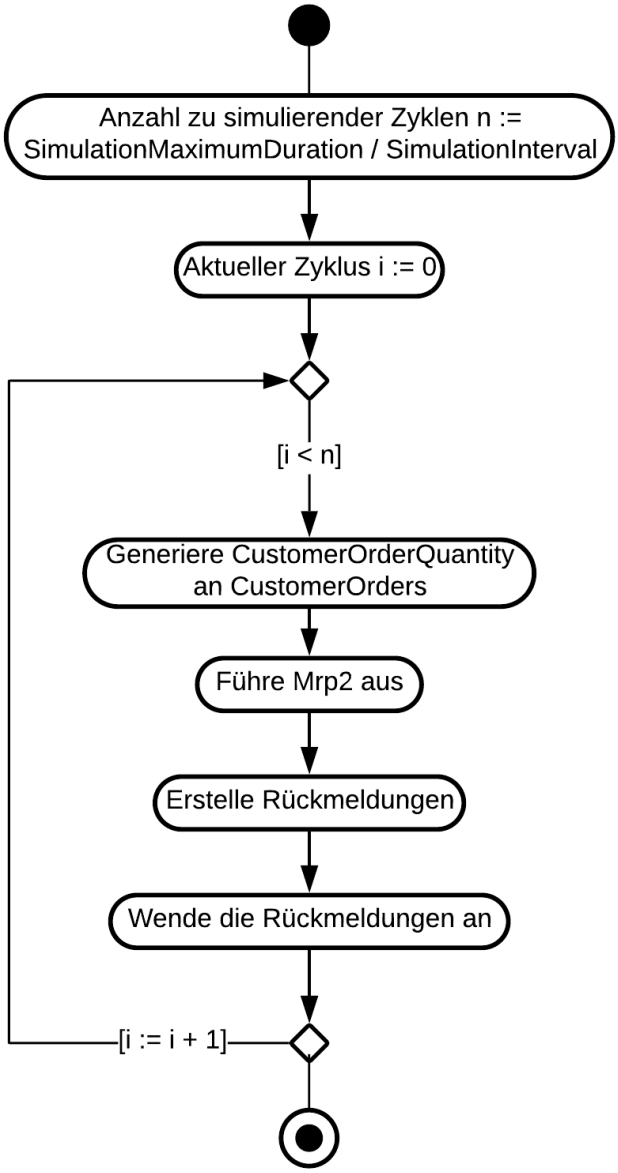

Abbildung 5.1.: Algorithmus der ZPP als Aktivitätsdiagramm der UML 2.0.

5.4. Architektur und Grobentwurf der zentralen Produktionsplanung

Um den Anforderungen an die Struktur des Quelltextes auch auf oberster Ebene zu genügen, ist eine verständliche Architektur unabdingbar. Das Primärziel ist folglich die lose Kopplung und die Abstraktion der Implementierung bis auf die Ebene des MRP-Laufs.

So kann sich ein Kenner des MRP-Lauf umgehend in der ZPP zurechtfinden. Die lose Kopplung ist so umgesetzt, dass jeder Algorithmusschritt des MRP-Laufs von einem Paket abgebildet ist. Falls ein Schritt des MRP-Lauf umfangreicher ist, ist ein Schritt ebenfalls in einem Subpaket abgebildet.

Die Abb. 5.2 zeigt die grobe Architektur der ZPP als Component/Block Diagramm. Ein Component/Block Diagramm beschreibt die statische Struktur eines Software-Systems und bietet einen konzeptionellen Überblick auf die Architektur des Systems.[3] Der Nutzer startet die Komponente ZppSimulator, der ZppSimulator führt den im vorherigen Kapitel beschriebenen Algorithmus aus. Aufgrund einer Software-Einschränkung des Diagrammzeichners ist rechts oben im Kasten der ZppSimulator-Komponente das Komponentensymbol nicht sichtbar. Die Agenten CustomerCreator, Mrp2, ConfirmationCreator greifen lesend und schreibend auf die Datenbank Transaktionsdaten zu. Der Agent ConfirmationAppliance greift lesend und schreibend auf die Datenbank Transaktionsdaten und auf die Datenbank Transaktionsdatenarchiv zu, was nötig ist, um abgeschlossene Transaktionsdaten zu archivieren. Nur der Agent Mrp2 liest Masterdaten. Wie aus den Storages ersichtlich sind die beiden Storages Transaktionsdata und Masterdata, also alle drei Datenbanken, gecached.

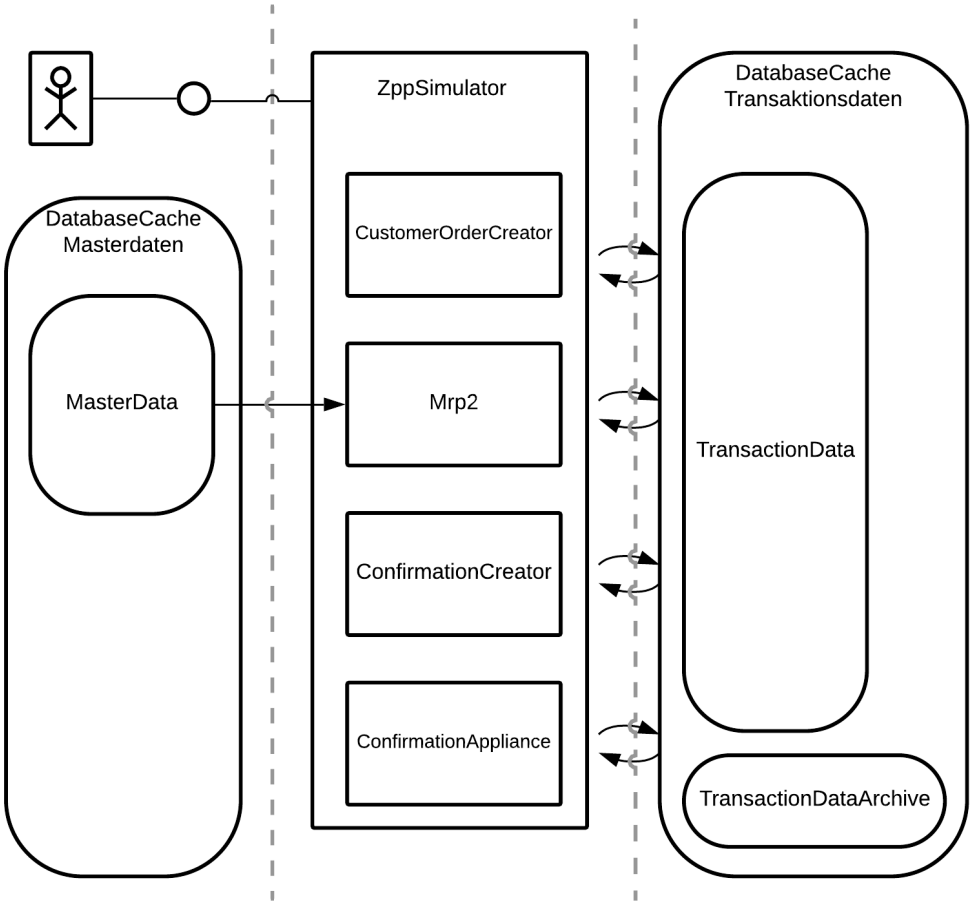

Abbildung 5.2.: Statische Struktur der ZPP als Component/Block Diagramm (TAM)

Die Abb. 5.3 zeigt die detaillierte Architektur der ZPP als Paketdiagramm der UML

[3] Vgl. SE, *Standardized Technical Architecture Modeling 1.0*, S. 5.

2.0. In C# repräsentiert jedes Paket einen Namespace und gruppiert Klassen.[4] Zunächst gibt es das Paket Test, das grundsätzlich auf jedes oberste Paket der Hierarchie zugreifen muss, um alles testen zu können. Der ZppSimulator ist das Herzstück der ZPP. Der ZppSimulator führt zyklisch den OrderGenerator, den MRP-Lauf (MRP2), die Erstellung der Rückmeldungen (ConfirmationCreator im Paket ConfirmationManager) und die Anwendung der Rückmeldungen (ConfirmationAppliance im Paket ConfirmationManager) aus. Der OrderGenerator ist Bestandteil der SSOP und generiert Kundenaufträge.

Der MRP-Lauf führt die Materialbedarfsplanung (MRP1), die Rückwärts-Vorwärts-Rückwärtsterminierung (BackwardsScheduler, ForwardScheduler im Paket Scheduler) und die Maschinenbelegung aus (JobShopScheduler im Paket Scheduler). Die Materialbedarfsplanung erstellt Provider für die Bedarfe und abhängige Demands der Provider mit dem ProviderManager. Der ProviderManager erstellt entsprechend die nötigen Entitäten und Zuordnungen mit dem Production-, Purchase- und Stockmanager.

Alle Pakete nutzen grundsätzlich den DataLayer. Der CacheManager steuert den Zugriff auf den DataLayer und ist ein Singelton. Allerdings steuert der CacheManager nicht den Zugriff auf die im DataLayer liegenden Wrapper der Datenbankentitäten und deren Collectionwrapper. Nur die Tests und Debuggingmethoden nutzen die graphischen Repräsentationen im Paket GraphicalRepresentation. Der Autor nutzt zwei Repräsentationsmöglichkeiten für Graphen in dieser Arbeit, zum einen die Graphenvisualisierung mittels Graphviz, zum anderen Ganttcharts.

Die Klasse „Graphviz“ erstellt die Graphenrepräsentationen in der Beschreibungssprache „DOT“. Graphviz-Layout-Programme nehmen Graphen, beschrieben durch Beschreibungssprache „DOT“, und erstellen daraus Diagramme u. a. im SVG-Format.[5][6] In dieser Arbeit verwendet der Autor eine Webseite von Marcin Stefaniuk, um die DOT-Darstellung in eine graphische Repräsentation im SVG-Format zu erstellen.[7] Die Klasse „GanttChart“ erstellt Ganttchart-Repräsentationen im JSON-Format, der Autor erstellt daraus eine graphische Repräsentation mit einer Webseite von Martin Krockert.[8]

[4] Vgl. Fowler, *UML Distilled*, S. 89.
[5] graphviz.org, *The DOT Language*.
[6] graphviz.org, *About Graph Visualization*.
[7] Stefaniuk, *Graphviz it*.
[8] Krockert, *Gantt live view*.

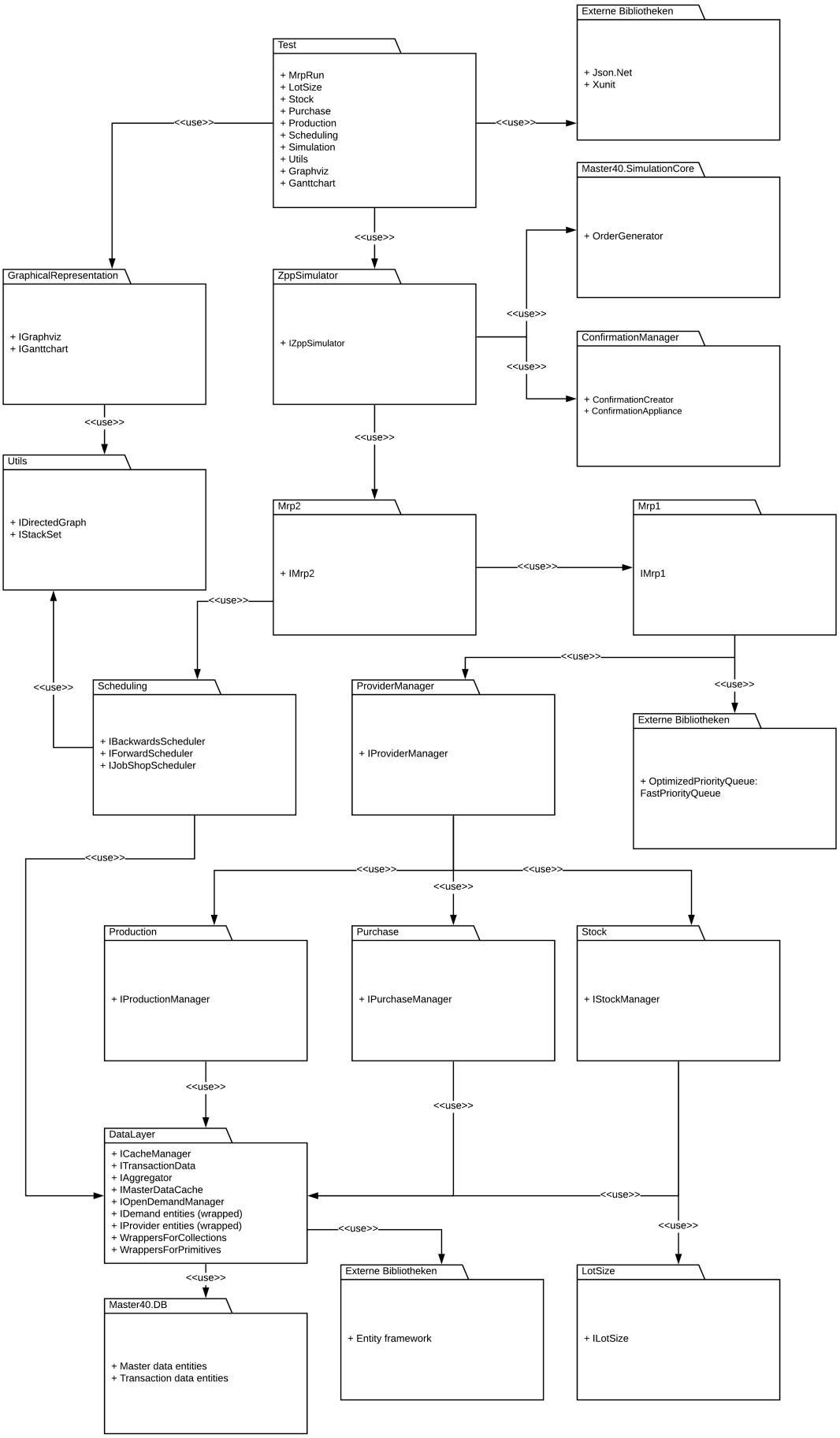

Abbildung 5.3.: Architektur der ZPP als Paketdiagramm der UML 2.0.

5.5. Umsetzung der Materialbedarfsplanung

5.5.1. Allgemeine Umsetzung Materialbedarfsplanung

Die Materialbedarfsplanung weist jedem Bedarf (Demand) einen Bedarfsdecker (Provider) zu. Der folgende Text beschreibt die Zuweisung allgemein ohne spezifische Ausprägung der Demands und Providern anhand des Listings 5.1 beschrieben. Zunächst initialisiert der Algorithmus eine Prioritätswarteschlange (dq) geordnet nach Startzeit. Wie im Kap. 2.6.1 beschrieben, könnte die dq ebenso nach Endzeit sortiert sein, bei den Demands gibt es nur einen Termin. Die Sortierung ist wichtig, denn die ZPP plant aufsteigend mit einer Breitensuche, jeder zukünftige zu planende Demand muss auf terminlich frühere Demands bzw. Provider zugreifen können.

In der Variablen entities (Menge) sammelt der Algorithmus die zu erstellenden Entitäten. Der Algorithmus initialisiert die dq mit allen bereits vorhandenen Demands aus den Transaktionsdaten. Die bereits vorhandenen Demands können generierte CustomerOrderParts oder ungestillte Demands aus dem vorherigen Zyklus sein. Der Algorithmus startet eine While-Schleife, die der Algorithmus solange ausführt, bis die dq leer ist. In der inneren Schleife entnimmt der Algorithmus der dq ein Element und speichert das Element in der Variablen demand. Da die dq aufsteigend nach Priorität sortiert ist, handelt es sich bei dem Element um den als nächstes fälligen Demand. Der Algorithmus erstellt für den Demand einen Provider und fügt den Provider der Variablen entities hinzu.

Danach erstellt der Algorithmus abhängige Demands für den Provider und fügt abhängige Demands der Variable entities hinzu. Abhängige Demands erklärt der Autor später im Kap. 5.5. Der Algorithmus fügt alle abhängige Demands zur dq hinzu und die dq sortiert die abhängige Demands entsprechend ihrer Priorität (Startzeit) ein. Die While-Schleife endet. Am Ende fügt der Algorithmus alle erstellten Entitäten aus der Variablen entities zu den Transaktionsdaten hinzu. Der Algorithmus endet.

```
1  dq := leere Demand-Prioritätswarteschlange geordnet nach Startzeit
2  entities = {}
3  Füge alle Demands der Transaktionsdaten zur dq hinzu
4  while dq NOT empty do
5      demand := Entnehme ein Element der dq
6      provider := Erstelle einen Provider für demand
7      entities := entities vereinigt {provider}
8      dependingDemands := Erstelle abhänge Demands für provider
9      entities := entities vereinigt dependingDemands
10     foreach dependingDemand in dependingDemands
11         Füge dependingDemand zur dq hinzu
12 Füge alle entities zu den Transaktionsdaten hinzu
```

Listing 5.1: Materialbedarfsplanung allgemein anhand von Demands und Providern

5.5.2. Spezifische Umsetzung Materialbedarfsplanung

Der nächste Abschnitt erklärt die Materialbedarfsplanung spezifisch für die verschiedenen Ausprägungen von Demands und Providern. Dafür muss der Autor unter anderem klären, was es bedeutet, einen Provider bzw. abhängige Demands zu erstellen und offene StockExchangeDemands zu reservieren. Zunächst muss der Autor klären, welche Verknüpfungen zwischen Demands und Providern gültig sind. Welcher Typ von Provider darf welchen Demand stillen. Außerdem muss festgelegt sein, welcher Typ von Provider welche Typen von abhängigen Demands haben darf. Die Tabelle 5.1 zeigt die Spezifikation der gültigen

Zuordnungen zwischen Demand und Provider. Die Pfeilrichtung ist willkürlich festgelegt und zeigt top-down vom CustomerOrderPart ausgehend im DemandToProvider-Graphen.

- Ein CustomerOrderPart darf nur auf genau einen StockExchangeProvider zeigen. Denn in dieser Arbeit laufen alle Kundentransaktionen über das Lager.
- Eine ProductionOrderBom darf nur auf genau einen StockExchangeProvider zeigen. Wie CustomerOrderParts laufen die abhängigen Transaktionen der ProductionOrderBom (Produktionsauftrag bzw. Bestellung) ebenfalls über das Lager.
- Ein StockExchangeDemand darf nur auf genau ein PurchaseOrderPart oder genau eine ProductionOrder zeigen. Abschließend endet der Lagerbedarf also in einer Bestellung bzw. in einem Produktionsauftrag. Eine Ausnahme ist folgende. Die Lagerstrategie benötigt es zu Modellierungszwecken, dass StockExchangeDemands auch keinen Provider haben. Näheres ist im Kap. 5.5.5 beschrieben.
- Ein PurchaseOrderPart darf auf keinen anderen Knoten zeigen. Ein PurchaseOrderPart hat also keine abhängigen Demands, denn mit einer Bestellposition endet ein Pfad im DemandToProvider-Graphen, es sind keine weiteren Bedarfe nötig. Der Grund ist, dass in dieser Arbeit die Wertschöpfungskette mit einer Bestellung endet und eine weitere Rückverfolgung nicht nötig ist.
- Eine ProductionOrder muss auf mindestens eine ProductionOrderBom zeigen. Jede ProductionOrder hat also ProductionOrderBoms als abhängige Demands, denn wie sollte ein Produktionsauftrag ohne Produktionsauftragsstücklistenposition produziert werden.
- Ein StockExchangeProvider muss mindestens auf einen StockExchangeDemand zeigen. Da mehrere StockExchangeProvider auf den gleichen StockExchangeDemands zeigen können, ist es eine m-n-Beziehung. Jeder StockExchangeProvider hat also StockExchangeDemands als abhängige Demands, sonst könnte ein negativer Lagerbestand entstehen. Näheres ist im Kap. 5.5.5 beschrieben.

Demand	Pfeilrichtung	Provider
1 CustomerOrderPart	$\rightarrow$	1 StockExchangeProvider
1 ProductionOrderBom	$\rightarrow$	1 StockExchangeProvider
1 StockExchangeDemand	$\rightarrow$	1 PurchaseOrderPart | 1 ProductionOrder
NONE	$\leftarrow$	1 PurchaseOrderPart
n ProductionOrderBoms	$\leftarrow$	1 ProductionOrder
n StockExchangeDemand	$\leftarrow$	m StockExchangeProvider

Tabelle 5.1.: Spezifikation der gültigen Zuordnungen zwischen Demand und Provider

5.5.3. Einführung der ProviderToDemand-Tabelle

Der Autor muss die Verbindungen zwischen einem Demand und einem Provider technisch realisieren. Die bereits gegebene Tabelle T_DemandToProvider verbindet Demands und Provider. Eine Richtung ist der Entität nicht zu entnehmen. Deshalb führt der Autor eine T_ProviderToDemand-Tabelle ein. Die T_ProviderToDemand-Tabelle hat exakt die gleichen Felder wie die T_DemandToProvider-Tabelle: Id, DemandId, ProviderId, Quantity.

Sei die Richtung einer DemandToProvider-Entität von dem Demand auf den Provider zeigend. Sei die Richtung einer ProviderToDemand-Entität von dem Provider auf den Demand zeigend.

5.5.4. Erstellung eines Providers für einen gegebenen Demand

Die Abb. 5.4 zeigt den Algorithmus für die Erstellung eines Providers für einen Demand als Aktivitätsdiagramm der UML 2.0. Der Algorithmus startet in der inneren Schleife des allgemeinen Materialbedarfsplanungsalgorithmus Zeile fünf in Listing 5.1 mit „Entnahme des Demands aus WS“. Der Algorithmus bestimmt den Typ des entnommenen Demands. Wenn der Demand ein

- CustomerOrderPart ist, erstellt der Algorithmus einen StockExchangeProvider mit der Menge des CustomerOrderPart (Demand).
- ProductionOrderBom ist, prüft der Algorithmus, ob die ProductionOrderBom Material benötigt, d.h., ein ArtikelChild gesetzt ist. Benötigt die ProductionOrderBom kein Material, endet der Algorithmus. Benötigt die ProductionOrderBom Material, erstellt der Algorithmus ein StockExchangeProvider mit der Menge der ProductionOrderBom (Demand).
- StockExchangeDemand ist, prüft der Algorithmus, ob der Artikel des StockExchangeDemands ein Bestell- oder Produktionsartikel ist. Ist der Artikel des StockExchangeDemands ein Produktionsartikel, erstellt der Algorithmus eine ProductionOrder. Ist der Artikel des StockExchangeDemands ein Bestellartikel, erstellt der Algorithmus eine PurchaseOrder mit einem PurchaseOrderPart. Der erstellte PurchaseOrderPart hat die Menge des StockExchangeDemands, denn die Losgröße wird bereits bei der Erstellung der StockExchangeDemands berücksichtigt. Die Menge stimmt also mit der Losgröße der Beschaffung überein.

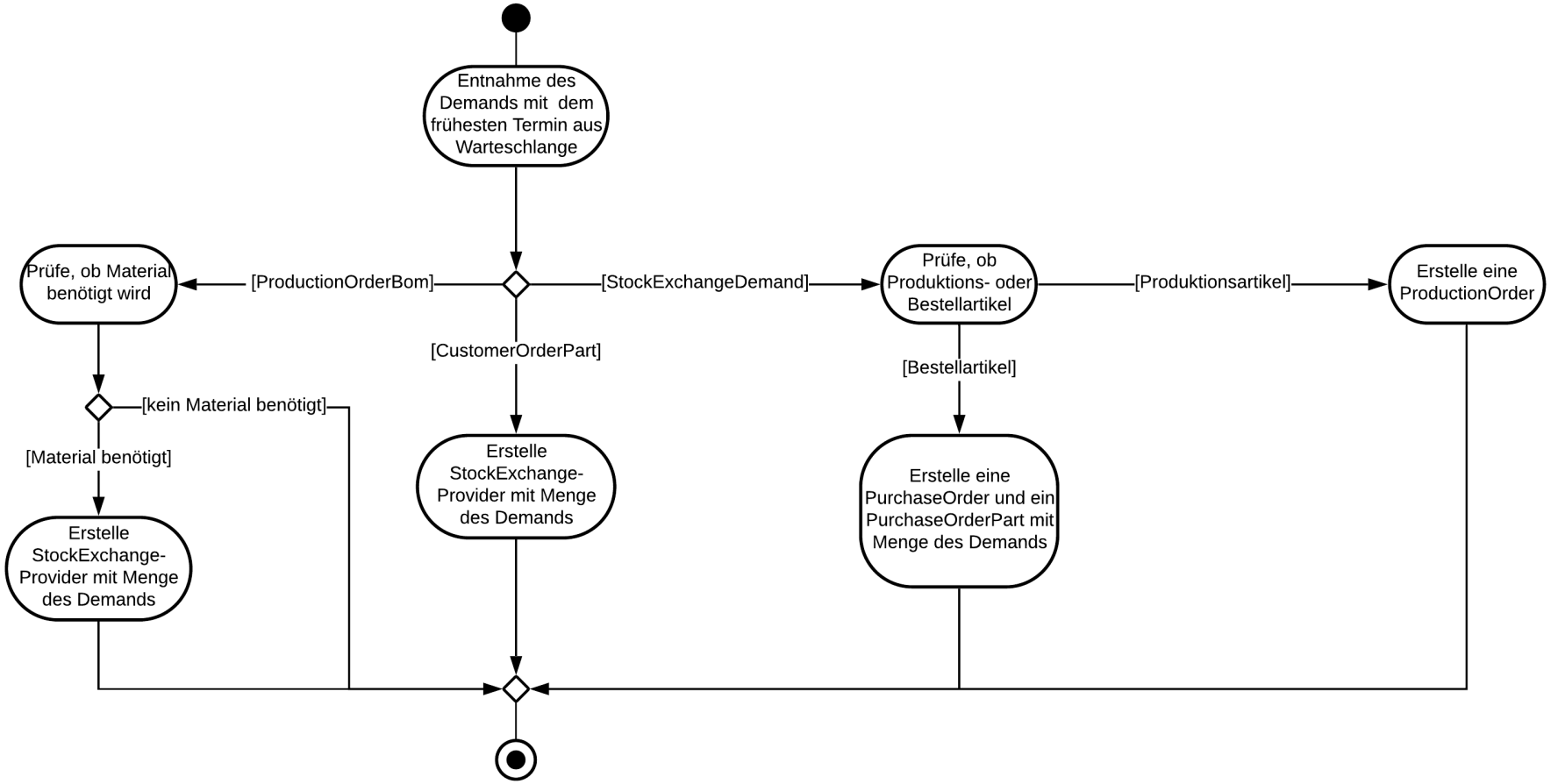

Abbildung 5.4.: Algorithmus: Erstellung eines Providers für einen gegebenen Demand

5.5.5. Umsetzung der Lagerstrategie

Für jeden Artikel muss es mindestens ein Lagerplatz geben. Ein Lagerplatz für ein Produktionsartikel hat weder Mindestbestand noch einen Maximalbestand, da es sich um eine Auftragsfertigung handelt und Produktionsartikel nur kurzfristig im Lager sind. Ein Lagerplatz für Einkaufsartikel hat einen Mindestbestand und einen Maximalbestand. Wird der Mindestbestand unterschritten, muss ein StockExchangeDemand erstellt werden, der als Menge den Maximalbestand abzüglich den momentanen Bestand bekommt.

Lagerobjekte mit initialen Lagerbeständen zu erstellen und deren Bestände zu jedem Zeitpunkt sicher zu verwalten ist nicht trivial. Der Autor umgeht das Problem wie folgt. Jeder StockExchangeProvider muss auf einen StockExchangeDemand zeigen, beide haben einen Termin. Auf einen StockExchangeDemand können so viele StockExchangeProvider mit einem Termin kleiner als der Termin des StockExchangeDemand zeigen, bis die gesamte Menge des StockExchangeDemands vergeben ist. Hat ein StockExchangeDemand noch eine freie Menge, ist der StockExchangeDemand offen, sonst geschlossen.

Die initialen Lagerbestände können man simulieren, indem man für jedes Lager einen StockExchangeDemand mit der Menge des initialen Lagerbestands und dem Zeitpunkt Null erstellen. Nun würde eventuell der StockExchangeDemand für StockExchangeProvider, die in der Vergangenheit liegen (im ersten ersten MRP-Lauf ist das möglich), nie genutzt werden. Das Problem umgeht der Autor für den ersten MRP-Lauf, indem er die Zeitpunkte nicht null, sondern sehr weit in der Vergangenheit setzt, z.B. -100.000. Die erste Vorwärtsterminierung wird die StockExchangeDemands auf den Zeitpunkt null oder größer ziehen. Die zweite Rückwärtsterminierung wird die StockExchangeDemands möglicherweise auf einen Zeitpunkt größer als null ziehen. Das entspricht nicht der Realität, beeinflusst dennoch die Planung nicht. Denn die nachfolgenden Zyklen bekommen CustomerOrderParts in einem zukünftigen Zeitabschnitt.

Ein weiteres Problem des Ansatzes ist, dass ein StockExchangeDemands als initialer Lagerbestand keinen Provider hat, ein solcher StockExchangeDemand verletzt also die Spezifikation 5.1. Ob ein StockExchangeDemand auf NONE zeigen darf und der Modellierungsansatz in Ordnung ist, oder die Modellierung anderweitig realisiert werden muss, müssen zukünftige Arbeiten ermitteln.

5.5.6. Umsetzung der Losgrößenerstellung

Die Materialbedarfsplanung nutzt die Schnittstelle für die Online-Losgrößenerstellung. Die Schnittstelle für die Online-Losgrößenerstellung ist im Interface ILotSize mit einer Methode „GetLotSizes“ definiert. Die Methode gibt eine Liste von Losgrößen zurück. Die Losgrößenbildung obliegt der Implementierung des Interfaces. Diese Arbeit stellt eine Implementierung die in den Eingangsdaten definierte konstante Losgröße bereit, sofern keine Losgröße am Material definiert ist.

Eine Offline-Losgrößenbildung erfordert eine erneute Ausführung des MRP-Laufs, denn dadurch verändern sich die Demand-Provider-Zuordnungen. Durch die Modularisierung der Architektur kann der MRP-Lauf erneut ausgeführt werden, allerdings sollten die vorhandenen Transaktionsdaten (außer Kundenaufträgen) gelöscht werden.

5.5.7. Erstellung der abhängigen Demands eines Providers

Das Listing 5.2 zeigt den Algorithmus für die Erstellung der abhängigen Demands eines Providers als Pseudocode. In der Variablen entities (Menge) sammelt der Algorithmus die zu erstellenden Entitäten. Die Variable provider ist der Provider, für den abhängige

Demands erstellt werden sollen. Zunächst ist eine Fallunterscheidung nötig, handelt es sich bei provider um eine ProductionOrder oder um einen StockExchangeProvider. Zunächst behandelt der Algorithmus den Fall, provider ist eine ProductionOrder.

Der Algorithmus erstellt ProductionOrderBoms für die ProductionOrder mit der Menge und dem Artikeltyp der ProductionOrder und speichert das Ergebnis in der Variablen dependingDemands. Danach iteriert der Algorithmus über alle dependingDemands und nennt ein Element dependingDemand. Der Algorithmus erstellt für dependingDemand einen ProviderToDemand mit der Menge des dependingDemand, der Id des dependingDemand für DemandId und der Id von provider für ProviderId. Der Algorithmus fügt den providerToDemand und dependingDemand zu entities hinzu. Der Fall „Provider ist eine ProductionOrder“ endet.

Danach behandelt der Algorithmus den Fall, provider ist eine StockExchangeProvider. Der Algorithmus erstellt die Variable remainingQuantity und initialisiert remainingQuantity mit der Menge des Providers in provider. Zunächst versucht der Algorithmus provider durch offene Demands zu stillen und weist die erstellten ProviderToDemands der Variablen providerToDemands zu. Der dahinterstehende Subalgorithmus versucht also offene (StockExchange-) Demands zu reservieren. Der Algorithmus reduziert remainingQuantity um die Summe aller reservierten Mengen, die an den providerToDemands stehen. Der Algorithmus fügt die erstellten Entitäten zu entities hinzu. Falls remainingQuantity größer als Null ist, muss der Algorithmus weitere StockExchangeDemands erstellen. für die Erstellung von StockExchangeDemands erstellt der Algorithmus, durch den Ruf einer Schnittstelle für Losgrößenerstellung mit dem Artikeltyp des in provider gespeicherten Providers, die Losgrößen für die in remainingQuantity gespeicherte Menge und speichert die Losgrößen in lotSizes. Es handelt sich bei der Nutzung der Schnittstelle für Losgrößenerstellung um eine Online-Losgrößenbildung.

Dann iteriert der Algorithmus über lotSizes und nennt ein Element lotSize. Der Schleifenrumpf startet. Der Algorithmus erstellt einen StockExchangeDemand mit der Menge lotSize, Startzeit von provider und Artikeltyp von provider und fügt den erstellten StockExchangeDemand zu entities hinzu. Der Algorithmus erstellt die Variable quantityToReserve und initialisiert quantityToReserve mit 0. Wenn remainingQuantity größer gleich lotSize ist, belegt der Algorithmus quantityToReserve mit lotSize. Ansonsten belegt der Algorithmus quantityToReserve mit remainingQuantity und merkt sich den erstellten StockExchangeDemand als offenen Demand, denn von der letzten Losgröße bleibt etwas übrig. Zukünftige StockExchangeProvider können somit den StockExchangeDemand reservieren.

Danach reduziert der Algorithmus remainingQuantity um die Losgröße. Der Algorithmus erstellt einen ProviderToDemand mit der Menge quantityToReserve, Id des erstellten StockExchangeDemands für DemandId, und Id von provider für ProviderId und fügt den erstellten ProviderToDemand zu entities hinzu. Der Schleifenrumpf endet. Am Ende fügt der Algorithmus alle erstellten Entitäten aus der variablen entities zu den Transaktionsdaten hinzu. Der Algorithmus endet.

```
entities := {}
provider := ein Provider, für den abhängige Demands erstellt werden sollen
if typeof(provider) = typeof(ProductionOrder)
    dependingDemands := Erstelle ProductionOrderBoms für provider mit Menge(provider)
      und Artikeltyp(provider)
    foreach dependingDemand in dependingDemands)
        providerToDemand := Erstelle einen T_ProviderToDemand mit der
          Menge = Menge(dependingDemand), DemandId = Id(dependingDemand),
          und ProviderId = Id(provider)
        entities := entities vereinigt {providerToDemand, dependingDemand}

else if typeof(provider) = typeof(StockExchangeProvider)
    remainingQuantity := Menge(provider)
    // Versuche provider durch offene Demands zu stillen
    providerToDemands := Stille Provider durch offene Demands (Reserviere offene Demands)
    remainingQuantity := remainingQuantity - Sum(providerToDemands)
    entities := entities vereinigt providerToDemands
    if remainingQuantity > 0
        lotSizes := Erstelle Losgrößen für remainingQuantity abhängig
          vom Artikeltyp(provider)
        foreach lotSize in lotSizes
            stockExchangeDemand := Erstelle StockExchangeDemand mit Menge lotSize,
              StartBackward(provider) und Artikeltyp(provider)
            entities := entities vereinigt {stockExchangeDemand}
            quantityToReserve := 0
            if remainingQuantity >= lotSize
                quantityToReserve = lotSize
            else
                quantityToReserve := remainingQuantity
                Remember created demand as openDemand
            remainingQuantity = remainingQuantity - lotSize
            providerToDemand := Erstelle einen T_ProviderToDemand mit der
              Menge = quantityToReserve, DemandId = Id(stockExchangeDemand),
              und ProviderId = Id(provider)
            entities := entities vereinigt {providerToDemand}
else
    // PurchaseOrderPart hat keine abhängigen Demands
Füge entities zu den Transaktionsdaten hinzu
```

Listing 5.2: Algorithmus: Erstellung der abhängigen Demands eines Providers als Pseudocode

5.5.8. Reservierung offener Demands

Das Listing 5.3 zeigt den Algorithmus für die Reservierung offener Demands als Pseudocode. Die Variable provider ist der Provider, für den der Algorithmus abhängige Demands reservieren soll. In der Variablen entities (Menge) sammelt der Algorithmus die zu erstellenden Entitäten. Zunächst bestimmt der Algorithmus die offenen Demands für den Artikeltyp von provider und speichert die offenen Demands in der Variablen openDemands. Dann initialisiert der Algorithmus die Variable remainingQuantity mit der Menge von provider.

Dann iteriert der Algorithmus über openDemands und nennt ein Element der Menge openDemand. Der Schleifenrumpf beginnt. Falls die Startzeit von provider größer gleich der Startzeit von openDemand ist, führt der Algorithmus den folgenden Rest des Schleifenrumpfes aus. Der Algorithmus initialisiert die Variable quantityToReserve mit einer Null. Wenn remainingQuantity größer gleich der Menge des openDemand ist, belegt der Algorith-

mus quantityToReserve mit der Menge von openDemand und entfernt den openDemand aus openDemands. Ansonsten belegt der Algorithmus quantityToReserve mit dem Wert von remainingQuantity. Nach der Fallunterscheidung reduziert der Algorithmus remainingQuantity um quantityToReserve. Der Algorithmus erstellt einen ProviderToDemand mit der Menge von quantityToReserve, der Id von openDemand für DemandId und der Id von provider für ProviderId. Danach fügt der Algorithmus den erstellten providerToDemand zu entities hinzu. Wenn die remainingQuantity kleiner gleich null ist, bricht die Schleife ab. Der Schleifenrumpf endet.

```
1  provider := ein Provider, für den abhängige Demands erstellt werden sollen
2  openDemands := offene Demands für den Artikeltyp des Providers
3  entities := {}
4  remainingQuantity := Menge(provider)
5
6  foreach openDemand in openDemands
7      if StartTimeBackward(provider) >= StartTimeBackward(openDemand)
8          quantityToReserve := 0
9          if remainingQuantity >= OpenQuantity(openDemand)
10             quantityToReserve := Menge(openDemand)
11             Entferne openDemand aus den openDemands
12         else
13             // Letzte Iteration, remaining < openQuantity
14             quantityToReserve := remainingQuantity
15
16         Reduziere remainingQuantity um quantityToReserve
17         providerToDemand := Erstelle einen T_ProviderToDemand mit der
18             Menge = quantityToReserve, DemandId = Id(openDemand),
19             und ProviderId = Id(provider)
20         entities := entities vereinigt providerToDemand
21         if remainingQuantity <= 0
22             break;
```

Listing 5.3: Algorithmus: Reservierung offener Demands als Pseudocode

5.5.9. Ergebnis der Materialbedarfsplanung für das Tischbeispiels

Der folgende Abschnitt wendet die allgemeine Materialbedarfsplanung auf das vorgestellte Tischbeispiel aus Kap. 2.3 an. Bevor die Materialbedarfsplanung gestartet werden kann, müssen zunächst Kundenaufträge erstellt werden. Als Beispiel startet der Autor den Simulator mit der Vorgabe, zwei Kundenaufträge mit je einem Tisch als Kundenauftragsposition zu erstellen. Die Losgröße sei konstant und auf zwei gesetzt. Abb. 5.5 zeigt das Ergebnis der Materialbedarfsplanung als Bedarf-Bedarfsdecker-Graphen. Die restliche Arbeit bezeichnet den Bedarf-Bedarfsdecker-Graph als DemandToProvider-Graphen.

Die Stücklistenauflösung und Stillung der DemandToProvider-Beziehungen erfolgt gemäß der Spezifikation 5.1. D steht für Demand und P steht für Provider. Die Syntax eines Knotennamens lautet: „<P oder D>: <Typ>; <Id>: <Artikelname>; Menge: <Menge>; Start/End: <Zeiten>“. Ein Knotenname hat am Ende Start- und Endzeiten, das ist eine erste grobe Terminierung, die durch die Erstellung der Entitäten während der Stücklistenauflösung geschieht. Die Startzeiten nutzt MRP1 für die Sortierung der WS. „<Typ>“ kann für einen der folgenden Begriffe stehen: CustomerOrderPart (D), ProductionOrderBom (D), StockExchangeDemand (D), PurchaseOrderPart (P), ProductionOrder (P) und StockExchangeProvider (P). Der Simulator hat zwei CustomerOrderParts je mit der Menge eins erstellt. Der Simulator übergibt die zwei CustomerOrderPart der Materialbedarfspla-

nung (MRP1) als Demands zeitlich sortiert übergeben. Zunächst erstellt MRP1 für den linken CustomerOrderPart einen Bedarf: Ein StockExchangeProvider mit der im Demand angeforderten Stückzahl eins. MRP1 muss auch den StockExchangeProvider stillen und sucht einen offenen StockExchangeDemand mit einem Tisch, der noch nicht vollständig reserviert ist. MRP1 findet keinen offenen StockExchangeDemand mit einem Tisch, also erstellt MRP1 den StockExchangeDemand Tisch. Die Menge für den StockExchangeDemand bekommt MRP1 durch die Losgrößenschnittstelle. Im gewählten Beispiel ist die Losgröße konstant zwei, also ist die Menge für den StockExchangeDemand zwei. MRP1 fügt den erstellten StockExchangeDemand als Demand in die Warteschlange (WS) ein.

Der StockExchangeDemand Tisch hat einen früheren Termin als der rechte CustomerOrderPart. Folglich ist der StockExchangeDemand Tisch vor dem rechten CustomerOrderPart in der WS und MRP1 bearbeitet den StockExchangeDemand als nächstes. Der StockExchangeDemand Tisch ist ein Produktionsartikel. MRP1 erstellt eine ProductionOrder mit der angeforderten Menge zwei und weist die erstellte ProductionOrder dem StockExchangeDemand Tisch zu. Eine ProductionOrder muss abhängige Demands haben und MRP1 erstellt der Stückliste entsprechend alle nötigen ProductionOrderBoms sowie die dafür nötigen ProductionOrderOperations. MRP1 sortiert die erstellten abhängigen Demands (ProductionOrderBoms) in die WS ein. Die weitere Stücklistenauflösung und Stillung der DemandToProvider-Beziehungen funktioniert analog des bereits demonstrierten Teils und anhand der beschriebenen Algorithmen. Eine Besonderheit ist der StockExchangeDemand Schrauben. Der StockExchangeDemand Schrauben hat keinen Provider, daraus folgt nach der Lagerstrategie Kap. 5.5.5, dass er ein initialer Lagerbestand ist. Der StockExchangeDemand Schrauben stand also schon beim Start des MRP1 zur Verfügung.

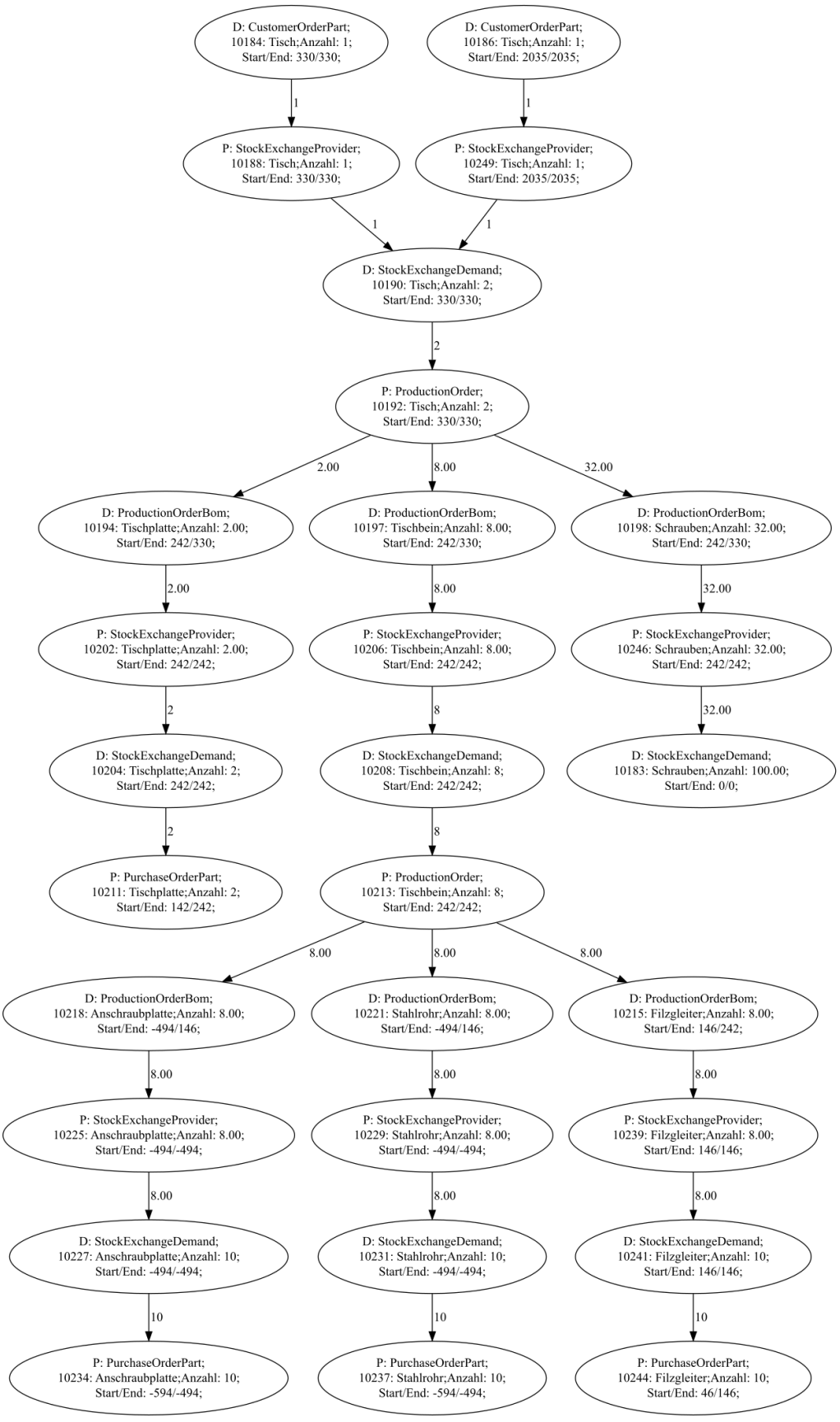

Abbildung 5.5.: Ergebnis der Materialbedarfsplanung als DemandToProvider-Graph

5.6. Umsetzung der Kapazitätsplanung gegen unbegrenzte Kapazität

5.6.1. Umsetzung der Erstellung des Auftragsoperationsgraphen

Zunächst muss der Algorithmus den DemandToProvider-Graphen erstellen. Dann muss der Algorithmus während der Erstellung für die Kinder jeder ProductionOrder löschen (ProductionOrderBoms). ProductionOrderBoms haben keine Start- und Endzeit, daher muss der Algorithmus jede ProductionOrderBom durch ihre Operation ersetzen, jede Operation darf nur einmal vorkommen. Der Algorithmus erstellt den Operationsgraphen für jede ProductionOrder und ersetzt die entsprechende ProductionOrder im DemandToProvider-Graphen durch den erstellten Operationsgraph der ProductionOrder. Der Operationsgraph einer ProductionOrder ist ein gerichteter Graph, der als Wurzel die ProductionOrder enthält und alle Operationen als Knoten in der Fertigungsreihenfolge enthält. Der Auftragsoperationsgraph bleibt nach der Materialbedarfsplanung konstant, daher kann der Auftragsoperationsgraph für folgende Durchlaufterminierungen wiederverwendet werden.

5.6.2. Berechnung der Start- und Endzeiten von Aufträgen und Operationen

Die Formeln für die Berechnung der Start- und Endzeiten für ProductionOrderOperations sind folgende:

- Die Menge stammt aus dem Ergebnis des MRP1 des entsprechenden Knotens.
- Der Übergangsfaktor ist in der Arbeit konstant drei.
- „Dauer Einzelfertigung“ stammt aus der Operation eines Artikels in der Tabelle M_Ope- ration.
- $Dauer = Menge *$ Dauer Einzelfertigung.
- Sei Materialbereitstellungszeitpunkt: mbz.
- $mbz(y) = max(\{Termin(x) | x$ ist ein Kindeskind von y im Auftragsoperationsgraphen und $(Typ(y) = ProductionOrderOperation)\} \cup \{$Endzeit(Vorgängeropertation)$\})$
- $Startzeit = mbz +$ Übergangszeit
- $Endzeit = Startzeit + Dauer$
- Übergangszeit $= Dauer *$ Übergangsfaktor

Die Formeln für die Berechnung der Start- und Endzeiten für PurchaseOrderParts sind folgende:

- Die Dauer ist Dauer der Lieferung aus aus der Tabelle M_ArticleToBusinessPartner.
- Die Menge stammt aus dem Ergebnis des MRP1 des entsprechenden Knotens.
- $Startzeit = Endzeit - Dauer$
- $Endzeit = Startzeit$(Vorgänger)

5.6.3. Umsetzung der Rückwärtsterminierung

Das Listing 5.4 zeigt den Algorithmus der Rückwärtsterminierung als Pseudocode. Der Algorithmus initialisiert die Variable og mit dem Auftragsoperationsgraphen. Der Algorithmus initialisiert den Stack S mit den zu terminierenden Wurzeln. Bei der ersten Rückwärtsterminierung sind die Wurzeln die CustomerOrderParts, bei der zweiten Rückwärtsterminierung sind die Wurzeln die Kinder der CustomerOrderParts (StockExchangeProvider). Wenn es die erste Rückwärtsterminierung ist, muss der Algorithmus alle Start- und Endzeiten der Aufträge bzw. Operationen löschen, um Seiteneffekte auszuschließen. Vorhandene Start- und Endzeiten könnten aus der Materialbedarfsplanung bzw. aus einem vorherigen Planungslauf vorhanden sein und die Terminierung beeinflussen.

Eine While-Schleife startet und führt den Schleifenrumpf, solange S Elemente hat, aus. Der Algorithmus initialisiert die Variable i mit dem obersten Element von S (Pop). Der Algorithmus initialisiert die Variable successors mit den Nachfolgeknoten von i in og. Danach iteriert der Algorithmus über successors und nennt ein Element successor. Der Schleifenrumpf der Foreach-Schleife beginnt. Der Algorithmus kellert successor in S ein. Falls successor schreibgeschützt ist, beendet der Algorithmus den Schleifenrumpf und führt die nächste Iteration aus, sofern die Schleifenbedingung erfüllt ist. Der Algorithmus bestimmt die minimale Startzeit aller Vorgänger von successor (einer davon ist i) und schreibt den Wert in minStartTime. Dann terminiert der Algorithmus die Endzeit von successor auf minStartTime. Die Foreach-Schleife endet. Die While-Schleife endet. Der Algorithmus endet.

```
1  og := Auftragsoperationsgraph
2  S := Stack mit zu terminierenden Wurzelknoten
3
4  Wenn diese Rückwärtsterminierung die erste Rückwärtsterminierung des Zykluses ist,
5    lösche alle Start-Endzeiten aller Knoten in og, die nicht read-only sind.
6
7  while S not empty do
8      i := Auskellern eines Elements aus S;
9      successors := Nachfolgeknoten(i) in og
10     foreach successor in successors
11         successor in S einkellern
12         if successor read-only
13             continue
14         // Minimale Startzeit der Vorgänger ermitteln
15         predecessors := Vorgängerknoten(successor) in og
16         minStartTime := StartTimeBackward(i)
17         foreach predecessor in predecessors
18             predecessorStartTime := StartTimeBackward(predecessor)
19             if predecessorsStartTime < minStartTime
20                 minStartTime = predecessorsStartTime;
21         // Rückwärts terminieren
22         Setze EndTimeBackward(successor) auf minStartTime
```

Listing 5.4: Algorithmus der Rückwärtsterminierung als Pseudocode

Die Terminierung erklärt der Autor anhand des Beispiels aus Abb. 5.6. Knoten a sei i, der Nachfolger ist c. Knoten c wird eingekellert und sei nicht schreibgeschützt. Die Variable minStartTime ist zunächst 10 (Wert von i). Nun bestimmt der Algorithmus die minimale Startzeit aller Vorgänger des Nachfolgers von i (Knoten c). Die Variable minStartTime ist dann 5 und der Algorithmus terminiert c auf 5. Denn a und b sind von c abhängig, können also erst starten, wenn c zur Verfügung steht. Würde c auf 10 terminiert werden, würde b

zu früh anfangen.

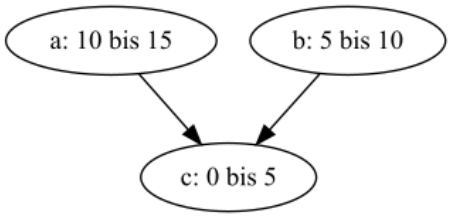

Abbildung 5.6.: Terminierungsbeispiel rückwärts

5.6.4. Umsetzung der Vorwärtsterminierung

Das Listing 5.5 zeigt den Algorithmus der Vorwärtsterminierung als Pseudocode.

Der Algorithmus initialisiert die Variable og mit dem Auftragsoperationsgraphen. Der Algorithmus initialisiert die Variable S als leeren Stack und die Variable si mit dem Simulationsintervall. Der Algorithmus benötigt das Simulationsintervall, damit der Algorithmus vom gegenwärtigen Zeitpunkt aus terminieren kann. Dann iteriert der Algorithmus über die Blattknoten von og und nennt ein Element node. Der Schleifenrumpf beginnt. Wenn der Knoten nicht schreibgeschützt ist und die Startzeit von node kleiner ist als der Startzeitpunkt des Simulationsintervall, dann setzt der Algorithmus die Startzeit von node auf den Startzeitpunkt des Simulationsintervall und kellert node in S ein. Der Schleifenrumpf endet.

Eine While-Schleife startet und führt den Schleifenrumpf, solange S Elemente hat, aus. Der Schleifenrumpf startet. Der Algorithmus initialisiert die Variable i mit dem obersten Element von S (Pop). Der Algorithmus initialisiert die Variable predecessors mit den Vorgängerknoten von i in og.

Danach iteriert der Algorithmus über predecessors und nennt ein Element predecessor. Der Schleifenrumpf der Foreach-Schleife beginnt. Der Algorithmus kellert predecessor in S ein. Falls predecessor nicht schreibgeschützt ist, die Startzeit von predecessor kleiner ist als die Endzeit von i und der predecessor kein CustomerOrderPart ist, bestimmt der Algorithmus die maximale Endzeit aller Nachfolger von predecessor (einer davon ist i) und schreibt den Wert in maxEndTime. Dann terminiert der Algorithmus die Startzeit von predecessor auf maxEndTime. Die Foreach-Schleife endet. Die while-Schleife endet. Der Algorithmus endet.

Die Terminierung erklärt der Autor anhand des Beispiels aus Abb. 5.7. Knoten b sei i, der Vorgängerknoten ist a. Knoten a wird eingekellert und sei nicht schreibgeschützt. Die Variable maxEndTime ist zunächst 5 (Endzeit von b). Nun bestimmt der Algorithmus die maximale Endzeit aller Nachfolger des Vorgängers von i (Knoten a). Die Variable maxEndTime ist dann 10 (denn c hat Endzeit 10) und der Algorithmus terminiert die Startzeit von a auf 10. Denn von a sind b und c abhängig, a kann also erst starten wenn b und c zur Verfügung stehen. Würde a auf 5 (Endzeit von b) terminiert werden, würde a zu früh anfangen.

```
S := leerer Stack
og := Auftragsoperationsgraph
si := Simulationsintervall

foreach node in Blattknoten(og)
    // Die Vorwärtsterminierung startet nur von Blättern,
      die folgende Bedingungen erfüllen
    if node is NOT read-only AND StartTimeBackward(node) < Start(si)
        StartTimeBackward(node) := Start(si)
        Einkellern von node in S
while S not empty do
    i := Auskellern eines Elements aus S
    predecessors := Vorgängerknoten(i)
    foreach predecessor in predecessors
        predecessor in S einkellern
        if predecessor is NOT read-only AND
          StartTimeBackward(predecessor) < EndTimeBackward(i)
          And predecessor ist kein CustomerOrderPart
            // Bilde maximum endTime aller Nachfolger von predecessor
            maxEndTime := EndTimeBackward(i)
            successors := Nachfolgeknoten(predecessor) in og
            foreach successor in successors
                if EndTimeBackward(successor) > maxEndTime
                    maxEndTime = successorsEndTime;
            // Vorwärts terminieren
            StartTimeBackward(predecessor) := maxEndTime
```

Listing 5.5: Algorithmus der Vorwärtsterminierung als Pseudocode

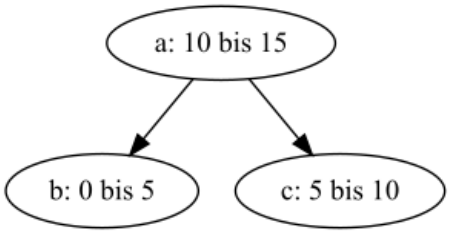

Abbildung 5.7.: Terminierungsbeispiel vorwärts

5.6.5. Ergebnisse der ersten Rückwärtsterminierung für das Tischbeispiel

Abb. 5.8 zeigt das Ergebnis der ersten Rückwärtsterminierung als Auftragsoperationsgraphen für das Tischbeispiel. Die Abb. zeigt den Auftragsoperationsgraphen versehen mit Start- und Endzeiten nach der ersten Rückwärtsterminierung. Der Leser kann die Zahlen mit den Formeln aus Kap. 5.6.2 berechnen und die Ergebnisse aus den Berechnung übersichtlicher mit der Tabelle B.1 aus dem Anhang vergleichen. „bs“ bzw. „be“ steht für die Start- bzw. Endzeit aus der Rückwärtsterminierung (eng. backwards start bzw. backwards end).

Die Zeiten der Aufträge bzw. Operationen des Auftragsoperationsgraphen sind am kleinsten an den Blättern und werden größer entgegengesetzt der Pfeilrichtung (Bottom-Up). Die Pfeile enthalten keine Mengen, da die Mengenzuordnung nicht Gegenstand einer Durchlaufterminierung ist. Die Pfeilrichtung ist willkürlich festgelegt und könnte ebenso umgekehrt sein, die Pfeilrichtung passt im festgelegten Fall zur Traversierungsrichtung der Rückwärtsterminierung, von den Wurzeln zu den Blättern (Top-Down, Tiefensuche). Dem Betrachter fällt sofort auf, dass die Blätter des Auftragsoperationsgraphen deutlich in der Vergangen-

heit liegen. Der Termin des linken CustomerOrderPart kann also nicht eingehalten werden.

Außerdem erkennt der Betrachter, dass die Operationen Übergangszeiten haben. Z. B. Operation „Filzgleiter anstecken“ startet in der Minute 218. Die vorangegangene Operation „Anschraubplatte anschweißen“ war bereits zur Minute 146 fertig. $218 - 146 = 72$, $72/3 = 24$, was exakt der Operationsdauer $242 - 218 = 24$ entspricht. Die Übergangszeit ist also 72 zwischen „Anschraubplatte anschweißen“ und „Filzgleiter anstecken“. Die negativen Zeiten (PurchaseOrderParts: Stahlrohr und Anschraubplatte) werden mit der Vorwärtsterminierung verschwinden.

Abbildung 5.8.: Ergebnis der ersten Rückwärtsterminierung als Auftragsoperationsgraph

5.6.6. Ergebnisse der Vorwärtsterminierung für das Tischbeispiel

Abb. 5.9 zeigt das Ergebnis der Vorwärtsterminierung als Auftragsoperationsgraphen für das Tischbeispiel. Die Abb. zeigt den Auftragsoperationsgraphen versehen mit Start- und Endzeiten nach der Vorwärtsterminierung. Der Leser kann die Zahlen mit den Formeln aus Kap. 5.6.2 berechnen und die Ergebnisse aus den Berechnung übersichtlicher mit der Tabelle B.2 aus dem Anhang vergleichen. „fs“ bzw. „fe“ steht für die Start- bzw. Endzeit aus der Vorwärtsterminierung (eng. forward start bzw. forward end).

Dem Betrachter fällt auf, dass die Blätter des Auftragsoperationsgraphen genau null bzw. positiv sind. Die Termine der CustomerOrderParts können also alle eingehalten werden. Außerdem fällt dem Betrachter auf, dass einige Provider deutlich zu früh bereitstehen, z. B. der StockExchangeProvider Tischplatte ist bereits zur Minute 242 bereitgestellt, wird erst zur Minute 902 benötigt. Das Gleiche wie für die Tischplatte gilt auch für die Schrauben (StockExchangeProvider). Die langen Liegezeiten werden in der zweiten Rückwärtsterminierung eliminiert.

Zur besseren graphischen Visualisierung zeigt Abb. 5.10 alle Operationen und Auftrage mit einer Dauer ungleich Null (PurchaseOrderParts).

Abbildung 5.9.: Ergebnis der Vorwärtsterminierung als Auftragsoperationsgraph

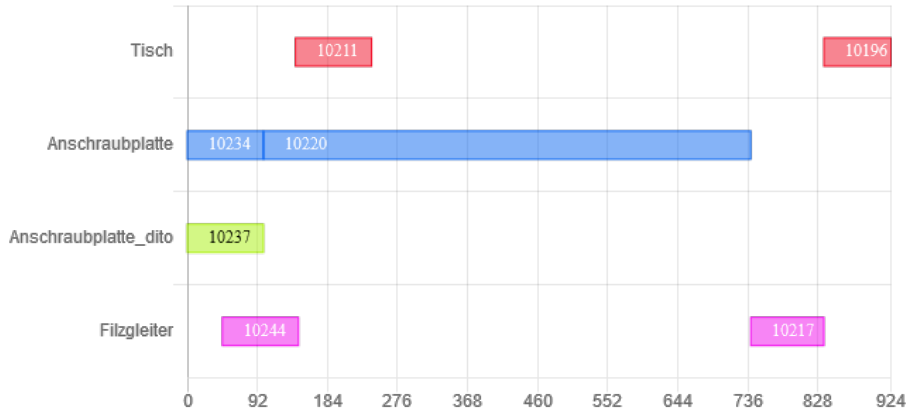

Abbildung 5.10.: Ergebnis der Vorwärtsterminierung als Gantt-Diagramm

5.6.7. Ergebnisse der zweiten Rückwärtsterminierung für das Tischbeispiel

Abb. 5.11 zeigt das Ergebnis der zweiten Rückwärtsterminierung als Auftragsoperationsgraphen für das Tischbeispiel. Die Abb. zeigt den Auftragsoperationsgraphen versehen mit Start- und Endzeiten nach der zweiten Rückwärtsterminierung. Der Leser kann die Zahlen mit den Formeln aus Kap. 5.6.2 berechnen und die Ergebnisse aus den Berechnung übersichtlicher mit der Tabelle B.3 aus dem Anhang vergleichen.

Dem Betrachter fällt sofort auf, dass die Startzeiten der Blätter des Auftragsoperationsgraphen immer noch genau null bzw. positiv sind. Nur der Termin des rechten CustomerOrderParts kann eingehalten werden, der Termin des linken CustomerOrderParts wird verfehlt.

Die Operationen haben die gleichen Start- und Endzeiten aus der Vorwärtsterminierung. Allerdings stehen einige Provider jetzt deutlich später bereit (= keine Liegezeiten), z.B. der StockExchangeProvider für die Tischplatte ist erst zur Minute 836 bereitgestellt (vorher 242) und wird zur Minute 902 benötigt. Die Minute 836 ist also optimal (keine Liegezeit), denn zur Minute 902 startet die nächste Operation und die Übergangszeit $(924 - 902) * 3 = 66$, folglich muss die Tischplatte $902 - 66 = 836$ bereitstehen. Das Gleiche wie für die Tischplatte gilt auch für die Schrauben (StockExchangeProvider). Die zweite Rückwärtsterminierung hat die langen Liegezeiten also eliminiert. Zur besseren graphischen Visualisierung zeigt Abb. 5.12 alle Operationen und Auftrage mit einer Dauer ungleich Null (PurchaseOrderParts).

Abbildung 5.11.: Ergebnis der zweiten Rückwärtsterminierung als Auftragsoperationsgraph

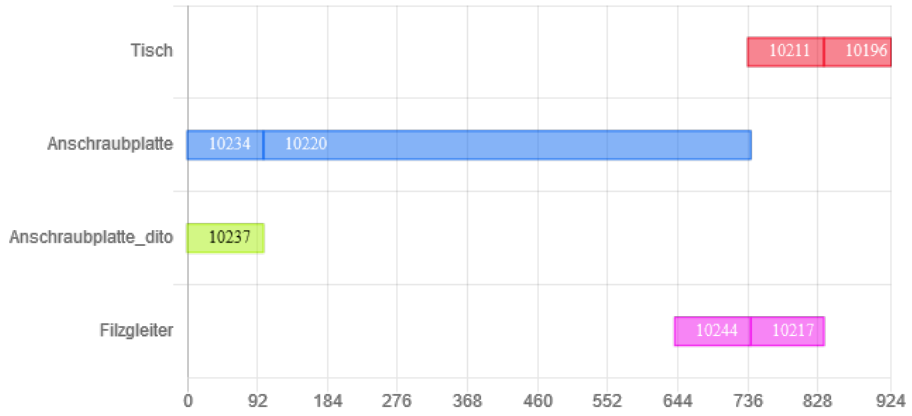

Abbildung 5.12.: Ergebnis der zweiten Rückwärtsterminierung als Gantt-Diagramm

5.7. Umsetzung der Kapazitätsplanung gegen begrenzte Kapazität

5.7.1. Umsetzung der Erstellung des Operationsgraphen

Die ZPP erstellt den Operationsgraphen aus dem Auftragsoperationsgraphen der Materialbedarfsplanung. Alle Knoten die keine ProductionOrderOperations sind, löscht die ZPP. Bei jeder Löschoperation verbindet die ZPP die Eltern des Löschknoten mit den Kindern des Löschknotens. Die Blätter des Operationsgraphen sind die ProductionOrderOperations, die einplanbar sind.

Abb. 5.13 zeigt den Operationsgraphen als gerichteten Graphen für das Tischbeispiel. Wie erwartet zeigt der Operationsgraph genau die drei Operationen entsprechend ihrer Fertigungsreihenfolge angeordnet: „Tisch zusammenbauen“, „Filzgleiter anstecken“ und „Anschraubplatte anschweißen“. „Filzgleiter anstecken“ ist ein Blatt, also eine einplanbare Operation. Der Operationsgraph zeigt die Start- und Endzeiten von der Rückwärtsterminierung, die die Ausgangszeiten für die Maschinenbelegung sind.

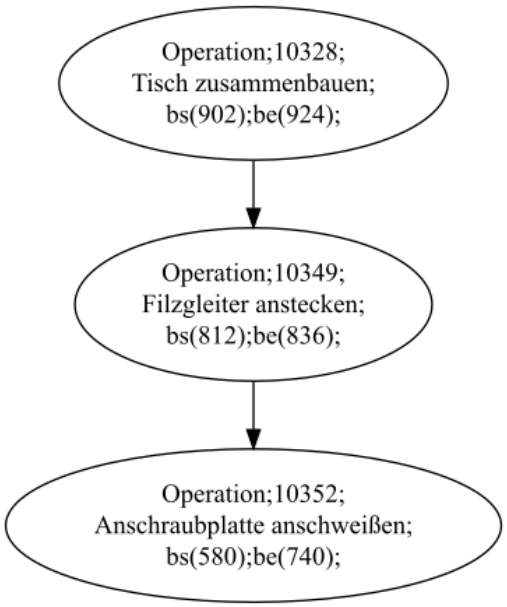

Abbildung 5.13.: Der Operationsgraph des Tischbeispiels als gerichteter Graph

5.7.2. Berechnung der Start- und Endzeiten von Operationen

Die Formeln für die Berechnung der Start- und Endzeiten für ProductionOrderOperations sind folgende Formeln:

- Die Menge stammt aus dem Ergebnis des MRP1 des entsprechenden Knotens.
- „Dauer Einzelfertigung“ stammt aus der Operation eines Artikels in der Tabelle M_Ope- ration.
- $Dauer = Menge *$ Dauer Einzelfertigung.
- Sei Materialbereitstellungszeitpunkt: mbz.

- $mbz(y) = max(\{Termin(x) | x$ ist ein Kindeskind von y
  im Auftragsoperationsgraphen
  und $(Typ(y) = ProductionOrderOperation)\}$
  $\cup\{$Endzeit(Vorgängeropertation)$\}$)
- $Startzeit = mbz$
- $Endzeit = Startzeit + Dauer$

5.7.3. Umsetzung des Giffler-Thompson-Algorithmus

Das Listing 5.7 zeigt den angepassten Giffler-Thompson-Algorithmus als Pseudocode. Das Listing 5.6 zeigt den Giffler-Thompson-Algorithmus als Pseudocode wie er im Buch von Herrn Zäpfel beschrieben ist[9], damit eine bessere Vergleichbarkeit zum angepassten Algorithmus gewährleistet ist.

Im Listing 5.6 sind zunächst alle Mengen und Variablen definiert, danach der Algorithmus. Der angepasste Algorithmus nutzt ebenfalls die definierten Mengen und Variablen aus Listing 5.6. Da Herr Zäpfel den Giffler-Thompson nur für eine Maschine pro Maschinentyp beschreibt, hat der Autor den Algorithmus auf Mehrmaschinenfähigkeit angepasst.

Sei og der erstellte Operationsgraph aus allen Operationen. Korrigiere die Leerlaufstartzeit jeder Maschine für alle Operationen der letzten Zyklen wie folgt. Iteriere über alle Operationen und iteriere für jede Operation über alle Maschinen der Maschinengruppe von der Operation. Wenn die Maschine die Maschine der Operation ist und die Leerlaufstartzeit der Maschine kleiner als die Endzeit der Operation ist, setze die Leerlaufstartzeit der Maschine auf die Endzeit der Operation.

Bestimme die Menge der einplanbaren Operationen (S) durch Ermittlung der Blätter von og. Setze die Startzeit aller Operationen aus S auf die Startzeit ihrer Rückwärtsterminierung. Der Algorithmus darf nicht wie in Zeile 14 Listing 5.6 die Startzeiten auf 0 setzen, sonst starten die Operationen zu früh. Solange S nicht leer ist, tue folgendes im Rumpf der While-Schleife. Berechne die Endzeit aller Operationen aus S durch die Addition der Operationsdauer auf die Startzeit. Bestimme die minimale Endzeit aller Operation aus S, speichere sie in d_min und speichere die dazugehörige Operation in o_min. Bilde die Konfliktmenge K für alle Operationen in S, deren Maschinengruppe mit der von o_min übereinstimmt und deren Startzeit kleiner als d_min ist. Solange K nicht leer ist, tue folgendes im Rumpf der inneren While-Schleife. Entnehme die Operation mit der höchster Priorität aus K und plane die Operation auf der nächsten freien Maschine ein. Herr Zäpfel beschreibt einen Teil des Algorithmus nicht näher und schreibt stattdessen „[...] und plane diese (Operation auf der nächsten Maschine) ein“ Zeile 21 Listing 5.6. Den nicht näher beschriebenen Teil des Algorithmus sowie die nachfolgenden Zeilen bis zum Ende beschreibt der Autor im folgenden.

Sei allO1 eine leere Menge. Für jede Maschine der Maschinen aus der Maschinengruppe von o_min aufsteigend geordnet nach Leerlaufstartzeit, tue folgendes im Schleifenrumpf der Foreach-Schleife. Sei o1 die Operation mit der höchsten Priorität aus K. Dafür wendet der Algorithmus die vorgeschriebene Prioritätsregel Schlupfzeitregel an. Füge o1 zu allO1 hinzu. Entferne o1 aus K. Sei ls die Leerlaufstartzeit der Maschine. Falls ls größer als die Startzeit von o1 ist, setze die Startzeit von o1 auf ls und passe die Endzeit von o1 entsprechend an. Sei ms der früheste Materialbereitstellungszeitpunkt von o1. Falls ms größer als die Startzeit von o1 ist, setze die Startzeit von o1 auf ms und passe die Endzeit

[9] Vgl. Zäpfel, *Moderne Heuristiken der Produktionsplanung*, S. 32.

von o1 entsprechend an. Aktualisiere die Leerlaufstartzeit der Maschine auf die Endzeit von o1. Die Foreach-Schleife endet.

Setze die Startzeit aller Operationen von K (die nicht mehr die Operationen in allO1 enthält) auf die Endzeit von der letzten Operation aus allO1. Für alle Operationen aus allO1 führe folgendes im Schleifenrumpf der Foreach-Schleife aus. Falls die Menge aller Nachfolger der Operation nicht leer ist, setze die Startzeiten aller Nachfolger auf die Endzeit der Operation. Entferne o1 aus og. Die Foreach-Schleife endet. Setze S auf die Menge aller Blätter von og. Die innere While-Schleife endet. Die äußere While-Schleife endet. der Algorithmus endet.

```
1  S: Menge der aktuell einplanbaren Arbeitsvorgänge
2  a: Menge der technologisch an erster Stelle eines Fertigungsauftrags stehenden
3    Arbeitsvorgänge
4  N(o): Menge der technologisch direkt nachfolgenden Arbeitsoperationen von
5    Arbeitsoperation o
6  M(o): Maschine auf der die Arbeitsoperation o durchgeführt wird
7  K: Konfliktmenge (die auf einer bestimmten Maschine gleichzeitig einplanbaren
8    Arbeitsvorgänge)
9  p(o): Bearbeitungszeit von Arbeitsoperation o (=Duration)
10 t(o): Startzeit der Operation o (=Start)
11 d(o): Fertigstellungszeitpunkt von Arbeitsoperation o (=End)
12 d_min: Minimum der Fertigstellungszeitpunkte
13 o_min: Operaton mit minimalem Fertigstellungszeitpunkt
14 o1: beliebige Operation aus K (o_dach bei Zäpfel)
15
16 Bestimme initiale Menge: S := a
17 t(o) := 0 für alle o aus S
18 while S not empty do
19     Berechne d(o) := t(o) + p(o) für alle o aus S
20     Bestimme d_min := min{ d(o) | o aus S }
21     Bilde Konfliktmenge K := { o | o aus S UND M(o) == M(o_min) UND t(o) < d_min }
22     while K not empty do
23         Entnehme beliebige Operation o1 aus K
24         und plane diese ein
25         t(o) := d(o1) für alle o aus K ohne o1
26         if N(o1) not empty then
27             S := S vereinigt N(o1) ohne o1
28             t(o) := d(o1) für alle o aus N(o1)
```

Listing 5.6: Der originale Giffler-Thompson-Algorithmus nach Herrn Zaepfel als Pseudocode

```
[...]
bs(o): Startzeit der Operation o aus der Rückwärtsterminierung
m: Maschine einer Maschinenmenge
ls(m): Leerlaufstart, Zeitpunkt, ab dem die Maschine im Leerlauf ist (untätig)
mb(o): Zeipunkt, an dem sämtliches für Operation o benötigtes Material bereit steht

Erstelle ProductionOrderToOperationGraph
Bestimme initiale Menge: S = a
foreach o in S
    t(o) = bs(o)

while S not empty do
    d_min = NULL
    foreach o in S
        Berechne d(o) = t(o) + p(o) für o
        if d_min is NULL OR d(o) < d_min
            d_min = d(o)
    K = {}
    foreach o in S
        if M(o) == M(o_min) AND t(o) < d_min
            K = K vereinigt {o}
    while K not empty do
        Entnehme Operationen und plane diese auf den Maschinen der Maschinengruppe
        von o_min ein
```

Listing 5.7: Der angepasste Giffler-Thompson-Algorithmus als Pseudocode

5.7.4. Ergebnis der Maschinenbelegung für das Tischbeispiel

Abb. 5.2 zeigt das Ergebnis der Maschinenbelegung für das Tischbeispiel als Tabelle. Der Leser kann die Zahlen mittels der Formeln aus Kap. 5.7.2 nachrechnen. Abb. 5.14 zeigt das Ergebnis der Maschinenbelegung für das Tischbeispiel als Gantt-Diagramm. Zu beachten ist, dass das mittlere Drittel ausgeblendet ist, denn da befindet sich keine Operation. Der Betrachter erkennt, dass es keine Übergangszeiten zwischen den Operationen gibt, denn die Maschinenbelegung ignoriert die Übergangszeiten aus den Durchlaufterminierungen. Dennoch gibt es eine deutliche Pause zwischen den Operationen, denn eine Operation kann erst beginnen, wenn das benötigte Material vorhanden ist.

Der Algorithmus traversiert nach unten (Top-Down, Tiefensuche) via den DemandToProvider/ProviderToDemand Strukturen (entspricht dem DemandToProvider-Graphen) bis einschließlich der nächsten StockExchangeDemands (falls vorhanden, sonst StockExchangeProvider) und ermittelt die maximale Endzeit aus den Kindeskindern und der Vorgängeroperation, um den Materialbereitstellungszeitpunkt herauszufinden.

Abbildung 5.14.: Ergebnis der Maschinenbelegung für das Tischbeispiel als Ganttchart (mittleres Drittel ausgeblendet)

Artikel	Operation	Material-bereitstel-lungszeit-punkt	Startzeit	Endzeit	Dauer	Menge	Dauer Einzelfer-tigung
Tisch	Tisch zusam-menbauen	836	836	858	22	2	11
Tischbein	Anschraub-platte an-schweißen	100	100	260	160	8	20
Tischbein	Filzgleiter anstecken	740	740	764	24	8	3

Tabelle 5.2.: Ergebnis der Maschinenbelegung für das Tischbeispiel

5.8. Umsetzung der Neuplanung unter Verrechnung von Rückmeldungen

5.8.1. Umsetzung der Erstellung von Rückmeldungen

Rückmeldungen erfolgen für ein bestimmtes in der Vergangenheit liegendes Zeitintervall, in unserem Fall für das momentane Simulationsintervall. Der Endzeitpunkt des Simulationsintervalls ist somit der gegenwärtige Zeitpunkt. Eine Rückmeldung ist die Statusanpassung einer Entität. Der Status einer Entität kann „Erstellt“, „In Bearbeitung“ oder „Beendet“ sein. Folgende Entitäten haben einen Status: StockExchangeDemands, StockExchangeProvider, PurchaseOrderParts und ProductionOrderOperations. Für alle anderen Entitäten kann ein Algorithmus den Status aus den vorher genannten Entitäten ermitteln.

Als nächstes führt der Autor zwei Maßnahmen ein. Die erste Maßnahme ist der Schreibschutz für ein Objekt. Jede erstellte Entität wird vor möglichen zukünftigen Änderungen an Zeiten geschützt. Der Schreibschutz sorgt dafür, dass eine Laufzeitexception auftritt, wenn ein Algorithmus versucht, Zeiten zu ändern. Der schützenswerte Zustand umfasst u. a. die Methoden für das Setzen einer Start- bzw. Endzeit. So schützt die Software die geplanten Objekte vor einer erneuten Durchlaufterminierung und Maschinenbelegung.

Als zweite Maßnahme führt der Autor eine Möglichkeit zur Archivierung von Objekten ein. Die Software hat daher neben der Datenbank mit den Master- und Transaktionstabellen eine Archivierungsdatenbank mit exakt den gleichen Tabellenschema. Wenn Objekte den Status „Beendet“ erreichen und eine erneute Verwendung in nachfolgenden Simulationsläufen nicht nötig ist, können die Objekte in die Archivierungsdatenbank verschoben werden. Daraus folgt eine im allgemeinen erhebliche Laufzeitverkürzung, denn z. B. muss die ZPP nicht mehr sämtliche StockExchangeDemands durchlaufen, sondern nur noch die offenen und beendeten StockExchangeDemands.

Das Listing 5.8 zeigt den Algorithmus der Erstellung von Rückmeldungen als Pseudocode. Zunächst speichert der Algorithmus das Ende des Simulationsintervalls in der Variablen intervalEnd. Dann iteriert der Algorithmus über die Vereinigung aller StockExchanges und PurchaseOrderParts; der Algorithmus bezeichnet ein Element aus der Vereinigung als entity. Im Schleifenrumpf setzt der Algorithmus zunächst die Entität der Variablen entity schreibgeschützt. Falls die Startzeit der Rückwärtsterminierung kleiner gleich der Variablen intervalEnd ist, setzt der Algorithmus den Status von entity auf „In Bearbeitung“. Wenn die Endzeit der Rückwärtsterminierung kleiner gleich der Variablen intervalEnd ist,

setzt der Algorithmus den Status der Variablen entity auf „Beendet“. Der Schleifenrumpf endet.

Dann iteriert der Algorithmus über die alle ProductionOrderOperations; der Algorithmus bezeichnet ein Element aus der Vereinigung als entity. Im Schleifenrumpf setzt der Algorithmus zunächst die Entität von entity schreibgeschützt. Falls die Startzeit der Maschinenbelegung kleiner gleich der Variablen intervalEnd ist, setzt der Algorithmus den Status der Variablen entity auf „In Bearbeitung“. Wenn die Endzeit der Maschinenbelegung kleiner gleich der Variablen intervalEnd ist, setzt der Algorithmus den Status der Variablen entity auf „Beendet“. Der Schleifenrumpf endet.

Dann iteriert der Algorithmus über alle CustomerOrderParts und bezeichnet ein Element der Menge als cop. Im Schleifenrumpf setzt der Algorithmus zunächst die Entität der Variablen entity schreibgeschützt. Danach führt der Algorithmus die Variable isfinished ein, die den initialem Wert wahr bekommt. Dann iteriert der Algorithmus in einer inneren Schleife über alle DemandToProviders und bezeichnet ein Element als demandToProvider. Im Schleifenrumpf der inneren Schleife setzt der Algorithmus den Wert der Variable isfinished auf falsch, falls der Provider des demandToProvider nicht beendet ist und der Demand des demandToProvider der cop entspricht. Die innere Schleife endet. Der Algorithmus setzt den Status der cop auf „Beendet“, falls isfinished den Wert wahr hat. Die äußere Schleife endet.

Abschließend iteriert der Algorithmus über die Vereinigung aller ProductionOrderBoms und ProductionOrders; der Algorithmus bezeichnet ein Element aus der Vereinigung als entity. Im Schleifenrumpf setzt der Algorithmus die Entität der Variablen entity schreibgeschützt. Die Schleife endet. Der Algorithmus endet.

```
intervalEnd := Ende des Simulationsintervalls

foreach entity in (StockExchanges vereinigt PurchaseOrderParts)
    Setze die entity read-only
    if StartTimeBackward <= intervalEnd
        Setze den Status der entity auf InProgress
    if EndTimeBackward <= intervalEnd
        Setze den Status der entity auf Finished

foreach entity in ProductionOrderOperations
    Setze die entity read-only
    if StartTime <= intervalEnd
        Setze den Status der entity auf InProgress
    if EndTime <= intervalEnd
        Setze den Status der entity auf Finished

foreach cop in CustomerOrderParts
    Setze die cop read-only
    isfinished := true
    foreach demandToProvider in DemandToProviders
        if demandToProvider.DemandId = cop.Id AND der Provider mit der
          demandToProvider.ProviderId
          ist nicht beendet
            isfinished := false
    if isFinshed
        Setze den Status der cop auf Finished

foreach entity in (ProductionOrderBoms vereinigt ProductionOrders)
    Setze die entity read-only
```

Listing 5.8: Algorithmus der Erstellung von Rueckmeldungen als Pseudocode

5.8.2. Umsetzung der Anwendung von Rückmeldungen

Nachdem die ZPP die Rückmeldungen für die Entitäten simuliert hat, muss die ZPP die Rückmeldungen anwenden. Der folgende Abschnitt betrachtet die Ermittlung des Status einer ProductionOrder. Wie das vorherige Unterkapitel darlegt, haben folgende Entitäten einen Status: StockExchangeDemands, StockExchangeProvider, PurchaseOrderParts und ProductionOrderOperations.

Den Status einer ProductionOrder kann ein Algorithmus wie folgt bestimmen. Alle Operationen einer ProductionOrder lassen sich in drei Mengen einteilen: die Menge der erstellten (A), sich in Bearbeitung befindenden (B) und beendeten Operationen (C). Alle Kombinationen aus der Vereinigung der drei Mengen können auftreten, es kann nur eine Kombination zu einem Zeitpunkt existieren. Über das Venn-Diagramms 5.15 kann der Status einer ProductionOrder ermittelt werden.

Das Venn-Diagramm zeigt alle möglichen Schnittmengen, die so nicht existieren, sondern hier nur jede Vereinigungsmöglichkeit demonstrieren, es gibt also sieben Fälle, die sich auf drei boolesche Ausdrücke reduzieren lassen. Eine ProductionOrder hat den Status „Erstellt“, wenn jede seiner Operationen den Status 0 („Erstellt“) hat. Eine ProductionOrder hat den Status 1 („In Bearbeitung“), wenn folgender aus dem Venn-Diagramm 5.15 herleitbarer logischer Ausdruck erfüllt ist: mind. eine Operation hat den Status „In Bearbeitung“ ODER (mind. eine Operation hat den Status „Beendet“ UND mind. eine Operation hat den Status „Erstellt“). Eine ProductionOrder hat den Zustand 2 „Beendet“, wenn alle Operationen den Zustand „Beendet“ haben.

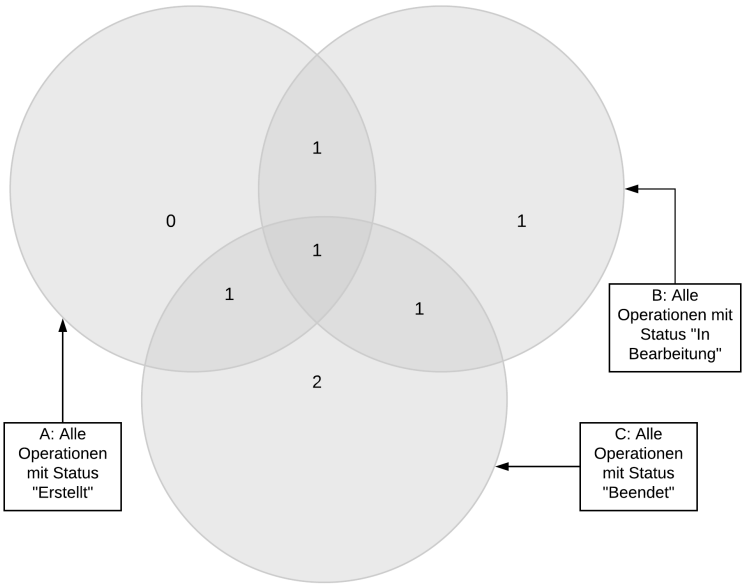

Abbildung 5.15.: Demonstrierung aller Vereinigungsmöglichkeiten der drei Statusmengen von Operationen

Der folgende Abschnitt beschreibt den Algorithmus der Anwendung von Rückmeldungen. Das Listing 5.9 zeigt Algorithmus der Anwendung von Rückmeldungen als Pseudocode. Zunächst iteriert der Algorithmus über alle ProductionOrders und bezeichnet ein Element aus der Menge als PrO. Im Schleifenrumpf bestimmt der Algorithmus zunächst den Status von PrO und speichert den ermittelten Status in der Variable state. Ein Switch-Konstrukt startet. Der folgende Abschnitt beschreibt den Fall (case) „Erstellt“. Zunächst löscht der Algorithmus alle Operationen von PrO und erstellt eine Variable g als leere Menge. Der Algorithmus fügt zur Menge g PrO, das Elter von PrO (StockExchangeDemand) und die Eltern des Elter (StockExchangeProviders), die nur ein Kind haben (StockExchangeDemand), hinzu. Außerdem fügt der Algorithmus die Kinder von PrO (ProductionOrderBoms) und deren Kinder (StockExchangeProviders)) zu g hinzu. Danach löscht der Algorithmus alle Elemente in g und alle Pfeile an dessen Knoten. Der Fall „Erstellt“ endet.

Im Fall (case) „In Bearbeitung“ ist nichts zu tun; die ProductionOrder PrO und deren Umgebung bleibt erhalten. Der folgende Abschnitt beschreibt den Fall (case) „Beendet“. Zunächst archiviert der Algorithmus die Operationen der PrO. Dann erstellt der Algorithmus die Variable g als leere Menge. Der Algorithmus fügt zu g PrO und ihre Kinder (ProductionOrderBoms). Danach entfernt der Algorithmus die Pfeile, die zu PrO zeigen und die weg von den ProductionOrderBoms zeigen. Danach archiviert der Algorithmus PrO und die verbundenen ProductionOrderBoms inklusive der Pfeile dazwischen. Der Fall „Beendet“ endet. Das Switch-Konstrukt endet.

Danach entfernt der Algorithmus alle Pfeile, die auf den StockExchangeProvider zeigen. Dann entfernt der Algorithmus alle Pfeile, die von StockExchangeDemands weg zeigen. Es bleiben die Pfeile zwischen StockExchangeProvidern und StockExchangeDemands übrig.

Danach iteriert der Algorithmus über alle StockExchangeDemands, die beendet und geschlossen sind und nennt ein Element der beendeten und geschlossenen StockExchangeDemands sed. Im Schleifenrumpf archiviert der Algorithmus zunächst sed und seine Eltern (StockExchangeProviders), die nur ein Kind haben, und die Pfeile zwischen sed und seinen Eltern (StockExchangeProviders), die nur ein Kind haben. Der Algorithmus darf nur

StockExchangeProvider mit einem Kind archivieren, denn sonst stimmt die Mengenzuordnung für die anderen StockExchangeDemands nicht mehr. Der Schleifenrumpf endet.

Danach archiviert der Algorithmus alle StockExchangeProvider ohne Kinder. Dann archiviert der Algorithmus beendete CustomerOrderParts. Danach archiviert der Algorithmus beendete PurchaseOrderParts. Dann archiviert der Algorithmus CustomerOrders ohne CustomerOrderParts. Danach archiviert der Algorithmus PurchaseOrders ohne PurchaseOrderParts. Der Algorithmus endet.

```
1  foreach PrO in productionOrders
2
3      state := Bestimme den Status der PrO.
4      switch (state)
5          case "Erstellt":
6              Lösche alle Operationen der PrO.
7              g := {}
8              Füge folgende Elemente zu g hinzu: PrO, Elter von
9                PrO (StockExchangeDemand) und deren
10               Eltern(StockExchangeProviders), die nur ein Kind haben
11               (StockExchangeDemand), Kinder der PrO (ProductionOrderBoms)
12               und deren Kinder (StockExchangeProviders))
13             Lösche alle Elemente in g und alle Pfeile an dessen Knoten.
14         case "In Bearbeitung":
15             Nichts zu tun
16         case "Beendet":
17             Archiviere die Operationen der PrO
18             g := {}
19             Füge folgende Elemente zu g hinzu: PrO und ihre Kinder
20               (ProductionOrderBoms).
21             Entferne Pfeile, die zu PrO zeigen und die weg von den ProductionOrderBoms
22               zeigen.
23             Archiviere PrO und die verbundenen ProductionOrderBoms inklusive Pfeile
24               dazwischen.
25
26 Entferne alle Pfeile auf StockExchangeProvider zeigend.
27 Entferne alle Pfeile von StockExchangeDemands weg zeigend.
28 // --> übrig beleiben Pfeile zwischen StockExchangeProvider und StockExchangeDemands
29
30 foreach sed in beendeten und geschlossenen StockExchangeDemands
31     Archiviere sed und seine Eltern (StockExchangeProvider), die nur ein Kind haben,
32       und die Pfeile dazwischen.
33
34 Archiviere alle StockExchangeProvider ohne Kinder.
35
36 Archiviere beendete CustomerOrderParts.
37 Archiviere beendete PurchaseOrderParts.
38
39 Archiviere CustomerOrders ohne CustomerOrderParts.
40 Archiviere PurchaseOrders ohne PurchaseOrderParts.
```

Listing 5.9: Algorithmus der Anwendung von Rueckmeldungen als Pseudocode

Nach der Anwendung der Rückmeldungen ist es möglich, dass die T_ProviderToDemand- und die T_StockExchange-Tabelle des Transaktionsdatenarchivs nicht mehr konsistent sind. Denn es könnte ein ProviderToDemand x geben, der von einem StockExchangeProvider auf einen StockExchangeDemand zeigt. Die Variable x ist möglicherweise noch nicht archiviert, da x noch in der Neuplanung für die Mengenzuordnung in den Transaktionsdaten benötigt wird und in den Transaktionsdaten auf weitere StockExchangeDemands

zeigt.

5.8.3. Ergebnis der Anwendung von Rückmeldungen für das Tischbeispiel

Der Standardzyklus geht von 0 bis 1440 (in Minuten). Da alle Aufträge und Operationen des Tischbeispiels innerhalb des Standardzyklus liegen, haben alle Aufträge und Operationen den Status „Beendet“. Abb. 5.16 zeigt das Ergebnis der Anwendung der Rückmeldungen für das Tischbeispiel als DemandToProvider-Graph der Transaktionsdaten.

Da alle Aufträge und Operationen den Status „Beendet“ haben, sind folglich alle Entitäten aus dem DemandToProvider-Graphen außer den offenen StockExchangeDemands und deren Eltern (StockExchangeProvider) archiviert. Es sind noch zwei Anschraubplatten, zwei Stahlrohre, zwei Filzgleiter und 68 Schrauben übrig.

Abb. 5.17 zeigt den DemandToProvider-Graphen des Transaktionsdatenarchivs. Die StockExchangeDemands Tisch, Tischplatte und Tischbein sind im Archiv, da sie geschlossen sind, denn sie haben keine Stückzahl übrig. Die beiden ProductionOrders Tisch und Tischbein sind ebenfalls im Archiv, da ihre Operationen beendet sind.

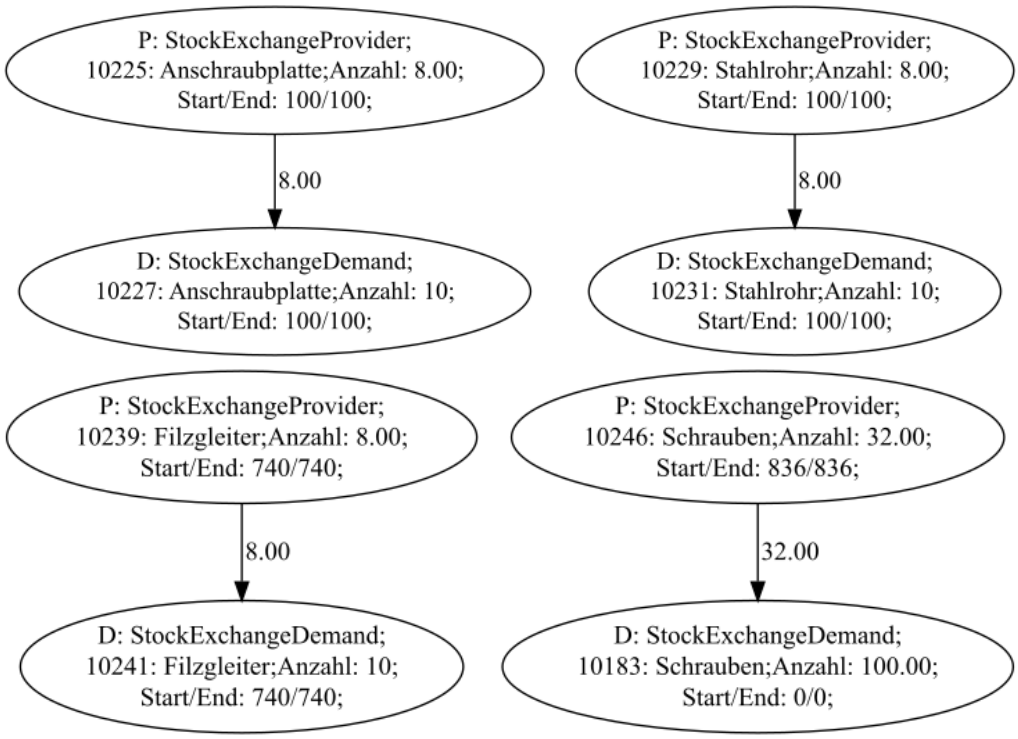

Abbildung 5.16.: Ergebnis der Anwendung der Rückmeldungen auf die Transaktionsdaten für das Tischbeispiel

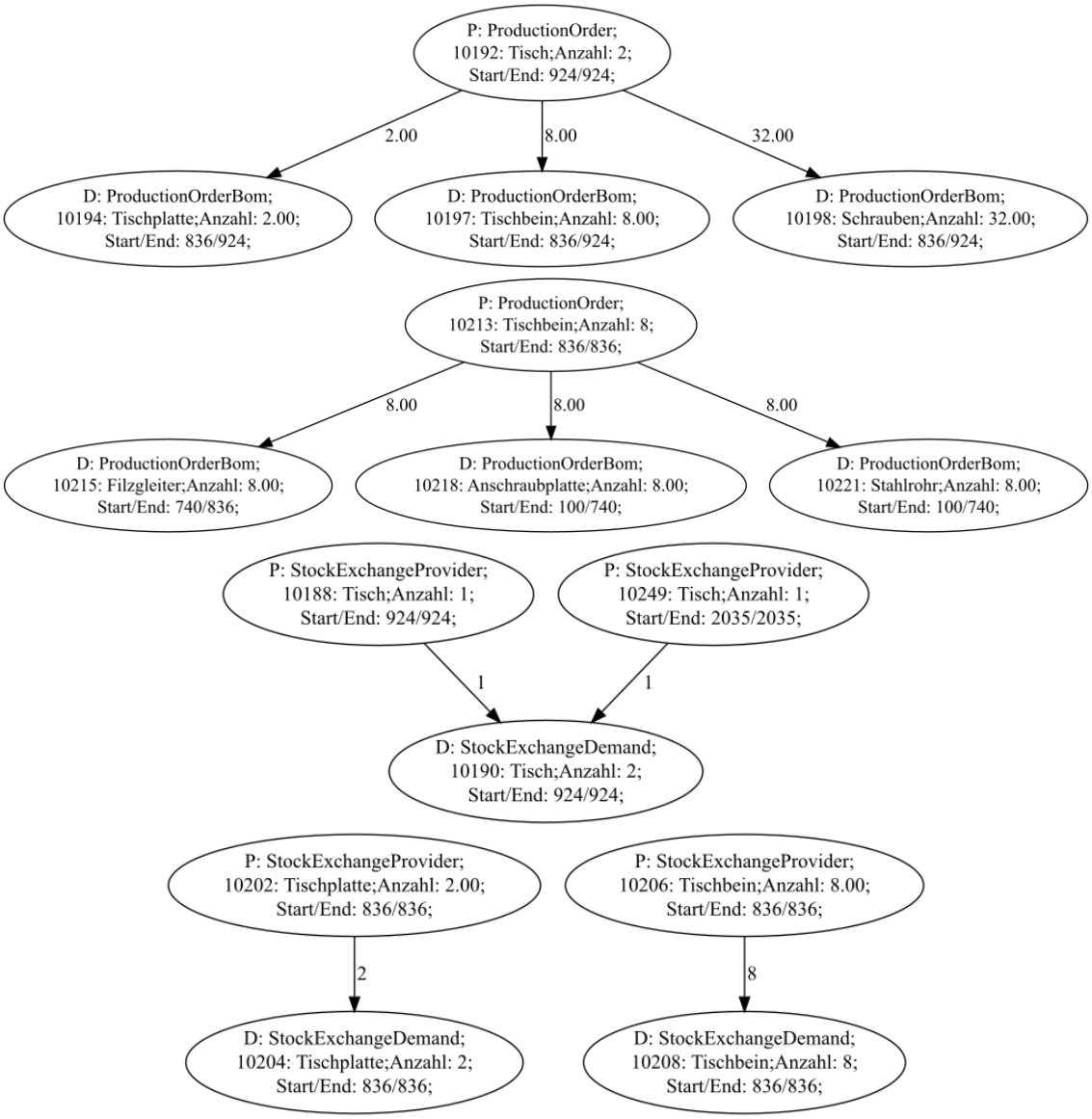

Abbildung 5.17.: Ergebnis der Anwendung der Rückmeldungen auf das Transaktionsdatenarchiv für das Tischbeispiel

5.8.4. Ausblick

Der implementierte Algorithmus für die Erstellung von Rückmeldungen ist trivial. Entitäten bekommen den Status „In Bearbeitung“, wenn die Startzeit aus der Planung kleiner gleich der Endzeit des Simulationsintervalls ist. Entitäten bekommen den Status „Beendet“, wenn die Endzeit aus der Planung kleiner gleich der Endzeit des Simulationsintervalls ist. Wünschenswert wäre es, dass statt der Endzeit aus der Planung eine um die Endzeit der Planung schwankende Zeit zurückgemeldet wird. Die Schwankung um die Endzeit würde die Realität besser abbilden.

5.9. Umsetzung der nicht funktionalen Anforderungen

5.9.1. Umsetzung der Anforderung an die Datenstrukturen

Die ZPP nutzt die Datenstrukturen der SSOP, allerdings passte der Autor die Datenstrukturen im Rahmen diverser Absprachen an. Kapitel 5.1 beschreibt die angepassten Datenstrukturen.

5.9.2. Umsetzung der Anforderung an die Integration in das vorhandene Projekt der selbstorganisierenden Produktion

Die Integration der ZPP steht noch aus. Ein Entwickler kann die ZPP wie folgt in die SSOP integrieren. Zunächst muss der Entwickler die aktualisierten Datenstrukturen des umgebenden Projektes der ZPP übernehmen (Modul DB). Dann muss der Entwickler das Modul ZPP vollständig übernehmen. Damit ist die Integration grundsätzlich abgeschlossen. Weitere Anpassungen wie das Heraustrennen von Paketen z. B. der Simulation kann ein Entwickler durch Orientierung an der Paketstruktur des Kapitels 5.4 vornehmen.

5.9.3. Umsetzung der Anforderung an das Simulationsmodell

Die ZPP nutzt das Simulationsmodell der SSOP im Paket „SimulationCore“ für die Erstellung von Kundenaufträgen. Die Simulation der Rückmeldungen setzt nicht auf dem Simulationsmodell der SSOP auf, sondern basiert auf dem im Kapitel 5.8.1 beschriebenen Algorithmus. Damit die ZPP für die Simulation der Rückmeldungen das Simulationsmodell der SSOP nutzt, sind weitere Arbeiten zu tun und waren im Rahmen dieser Arbeit nicht schaffbar.

5.9.4. Umsetzung der Anforderung an die Performanz

Der Autor hat bei der Entwicklung der ZPP stets die Performance berücksichtigt. Die wichtigste Maßnahme ist die Vermeidung von Datenbankrufen bei jeder Erstellung einer Entität , Entitätsänderung oder beim Lesen von Masterdaten. Der Autor nutzt stattdessen einen Cache, eine Art Spiegelung der nötigen Entitäten im Hauptspeicher. Damit die ZPP dennoch skaliert, ist eine zweite Datenbank für die Transaktionsdaten als Archiv für die Auslagerung aller erledigten Entitäten notwendig, sodass die ZPP den Hauptspeicher nicht soweit beansprucht, dass das Betriebssystem die Daten auf die Festplatte auslagern muss bzw. das Betriebssystem einfriert. Weitere Maßnahmen sind Codeoptimierung im Rahmen der Performancestudie mit einem Profiler.

5.9.5. Umsetzung der Anforderung an die Struktur: Objektgymnastik

Grundsätzlich wendet der Autor die Regeln der Objektgymnastik bei der Umsetzung der ZPP an. Im folgenden Unterkapitel geht der Autor auf jede der in Kap. 2.9 beschriebenen Regeln ein.

1. Der Autor wendet die zweite Regel inkonsequent an. Das Hauptproblem ist, dass zu viele Methoden entstehen würden und eine Klasse schnell unübersichtlich werden würde. Dann würde sofort die sechste Regel greifen und es würden viele Klassen entstehen. Konsequent angewendet würde die zweite Regel den Prototypen massiv aufblähen.

2. Die zweite Regel wendet der Autor nicht an. Zunächst versuchte der Autor die zweite Regel anzuwenden. Nachdem das dritte Mal Seiteneffekte durch die gleiche Art von Programmierfehler auftraten, verwarf der Autor die Regel. Das Listing 5.10 veranschaulicht das Problem. Zunächst programmiert der Autor eine gewöhnliche Prozedur (Funktion ohne Rückgabewert); das Else umgeht der Autor durch den vorgeschlagenen Early-Return.

   Eine Weile später erweitert der Autor die Prozedur um einen weiteren Funktionsaufruf. Es treten Seiteneffekte auf, denn die Prozedur ruft die Funktion „c“ im Else-Zweig auf. Es ist kein Else-Zweig zu sehen, er ist implizit programmiert. Beim nächsten Ansehen des Codeabschnitts muss dem Autor also bewusst sein, dass es sich um ein Else-Zweig handelt; es steht nicht explizit da. Was der Autor eigentlich beabsichtigt hatte, zeigt die Abb. im zweiten Teil ab Zeile 17. Der Autor wollte die Prozedur erweitern, nach Jeff Bay hätte er die Erweiterung nach dem Aufruf der Prozedur tätigen müssen. Bei konsequenter Anwendung der zweiten Regel muss der Programmierer also stets sich überlegen, ob ein impliziter If-Else Abschnitt in einer Prozedur vorhanden sein könnte. In der Praxis des schnellen Prototypings sind implizite Programmierabschnitte inakzeptabel.

3. Der Autor wendet die dritte Regel konsequent an. Die Vorteile sind offensichtlich und nach wenigen versuchen praktisch bemerkbar. Für numerische Primitive legt der Autor ein Interface „INumericPrimitive“ an (implementiert für double und int) und legt sechs Wrapper an.

4. Die vierte Regel steht im Widerspruch zur achten und dritten Regel (Wrapper), denn man muss innerhalb einer Klasse auf die Werte der gewrappten Entitäten kommen. Man benötigt also mind. zwei Punkte. Der Autor versucht dennoch die vierte Regel anzuwenden. Der Autor hat mit einem Unittest „TestNoMoreThanTwoPoints“ in der Klasse „TestStructure“ die Möglichkeit geschaffen, die Regel vollständig umzusetzen. Allerdings ist der Unittest deaktiviert, weil der Autor hin- und wieder gegen die Regel verstößt. In der Kürze der Zeit ist das vollständige Umsetzen nicht möglich.

5. Der Autor wendet die fünfte Regel immer an. Der Grund ist, dass die Regel sich mit der Einführung und Verbreitung von Java in den modernen Programmiersprachen etabliert hat und der Autor mit der Regel Programmieren gelernt hat. Die einzige Ausnahme ist die Umsetzung des Giffler-Thompson-Algorithmus, denn damit die Implementierung mit der Vorlage vergleichbar bleibt, wendet der Autor die Notation von Herrn Zäpfel an.

6. Der Autor wendet die sechste Regel aus Übersichtsgründen meistens an. Der Autor hat mit einem Unittest „TestKeepEntitiesSmall“ in der Klasse „TestStructure“ die

Möglichkeit geschaffen, die Regel vollständig umzusetzen. Allerdings ist der Unittest deaktiviert, weil der Autor hin- und wieder gegen die Regel verstößt. In der Kürze der Zeit ist das vollständige Umsetzen nicht möglich.

7. Grundsätzlich wendet der Autor die siebte Regel an. Als Beispiel seien hier sämtliche Wrapper für die T_*Entitäten genannt, die nur die Entität nutzen. Würde man versuchen die siebte Regel auf die Entitäten der Datenbank anzuwenden (die Entitäten sind vorgegeben), würde das gesamte Projekt massiv aufgebläht werden. Davon kann der Autor nur abgeraten.

8. Der Autor wendet achte Regel an und nutzt grundsätzlich Collection-Wrapper. Der Autor führt das Interface „ICollectionWrapper“ ein, dass im Normalfall die Collection-Wrapper implementieren. Daher kann der Autor mit Sicherheit sagen, dass es mindestens 17 Collection-Wrapper gibt.

9. Die Anwendung der neunten Regel (Tell, don't ask) kann nicht immer angewendet werden. Die neunte Regel steht im Widerspruch zur achten Regel und zur dritten Regel (Wrapper), denn man muss innerhalb einer Klasse auf die Werte der gewrappten Entitäten kommen. Sämtliche Logik in die Wrapper zu packen würde bedeuten, keine lose Kopplung realisieren zu können. Deswegen ist es stets kontextabhängig, ob die Regel überhaupt angewendet werden sollte.

```
1  // Regel 2 angewandt: ohne else
2
3  // Ursprüngliche Version
4  procedure
5      if x = 1
6          a()
7      b()
8
9  // Programmerweiterung
10 procedure
11     if x = 1
12         a()
13     b()
14     c()
15
16 // --------------------------------
17 // Regel 2 NICHT angewandt: mit else
18
19 // Ursprüngliche Version
20 procedure
21     if x = 1
22         a()
23     else
24         b()
25
26 // Programmerweiterung
27 procedure
28     if x = 1
29         a()
30     else
31         b()
32     c()
```

Listing 5.10: Anwendung der Regel 2 und durch einen Programmierfehler ausgelöste Seiteneffekte

5.10. Beschreibung der verwendeten Bibliotheken

Die ZPP verwendet folgende Bibliotheken.

- Die ZPP verwendet das vorgegebene Entity-Framework. Das Entity-Framework ist ein Framework für die objektrelationale Abbildung (ORM).[10]
- Die ZPP verwendet die Bibliothek „Json.NET“. „Json.NET“ kann ein C#-Objekt in ein JSON-String serialisieren und umgekehrt.[11]
- Die ZPP verwendet die Bibliothek „OptimizedPriorityQueue“. „OptimizedPriorityQueue“ ist die Implementierung einer Prioritätswarteschlange, denn C# enthält keine Implementierung einer Prioritätswarteschlange.

Das Entity-Framework ist durch die SSOP vorgegeben.

[10] Microsoft, *Entity Framework*.
[11] Newtonsoft, *Json.NET Dokumentation*.

6. Verifikation der zentralen Produktionsplanung: Funktionstests

6.1. Beschreibung des betriebswirtschaftlichen Testszenarios für die zentrale Planung

Die Simulation soll innerhalb des Zeitintervalls Null und 20160 erfolgen, das soll auf zwei Arten geschehen. Die Simulation soll für jeden Tag einzeln planen, dann gibt es nach jedem Tag Rückmeldungen. Und die Simulation soll den gesamten Zeitraum planen, dann gibt es einmalig am Ende Rückmeldungen. Die Tests führen beide Szenarien aus und überprüfen die zu beschreibenden Bedingungen. In diesem Kapitel ist mit Test immer ein Integrationstests gemeint. Die ZPP führt also mind. ein Planungszyklus aus. Ein Unittest testet eine einzelne Routine. Ein Integrationstest testet ein gesamtes Programm auf eine korrekte Funktionsweise.[1]

Ein drittes Testszenario führt den Test aus, nach dem der MRP-Lauf (MRP2) abgeschlossen ist. Die Simulation wendet also keine Rückmeldungen an und führt nur einen Zyklus aus. Die Beschränkung auf einen Zyklus sorgt dafür, dass die Tests keine Neuplanungen durchführen und die zu prüfenden Bedingungen nach mehreren Zyklen möglicherweise nicht eingehalten werden. Tests nutzen das dritte Testszenario nur in Ausnahmefällen innerhalb der Verifikation, wenn das aus technischen Gründen nicht anders möglich ist. Die Testszenarien nutzen das Tischbeispiel und die Trucks.

6.2. Beschreibung der Methodik

Die entwickelte ZPP nutzt das Test-Framework Xunit, um die in C# geschriebenen Tests auszuführen. Travis-CI führt nach jedem Push von Commits (Git) einen Test-Job aus, der die Tests in einer virtuellen Maschine innerhalb einer gehosteten Umgebung ausführt (Cloud). Travis-CI ist ein gehosteter Continuous-Integration-Dienst, der Softwareprojekte baut und deren Tests ausführt.[2] Ein Testablauf ist wie folgt. Zunächst führt ein Test eine Simulation nach dem definierten Testszenario aus. Dann prüft der Test die angegebenen Bedingungen. Sind alle Bedingungen erfüllt, ist der Test erfolgreich, sonst ist er fehlgeschlagen. Abb. 6.1 zeigt einen erfolgreichen Test-Job auf Travis-CI.

[1] Vgl. ISO/IEC/IEEE, *24765*, S. 231, 490.
[2] CI, *Travis CI*.

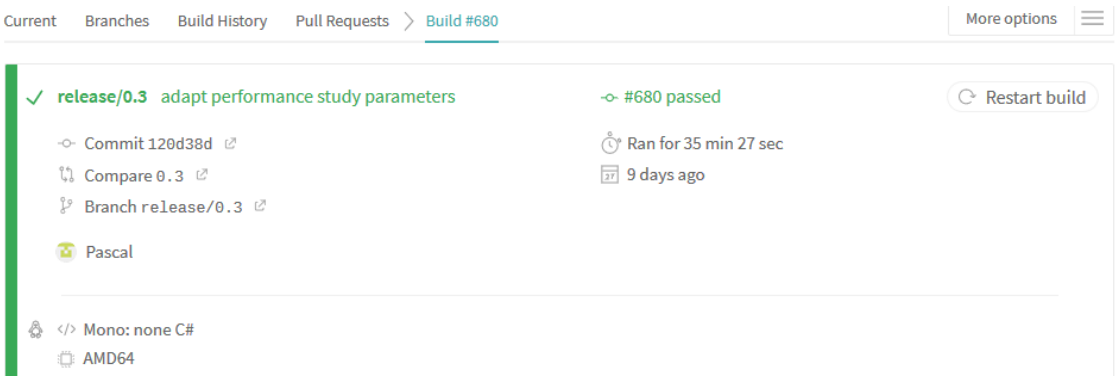

Abbildung 6.1.: Erfolgreicher Test-Job auf Travis-CI

6.3. Test der Materialbedarfsplanung

In den folgenden Unterkapiteln nutzt der Autor die Begriffe des Datenmodells, um sämtliche Bedingungen übersichtlich und exakt beschreiben zu können.

Der MRP1-Algorithmus ordnet Mengen zwischen Demands und Providern mit DemandToProvidern bzw. ProviderToDemands zu. Jeder DemandToProvider bzw. ProviderToDemand enthält die Id eines Demands (DemandId), eines Providers (ProviderId) und die Mengenzuordnung. Für jede DemandId bzw. ProviderId muss der Test das dazugehörige Objekt aus dem Cache bzw. der Datenbank holen.

Zunächst prüft der Test „TestMrp1“ in der Klasse „VerifyMrp1“, dass Demands und Provider nur gemäß Spezifikation 5.1 miteinander verbunden sind. Ein weiterer Test prüft, dass die Mengen der DemandToProvider, ProviderToDemands und deren Demands und Provider folgenden Bedingungen genügen.

Es muss mindestens einen DemandToProvider bzw. ProviderToDemand geben. Für jeden DemandToProvider muss gelten:

- Die Menge des DemandToProvider muss größer als Null sein.
- Die Menge seines Demands und Providers muss größer als Null sein.
- Ist sein Demand eine ProductionOrderBom, dann muss die Menge seiner ProductionOrderBom gleich der Menge seines StockExchangeProviders sein.
- Ist sein Demand eine StockExchangeDemands, dann muss die Menge seines StockExchangeDemands gleich der Menge seines PurchaseOrderParts oder seiner ProductionOrder sein. Denn die Losgröße entspricht in dieser Arbeit der Bestellgröße und die Losgröße steckt im StockExchangeDemand.
- Aus den vorigen Bedingungen folgt: Die Menge seines Demands muss gleich der Menge seines Providers sein. Die Menge des DemandToProvider muss gleich der Menge seines Demands bzw. Providers sein.

Für jeden ProviderToDemand muss gelten:

- Die Menge des ProviderToDemand muss größer als Null sein.
- Die Menge seines Demands und Providers muss größer als Null sein.
- Ist sein Provider StockExchangeProviders, dann muss die Menge seines StockExchangeProviders größer gleich der Menge des ProviderToDemands sein.

- Zwischen ProductionOrder und ProductionOrderBom gibt es keine Bedingung im Bezug auf die Menge.

6.4. Test der Kapazitätsplanung gegen unbegrenzte Kapazität

Folgende Bedingungen prüft der Test der Kapazitätsplanung gegen unbegrenzte Kapazität anhand der Transaktionsdaten als auch anhand des Transaktionsdatenarchivs.

- Alle ProductionOrderOperations sind nach ihrem Start ohne Unterbrechung bis zum Ende durchzuführen. Daraus folgt, dass Endzeit minus Startzeit der definierten Dauer in M_Operation entsprechen muss.

Die folgenden Bedingungen kann der Test der Kapazitätsplanung gegen unbegrenzte Kapazität nur im dritten Testszenario (ohne Anwendung von Rückmeldungen und nur ein Zyklus) testen, denn die Anwendung von Rückmeldungen löscht die Verbindungen zwischen den ProductionOrderBoms und ihren Kindern (StockExchangeProvider) und löscht die Verbindungen zwischen ProductionOrders und seinen Eltern (StockExchangeDemands). Somit könnte der Test z. B. den Auftragsoperationsgraphen nicht mehr aufbauen und keine Beziehungen zwischen ProductionOrders und StockExchanges herstellen.

Die folgenden Bedingungen prüft der Test anhand des Auftragsoperationsgraphen, den er aus der Datenbank bzw. aus dem Cache aufbaut. Seien in diesem Kapitel zur Vereinfachung alle Elemente des Auftragsoperationsgraphen (Aufträge und Operationen) Knoten genannt. Vorgänger und Nachfolger und die Pfeilrichtungen sind so, wie es für den Auftragsoperationsgraphen definiert ist.

- Die Startzeit eines Knotens muss größer gleich der Endzeit seiner Nachfolger sein. Die Bedingung gilt nicht für CustomerOrderParts, deren Startzeit kann kleiner als sein Nachfolger sein, denn es kann sein, dass die Startzeit nicht eingehalten werden kann.

- Zur geplanten Startzeit aus der zweiten Rückwärtsterminierung einer Operation muss das benötigte Material vorliegen. Der Test „TestEveryOperationHasNeededMaterialAtStartBackward“ führt die Prüfung durch, dass zur geplanten Startzeit das benötigte Material vorliegt.

Außerdem führt die Klasse „TestScheduling“ weitere Prüfungen gegen einen Zyklus aus:

- Prüfe, ob Start- und Endzeit ungleich NULL ist. Das Einhalten der Bedingung stellt sicher, dass die ZPP die Aufträge bzw. Operationen überhaupt terminiert.

- Prüfe, ob Start- und Endzeit negativ sind, wenn ja, dann hat die ZPP die Vorwärtsterminierung nicht ausgeführt.

- Prüfe, dass die Übergangszeit vor einer Operationen mind. das dreifache der Dauer der vorherigen Operation ist. Damit ist sichergestellt, dass jede Operation eine Übergangszeit hat.

6.5. Test der Kapazitätsplanung gegen begrenzte Kapazität

Ein zulässiger Plan der Maschinenbelegung ist durch folgende Bedingungen definiert:[3][4]

[3] Vgl. Zäpfel, *Moderne Heuristiken der Produktionsplanung*, S. 6.
[4] Vgl. Corsten, *Produktionswirtschaft*, S. 488.

1. Zunächst muss der Test prüfen, dass jede Operation geplant ist, dass heißt in jeder Operation muss eine Maschine, eine Startstart und eine Endzeit gesetzt sein. Für die Maschine reicht eine ungleich NULL Prüfung, für die Start- und Endzeit muss der Test prüfen, dass die Startzeit einer Operation ungleich der Endzeit einer Operation ist, denn es handelt sich um int-Werte, also ein Primitiv, das immer einen Wert besitzt.

2. Jede ProductionOrder muss eine bestimmte Anzahl und Reihenfolge an Maschinen durchlaufen.

3. Jede ProductionOrder wird auf jeder Maschine genau einmal bearbeitet.

4. Keine ProductionOrder kann gleichzeitig auf mehreren Maschinen bearbeitet werden.

5. Keine Maschine kann gleichzeitig mehrere ProductionOrderOperations bearbeiten, d.h., zu jedem Zeitpunkt darf eine Maschine nur eine Operation ausführen. Der Test „TestJobShopScheduling“ führt die Prüfung durch Aufrufen der Methode „VerifyEveryMachineHasOnlyOneOperationAtAnyTime“ im Test aus.

6. Alle ProductionOrderOperations sind nach ihrem Start ohne Unterbrechung bis zum Ende durchzuführen. Daraus folgt, dass Endzeit minus Startzeit der definierten Dauer in M_Operation entsprechen muss. Zur Zeit prüft der Test nur, dass die Endzeit minus Startzeit der Dauer der ProductionOrderOperation entspricht. Es ist zur Zeit mit dem Datenmodell nicht möglich, von der ProductionOrderOperation auf die dazugehörige M_Operation zu schließen. Wenn die Dauer in der ProductionOrderOperation falsch gesetzt ist, kann ein Test das nicht überprüfen und müsste in einem gesonderten Unittest überprüft werden.

7. Die Maschinen stehen zur Bearbeitung der ProductionOrders uneingeschränkt zur Verfügung. Die Bedingung ist in dieser Arbeit nicht der Fall, durch die Neuplanung kann es ab dem zweiten Zyklus dazu kommen, dass Maschinen zum Intervallstart belegt sind und erst später zur Verfügung stehen. Die Prüfung der fünften Bedingung stellt sicher, dass es dennoch nicht zu Mehrfachbelegungen kommt. Der Algorithmus der Maschinenbelegung ist entsprechend angepasst.

Die aufgezählten Bedingungen prüft der Test „TestJobShopScheduling“ der Klasse „VerifyJobShopScheduling“ durch Aufruf der Methode „VerifyEveryOperationIsCorrectlyPlanned“ soweit nicht anders erwähnt.

Es gibt weitere Bedingungen, die bereits durch das Datenmodell garantiert sind und nicht getestet werden müssen:

- Bearbeitungs-, Rüst- und Transportzeiten sind bekannt und konstant: Bearbeitungszeiten sind in der Tabelle M_Operation definiert. Rüst- und Transportzeiten sind in dieser Arbeit nicht vorgesehen.

- Im Gegensatz zum Job-Shop-Fall kann es für einen Maschinentyp mehrere Maschinen geben. Der Algorithmus ist, wie im Kapitel 5.7 beschrieben, entsprechend angepasst.

Es gibt weitere Bedingungen, die bereits durch den Entwurf garantiert sind und nicht getestet werden müssen:

- Die ProductionOrders mit ihren ProductionOrderOperations stehen zu Beginn der Maschinenbelegung in Menge und Startzeiten eindeutig fest, wobei die Startzeiten aus

einer Durchlaufterminierung stammen. Die ZPP stellt die Bedingung sicher, indem zuerst die Materialbedarfsplanung, dann die Durchlaufterminierung und abschließend die Maschinenbelegung durchführt, d.h. zu Beginn der Maschinenbelegung sind die Mengen und Startzeiten vorhanden.

Wenn eine Operation startet, muss das benötigte Material vorliegen. Die Bedingung kann ebenso wie im vorherigen Kapitel nur im dritten Testszenario laufen (ohne Anwendung von Rückmeldungen und nur ein Zyklus). Denn die Anwendung von Rückmeldungen löscht die Verbindungen zwischen den ProductionOrderBoms und ihren Kindern (StockExchangeProvider). Ein Test kann somit nicht mehr nachvollziehen, welche StockExchangeProvider den ProductionOrderBoms zugeordnet sind. Der Test „TestEveryOperationHasNeededMaterialAtStart“ führt die Prüfung durch, dass jede Operation erst startet, wenn das benötigte Material vorliegt.

6.6. Test der Neuplanung unter Verrechnung von Rückmeldungen

Grundsätzlich muss es einen Test für das Erstellen von Rückmeldungen und das Anwenden von Rückmeldungen geben. Es ist nicht möglich den Status aus den Planungszeiten zu folgern und entsprechend zu testen, denn die Erstellung von Rückmeldungen kann um die Endzeit schwanken. Z. B. kann die Endzeit einer ProductionOrderOperation vor oder nach dem geplanten Zeitpunkt liegen. Die Anwendung der Rückmeldungen kann getestet werden.

Folgende Bedingungen prüft der Test auf den Transaktionsdaten (und nicht auf dem Archiv) für die Anwendung von Rückmeldungen:

- Ein CustomerOrderParts darf kein Kind haben und darf nicht beendet sein.
- Ein PurchaseOrderPart darf kein Elter haben und darf nicht beendet sein.
- Für ein CustomerOrder muss mind. ein CustomerOrderPart vorhanden sein.
- Für eine PurchaseOrder muss mind. ein PurchaseOrderPart vorhanden sein.
- Ein StockExchangeProvider muss mind. ein Kind haben.
- Ein StockExchangeDemand darf nicht beendet und geschlossen sein.
- Eine ProductionOrder darf nicht beendet sein und für eine ProductionOrder muss es mind. eine Operation geben.
- Für jede ProductionOrderBom muss die dazugehörige Operation vorhanden sein.
- Für jeden DemandToProvider und ProviderToDemand müssen die dazugehörigen Demands und Provider existieren.

Die Bedingungen prüft der Test „TestApplyConfirmations“ in der Klasse „VerifyApplyConfirmations“.

Folgende Bedingungen prüft der Test auf dem Transaktionsdatenarchiv für die Anwendung von Rückmeldungen:

- Ein CustomerOrderParts darf kein Kind haben und muss beendet sein.
- Ein PurchaseOrderPart darf kein Elter haben und muss beendet sein.

- Für ein CustomerOrder muss mind. ein CustomerOrderPart vorhanden sein.
- Für ein PurchaseOrder muss mind. eine PurchaseOrderPart vorhanden sein.
- Ein StockExchangeProvider muss mind. ein Kind haben.
- Ein StockExchangeDemand muss beendet und geschlossen sein.
- Eine ProductionOrder muss beendet sein und für eine ProductionOrder muss es mind. eine Operation geben.
- Für jede ProductionOrderBom muss die dazugehörige Operation vorhanden sein.
- Für jeden DemandToProvider und ProviderToDemand müssen die dazugehörigen Demands und Provider existieren.

  Die Bedingungen prüft der Test „TestApplyConfirmationsArchive“ in der Klasse „VerifyApplyConfirmations“.

6.7. Test der Struktur

Für die Objektgymnastik hat der Autor soweit möglich Tests geschrieben. Folgende Regeln lassen sich durch Tests überprüfen.

- Regel Nr. 2: Nutze nicht das „Else“-Schlüsselwort.
- Regel Nr. 4: Nutze nur einen Punkt pro Anweisung (statement).
- Regel Nr. 6: Alle Entitäten kleinhalten.

Die angegebenen Tests sind in der Klasse „TestStructure“ implementiert. Der Test der Regel Nr. 2 ist aus den genannten Gründen in Kapitel 5.9.5 deaktiviert. Der Test für die Regel Nr. 4 und Regel Nr. 6 sind entsprechend der in Kapitel 5.9.5 beschriebenen Toleranzen nicht aktiviert.

7. Empirische Studie zur Überprüfung der Performance der zentralen Produktionsplanung

7.1. Einführung

Die O-Notation beschreibt die Laufzeit von Algorithmen. Erstrebenswert ist eine Laufzeit O(n), denn dann wächst die benötigte Berechnungszeit mit der Eingabe linear. Die Komplexitätsklasse P umfasst alle Algorithmen, die eine polynomielle Laufzeit haben: $P = \bigcup_k O(n^k)$. Damit ist O(n) ebenfalls in P enthalten. Die ZPP muss in P liegen, sonst ist sie nicht auf einer deterministischen Maschine ausführbar und liegt in NP.[1] Allerdings kann eine Laufzeit in P nicht garantieren, dass das Laufzeitziel aus den Anforderungen erreicht wird, denn für größere k wächst die polynomielle Laufzeit sehr schnell.

7.2. Anforderungen an die Performancestudie

Die Laufzeit muss in Befehlszyklen gemessen werden. Es muss mindestens dreimal gemessen werden. Die Messung muss nach dem Warmlaufen der ZPP geschehen. Die Speichermessung muss den Maximalwert über der gesamten Laufzeit der ZPP ergeben. Außerdem kann der Speicherbedarf über die Zeit gemessen werden. Die Messungen müssen für mindestens zwei verschiedene Stücklisten ausgeführt werden. Es muss mindestens eine Optimierung durchgeführt werden.

Die Messungen müssen für zehn, 100 und 1000 Kundenaufträge durchgeführt werden. Außerdem muss eine Vermutung für die Laufzeit- und Speicherbelegung für 10000 und mehr Kundenaufträge aufgestellt werden. Die Simulationsdauer muss 20160 Minuten betragen, das entspricht 2 Wochen. Die Messungen müssen für das Simulationsintervall 1440 und 20160 Minuten betragen. Die Simulation muss also mind. einen bzw. 14 Zyklen ausführen.

7.3. Eingangsdaten für die Performancestudie

Kap. 5.2 erklärt die Eingangsparameter der ZPP. Statt die Messungen für zehn Kundenaufträgen durchzuführen, führt die Performancestudie die Messungen zwischen 100 und 1000 Kundenaufträgen linear aufsteigend in Hunderterschritten aus. Denn damit eine Software skaliert, muss die Laufzeit einer Software linear mit den Eingabedaten steigen (O(n)). Das Ziel der Skalierbarkeit lässt sich also am einfachsten visuell anhand einer linear eingeteilten x-Achse überprüfen.

Die Tabelle 7.1 beschreibt die Eingangsdaten der vier gewählten Fälle für die Performancestudie. Die Performancestudie nutzt das Tischbeispiel und die Trucks. Die Losgröße ist mit Zwei gering gewählt, bei der Produktion kundenspezifischer Produkte ist eine niedrige Losgröße allerdings realistisch.

Es erfolgt keine Kapazitätsanpassung an die Last, d.h. Aufträge kommen mit steigender Anzahl an Kundenaufträge immer später. In einer ausgeglichenen Kapazitätssituation ist die Simulation schneller, da nicht so viele Entitäten verworfen werden für die Neuplanung. Die Performancestudie behandelt also das Worst-Case-Szenario.

[1] Vgl. Sipser, *Introduction to the Theory of Computation*, S. 286.

Fall	Customer-OrderQuantity	LotSize	DbSetInitializer	Simulation-Maximum-Duration [m]	Simulation-Interval [m]	x-Achse
1	100-1000	2	Tischbeispiel	20160	20160	# Kundenaufträge
2	100-1000	2	Tischbeispiel	20160	1440	# Kundenaufträge
3	100-1000	2	Trucks	20160	1440	# Kundenaufträge
4	100-1000	2	Trucks	20160	20160	# Kundenaufträge
5	500	2	Tischbeispiel	20160	1440	# Zyklen
6	500	2	Trucks	20160	1440	# Zyklen

Tabelle 7.1.: Zu simulierende Fälle für die Performancestudie

7.4. Optimierung der ZPP

Zunächst muss der Autor die ZPP soweit optimieren, dass die ZPP 1000 Kundenaufträge in vertretbarer Zeit simuliert (eine Messung in unter zwei Stunden). Denn zunächst ist es nicht möglich, eine derart große Anzahl zu simulieren. Der Fokus bei der Entwicklung der ZPP lag zunächst darauf, die ZPP funktionell korrekt und vollständig den funktionellen Anforderungen genügend zu entwickeln.

Grundsätzlich ist die Skalierung eines Programmablaufs mit O(n) nur möglich, wenn entlang der n-Schritte jeder Schritt O(1) benötigt. In der Praxis heißt das, dass jeder Schritt im Verhältnis zu O(n) gegen O(1) gehen muss. Denn eine verkette For-Schleife hat per Definition O(n^2), nur muss das n im Verhältnis zu den n Schritten des Programmablaufs sehr klein sein. Folgende Schritte sind notwendig.

1. Das Persistieren des Cache nach jedem Zyklus ist sehr zeitaufwändig und lässt die Simulation nicht terminieren. Zunächst soll die ZPP die am Ende der Simulation die Daten persistieren. Doch es zeigt sich, dass die Persistierung des Entity-Framework einen Laufzeitfehler hat. Das Entity-Framework führt die vielen Inserts nicht in einer Transaktion durch, sondern pro Insert eine Transaktion. Das bedeutet, dass bei z.B. 1000 zu persistierenden Entitäten 1000 Transaktionen durchgeführt werden. Die Persistierung durch das Entity-Framework skaliert also nicht und ist im Rahmen der Performance-Studie unbrauchbar. Daher ist für die Performancestudie das abschließende Abspeichern in der Datenbank deaktiviert.

2. Das Halten aller Transaktionsdaten in einer Datenbank durch das vorgegebene Schema skaliert ebenfalls nicht. Der Cache wächst mit steigender Anzahl an Kundenaufträgen. Die ZPP muss also jedes Mal über einen prozentual steigenden Anteil an erledigten Entitäten iterieren. Der Autor führt daher eine Archivierungsdatenbank (und entsprechend einen Archivierungscache) ein, die vom Schema eine Spiegelung der Transaktionsdatenbank ist. Alle erledigten Entitäten verschiebt die ZPP in den Archivierungscache.

3. Sämtliche Zugriffe auf die Entitäten des Caches muss die ZPP in O(1) erledigen. Die Collections des Cache sind als Menge implementiert, der Zugriff über die Menge muss

über einen Index erfolgen. Außerdem muss der Autor für aggregierte Daten jeweils einen Index anlegen. Das betrifft folgende Zugriffe auf: DemandToProviders/ProviderToDemands über die DemandId und ProviderId, ProductionOrderBoms über die OperationId und ProductionOrderOperations über die ProductionOrderId.

4. Die Implementierung des gerichteten Graphen muss, statt auf Kanten, auf Knoten basieren. Jeder Knoten muss seine Vorgänger und Nachfolger enthalten. Das ermöglicht Zugriffe auf Vorgänger und Nachfolger von Knoten in O(1) statt O(n) bei der Kantenimplementierung. Da alle Algorithmen auf gerichteten Graphen basieren, ist der Wechsel von Kanten auf Knoten ein sehr großer Optimierungsschritt.

5. Der Autor unternimmt weitere kleinere Optimierungen wie der direkte Zugriff auf Felder statt über Methoden, wenn die Anzahl der Aufrufe weit überdurchschnittlich ist.

Mit den genannten Optimierungen ist die Performancestudie durchführbar, die ZPP skaliert nun zumindest innerhalb des Messbereichs so, dass die Ausführungszeit der Performancestudie vertretbar ist.

7.5. Methoden zur Durchführung der Performancestudie

Die Performancestudie ist in einem Integrationstest implementiert. Der Autor startet den Integrationstest für die Messungen mittels Visual Studio im normalen Modus, also nicht im Debug-Modus. Der Integrationstest startet eine Simulation wie sie im Kap. 5.3 beschrieben ist. Der Integrationstest führt also entsprechend der Eingangsdaten eine Anzahl von Zyklen aus. Während der Ausführung misst der Integrationstest die Ausführung der einzelnen Schritte des MRP-Laufs pro Zyklus. Außerdem misst der Integrationstest die gesamte Ausführungsdauer und Speicherbedarf pro Simulationszyklus. Die Messungen starten frühestens nach der Initialisierung der ZPP, dann ist die ZPP warmgelaufen und hat u.a. die Datenbanken erstellt.

Die Ausführungsdauer misst der Autor in Befehlszyklen (CPU-Cycles) anhand der Methode „QueryThreadCycleTime“ der „kernel32.dll“ in C#. Die Methode „QueryThreadCycleTime“ gibt die Anzahl an CPU-Zyklen, die vom übergebenen Thread genutzt wurden. Die Anzahl an CPU-Zyklen umfassen die Zyklen im User- und Kernelmodus.[2] CPU-Zyklen haben nichts mit den Ticks einer Systemuhr zu tun (kleinste Zeitauflösung).

Den Speicherbedarf misst der Autor mittels des Debugging-Modus von Visual Studio. Der Debugging-Modus zeigt den Speicherbedarf über die Zeit an. Ein Maximum ist so einfach ermittelbar. Allerdings hat der Debugging-Modus einer IDE natürlich großen Overhead, die Genauigkeit ist fraglich. Die Messungen sind also als obere Schranke zu betrachten.

Versuche, den Speicherbedarf nach jedem Zyklus innerhalb des Programms zu ermitteln, schlugen fehl, da der Garbage-Collector stets um den Messzeitpunkt herum einsetzte und somit die Messungen verzehrten und unbrauchbar machten. Der Grund für das Einsetzen des Garbage-Collectors mit starker Verzerrung ist, dass als letztes die Anwendung der Rückmeldungen im Zyklus stattfindet und und die Anwendung der Rückmeldung einen großen Anteil an Entitäten aus den Transaktionsdaten löscht. Ein Maximum war so nicht ermittelbar. Den Speicherbedarf innerhalb des Programms hat der Autor anhand des Feldes „WorkingSet64“ der Klasse „System.Diagnostics.Process“ in C#. Das Feld „WorkingSet64“

[2] Microsoft, *QueryThreadCycleTime function*.

gibt die momentane Größe der Speicherarbeitsmenge in Bytes an, die vom Prozess genutzt wird. Die Speicherarbeitsmenge ist die Menge der Speicherseiten des physikalischen Speichers, die für den Prozess sichtbar ist. Die Seiten sind für den Prozess ohne Page fault verfügbar.[3]

7.6. Ergebnisse der Messungen

Die Ergebnisse stellt der Autor anhand von Diagrammen dar. Die den Diagrammen zugrunde liegenden Zahlen sind im Anhang C zu finden. Die in diesem Kapitel erwähnten Fälle beziehen sich auf in Tabelle 7.1 eingeführten Fälle. Die Anzahl Zyklen berechnet sich aus MaximumDuration geteilt durch SimulationInterval. Allerdings kann es vorkommen, dass die ZPP mehr Zyklen benötigt, denn im letzten Zyklus könnten Kundenaufträge noch offen sein. Das kann kann an zu wenigen Maschinen oder an einem zu kurz gewählten Intervall liegen.

Für die ersten vier Fälle enthält jedes Diagramm eine Gerade zwischen dem ersten und dem letzten Messwert. Der Autor verwendet für die Gerade jeweils den Median aller drei Messungen. Die Gerade stellt das Ziel dar, höchstens lineares Wachstum der Laufzeit und des Speicherbedarfs mit wachsender Anzahl an Kundeaufträgen zu erreichen. Allerdings ist die richtige Interpretation wichtig. Liegen alle Messwerte zwischen dem ersten und letzten Messwert deutlich unter der Geraden, verhält sich die ZPP deutlich nichtlinear, die Laufzeit wächst also deutlich schneller. Ein Schwanken der Messwerte um die Gerade herum spricht für lineares Wachstum. Sind die meisten Messwerte zwischen dem ersten und letzten Messwert über der Geraden, spricht das für höchstens lineares Wachstum.

Eine Messung über alle sechs Fälle ist deutlich unter zwei Stunden laut Abb. 7.1 erledigt. Der fünfte bzw. sechste Fall sind implizit enthalten im zweiten bzw. dritten Fall. Allerdings enthält jede Zeile die Planung für 100, 200, ... und 1000 Kundenaufträge. Nur die Simulation der geforderten 500 Kundenaufträge dauert maximal fünf Minuten. Das Ziel ist also deutlich von der Laufzeit her unterschritten und damit die Anforderung erfüllt.

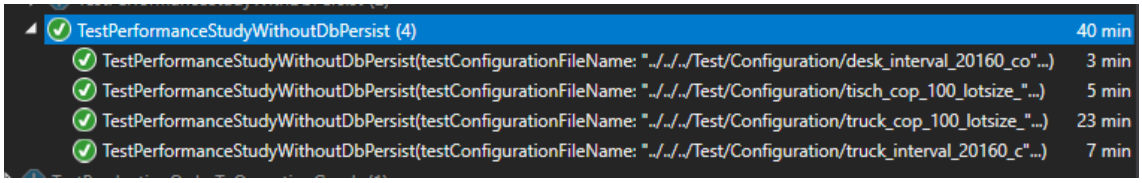

Abbildung 7.1.: Die Dauer einer Messung für Fall 1-4

Die Trucks sind deutlich komplexer als das Tischbeispiel. Die Messwerte für die Laufzeit sind für die Trucks größer. Daher betrachtet der Autor im übrigen Kapitel zunächst nur die Fälle für die Trucks. Für die übrigen Fälle befinden sich die Diagramme im Anhang C. Ebenso befinden sich die den Diagrammen zugrunde liegenden Zahlen im Anhang für Fall 1-4 C.4 und Fall 5-6 C.5. Tabellarische Zahlen für die Diagramme der Speichermessungen gibt es nicht. Visual Studio stellt die Diagramme nur als fertige Diagramme ohne Datengrundlage zur Verfügung.

[3] Microsoft, *Process.WorkingSet64 Property*.

7.7. Ergebnisse der Messung von CPU-Zyklen Fall 1-4

Abb. 7.2 zeigt die CPU-Zyklen für den dritten Fall. Die Messungen liegen zwischen dem Anfangs- und Endwert deutlich unter der Geraden. Das gilt ebenfalls für den vierten Fall Abb. 7.3. Die Laufzeit ist größer gleich O(n^2) und damit nichtlinear. Der fünfte Fall wird zeigen, welche Komponente(n) für die Nichtlinearität verantwortlich ist. Der Laufzeitunterschied zwischen dem dritten und dem vierten Fall beträgt etwa die doppelte Laufzeit. Es macht also einen deutlichen Unterschied, ob die ZPP alle Kundenaufträge in einem oder in vierzehn Intervallen simuliert, obwohl die Simulationsdauer gleich bleibt. Der Rechenaufwand ist offensichtlich viel größer, wenn die ZPP alle Kundenaufträge in einem Zyklus simuliert, denn der Archivierungsmechanismus kann seine Vorteile nicht ausspielen und die Maschinenbelegung hat ca. die 14-fache Menge an Operationen zu berechnen.

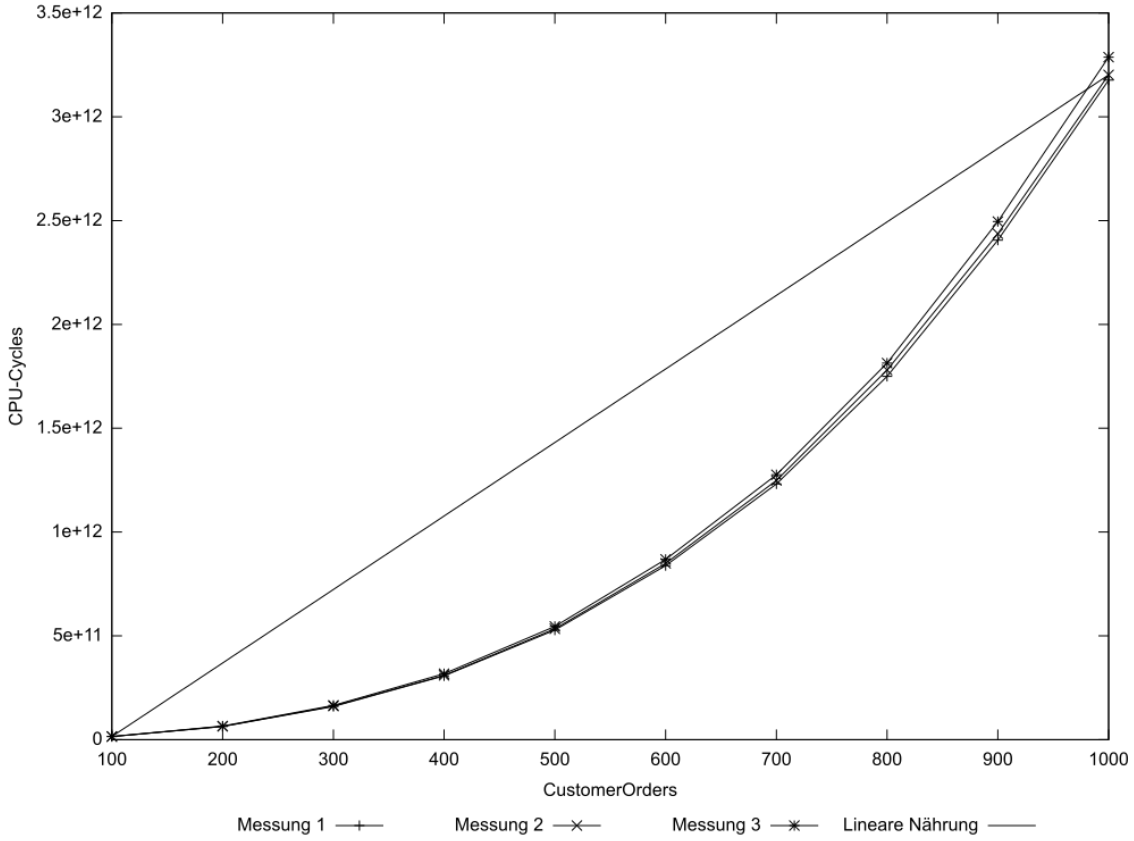

Abbildung 7.2.: CPU-Zyklen für Fall 3

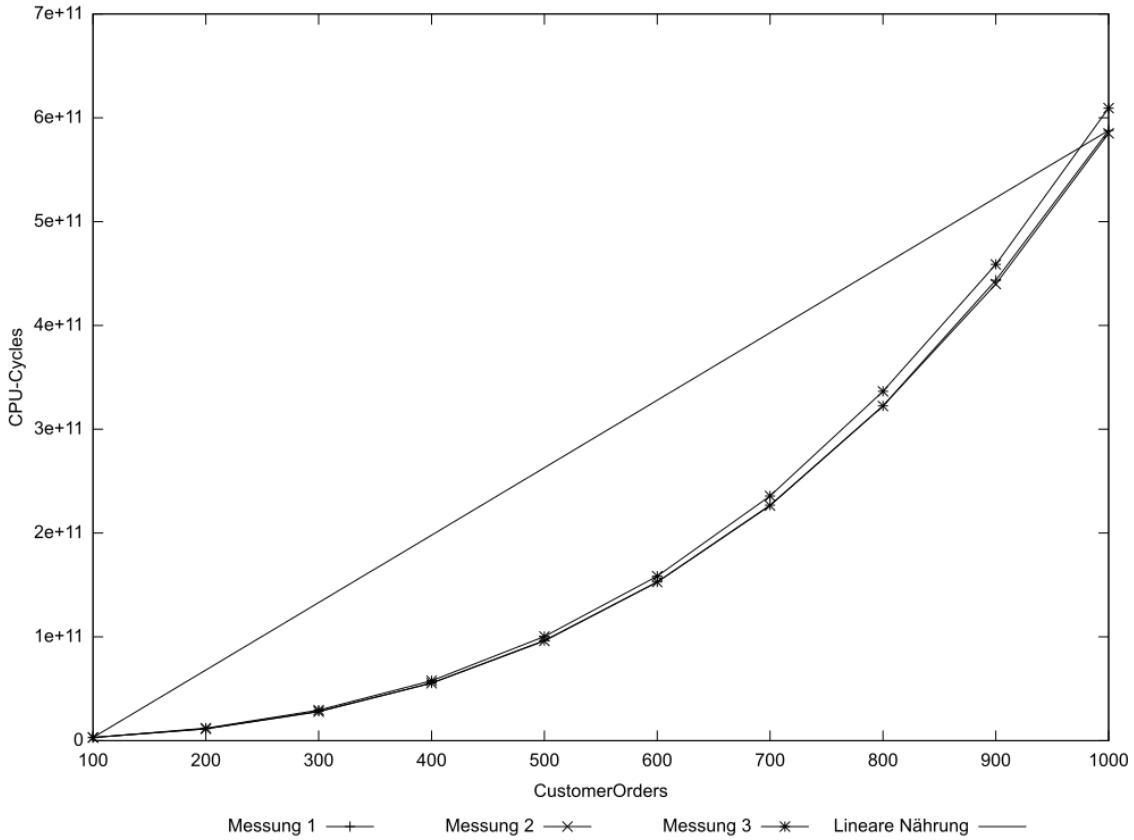

Abbildung 7.3.: CPU-Zyklen für Fall 4

7.8. Ergebnisse der Messung des Speicherbedarfs Fall 1-4

Die Messungen für den Speicherbedarf unterscheiden sich zwischen den Fällen kaum. Der Verdacht ist, dass der Debugging-Modus von Visual Studio mit seinem Overhead den Speicherbedarf stark verzehrt. Allerdings können die Messungen als oberere Schranke dienen. Die Messungen legen einen höchstens linear wachsenden Speicherbedarf nahe. Exemplarisch verweist der Autor auf die erste Messung des dritten Falls 7.4. Ein Speicherleck ist offensichtlich nicht vorhanden, denn sonst würde der Speicherbedarf nichtlinear steigen.

Das Maximum des Speicherbedarf ist laut Messungen ca. 491 MB. Der Autor hat den Speicherbedarf nicht optimiert. So könnte der Speicherbedarf deutlich reduziert werden, indem Objekte wiederverwendet (Objekt-Pooling) bzw. gar nicht erst durch z.B. Boxing oder Konvertierungen erzeugt werden. Die Objektgymnastik erzwingt einen hohen Speicherbedarf durch das Wrappen von Primitiven und Klassen.

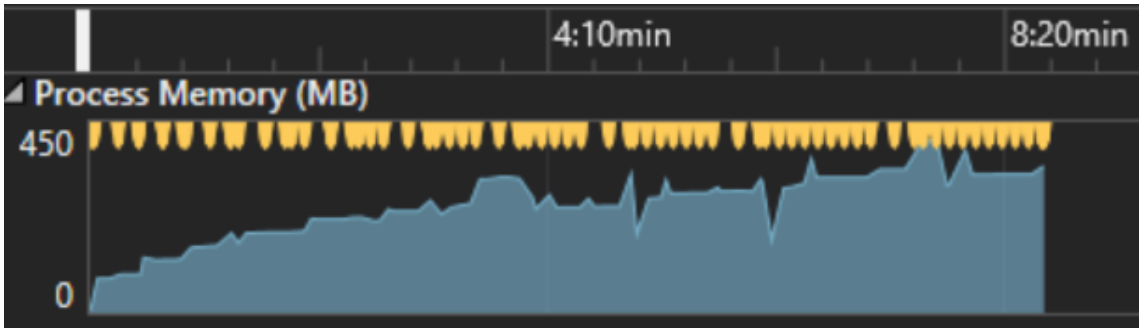

Abbildung 7.4.: 1. Messung des Speicherbedarfs für den 3. Fall

7.9. Ergebnisse der Speicherbedarf von CPU-Zyklen Fall 5-6

Der fünfte und sechste Fall betrachten die zyklische Messung der einzelnen Komponenten. Die Messung beantwortet die Frage, welche Komponente wie viele CPU-Zyklen in welchem Zyklus benötigt. Die beiden folgenden Diagramme zeigen die erste Messung. Die beiden weiteren Messungen weichen nur geringfügig ab und sind nur durch die Fehlerfortpflanzung in den letzten Zyklen optisch erkennbar. Eine Fehlerfortpflanzung gibt es, da für jeden Zyklus pro Komponente die Messung gestartet und gestoppt wird. Die kleine Abweichung akkumuliert sich also über mindestens 14 Zyklen.

Die Abb. 7.5 zeigt die erste Messung des sechsten Fall, also für den anspruchsvolleren Fall der Trucks. Die Messung zeigt, dass die Maschinenbelegung für die Nichtlinearität verantwortlich ist. Für weitere Optimierungen im Bezug auf die Laufzeit liegt also in der Maschinenbelegung das größte Potenzial. Die Laufzeit der Durchlaufterminierungen wächst schneller als die anderen Komponenten (mit Ausnahme der Maschinenbelegung). Allerdings verhalten sich die Durchlaufterminierungen linear für steigende Kundenaufträge.

Die Abb. 7.6 zeigt die erste Messung des fünften Fall. Der fünfte Fall verhält sich anders in der Komponentenverteilung. Die Maschinenbelegung hat zwar stets die größte Laufzeit, allerdings ist der Abstand zu den Durchlaufterminierungen und zum MRP1 nicht groß. Der Grund ist, dass die Komplexität des Tischbeispiel sehr gering mit nur drei Operation pro Produktionsauftrag ist. Die anderen Komponenten haben also einen größeren Einfluss auf das Messergebnis. Die Messung zeigt, dass die Komponenten MRP1, die Durchlaufterminierungen und die Maschinenbelegung nichtlinear für steigende Kundenaufträge verhalten, allerdings liegen sie deutlich in $O(n^2)$.

Die Messungen des fünften und sechsten Falls ermöglichen eine Laufzeitabschätzung der Komponenten für höhere Kundenauftragszahlen. Der Autor schätzt O(1) für die Komponente CreateCustomerOrder, die Erhöhung der Kundenauftragszahlen beeinflusst also nicht die Erstellung von Kundenaufträgen. Der Autor schätzt O(n) für die Komponenten CreateConfirmations und ApplyConfirmations. Der Autor schätzt $O(n^2)$ für die Komponenten MRP1, die Durchlaufterminierungen und die Maschinenbelegung.

Allerdings müssen weitere Messungen zeigen, ob die Laufzeit der Maschinenbelegung wirklich quadratisch mit der Anzahl an Kundenaufträgen wächst. Denn die Laufzeit könnte für höhere Anzahlen von Kundenaufträgen deutlich ansteigen. Ein Grund ist, dass keine Kapazitätsanpassung erfolgt, dadurch bleiben für steigende Kundenauftragszahlen immer mehr Operationen offen und die Maschinenbelegung muss über eine immer größer werdende Menge von Operationen ihren Algorithmus ausführen. Für eine reale Laufzeitabschätzung ist die Einbeziehung der Kapazitätsauslastung also unabdingbar.

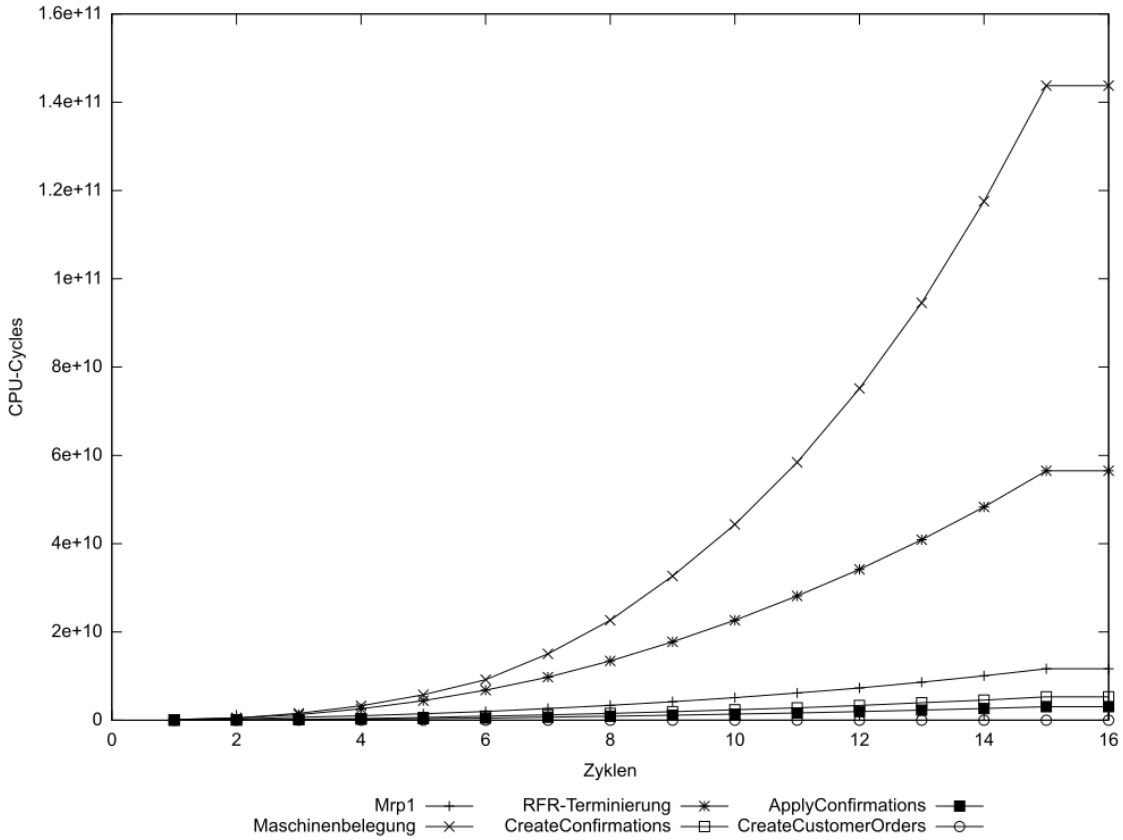

Abbildung 7.5.: Erste Messung der CPU-Zyklen für 6. Fall

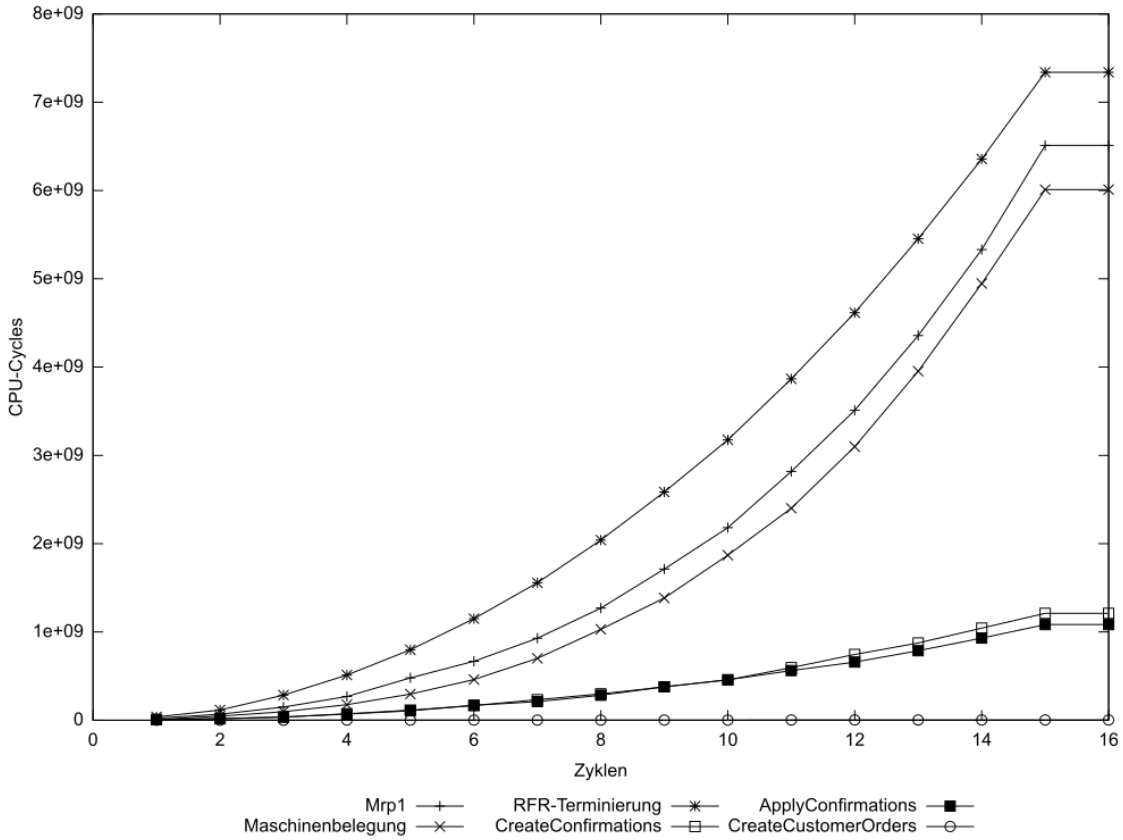

Abbildung 7.6.: Erste Messung der CPU-Zyklen für den 5.Fall

7.10. Zusammenfassung und Diskussion

Die Messungen zeigen, dass die Anforderungen an die Performance erfüllt sind. Insbesondere die Anforderung an die Laufzeit ist deutlich unterschritten worden. Die Messungen decken Optimierungspotenziale in der Laufzeit bei den Komponenten MRP1, den Durchlaufterminierungen und der Maschinenbelegung auf. Der Autor konnte aus den Messungen eine Laufzeitabschätzung für höhere Kundenauftragszahlen vornehmen. Genauere Messungen müssten die Kapazitätsauslastung mit einbeziehen, sodass genauere Laufzeitabschätzungen möglich wären.

8. Zusammenfassung und Ausblick

8.1. Zusammenfassung

Diese Arbeit beschreibt die Entwicklung einer performanten zentralen Produktionsplanung zu Vergleichs-und Integrationszwecken. Der entwickelte Prototyp genügt, mit kleinen Abstrichen an die Struktur und der Simulation von Rückmeldungen, den gestellten Anforderungen. Insbesondere genügt der Prototyp den Anforderungen an die Performance, damit wurde das wesentliche Ziel dieser Arbeit erreicht.

Diese Arbeit behandelt zunächst die Beschreibung einer zentralen Produktionsplanung mit dem MRP-Lauf und beschreibt detailliert die einzelnen notwendigen Schritte: die Stücklistenauflösung, die Losgrößenbildung, die Materialbedarfsplanung, die Kapazitätsplanung gegen unbegrenzte Kapazität, die Kapazitätsplanung gegen begrenzte Kapazität, die Erstellung und Anwendung der Rückmeldungen und die Neuplanung. Dann beschreibt die Arbeit eine entwickelte Stückliste „Tisch“. Außerdem beschreibt die Arbeit detailliert die Objektgymnastik.

Danach beschreibt der Autor die wesentlichen aus der Aufgabenstellung ableitbaren funktionalen und nicht funktionalen Anforderungen. Dann beschreibt die Arbeit detailliert die Umsetzung der ZPP. Zunächst beschreibt die Arbeit die nötigen Anpassungen und Erweiterungen an den Datenstrukturen. Dann definiert der Autor die für die Planung nötigen Eingangsdaten. Der Autor beschreibt die algorithmische Umsetzung des MRP-Laufs. Danach beschreibt der Autor die Architektur und den Grobentwurf der ZPP.

Dann folgt detailliert die algorithmische Umsetzung der einzelnen Komponenten: die Materialbedarfsplanung inklusive der Online- und Offline-Losgrößenbildung, die Kapazitätsplanung gegen unbegrenzte Kapazität, die Kapazitätsplanung gegen begrenzte Kapazität, die Erstellung und Anwendung von Rückmeldungen und die Neuplanung. Für jede der Komponenten sind die Ergebnisse der Algorithmen anhand des Tischbeispiels demonstriert. Die Arbeit beschreibt ausführlich die Architektur der ZPP. Der Autor beschreibt die Umsetzung der nicht funktionalen Anforderungen. Die Umsetzung schließt der Autor mit der Beschreibung der verwendeten Bibliotheken ab, der Autor hat nur kleine Hilfsmittel wie eine Prioritätswarteschlange verwendet.

An die Umsetzung schließt sich die Verifikation der ZPP an. Die Verifikation besteht aus Integrationstests für jede Komponente. Die Integrationstest prüfen die resultierenden Daten nach mehrfachen Planungsläufen (Simulation) auf zu erfüllenden Bedingungen. Abschließend führt der Autor eine empirische Studie zur Überprüfung der Anforderungen an die Performance der entwickelten ZPP aus. Eine Analyse von existierender Produktionsplanungssysteme hat der Autor mangels Zeit nicht gemacht.

Probleme in der Umsetzung traten wie erwartet durch die vorgegebene Verwendung des Entity-Framework auf. Die Verwendung des Entity-Framework in dieser Arbeit führte bereits zu Schwierigkeiten im Bezug auf die Laufzeit in der Masterarbeit von Martin Krockert und Sebastian Seifert.[1] Das Entity-Framework in der verwendete Version (dotNET Core 2.2) ist nicht in der Lage, mehrere Insert- und Update-Statements in einer Transaktion durchzuführen. Daher ist in der verwendeten Version ein Einsatz mit Tausenden bzw. Millionen Objekten eine Einsatzmöglichkeit nicht gegeben. So verzichtete der Autor bei der Umsetzung der ZPP soweit möglich auf das Entity-Framework. Das Entity-Framework initialisiert nur die Datenbank-Schemas, speist die Masterdaten ein und persistiert am Ende

[1] Vgl. Krockert und Seifert, „Selbstorganisation oder zentrale Planung im Kontext von Industrie 4.0“, S. 139.

die Simulationsdaten. Wobei der Autor die Persistierung in der empirischen Studie deaktivieren musste, um die empirischen Studie überhaupt mit der angeforderten Anzahl an Kundenaufträgen durchführen zu können.

8.2. Ausblick

Der entwickelte Prototyp befindet sich in einem Anfangszustand. Nun müssen weitere Arbeiten den Prototypen validieren. Der Prototyp muss also in der Praxis genutzt und auf die erwartete Funktionsfähigkeit geprüft werden. Ggf. müssen weitere Arbeiten die Struktur verbessern. Weitere Arbeiten müssen das Simulationsmodell der SSOP für die Simulation der Rückmeldungen einführen.

Außerdem müssen weitere Arbeiten offene Fragen klären. Z.B. ob die Modellierung der initialen Lagerzustände wie entwickelt verbleibt und die nötige Anpassung der Spezifikation vorgenommen wird (Zuordnungen zwischen Bedarfen und Bedarfsdeckern). Es müssen weitere Arbeiten klären, wie das Entity-Framework weiter genutzt wird, oder ob es durch ein ORM ersetzt wird, das die Verarbeitung von mehreren Insert/Update-Statements in einer Transaktion beherrscht. Eine Analyse existierender Produktionsplanungssysteme steht noch aus.

Literatur

CI, Travis. *Travis CI*. 2019. URL: https://github.com/travis-ci/travis-ci/blob/master/README.md (besucht am 01. 12. 2019).

Corsten, Hans. *Produktionswirtschaft*. 2004.

Dangelmaier, Wilhelm. *Theorie der Produktionsplanung und -steuerung*. 2009.

Dudenverlag. *Elter, das oder der*. 2019. URL: https://www.duden.de/rechtschreibung/Elter (besucht am 01. 12. 2019).

Fowler, Martin. *UML Distilled*. 2003.

graphviz.org. *About Graph Visualization*. URL: http://www.graphviz.org/about/ (besucht am 01. 12. 2019).

– *The DOT Language*. URL: http://www.graphviz.org/doc/info/lang.html (besucht am 01. 12. 2019).

Hollas, Boris. *Grundkurs Theoretische Informatik*. Springer Berlin, 2015.

IETF. *The JavaScript Object Notation (JSON) Data Interchange Format*. 2019. URL: https://tools.ietf.org/html/rfc8259 (besucht am 01. 12. 2019).

ISO/IEC/IEEE. *24765*. ISO/IEC/IEEE, 2017.

Krockert, Martin. *Gantt live view*. 2019. URL: http://krockema.github.io/ganttChart/ (besucht am 01. 12. 2019).

Krockert, Martin und Sebastian Seifert. „Selbstorganisation oder zentrale Planung im Kontext von Industrie 4.0“. Magisterarb. HTW Dresden, 24. Jan. 2018.

Kurbel, Karl. *Enterprise Resource Planning und Supply Chain Management in der Industrie*. 2016.

– *MRP II*. 2016. URL: http://www.enzyklopaedie-der-wirtschaftsinformatik.de/lexikon/informationssysteme/Sektorspezifische-Anwendungssysteme/MRP-II/index.html (besucht am 01. 12. 2019).

McDonald, Jason S. *Design Patterns Quick Reference*. URL: http://www.mcdonaldland.info/2007/11/28/40/ (besucht am 01. 12. 2019).

Microsoft. *Entity Framework*. 2019. URL: https://docs.microsoft.com/en-us/ef/ (besucht am 01. 12. 2019).

– *Process.WorkingSet64 Property*. 2019. URL: https://docs.microsoft.com/en-us/dotnet/api/system.diagnostics.process.workingset64?view=netframework-4.8 (besucht am 01. 12. 2019).

– *QueryThreadCycleTime function*. 2019. URL: https://docs.microsoft.com/en-us/windows/win32/api/realtimeapiset/nf-realtimeapiset-querythreadcycletime?redirectedfrom=MSDN (besucht am 01. 12. 2019).

Newtonsoft. *Json.NET Dokumentation*. 2019. URL: https://www.newtonsoft.com/json/help/html/Introduction.htm (besucht am 01. 12. 2019).

Scheer, August-Wilhelm. *Wirtschaftsinformatik*. 1995.

Schuh, Günther. *Produktionsplanung und -steuerung*. 2006.

SE, SAP. *Standardized Technical Architecture Modeling 1.0*. URL: http://www.fmc-modeling.org/download/fmc-and-tam/SAP-TAM_Standard.pdf (besucht am 01. 12. 2019).

Sipser, Michael. *Introduction to the Theory of Computation*. Cengage Learning, 2012.

Stefaniuk, Marcin. *Graphviz it*. 2019. URL: http://graphviz.it/ (besucht am 01. 12. 2019).

ThoughtWorks. *ThoughtWorks Anthology: Essays on Software Technology and Innovation*. 2008.

Zäpfel, Günther. *Moderne Heuristiken der Produktionsplanung*. 2005.

A. ERM-Diagramm der Datenbank

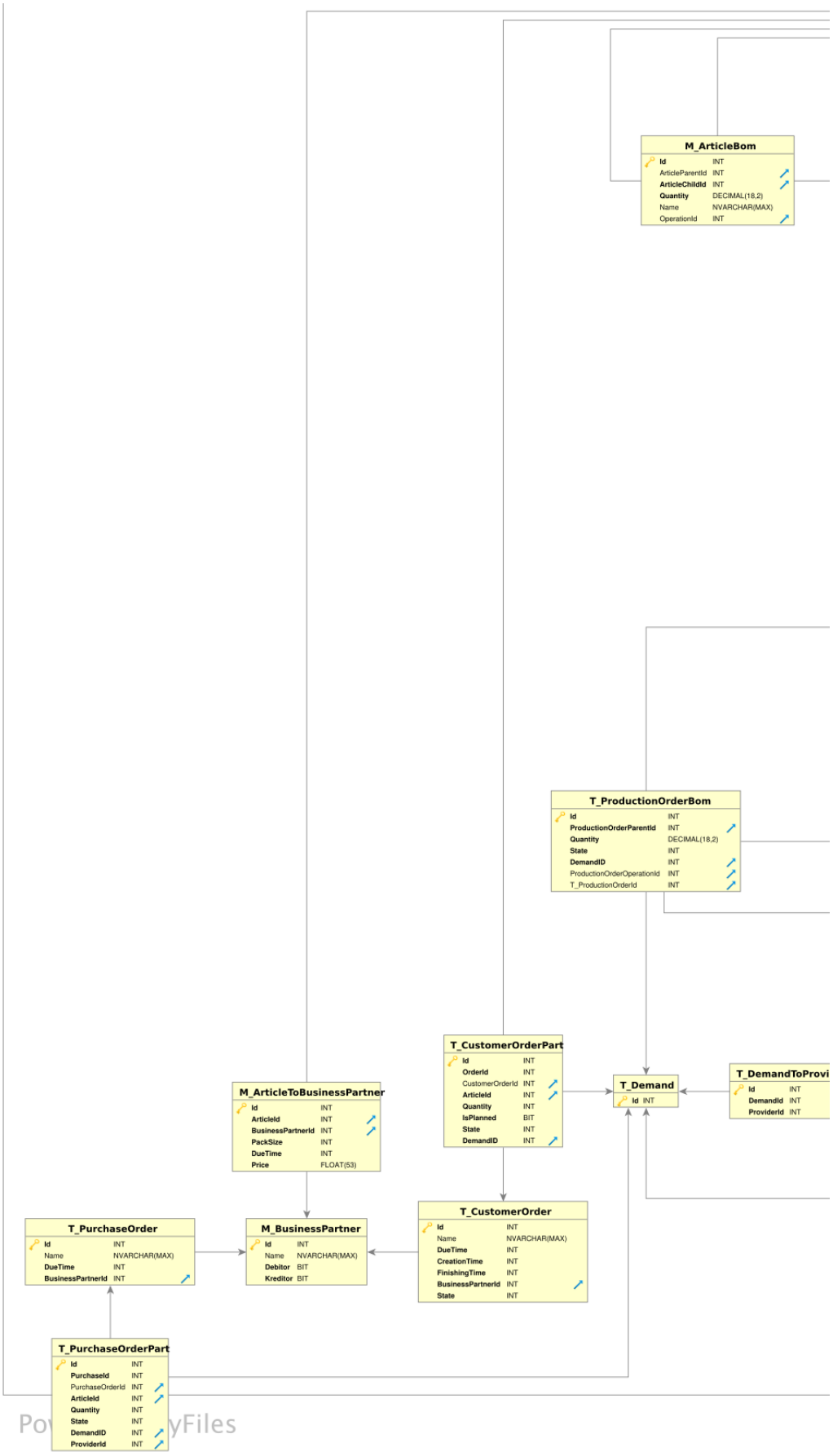

Abbildung A.1.: Das ERM-Diagramm der Datenbank linker Teil

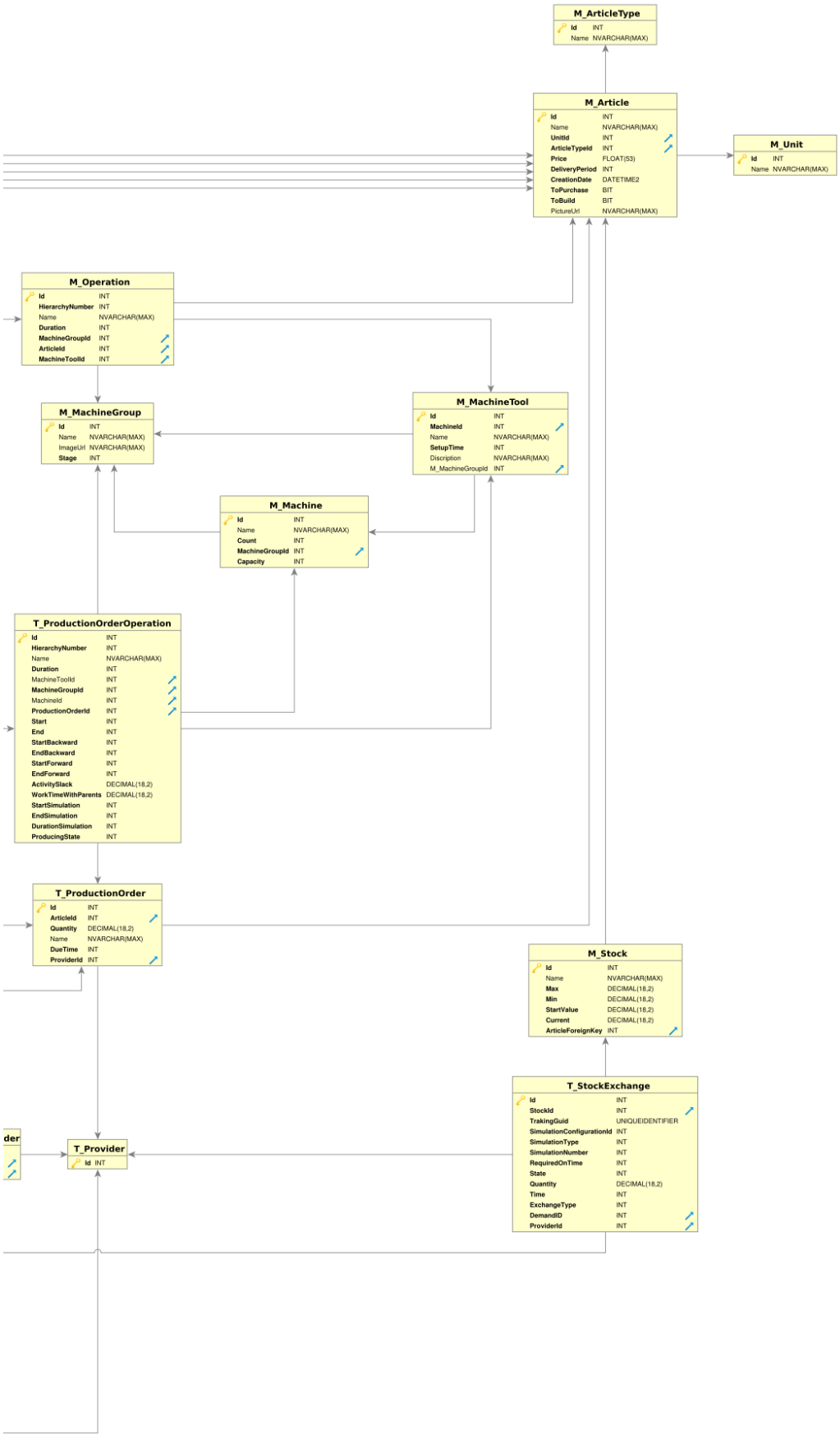

Abbildung A.2.: Das ERM-Diagramm der Datenbank rechter Teil

B. Berechnungen der Durchlaufterminierungen für das Tischbeispiel

Produktionsartikel	Operation	Startzeit	Endzeit	Dauer	Menge	Dauer Einzelfertigung	Materialbereitstellungszeitpunkt	Übergangszeit	Übergangsfaktor
Tisch	Tisch zusammenbauen	308	330	22	2	11	242	66	3
Tischbein	Filzgleiter anstecken	218	242	24	8	3	146	72	3
Tischbein	Anschraubplatte anschweißen	-14	146	160	8	20	-494	480	3

Einkaufsartikel	Startzeit	Endzeit	Dauer	Menge
Tischplatte	142	242	100	2
Filzgleiter	46	146	100	8
Anschraubplatte	-594	-494	100	10
Stahlrohr	-594	-494	100	10

Tabelle B.1.: Ergebnis der ersten Rückwärtsterminierung

Produktions-artikel	Operation	Startzeit	Endzeit	Dauer	Menge	Dauer Einzel-fertigung	Materialbe-reitstellungs-zeitpunkt	Übergangs-zeit	Übergangs-faktor
Tisch	Tisch zusam-menbauen	902	924	22	2	11	836	66	3
Tischbein	Filzgleiter an-stecken	812	836	24	8	3	740	72	3
Tischbein	Anschraub-platte an-schweißen	580	740	160	8	20	100	480	3

Einkaufsartikel	Startzeit	Endzeit	Dauer	Menge
Tischplatte	142	242	100	2
Filzgleiter	46	146	100	8
Anschraubplatte	0	100	100	10
Stahlrohr	0	100	100	10

Tabelle B.2.: Ergebnis der Vorwärtsterminierung

Produktions-artikel	Operation	Startzeit	Endzeit	Dauer	Menge	Dauer Einzel-fertigung	Materialbe-reitstellungs-zeitpunkt	Übergangs-zeit	Übergangs-faktor
Tisch	Tisch zusam-menbauen	902	924	22	2	11	836	66	3
Tischbein	Filzgleiter an-stecken	812	836	24	8	3	740	72	3
Tischbein	Anschraub-platte an-schweißen	580	740	160	8	20	100	480	3

Einkaufsartikel	Startzeit	Endzeit	Dauer	Menge
Tischplatte	736	836	100	2
Filzgleiter	640	740	100	8
Anschraubplatte	0	100	100	10
Stahlrohr	0	100	100	10

Tabelle B.3.: Ergebnis der zweiten Rückwärtsterminierung

C. Weitere Ergebnisse der Performancestudie

C.1. Ergebnisse der Messung von CPU-Zyklen Fall 1 und 2

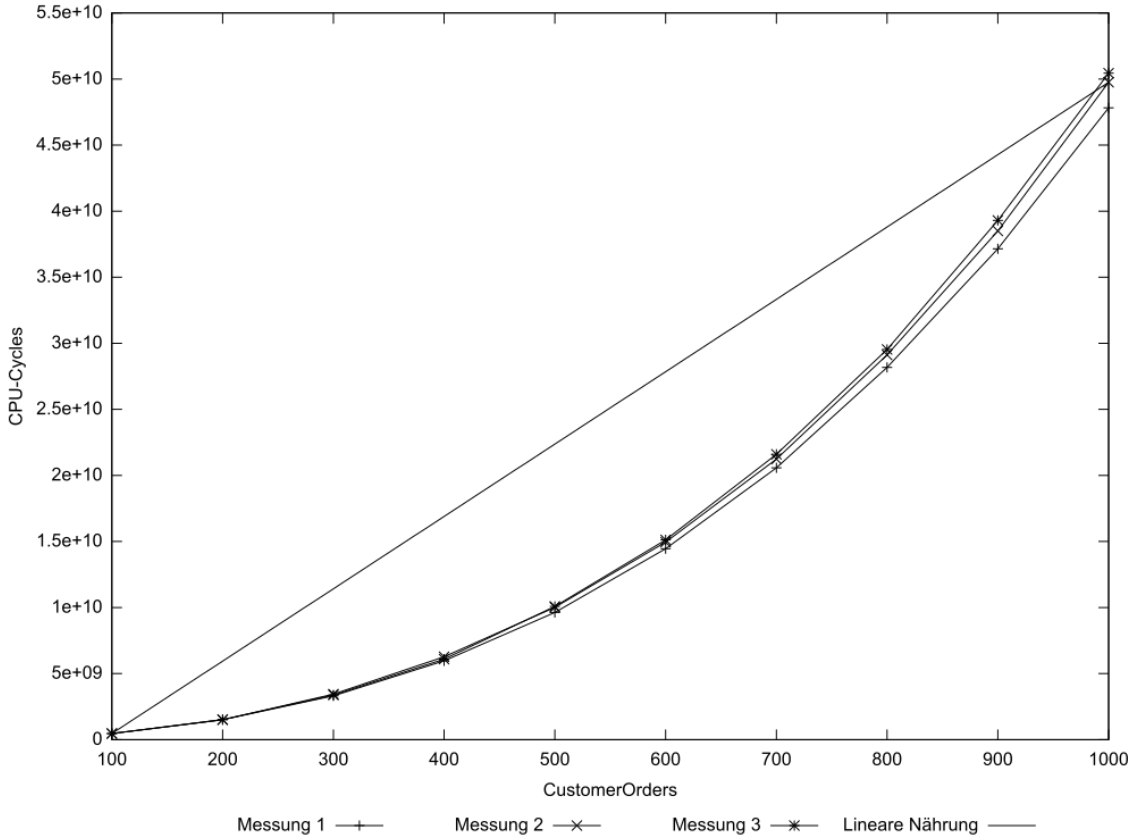

Abbildung C.1.: CPU-Zyklen für Fall 1 (Tisch, 100-1000 COs, mind. 1 Zyklen)

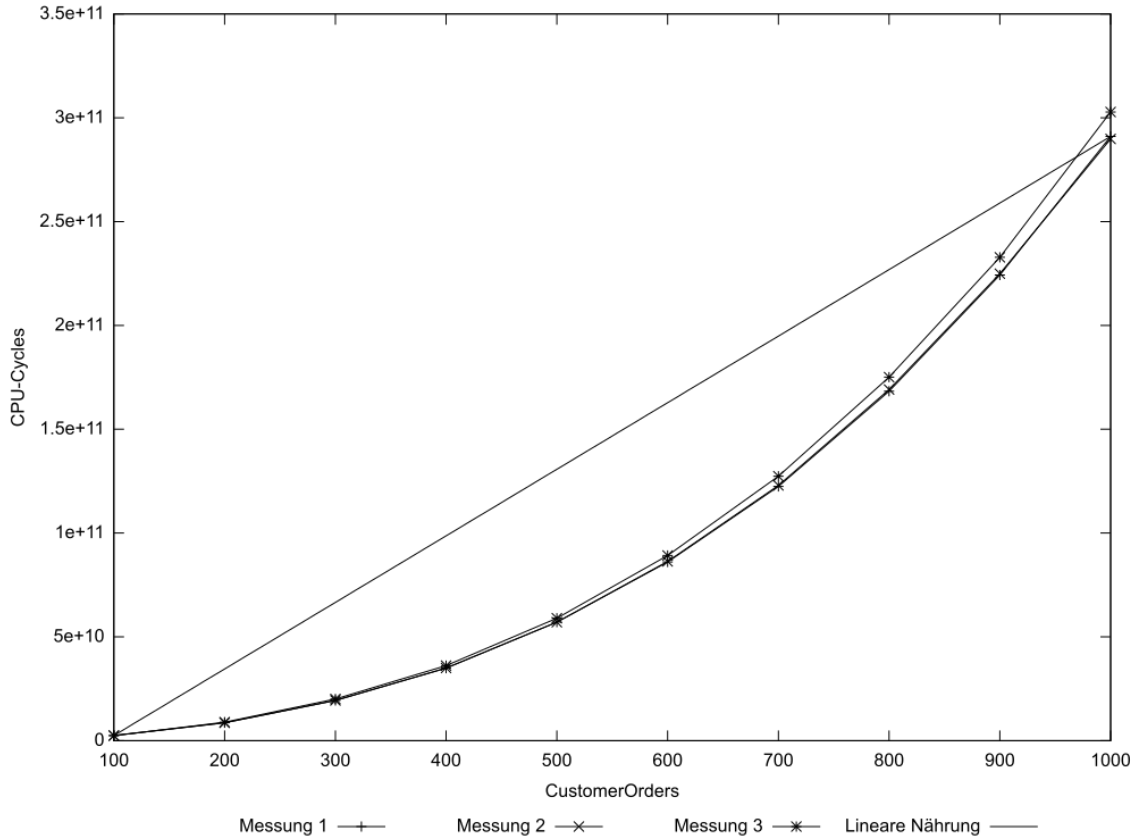

Abbildung C.2.: CPU-Zyklen für Fall 2 (Tisch, 100-1000 COs, mind. 14 Zyklen)

C.2. Weitere Ergebnisse der Messung des Speicherbedarfs Fall 1-4

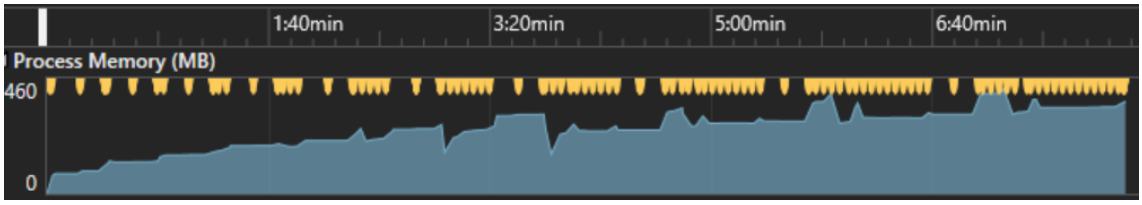

Abbildung C.3.: 1. Messung des Speicherbedarfs für den 1. Fall

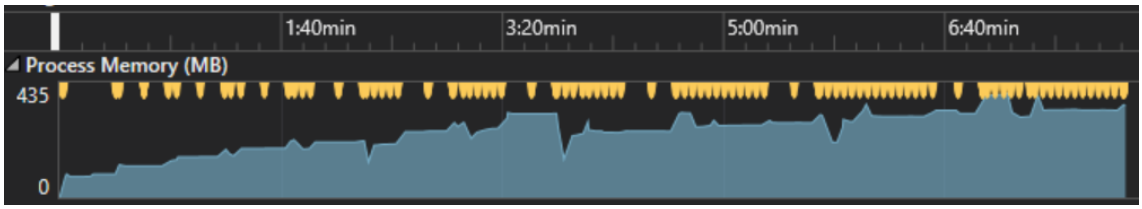

Abbildung C.4.: 1. Messung des Speicherbedarfs für den 2. Fall

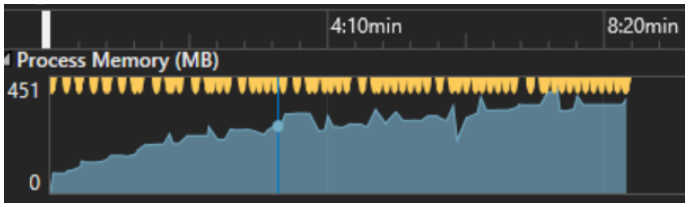

Abbildung C.5.: 1. Messung des Speicherbedarfs für den 4. Fall

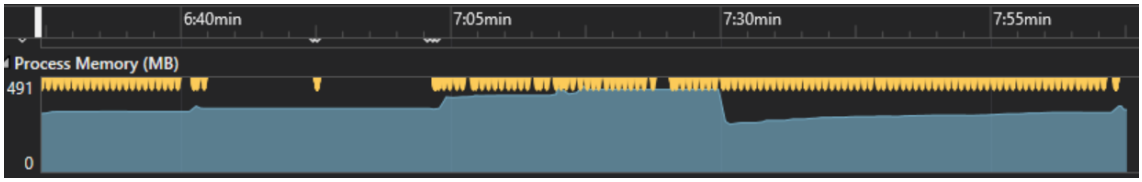

Abbildung C.6.: 2. Messung des Speicherbedarfs für den 1. Fall

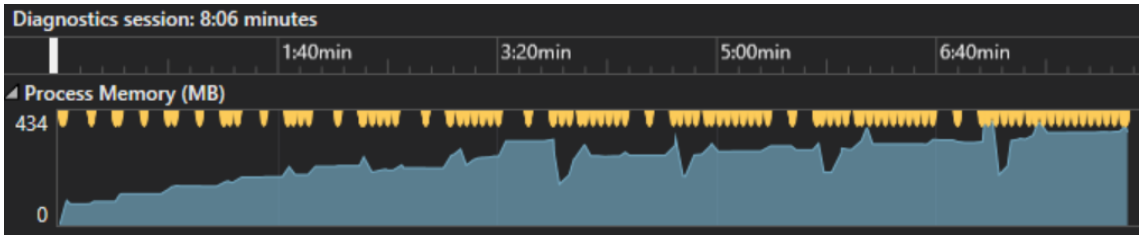

Abbildung C.7.: 2. Messung des Speicherbedarfs für den 2. Fall

Abbildung C.8.: 2. Messung des Speicherbedarfs für den 3. Fall

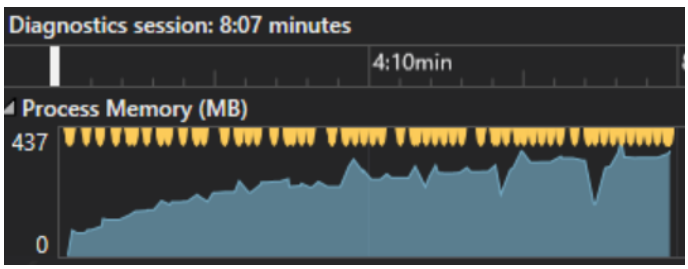

Abbildung C.9.: 2. Messung des Speicherbedarfs für den 4. Fall

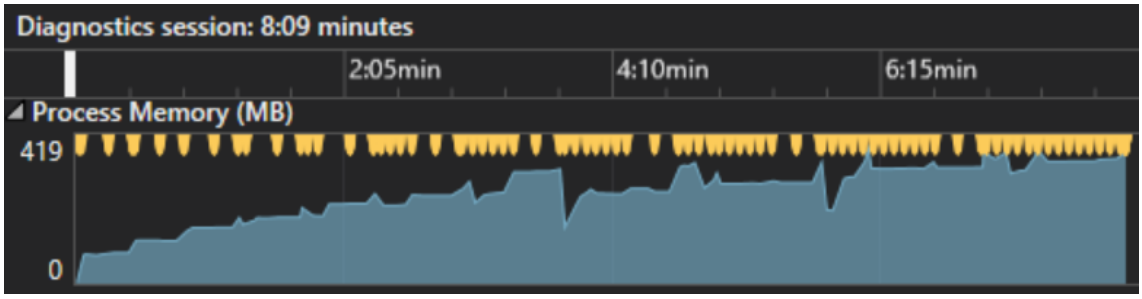

Abbildung C.10.: 3. Messung des Speicherbedarfs für den 1. Fall

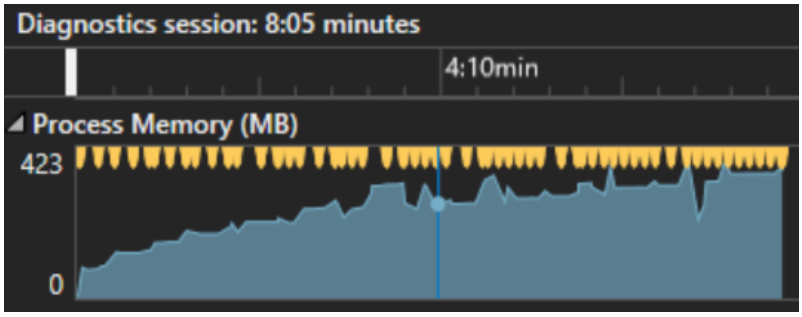

Abbildung C.11.: 3. Messung des Speicherbedarfs für den 2. Fall

Abbildung C.12.: 3. Messung des Speicherbedarfs für den 3. Fall

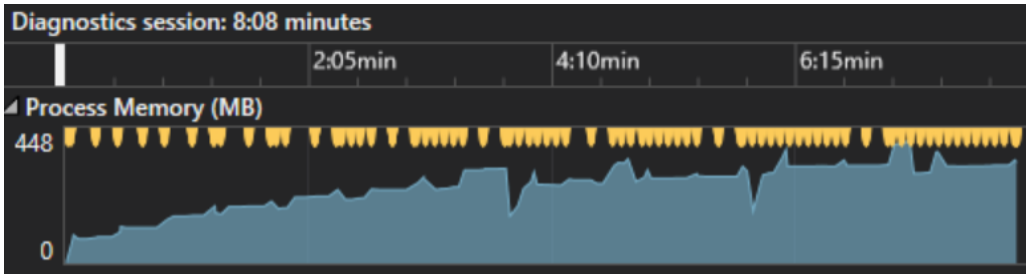

Abbildung C.13.: 3. Messung des Speicherbedarfs für den 4. Fall

C.3. Ergebnisse der 2. und 3 Messung von CPU-Zyklen Fall 5-6

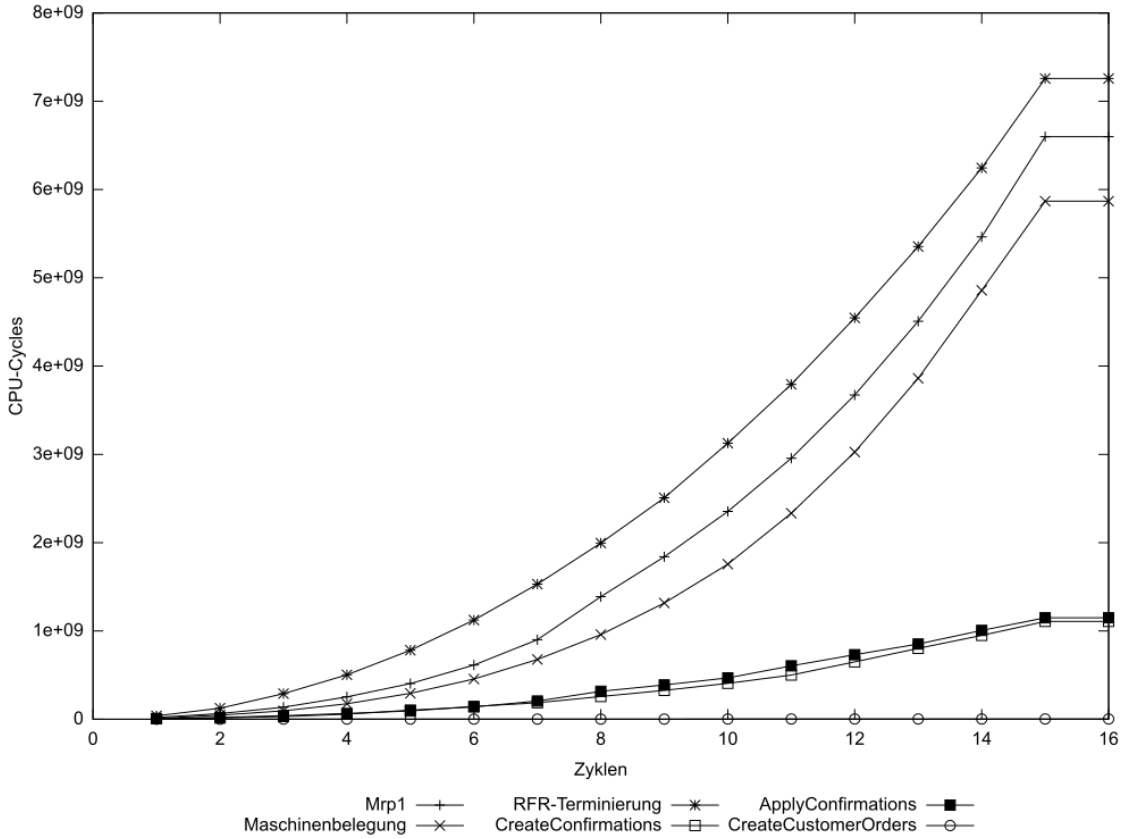

Abbildung C.14.: CPU-Zyklen für Fall 5 (Tisch, 500 COs, mind. 14 Zyklen)

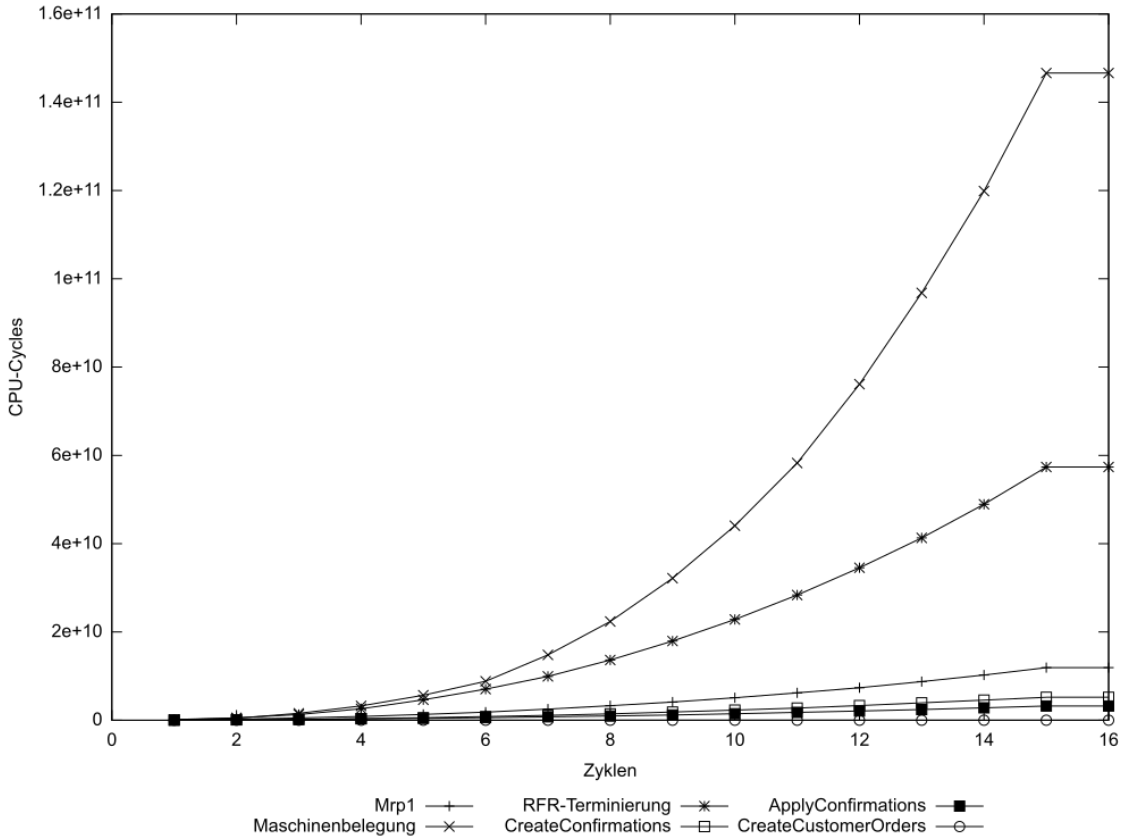

Abbildung C.15.: CPU-Zyklen für Fall 6 (Truck, 500 COs, mind. 14 Zyklen)

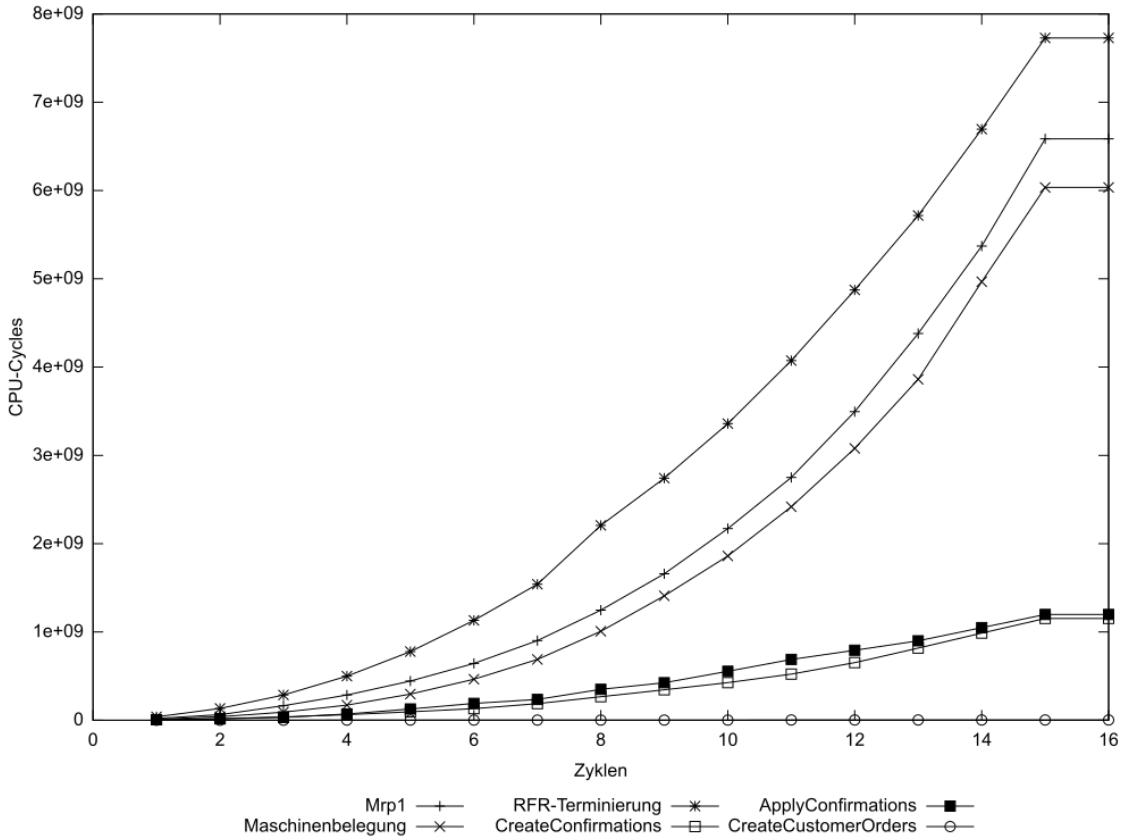

Abbildung C.16.: CPU-Zyklen für Fall 5 (Tisch, 500 COs, mind. 14 Zyklen)

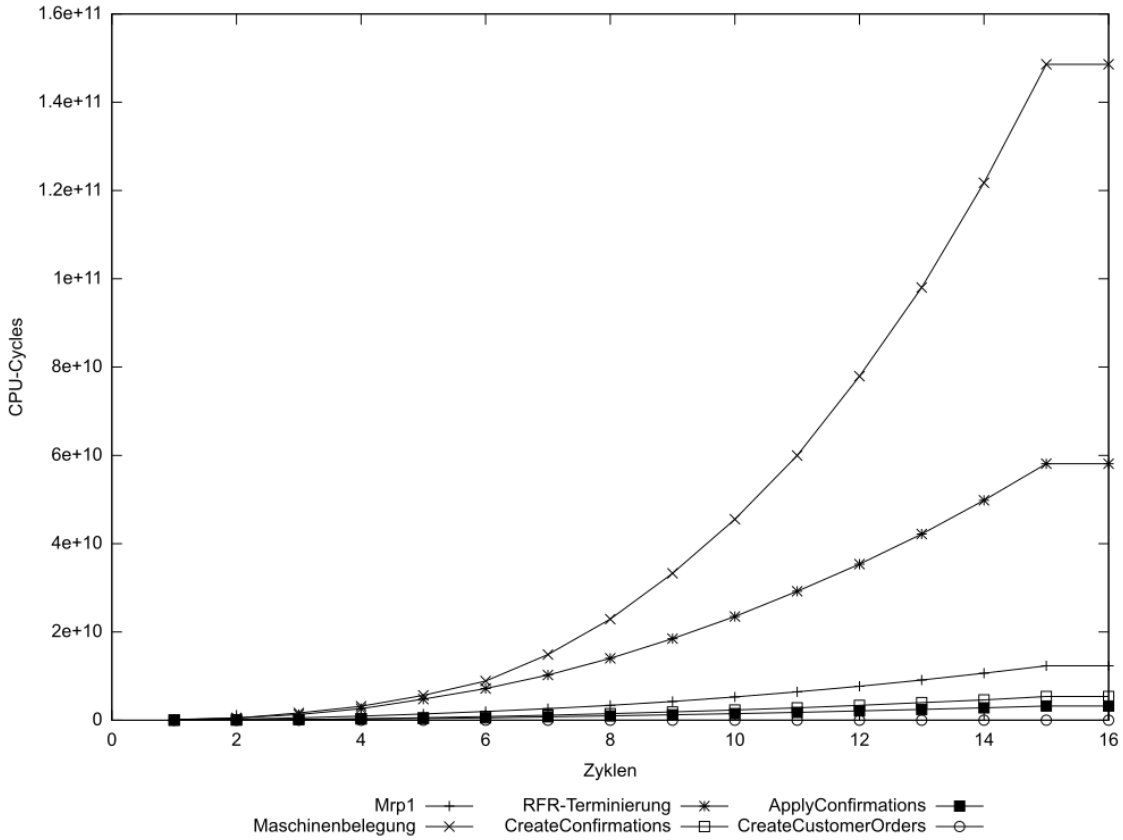

Abbildung C.17.: CPU-Zyklen für Fall 6 (Truck, 500 COs, mind. 14 Zyklen)

C.4. Ergebnisse der Messung von CPU-Zyklen Fall 1-4 tabellarisch

COs	CPU-Zyklen	Speicherbedarf	CPU-Zyklen	Speicherbedarf	CPU-Zyklen	Speicherbedarf	CPU-Zyklen Median	Speicherbedarf Median
100	460313560	415195136	467835439	425537536	467852353	405204992	467835439	415195136
200	1515190950	427986944	1500501048	436469760	1518887794	417882112	1515190950	427986944
300	3315476006	230891520	3449256033	172728320	3376992785	444047360	3376992785	230891520
400	5967623824	172699648	6276814371	161107968	6096294042	479367168	6096294042	172699648
500	9627382263	166129664	10020086011	177741824	10083447967	178106368	10020086011	177741824
600	14453980676	187592704	14952902753	190406656	15125652609	184565760	14952902753	187592704
700	20573552683	208203776	21239870340	200450048	21594289419	198180864	21239870340	200450048
800	28171350092	204554240	29124115036	205533184	29535910440	208887808	29124115036	205533184
900	37154060515	214437888	38504409070	221138944	39303880040	216666112	38504409070	216666112
1000	47820737744	226463744	49754404462	224382976	50446340896	225837056	49754404462	225837056

Tabelle C.1.: Ergebnis der Messung von CPU-Zyklen des 1. Falls

COs	CPU-Zyklen	Speicherbedarf	CPU-Zyklen	Speicherbedarf	CPU-Zyklen	Speicherbedarf	CPU-Zyklen Median	Speicherbedarf Median
100	2513103945	246382592	2467900508	157220864	2380246745	401772544	2467900508	246382592
200	8648167400	198909952	8576772016	192892928	8899559921	189911040	8648167400	192892928
300	19358842590	224206848	19359397571	227782656	20074860749	212729856	19359397571	224206848
400	34900321410	250728448	35017274984	242188288	36151955306	250081280	35017274984	250081280
500	57140049650	278245376	57057126877	273412096	58955051070	286515200	57140049650	278245376
600	86039677639	324227072	86327192236	318574592	89198301775	326348800	86327192236	324227072
700	122450111406	322359296	122956038472	328384512	127362484114	348770304	122956038472	328384512
800	168301467979	348012544	169145223471	342634496	175069174896	353873920	169145223471	348012544
900	224243253602	388964352	224920563635	386564096	232779302350	377344000	224920563635	386564096
1000	291027600881	409370624	289834905806	432418816	302759358169	405975040	291027600881	409370624

Tabelle C.2.: Ergebnis der Messung von CPU-Zyklen des 2. Falls

COs	CPU-Zyklen	Speicherbedarf	CPU-Zyklen	Speicherbedarf	CPU-Zyklen	Speicherbedarf	CPU-Zyklen Median	Speicherbedarf Median
100	15151135274	206471168	14996882112	216231936	15275639687	225943552	15151135274	216231936
200	63677718464	262504448	64073325325	249819136	65691234899	258359296	64073325325	258359296
300	160020463731	305176576	161841933112	318226432	165340147082	311226368	161841933112	311226368
400	307655501589	373465088	309499569836	353525760	317866324143	372236288	309499569836	372236288
500	528078793942	414580736	533868365810	427003904	545578047299	429170688	533868365810	427003904
600	838106149337	494071808	849004425211	470614016	868141441573	526258176	849004425211	494071808
700	1231611530165	527245312	1248605951079	527224832	1276028887848	561283072	1248605951079	527245312
800	1749859130283	591519744	1780476696953	642256896	1813911366982	556179456	1780476696953	591519744
900	2405767327259	660074496	2436836890773	699301888	2495650643964	681816064	2436836890773	681816064
1000	3176494325426	727793664	3201842561599	724344832	3287161351879	731500544	3201842561599	727793664

Tabelle C.3.: Ergebnis der Messung von CPU-Zyklen des 3. Falls

COs	CPU-Zyklen	Speicherbedarf	CPU-Zyklen	Speicherbedarf	CPU-Zyklen	Speicherbedarf	CPU-Zyklen Median	Speicherbedarf Median
100	3045556424	168796160	3054067277	156065792	3106083371	156622848	3054067277	156622848
200	11446441940	194199552	11350852286	179662848	11984679253	178720768	11446441940	179662848
300	28233513404	223191040	27941185099	198762496	29437337338	199065600	28233513404	199065600
400	55582750212	261595136	55406904708	260018176	57810265978	223354880	55582750212	260018176
500	96516358314	288440320	96115040159	259051520	100341173612	289845248	96516358314	288440320
600	152843431991	277004288	152751395503	356769792	158404358529	356741120	152843431991	356741120
700	226853226157	290738176	226367177048	286580736	235659269753	308035584	226853226157	290738176
800	322507272425	316825600	322338523253	318590976	336479667824	323100672	322507272425	318590976
900	443481464444	357015552	439741468807	340344832	458756122774	338472960	443481464444	340344832
1000	587885545885	376967168	585031484307	382529536	609264811039	391897088	587885545885	382529536

Tabelle C.4.: Ergebnis der Messung von CPU-Zyklen des 4. Falls

C.5. Ergebnisse der Messung von CPU-Zyklen Fall 5-6 tabellarisch

Zyklus	MRP1	MRP2	Maschinenbelegung	Durchlauftermi-nierungen	CreateConfirma-tions	ApplyConfirma-tions	CreateCustomer-Orders
1	22182040	67591089	8213499	37137707	4850535	9208744	162671
2	68297213	230948205	46159946	116304650	17289002	19435721	334201
3	149602878	530903468	95913572	284995933	35713208	36210985	508096
4	269516579	958563134	176118262	512228317	73810165	69931334	672565
5	479670845	1574036394	295419607	797876533	113892939	106497337	844704
6	667415421	2304671732	460920155	1150331986	166955967	168070764	1082694
7	929417067	3214580620	701366560	1557186361	232913628	210483920	1249495
8	1269791721	4367363674	1029266181	2040882616	299076880	282923727	1423789
9	1712512750	5709803891	1384195219	2584992227	377855106	378324522	1595219
10	2181093506	7256666595	1870145844	3176412443	456268313	457803089	1775317
11	2818698608	9116407544	2400523422	3866885726	595339528	561747075	1944755
12	3510355218	11256420274	3099070758	4615673852	744507437	658902828	2116773
13	4356957611	13796272502	3953661118	5453184683	875410520	787730770	2328358
14	5330557416	16667337592	4946465832	6356775644	1043311181	930094605	2534391
15	6510632506	19896162571	6010640172	7340204746	1210420895	1084109549	2697597
16	6510632506	19896162571	6010640172	7340204746	1210420895	1084109549	2697597

Tabelle C.5.: Ergebnis der 1. Messung des 5. Falls

Zyklus	MRP1	MRP2	Maschinenbelegung	Durchlauftermierungen	CreateConfirmations	ApplyConfirmations	CreateCustomerOrders
1	20843177	67675236	8635213	38140639	7548888	4669394	165730
2	63777909	234612803	46070227	124575522	20611684	14676922	458575
3	136519769	521052602	94793820	289336332	39781618	31402394	762514
4	254124688	932662004	175234452	502561494	65289048	56014555	951636
5	404535303	1480195557	293817235	780661743	94396934	102435512	1151700
6	613915916	2194120116	455381905	1123186270	143636196	139645713	1344992
7	900622361	3108778210	676447028	1529388947	186437596	206427436	1549587
8	1388120044	4345484809	958929524	1995507148	257148964	315986217	1723056
9	1839855712	5667757798	1316679148	2507594339	326954027	388807415	1905275
10	2354121915	7242458400	1757045954	3126825117	406824038	467887879	2071396
11	2955588427	9087339307	2332697473	3793643114	500158383	605208456	2246556
12	3672125780	11250861471	3026185893	4546210890	649373143	730779308	2426640
13	4508230557	13730919705	3861189549	5354111444	803181218	852014202	2608952
14	5463981094	16574977898	4857662583	6244777447	950111997	1006908358	2881399
15	6598171703	19732919324	5868049642	7256979987	1107961793	1149190580	3051569
16	6598171703	19732919324	5868049642	7256979987	1107961793	1149190580	3051569

Tabelle C.6.: Ergebnis der 2. Messung des 5. Falls

Zyklus	MRP1	MRP2	Maschinenbelegung	Durchlauftermierungen	CreateConfirmations	ApplyConfirmations	CreateCustomerOrders
1	20561528	70807944	11595779	38590791	5232035	5103882	170987
2	63653038	237136288	39424065	133864697	18513201	15722225	346528
3	164789582	542278615	91848260	285239598	37253790	32342798	523698
4	285104975	956258712	171182526	499229997	63537052	70302714	726537
5	444178085	1518188592	295716494	777189330	95095502	126791048	898799
6	643173569	2240408451	464567400	1131112952	130813793	190605563	1093046
7	901766590	3132799597	688977791	1539965789	188297670	235283547	1271191
8	1246911903	4463990980	1007494447	2206743793	266917485	349479955	1470126
9	1658369115	5814226982	1409605744	2742538637	345131596	424084412	1647855
10	2170951934	7395920191	1860990398	3359508524	425967030	554888823	1820860
11	2749777193	9246611181	2417692607	4073730821	522598474	688449356	1996920
12	3494501494	11478516707	3078787511	4874253074	652947899	792064757	2175557
13	4380684385	13992115332	3862067593	5717222310	817599978	899350437	2389059
14	5371477833	17086131129	4967622537	6695380043	987047141	1049118082	2721995
15	6585590435	20401972366	6034494441	7729067294	1152631146	1198407411	3038413
16	6585590435	20401972366	6034494441	7729067294	1152631146	1198407411	3038413

Tabelle C.7.: Ergebnis der 3. Messung des 5. Falls

Zyklus	MRP1	MRP2	Maschinenbelegung	Durchlauftermierungen	CreateConfirmations	ApplyConfirmations	CreateCustomerOrders
1	80974698	347487522	120526170	145929340	19527917	15637622	189986
2	249116618	1246849355	487900151	509642853	59401974	60483679	390343
3	658236716	3489513927	1576275004	1254587348	196259534	118341188	598836
4	996634683	6842189095	3256724761	2588121195	368189883	236818458	794486
5	1456833048	11640706672	5752701287	4430074358	609041505	351843561	992382
6	1989815706	18027552454	9187918961	6824367356	917204156	500920694	1279509
7	2651462098	27458937738	15037281889	9744200465	1217102151	698515411	1475957
8	3376892157	39479280753	22642379795	13433158326	1548432539	899524320	1699072
9	4196061853	54626812530	32653609243	17749588560	1932869724	1134556854	1891261
10	5127218336	72158985674	44354485510	22648944112	2356150898	1392628837	2085520
11	6166811774	92814112565	58431077397	28166768131	2812228142	1651867234	2286026
12	7301510655	116688554277	75160411656	34176241854	3354206661	1968136896	2491498
13	8622325230	144120470581	94544396552	40902361373	3964376976	2280450064	2694077
14	10058592354	175981841598	117566521483	48304041074	4567268953	2662174379	2934051
15	11657749562	211995063485	143755352197	56527949247	5313819503	3066548739	3129247
16	11657749562	211995063485	143755352197	56527949247	5313819503	3066548739	3129247

Tabelle C.8.: Ergebnis der 1. Messung des 6. Falls

Zyklus	MRP1	MRP2	Maschinenbelegung	Durchlauftermi-nierungen	CreateConfirma-tions	ApplyConfirma-tions	CreateCustomer-Orders
1	80974698	347487522	120526170	145929340	19527917	15637622	189986
2	249116618	1246849355	487900151	509642853	59401974	60483679	390343
3	658236716	3489513927	1576275004	1254587348	196259534	118341188	598836
4	996634683	6842189095	3256724761	2588121195	368189883	236818458	794486
5	1456833048	11640706672	5752701287	4430074358	609041505	351843561	992382
6	1989815706	18027552454	9187918961	6824367356	917204156	500920694	1279509
7	2651462098	27458937738	15037281889	9744200465	1217102151	698515411	1475957
8	3376892157	39479280753	22642379795	13433158326	1548432539	899524320	1699072
9	4196061853	54626812530	32653609243	17749588560	1932869724	1134556854	1891261
10	5127218336	72158985674	44354485510	22648944112	2356150898	1392628837	2085520
11	6166811774	92814112565	58431077397	28166768131	2812228142	1651867234	2286026
12	7301510655	116688554277	75160411656	34176241854	3354206661	1968136896	2491498
13	8622325230	144120470581	94544396552	40902361373	3964376976	2280450064	2694077
14	10058592354	175981841598	117566521483	48304041074	4567268953	2662174379	2934051
15	11657749562	211995063485	143755352197	56527949247	5313819503	3066548739	3129247
16	11657749562	211995063485	143755352197	56527949247	5313819503	3066548739	3129247

Tabelle C.9.: Ergebnis der 2. Messung des 6. Falls

Zyklus	MRP1	MRP2	Maschinenbelegung	Durchlauftermierungen	CreateConfirmations	ApplyConfirmations	CreateCustomerOrders
1	80974698	347487522	120526170	145929340	19527917	15637622	189986
2	249116618	1246849355	487900151	509642853	59401974	60483679	390343
3	658236716	3489513927	1576275004	1254587348	196259534	118341188	598836
4	996634683	6842189095	3256724761	2588121195	368189883	236818458	794486
5	1456833048	11640706672	5752701287	4430074358	609041505	351843561	992382
6	1989815706	18027552454	9187918961	6824367356	917204156	500920694	1279509
7	2651462098	27458937738	15037281889	9744200465	1217102151	698515411	1475957
8	3376892157	39479280753	22642379795	13433158326	1548432539	899524320	1699072
9	4196061853	54626812530	32653609243	17749588560	1932869724	1134556854	1891261
10	5127218336	72158985674	44354485510	22648944112	2356150898	1392628837	2085520
11	6166811774	92814112565	58431077397	28166768131	2812228142	1651867234	2286026
12	7301510655	116688554277	75160411656	34176241854	3354206661	1968136896	2491498
13	8622325230	144120470581	94544396552	40902361373	3964376976	2280450064	2694077
14	10058592354	175981841598	117566521483	48304041074	4567268953	2662174379	2934051
15	11657749562	211995063485	143755352197	56527949247	5313819503	3066548739	3129247
16	11657749562	211995063485	143755352197	56527949247	5313819503	3066548739	3129247

Tabelle C.10.: Ergebnis der 3. Messung des 6. Falls

D. Abkürzungsverzeichnis

- Abb: Abbildung
- BOM: Eng. bill of materials (Stückliste)
- Bzw: Beziehungsweise
- COs: CustomerOrders
- ERM: Entity-Relationship-Modell
- Ggf: Gegebenenfalls
- Kap: Kapitel
- Mind: Mindestens
- ORM: Objektrelationaler Mapper
- S: Seite
- SSOP: Prototyp für eine sich selbst organisierende Produktion
- U.a.: Unter anderem
- Vgl: Vergleich
- Z.B.: Zum Beispiel
- ZPP: Zentrale Produktionsplanung

E. Erklärung über die eigenständige Erstellung der Arbeit

Hiermit erkläre ich, dass ich die vorgelegte Diplomarbeit mit dem Titel „Entwicklung einer performanten zentralen Produktionsplanung zu Vergleichs- und Integrationszwecken“ selbständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie alle wörtlich oder sinngemäß übernommenen Stellen in der Arbeit als solche und durch Angabe der Quelle gekennzeichnet habe. Dies gilt auch für Zeichnungen, Skizzen, bildliche Darstellungen sowie für Quellen aus dem Internet. Mir ist bewusst, dass die Hochschule für Technik und Wirtschaft Dresden Prüfungsarbeiten stichprobenartig mittels der Verwendung von Software zur Erkennung von Plagiaten überprüft.

Dresden, 2. Dezember 2019 | _____

Ort, Datum | Student