Automatisierte Komposition und Konfiguration von Workflows
zur Planung mittels kombinatorischer Logik

Dissertation

zur Erlangung des Grades eines

D o k t o r s d e r I n g e n i e u r w i s s e n s c h a f t e n

der Technischen Universität Dortmund
an der Fakultät für Informatik

von

Jan Winkels

Dortmund

2019

Tag der mündlichen Prüfung: 10. Oktober 2019
Dekan: Prof. Dr.-Ing. Gernot Fink
Gutachter:

Prof. Dr. Jakob Rehof (Technische Universität Dortmund, Deutschland)

Prof. Dr. Bernhard Steffen (Technische Universität Dortmund, Deutschland)

Gefördert durch die Deutsche Forschungsgemeinschaft (DFG) – Projektnummer **276879186/GRK2193**

Danksagungen:

Ich danke an erster Stelle meinem Betreuer Jakob Rehof für die umfassende, kompetente und engagierte Betreuung dieser Dissertation. Nicht nur, weil du mir vor drei Jahren durch die Anstellung im Graduiertenkolleg die Arbeit an diesem Werk erst möglich gemacht hast, sondern auch deine Begeisterung für das Thema, deine Geduld und dein stets offenes Ohr haben maßgeblich dafür gesorgt, dass aus der anfänglichen Idee auch wirklich ein reales Ergebnis geworden ist.

Ich danke auch Bernhard Steffen, dafür dass er sich seinerzeit für mich stark gemacht, dass ich diese Rolle im GRK einnehmen konnte und für sein Feedback und seine Beratung bei der Erstellung dieser Arbeit.

Ganz besonderen Dank verdient Boris Düdder, der mir, auch nach seinem Wechsel ins ferne Kopenhagen, stets mit Rat und Tat zur Seite stand. Sein fachlicher und persönlicher Beistand hat diese Arbeit entscheidend mit voran gebracht.

Ich bedanke mich beim gesamten Team der Lehrstühle 14 und 5 der Fakultät für Informatik an der TU Dortmund für den regen fachlichen Austausch und die hervorragende und angenehme Arbeitsatmosphäre. Insbesondere Sevda Tarkun, Ute Joschko und Lars Hildebrand sollen hier Erwähnung finden. Ihr habt mir immer den Rücken frei gehalten und mich bestens unterstützt. Auch Jan Bessai, Andrej Dudenhefner, Tristan Schäfer und Anna Vasileva verdienen meinen großen Dank. Ihr habt mir alle durch euren Input sehr geholfen, diese Dissertation zu verfassen.

Vielen Dank auch an meine Mitstreiter und Mitstreiterinnen aus dem GRK 2193. An Julian Graefenstein, dafür, dass er mir in drei Jahren erfolgreich Fabrikplanung beigebracht hat; an Lisa Lenz für die die beste E-Mail aller Zeiten; an Christin Schumacher für ihren hervorragenden fachlichen Support vor allem in der Endphase der Dissertation und dafür, dass sie immer Zeit für einen Kaffee oder ein Eis hatte, wenn ich mal Ablenkung brauchte. Ebenso seien hier Anne Meyer, Philipp Regelmann, Tim Delbrügger, Daniel Müller, Carina Mieth, Hendrik Lager, Jürgen Schmelting, Andreas Wirtz, Matthias Meisner und Felix Zeidler erwähnt: Ihr habt mich fachlich wie persönlich drei Jahre lang hervorragend unterstützt, mir Einblicke in eure Fachbereiche gewährt und unser gemeinsames Ziel mit Leidenschaft voran getrieben. Außerdem seid ihr echt tolle Kollegen und es war gut zu wissen, dass es Menschen gab, die zum Teil mit den gleichen Problemen wie ich zu kämpfen hatten.

Last but not least danke ich meiner Mutter Gabi fürs Mitfiebern und die große moralische Unterstützung sowie meinem Bruder René, meiner Freundin Gesa und meinen Freunden fürs engagierte Korrekturlesen und dass sie mich immer wieder aufgebaut haben, wenn es mal nicht so lief. Besonders meinem Vater Ralf habe ich unglaublich viel zu verdanken: Ohne dich wäre ich nicht nur bei dieser Arbeit, sondern auch im Studium und als Mensch insgesamt nicht mal halb so weit gekommen.

Vielen Dank euch allen!

„Niemand plant zu versagen, aber die meisten versagen beim Planen.“

Lee Iacocca, ehem. Präsident der Ford Motor Company [70]

Abstract

If an adjustment requirement is identified in a factory system, an adaptation process must be started. This usually includes a planning phase in which a project team develops a procedure for making the adjustment. While production, logistics and manufacturing processes have already been and are automated to a large extent, the development of such a planning process still takes place manually and individually to a large extent. The planning team develops the required plan manuallyänd as required. The creation of a plan tailored to the project results from the specific requirements of each (adaptation) project. The creation of the plan itself under these requirements, however, takes place according to recognizable patterns. The aim of the dissertation is to develop a possibility to automate the creation of plans and the planning itself by developing a procedure (or a software), which makes it possible to generate plans dynamically under consideration of previously given basic conditions according to demand. In order to generate processes dynamically, the project follows a modular principle. A collection of standardized process modules is defined from which complex processes and plans can be assembled. The idea is comparable to a Lego construction kit. Similar to how such building blocks can be combined to almost any object, the process modules can result in any desired process. At the end of the dissertation project, software is to be developed that automatically delivers the appropriate workflow for each project. The planner only enters basic project information (e.g. budget and time restrictions) and receives a selection of possible plans for the realization of the project. If events occur during the execution of the plan that make plan adjustments necessary, this can also be carried out automatically by regenerating the plan. To achieve this goal, methods of combinatorial logic and constraint solving are used. By using combinatorial logic, software synthesis has already been successfully performed at the Chair of Software Engineering. This means that individual programs can be generated from a given set of software components. Constraintsolving, on the other hand, refers to procedures for the determination of solutions for (mathematical) problems under consideration of restrictive constraints. These two technologies are to be combined in an extension of common project planning software. In addition, by using suitable code generation frameworks, the applications needed to implement the respective plan are to be automatically generated.

Inhaltsverzeichnis

Kapitel 1

Motivation und Einleitung

Wird in einem Fabriksystem ein Anpassungsbedarf erkannt, muss ein Anpassungsprozess gestartet werden. Ein solcher Prozess beinhaltet in der Regel eine Planungsphase, in der ein Projektteam ein Vorgehen erarbeitet, wie die Anpassung vorzunehmen ist. Während Produktions-, Logistik- und Fertigungsprozesse bereits weitgehend automatisiert wurden und werden, findet die Entwicklung eines solchen Planungsprozesses in der Regel nach wie vor manuell und individuell statt. Das Planungsteam entwickelt den jeweils benötigten Plan „von Hand“ und nach Bedarf. Die auf das Projekt zugeschnittene Planerstellung ergibt sich aus den spezifischen Anforderungen, die jedes (Anpassungs-) Projekt mit sich bringt. Die Erstellung des Plans erfolgt unter diesen Anforderungen allerdings nach wiedererkennbaren Mustern. Ziel der Dissertation ist es, eine Möglichkeit zu entwickeln, das Erstellen von Plänen und das Planen zu automatisieren. Hierzu soll ein Verfahren (bzw. eine Software) entwickelt werden, das die Möglichkeit bietet, Pläne unter Berücksichtigung von zuvor angegebenen Rahmenbedingungen dynamisch nach Bedarf zu generieren.

Um Prozesse dynamisch zu generieren, folgt das Projekt einem Baukastenprinzip. Es wird eine Sammlung von standardisierten Prozessmodulen definiert, aus denen sich komplexe Prozesse und Pläne zusammenfügen lassen. Die Idee ist vergleichbar mit einem Lego-Baukasten. So wie solche Bausteine je nach Wunsch zu beinahe jedem beliebigen Objekt zusammengefügt werden können, sollen auch die Prozessmodule jeden gewünschten Prozess abbilden können. Am Ende des Projektes soll eine Software entstehen, die für jedes Projekt automatisch den passenden Workflow liefert. Ein Projektplaner gibt nur noch grundlegende Informationen (z.B. Budget- und Zeitbeschränkungen) an und erhält eine Auswahl an möglichen Plänen zur Realisierung des Projektes. Treten während der Durchführung des Plans Ereignisse auf, die Plananpassungen notwendig machen, können auch diese durch eine Neugenerierung des Plans automatisiert durchgeführt werden.

Um dieses Ziel erreichen zu können, werden Methoden der kombinatorischen Logik und des Constraintsolvings genutzt. Durch die Nutzung der kombinatorischen Logik wird am Lehrstuhl für Software Engineering der TU Dortmund bereits erfolgreich Softwaresynthese betrieben [20], was bedeutet, dass aus einer gegebenen Menge unterschiedlicher Software-Komponenten individuelle Programme generiert werden können. Constraintsolving wiederum bezeichnet Verfahren zur Ermittlung von Lösungen für (mathematische) Probleme unter Berücksichtigung von einschränkenden Nebenbedingungen (Constraints) [102, 1]. In dieser Arbeit werden beiden Technologien in einer Erweiterung eines gängigen Projektplanungverfahrens zusammengeführt.

Dazu wird zunächst auf die modernen Herausforderungen der Fabrikplanung im Kontext der Industrie 4.0 eingegangen. Es wird gezeigt, warum Bedarf nach modernen, schnel-

len und flexiblen Planungsansätzen besteht und wie die Informatik diese unterstützen kann. Im weiteren Verlauf der Arbeit werden dann die methodischen und theoretischen Grundlagen der Fabrikplanung dar- und mögliche Planungssystematiken zur Umsetzung von automatisierter Plan-Generierung vorgestellt. Anschließend werden Technologien aus dem Bereich der Synthese und des Constraintsolving erörtert und in einer prototypischen Softwareanwendung zusammengeführt. Den Abschluss der Arbeit bildet eine Reihe von Experimenten, die das in dieser Arbeit erarbeitete Vorgehen anhand realer Planungsszenarien validieren.

1.1 Anpassungsintelligenz von Fabriken

Die Komplexität moderner Wertschöpfungsketten ist in den letzten Jahrzehnten in Folge zunehmender Globalisierung, Regularien, Produktentwicklungen und veränderten Marktbedürfnissen immer anspruchsvoller geworden[112]. Aufgrund dieser Entwicklungen steht die wirtschaftliche und technische Betriebsführung vor der immanenten Herausforderung, neue Lösungsansätze zur Planung, Steuerung und Modellierung von unternehmensübergreifenden Wertschöpfungsketten zu entwickeln. Vor allem für klassische Produktionsunternehmen zeichnet sich dabei ein Wandel ab: Kennzeichneten früher Verkäufermärkte mit langen, unflexiblen Produktionszyklen, kombiniert mit einer hohen Maschinenauslastung, hohen Losgrößen und langen Durchlaufzeiten in der Produktion das wirtschaftliche Tun, sind es heute Käufermärkten mit kurzen Produktzyklen und Supply Chain Netzwerken, die hohe Lieferbereitschaften und kurze Durchlaufzeiten bereitstellen können. Die Losgrößen, also die Anzahl an zu produzierenden identischen Produkten, werden dabei immer kleiner.

Vor allem in der Automobilbranche macht sich dieser Trend bereits seit Jahren bemerkbar. Beispielsweise war der VW Käfer als erfolgreichstes produziertes Auto im Deutschland der 1960er und 1970er Jahre ein klassisches Großserienprodukt[108]. Die produzierten Modelle waren bis auf wenige Merkmale völlig identisch. Individualisierungsmöglichkeiten gab es nur über die Wagenfarbe sowie einige optionale Features wie ein Faltdach [108]. Die Produktion war hauptsächlich auf Effizienz ausgelegt. Es sollten möglichst viele Modelle schnell produziert werden. Flexibilität und Individualität waren bei der überschaubaren Produktvielfalt weder vorgesehen noch gefragt. Die Automobilindustrie setzte jahrelang auf Fließbandfertigung.

Heute gestaltet sich die Produktion im Automobilsektor anders. Zwar ist Effizienz auch heute noch ein wesentlicher Kosten- wie Erfolgsfaktor, allerdings sind die Fahrzeuge selbst deutlich individueller geworden[108]. Bei der Volkswagen-Tochter Audi sind von den täglich etwa 2000 produzierten Audi-A3s nur noch zwei Fahrzeuge absolut identisch [130, 12]. Die Möglichkeit zur Individualisierung hat über die Jahre hinweg deutlich zugenommen.

Heute lässt sich nahezu jedes Detail eines Fahrzeugs nach den eigenen Vorstellungen des Käufers konfigurieren, wodurch jedes Auto fast zu einem Unikat wird [12]. Dies ist ein Trend, der auch in anderen Branchen mehr und mehr Fuß fasst. Für die Produktion ist dies mit großen und neuen Herausforderungen verbunden, denn individuell gestaltete Produkte sind mit starrer Fließbandfertigung nicht zu produzieren. Zeitgleich verkürzen sich die Produktlebenszyklen der Modelle. Wurde der Käfer über einen Zeitraum von etwa 20 Jahren fast unverändert auf dem Markt angeboten, werden heute Automobil-Modelle bereits nach zwei Jahren erneuert. Dies ist auch eine Folge der sich stetig verändernden Erwartungen der Kunden an die Modelle und die in ihnen verbaute Technik.

Diese Dynamik und stetige Veränderung der Produktionsbedingungen und -Anforderungen machen Fabriksysteme notwendig, die flexibel auf Kundenwünsche und veränderte Produktionsparameter reagieren können und unter allen Voraussetzungen in der Lage sind, wirtschaftlich und erfolgreich zu arbeiten. Der nächste, logische Schritt in der technologiegetriebenen industriellen Entwicklung ist demnach der Weg zu einer „intelligenten Fabrik der Zukunft“ [154, 112]. Der Begriff *Industrie 4.0* ist eine Umschreibung für diesen Transformationsprozess, bei dem vor allem IT-Systeme, künstliche Intelligenz und eine zunehmende Autonomisierung der Systeme in die Entscheidungsfindung über die Produktionsprozesse eingebunden werden. Dieser Entwicklungsverlauf wird entscheidend unterstützt durch Entwicklungen im Hard- und Softwarebereich, die physische und virtuelle Welten zu einem „Internet der Dinge“ zusammenwachsen lassen [156]. Dieser Begriff beschreibt den Prozess, dass Computer als singuläre Geräte in unserer Alltagsumgebung verschwinden und durch neue, intelligente Systeme ersetzt werden. Diese ubiquitären Systeme kommunizieren autonom miteinander. Sie sammeln gemeinsam Inforationen und automatisieren auf Basis dieser Informationen Entscheidungen. Gleichzeitig wird deren Komplexität reduziert. Für Unternehmer, Führungskräfte und Entscheider stellen sich vor diesem Hintergrund Fragen, wie solche Technologien in Unternehmensprozesse integriert werden können und wie der größtmögliche Mehrwert im Hinblick auf die modernen Herausforderungen, die es im Rahmen der Industrie 4.0 zu meistern gilt, aus dem Einsatz solcher Technologien gezogen werden kann.

1.1.1 Motivation und Hintergrund des Graduiertenkollegs 2193

Diese Arbeit ist im Rahmen des *DFG-Graduiertenkollegs 2193 „Anpassungsintelligenz von Fabriken im dynamischen und komplexen Umfeld“* entstanden. Zielstellung des Graduiertenkollegs ist es, Fabrikanpassungsplanung betont interdisziplinär zu erforschen. Besonderer Fokus wird auf das Zusammenwirken verschiedener Fachdisziplinen bei der gemeinsamen Fabrikplanung und Fabrikanpassungsplanung gelegt. Wie bereits erwähnt, zwingt die stark ansteigende Dynamik und Intensität von Umfeldveränderungen Unternehmen immer

häufiger, flexible Fabriksysteme entwerfen und diese bei Bedarf schnell und effizient anzupassen. Diese Anpassungen betreffen zahlreiche Facetten eines Fabriksystems: Bauliche Aspekte, der Einsatz und die Positionierung der Maschinen, die Schulung der Mitarbeiter oder die Anpassung der korrespondierenden IT-Systeme. Dadurch zeichnen sich Fabrikanpassungen stets durch eine hohe Komplexität und Interdisziplinarität aus. Dem steht in der Wissenschaft bislang jedoch kein ausreichender methodischer oder terminologischer Austausch der notwendigen Disziplinen gegenüber [147].

Die zunehmende Individualisierung der Produkte, kürzer werdende Produktlebenszyklen, größere Absatzschwankungen sowie die zunehmende Digitalisierung und andere Entwicklungen (wie z.B. restriktivere Umweltgesetzgebungen) stellen Faktoren dar, die dazu beitragen, dass systemimmanente Flexibilität und Wandlungsfähigkeit notwendig sind. Die steigende Dynamik, Intensität und Häufigkeit von Veränderungen im Unternehmensumfeld bedingen einen permanenten Fabrikplanungsprozess bzw. eine zunehmend kurz-zyklische Anpassungsnotwendigkeit. Die Geschwindigkeit, mit der Fabriksysteme angepasst werden können, stellt insbesondere für Produktionsunternehmen in Hochlohnländern einen zentralen strategischen Erfolgsfaktor dar [147].

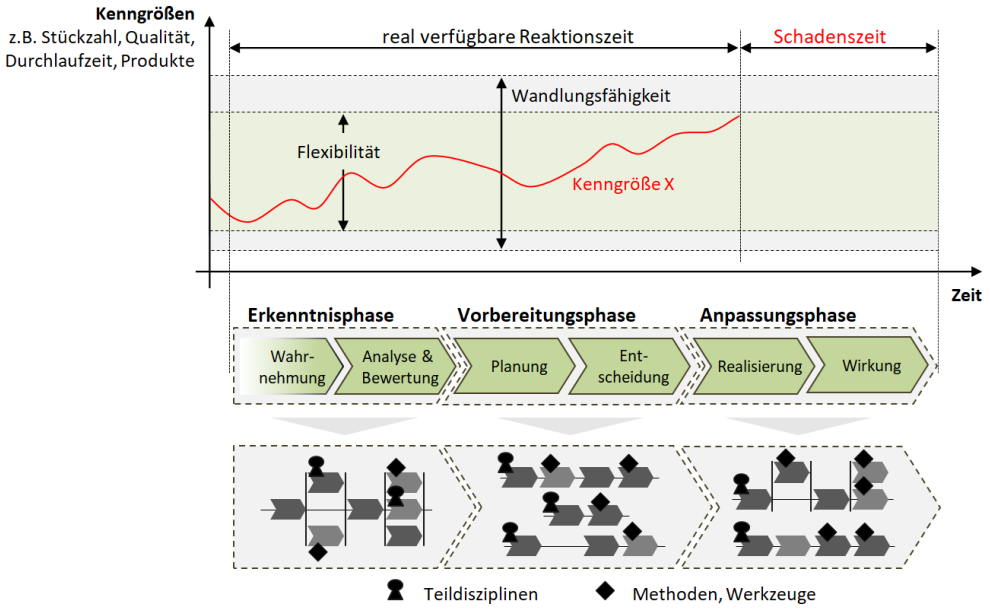


Abbildung 1.1: Flexibilität und Wandlungsfähigkeit nach [147]

Eine Systemanpassung ist genau dann erforderlich, sobald die Veränderung eines Einflussfaktors Anforderungen an das Produktionssystem erzeugt, welche außerhalb der spezifischen, systemischen Flexibilitätskorridore liegen. Das bedeutet, dass diese nicht mehr über das vorhandene System abgefangen werden können (vgl. Abbildung 1.1). In die-

sem Fall muss ein Anpassungsprozess am System gestartet werden. Das Graduiertenkolleg stützt sich dabei auf den Fabrikanpassungsprozess nach *Morales* [101] (Abbildung 1.2).

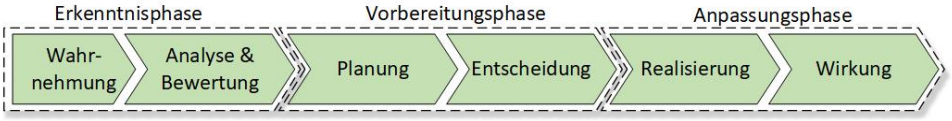


Abbildung 1.2: Der Fabrikanpassungsprozess nach [101]

Zu Beginn eines solchen Prozesses steht immer eine sogenannte Erkenntnisphase. Die Herausforderung in dieser Phase besteht darin, die Entwicklung systemrelevanter Kenngrößen zu erfassen und einzuordnen. Nur so kann erkannt werden, ob ein System aktuellen Anforderungen gerecht wird und ein Anpassungsbedarf vorliegt. Ein Beispiel für eine solche Erkenntnisphase ist die Zeit, in der ersichtlich wird, dass das vorhandene Produktionssystem selbst bei voller Auslastung nicht die gewünschten Stückzahlen eines Produktes herstellen kann. In diesem Fall liegt ein Anpassungsbedarf in Form einer notwendigen Kapazitätsaufstockung vor, was bedingt, dass mit der Planung einer Fabrikanpassung begonnen werden muss. In deren Verlauf werden in der Regel mehrere Lösungsalternativen erarbeitet. Wie die Planung im Fabrikkontext abläuft und wie konkrete Herausforderungen und Hemmnisse dabei aussehen, wird in Kapitel 2 dargelegt. Auf die Planung folgt die Entscheidung für eine Anpassungsvariante sowie deren anschließende Realisierung (auf Basis des in der Planungsphase definierten Ablaufs).

Im Rahmen der Erkenntnisphase ist es von großer Bedeutung, den aktuellen Zustand des Fabriksystems anhand geeigneter Kennzahlen verlässlich zu überwachen und (Fehl-) Entwicklungen erkennbar zu machen. Ebenso sollte die Wahrnehmungszeit signifikant reduziert werden. Die systematische Zukunftsplanung bzw. Veränderungsantizipation ist Gegenstand verschiedener Forschungsbemühungen [147], ebenso wie die Analyse und Bewertung von Produktionssystemen und Einflussfaktorveränderungen. Im Rahmen der Maßnahmenplanung und -entscheidung gilt es, Assistenzmethoden zu entwickeln, die fallspezifische Handlungsoptionen liefern. Ein besonderer Fokus liegt auf der Herausstellung der Interdependenzen der Maßnahmenwirkungen, die sich im Falle von Maßnahmenbündeln ergeben. Denn die Planung darf sich keineswegs auf die Maßnahmen zur Beseitigung der identifizierten Differenz zwischen existierendem und zukünftigem Leistungs- bzw. Kostenprofil beschränken, sondern muss das betrachtete System unter Beachtung sämtlicher Systemelemente und ihrer Zusammenhänge berücksichtigen. Im Bereich der Fertigung beispielsweise müssen verfahrenstechnische Anpassungsmaßnahmen (wie Veränderungen von Prozessparametern) bereits in der Planungsphase unter Berücksichtigung des gesamten Produktionssystems und aller Zielgrößen ganzheitlich aufeinander abgestimmt werden.

Am Ende des Anpassungsprozesses steht eine Evaluationsphase (*Wirkung*), bei der die Auswirkungen der durchgeführten Anpassungsmaßnahme im Hinblick auf deren Zielstellung ausgewertet wird. Die dort gewonnenen Daten wiederum können den Grundstein für einen neuen Anpassungsprozess bilden.

1.1.2 Rolle der Informatik in Bezug auf Anpassungsintelligenz

Um das Ziel und den Mehrwert dieser Arbeit im interdisziplinären Kontext der Industrie 4.0 und der damit verbundenen Anpassungsintelligenz von Fabriken zu verdeutlichen, ist es zunächst notwendig, die Rolle und Aufgaben der Informatik zu definieren und die Stellen, an denen sie zur Wertschöpfung in Fabriken und Unternehmen beitragen kann, herauszustellen.

Durch die steigende Bedeutung der Aspekte einer Vernetzung, künstlicher Intelligenz, Monitoring oder Daten-Management kommt der Informatik im Rahmen der Industrie 4.0 eine zentrale Bedeutung zu. Die Entwicklung ist in etwa vergleichbar mit den Veränderungen in der Heimelektronik oder Medienindustrie der vergangenen Jahre, bei denen IT- und Software-Lösungen allgegenwärtig wurden. Es ändert sich dadurch vor allem die Wahrnehmung der Informatik durch die anderen involvierten Disziplinen[86]. Während sie lange allein als Kostenfaktor gesehen wurde, sind ihre Beiträge zum Gesamterfolg eines Unternehmens nun direkt greif- und wahrnehmbar. In der Folge steigt auch das Bewusstsein, dass eine tiefgreifende Einbindung der Informatik in Entscheidungsprozesse nicht nur vorteilhaft, sondern fast unumgänglich ist[69]. Die Aufgaben und Herausforderungen gliedern sich dabei in drei Bereiche auf:

- *Produkt:* Informationstechnische Lösungen, Produkte und Dienstleistungen kommen im vom Unternehmen hergestellten Produkt zum Einsatz. Dabei handelt es sich beispielsweise um vernetzte Produktionsmaschinen oder Autos [69].

- *SmartFactory:* In der SmartFactory werden informationstechnische Lösungen, Produkte und Dienstleistungen vereint, welche zur Produktion von Gütern verwendet werden. Beispiele hierfür sind vernetzte Produktionsmaschinen mit Integration in ein ERP-System oder eine kundengetriggerte 3D-Druck-Produktion [69].

- *Business Support:* Die Rolle der Informatik im Business Support wird durch informationstechnische (Software-)Lösungen, Produkte und Dienstleistungen definiert. Die Prozesse des Unternehmens (z.B. Auftragsmanagement, Finanzen- und Controlling, aber auch Planungsabläufe) können auf diese Weise und unter Zuhilfenahme von Techniken mit künstlicher Intelligenz unterstützt, gesteuert und beschleunigt werden [69].

Im Rahmen der Fabrikanpassung und des oben definierten Anpassungsprozesses kann sich die Informatik vor allem im Bereich des Business Supports durch die Bereitstellung von Softwarelösungen einbringen. Entscheidend dafür sind Kompetenzen im Produktmanagement und der SmartFactory sowie der Sensorik und der Arbeitsweise der eingesetzten Maschinen und Prozesse im Unternehmen. Ebenfalls ist eine tiefgreifende Kenntnis über das zugrundeliegende Geschäftsmodell von Vorteil. Nur durch diese Kompetenzen, die einen regen Austausch mit den Beschäftigten aus dem Management und der Fabrikplanung notwendig macht, ist es gewährleistet, dass die durch die Tools bereitgestellten Daten für das Management von Nutzen und Relevanz sind.[69, 86]

Ziel der Informatik im Kontext des Graduiertenkollegs ist es, den Fabrikbetrieb bestmöglich vor allem bei der Entscheidungsfindung bei Anpassungs- und Konfigurationsmaßnahmen zu unterstützen. Dies kann neben der Aufbereitung sämtlicher für einen Entscheidungsprozess relevanten Daten auch durch die Integration von KI- und Simulationsmodellen in den Entscheidungsprozess geschehen. Dadurch ist es unter anderem möglich, die Konsequenzen einzelner Entscheidungen besser abzuschätzen, aber auch bereits vorhandene Prozesse zu optimieren oder im Problemfall Lösungsmöglichkeiten aufzuzeigen. Auch durch die automatische Generierung von Software-Werkzeugen und Modellen kann der Anpassungsprozess unterstützt werden [54]. Durch Automatisierung der Konfiguration von Fabriken und Planungsworkflows können den verantwortlichen Personen komplexe Aufgaben abgenommen werden. Die Planung und Konzeption der Pläne ist aktuell immer noch eine Aufgabe, die vom Planungsteam manuell und projektspezifisch ausgeführt wird. Durch die zunehmende Komplexität der Planungsaufgaben und unter dem Druck, diese in immer kürzerer Zeit durchzuführen, steigt der Bedarf an technologischer Unterstützung (vgl. Kapitel 2.2 und [109]). Ein wichtiger Faktor ist vor allem die Zeit. Durch einen hohen Automatisierungsgrad lassen sich Handlungsoptionen wesentlich schneller erkennen und evaluieren. Damit kann das Management wiederum schneller auf Veränderungen reagieren. Die Informatik nimmt insofern auch die Rolle eines Beschleunigers der Wertschöpfung im Unternehmen ein.

1.2 Aufbau und Ziele der Arbeit

Vor diesem Hintergrund ist es das Ziel der Arbeit, die technischen Möglichkeiten des Software-Engineerings und der Software-Synthese zu nutzen, um den Planungs- und Anpassungsprozess einer Fabrik effizient zu unterstützen. Dabei konzentriert sich die Arbeit auf den *Business Support* Charakter der Informatik innerhalb der Anpassungsintelligenz von Fabriken. Zu diesem Zweck soll eine Möglichkeit geschaffen werden, die es erlaubt, komplexe Planungsaufgaben automatisiert bewältigen zu können. Die im Rahmen dieser Arbeit entwickelte Technologie soll es ermöglichen, Planungsworkflows jederzeit nach Be-

darf passend zur jeweiligen Projekt-Situation und -Spezifikation generieren zu können, um den Plaungs- und Anpassungsprozess signifikant zu unterstützen. Es sollen explizit nicht nur einzelne Planparameter (zum Beispiel Pufferzeiten) automatisch erzeugt, sondern gesamte Planstrukturen und unterschiedliche Lösungswege passend zu einem Planungsfall entwickelt und aufgezeigt werden. Die generierten Pläne sollen zudem im jeweiligen Projekt durch- und ausführbar sein.

Um dieses Ziel zu erreichen, wird im folgenden Kapitel die Disziplin der Fabrikplanung näher betrachtet. Es wird ein Überblick über bestehende und etablierte Planungsvorgehen, -konzepte und -systeme gegeben. Diese werden im Kontext moderner Anforderungen bewertet und ihre Eignung zur Verwendung für das beschriebene Ziel dieser Arbeit untersucht. Ebenso werden Werkzeuge und Formalismen (wie die Netzplantechnik, Gantt-Charts oder so genannte Weg-Zeit-Diagramme) untersucht, die zur Erstellung, Darstellung und zur Verwaltung von Planungsworkflows verwendet werden. Auch hier wird besonderes Augenmerk auf die Eignung für eine Automatisierung gelegt. Am Ende des Kapitels soll aufgezeigt werden, welche Konzepte und Werkzeuge für den Einsatz in einer wandlungsfähigen Fabrik mit ihren permanenten Anpassungsbedürfnissen am besten geeignet sind.

Ein Blick wird auch den Punkten gewidmet, an welchen besonderer Bedarf an intelligenter technologischer Unterstützung besteht und wie diese Unterstützung in den Planungsprozess integriert werden kann. Dies alles ermöglicht eine Einordnung, wie und nach welchen Verfahren eine automatisierte Planung umgesetzt werden kann, welche Zusammenhänge zu beachten sind, welche Anforderungen gelten und welche Formalismen und Tools zur Darstellung und Umsetzung der Pläne geeignet sind. Im darauf folgenden Kapitel werden dann Ansätze aus der Informatik betrachtet, die sich potenziell eignen, eine automatisierte Planung umsetzen zu können. In diesem Zusammenhang werden insbesondere Synthese-Techniken, Logiken und Constraint-Solving-Ansätze betrachtet. Als Ergebnis von Kapitel 3 ist die Basis für ein Vorgehen zur automatisierten Erzeugung von Planungsworkflows in Verbindung mit einer taxonomisch strukturierten Sammlung an Bausteinen gelegt, um ein prototypisches Verfahren umsetzen zu können. Kapitel 4 widmet sich der Umsetzung dieses Verfahrens und erläutert anhand einer Implementierung dessen Funktionsweise. Die Funktionalität dieser Implementierung wird in Kapitel 5 anhand von Experimenten mit Planungsdaten aus realen Planungsprojekten evaluiert und in Kapitel 6 kritisch bewertet.

Zunächst sollen der folgende Abschnitt einen Überblick über die maßgeblichen wissenschaftlichen Beiträge und Erkenntnisse dieser Arbeit geben. Anschließend stellt Kapitel 1.4 die wissenschaftlichen zentralen Publikationen, , die im während der Arbeit an dieser Dissertation entstanden sind vor.

1.3 Wissenschaftliche Beiträge

Die wissenschaftlichen Beiträge dieser Arbeit lassen sich in theoretische und technische Beiträge differenzieren. Nach der Auflistung der Beiträge werden die Primärpublikationen, ihre jeweiligen Beiträge und eine Abgrenzung zu den Original- und Eigenbeiträgen der Arbeit vorgestellt.

Die Arbeit liefert die folgenden **theoretischen Beiträge**:

1. Zunächst werden Vorgehen und Modelle zum modularen Planen von Fabriksystemen erarbeitet und vorgestellt sowie passende Darstellungsformen gezeigt.
2. Durch Kombination der vorgestellten Planungsvorgehen wird eine umfassende Bibliothek von gekapselten und wiederverwertbaren Planungsbausteinen (Modulen) erstellt, aus denen je nach Bedarf passende Planungsworkflows erzeugt werden können.
3. Um die Module der in Kapitel 3.4 vorgestellten Synthese verwendbar zu machen, ist eine Überführung der Module in ein Typsystem erforderlich. Kapitel 4.3 zeigt deshalb eine eine Typisierung der Planungsmodule sowie deren Abhängigkeiten.
4. Darauf aufbauend wird die Anwendbarkeit des Combinatory-Logic-Synthesis-Frameworks (ein typ-basiertes Verfahren für die Software-Synthese) und die Kombination mit SMT-Solvern dargestellt, wodurch eine Methode zum Ad-Hoc-Planen nach Bedarf entsteht. Diese Methode bietet:
   - die Zuordnung eines abstrakten Kontextes und der Problemdomäne der Fabrikplanung hin zu einer prägnanten Spezifikation in kombinatorischer Logik (siehe auch Punkt 3), welche die Semantik der Domäne bewahrt.
   - eine effizient berechenbare Synthese von Planungsworkflows in Form von Netzplänen, die die Spezifikationen unter Verwendung der kombinatorischen Logik nutzt.
   - ein Mapping von den Synthese-Ergebnissen in das etablierte Planungswerkzeug der Netzpläne.
   - eine zielgerichtete Synthese und Generierung von Planungsworkflows, die die jeweiligen Projektgegebenheiten berücksichtigt.

Die **technischen Beiträge** ergänzen die theoretischen Beiträge und können wie folgt zusammengefasst werden:

5. Vorgestellt wird eine Verknüpfung von kombinatorischer Synthese und SMT-Solving Methoden zur praktischen Implementierung des Vorgehens aus Punkt 4. Die Implementierung nutzt die Vorteile, wie sie in den theoretischen Beiträgen entwickelt wurden.

6. Es wird eine Implementierung der Methodik zum automatischen Generieren von Netzplänen einschließlich der genannten Synthese vorgestellt. Experimentell validierte Ergebnisse belegen die praktische Einsetzbarkeit dieses Ansatzes.

7. Die entwickelte Softwarearchitektur zeigt unter Verwendung der Implementierung des kombinatorischen Logiksynthesealgorithmus und der Methode der automatischen Erzeugung von Netzplänen (auf Basis der getypten Planungsmodule), wie sie in Verbindung mit etablierten Softwarelösungen zur Planung arbeiten kann.

8. Die Softwarearchitektur und die Implementierung werden in drei Experimenten an realen Fabrikplanungsprojekten getestet, um die Einsetzbarkeit der Architektur und der Implementierung in realen Projekten zu belegen.

Vorarbeiten, die zur Erreichung dieser Ziele notwendig waren, konnten zuvor schon durchgeführt und veröffentlicht werden. Die wichtigsten Artikel und Beiträge werden im folgenden Abschnitt vorgestellt.

1.4 Eigene Publikationen

Im Rahmen der Vorarbeiten zu dieser Dissertation sind bereits einige Forschungsbeiträge und Publikationen entstanden, die das Thema der automatischen Konfiguration von Planungsworkflows behandeln. Die meisten dieser Artikel wurden in enger Kooperation mit *Julian Graefenstein* vom Lehrstuhl für Unternehmenslogistik der Fakultät Maschinenbau an der TU Dortmund erstellt und entsprechend gemeinsam publiziert.

Viele Ergebnisse dieser Publikationen sind in diese Dissertation eingeflossen, weshalb sie an dieser Stelle vorgestellt werden. Zunächst werden in den Abschnitten 1.4.1 bis 1.4.3 konzeptionelle Veröffentlichungen vorgestellt, die die Anküpfungspunkte für die Informatik in der Industrie 4.0 aufzeigen, das Anwendungsgebiet für diese Dissertation abstecken und das generelle Vorgehen der bei der automatischen Komposition von Fabrikplanungsworkflows beschreiben. Anschließend folgen in den Abschnitten 1.4.4 bis 1.4.6 Veröffentlichungen, die die Implementierung und die Anwendung des Vorgehens anhand diverser Beispielszenarien vorstellen.

1.4.1 Einfluss der Industrie 4.0 auf ausgewählte Kompetenz- und Rollenprofile – Entwicklungen von Berufsbildern unter besonderer Berücksichtigung von IT-Kompetenzen. In:

- **Autoren:** Christin Schumacher, Hendrik Lager, Philipp Regelmann, Jan Winkels, Julian Graefenstein
- **Erschienen in:** Industrie 4.0 Management 35 (2), S. 42-57, 2019, GITO Verlag, Berlin [134]

Ausgehend von der Gegenüberstellung der Entwicklung von Wissens-, Kompetenz- und Rollenprofilen operativer und strategischer Beschäftigtengruppen im Zuge der Industrie 4.0 von Lager et al. [86], wird in dieser Publikation die Rolle des überschneidenden Schwerpunktes IT näher untersucht und auf die taktische Ebene am Beispiel der Produktionsplanung ausgelegt. Dabei werden die Auswirkungen des Bedarfs verstärkter IT-Kompetenz in allen Bereichen produzierender Unternehmen auf eine Aufweichung von aktuellen Rollenprofilgrenzen dargestellt. Aus informatischer Sicht wurden vor allem Anknüpfungspunkte für automatisierungs- und entscheidungsunterstützende Softwarelösungen untersucht.

1.4.2 Intelligente Orchestrierung von Planungsprozessen - Anwendung von logikbasiertem Constraintsolving in der Fabrikplanung

- **Autoren:** Julian Graefenstein, Jan Winkels, David Scholz, Michael Henke und Jakob Rehof
- **Erschienen in:** Zeitschrift für Wirtschaftlichen Fabrikbetrieb, Jahrg. 112 (2017), Dokumentennummer ZW 111696, Carl Hanser Verlag, München [54]

In diesem Ansatz wird herausgestellt, dass bis dato existierende Modelle der Fabrikplanung aufgrund der komplexen und dynamischen Natur der Planung nicht in der Lage sind, den Herausforderungen einer ständigen Anpassungen von Fabriksystemen zu begegnen. Er stellt zudem den modularen Entwurf für Planungsprozesse vor, welcher mithilfe von Logik und Constraintsolving orchestriert wird. Der Entwurf verfolgt das Ziel der Entwicklung einer Planungshilfe, die eine flexible und dauerhafte Anpassung der Planungsprozesse an sich ständig ändernde Planungssituationen ermöglicht, um schnell und effizient das Fabriksystem auch unter dynamischen Bedingungen anpassen zu können.

1.4.3 Automatisierungspotenziale für Controlling-Kennzahlen aus den Daten eines BIM-Projekts

- **Autoren:** Lisa Theresa Lenz, Jan Winkels, Philipp Regelmann
- **Erschienen in:** Lenz, L. T.; Regelmann, P.; Winkels, J.; Gralla, M.: Automatisierungspotenziale für Controlling Kennzahlen aus den Daten eines BIM-Projektes. In: Sundermeier, Matthias; Meinen, Heiko (Hrsg.): Bauwirtschaft – Markt, Management, Recht, Heft 2, Werner Verlag, 2019. [92]

Der Artikel befasst sich mit dem Projektcontrolling in Bauprojekten und der zugehörigen IT-Umgebung. Speziell werden Potenziale für die weitere Automatisierung von Projekt-abläufen untersucht. Hierzu sind die Kennzahlen in der Baubranche zu identifizieren und eine IT-gerechte Übersetzung zu generieren, ohne dabei die vollständige Datenintegrität und -Konsistenz zu verletzen. Die Projektabwicklung und die gegebene IT-Infrastruktur werden anhand des Workflows zur Erstellung eines Baufortschritts aus Max Bögl analysiert.

Besonderes Augenmerk gilt den Schnittstellen zwischen verschiedenen Softwaresystemen, die während eines solchen Prozesses verwendet werden. Wenn Daten von einer Software zur anderen übertragen werden, bestehen automatisch Schwachstellen in der Datenintegrität. Insbesondere bei einer manuellen Datenübertragung besteht ein erhöhtes Risiko von Übertragungsfehlern. Eine mögliche Lösung ist die Implementierung einer Cloud-basierten Anwendung zur Verwaltung des Datenflusses.

1.4.4 Automated processing of planning modules in factory planning by means of constraint-solving using the example of production segmentation

- **Autoren:** Julian Graefenstein, Jan Winkels, David Scholz, Oliver Seifert, Michael Henke, Jakob Rehof
- **Erschienen in:** Hankammer, Stephan and Nielsen, Kjeld and Piller, Frank T. and Schuh, Günther and Wang, Ning, Customization 4.0, Springer proceedings in business and economics, 2018 [55]

Inhalt dieser Veröffentlichung [55] ist die Aufbereitung und Darstellung der wesentlichen Daten, die als Grundlage für die Fabrikplanung und die Anpassung von Fabriken benötigt werden. Häufig werden diese Daten in beliebigen und unstrukturierten Formen und Orten irgendwo auf einem Datenträger gespeichert oder stehen überhaupt nicht zur Verfügung. Eine planungsgerechte Aufbereitung dieser Daten für den Planungsprozess kann zu einem hohen Aufwand führen. Um dieser Situation entgegenzuwirken, kann ein Data-Warehouse-System im Rahmen von Business Intelligence verwendet werden, um die Daten in einer zentralisierten und konsistenten Form bereitzustellen. Die Vorteile einer aktuellen und konsistenten Datenbasis zeigt ein Beispiel für die Produktionssegmentierung. Mit der Planung der Werksanpassung über individuell einstellbare Planungsmodule können Planungsaufgaben automatisch oder teilweise automatisiert abgewickelt werden. An einem gegebenen Beispiel einer Schraubstockproduktion, die in vier Varianten hergestellt werden kann, werden die Vorteile aufgezeigt und der Ansatz detailliert erläutert. Constraint-Solving, der modulare Planungsprozess und die im Data Warehouse verfügbaren Daten ermöglichen eine automatische Segmentierung und reduzieren somit die Planungszeit.

1.4.5 Automatic Composition of Rough Solution Possibilities in the Target Planning of Factory Planning Projects by Means of Combinatory Logic

- **Autoren**: Jan Winkels, Julian Graefenstein, Tristan Schäfer, David Scholz, Jakob Rehof, Michael Henke
- **Erschienen in:** 8th International Symposium, ISoLA 2018, Limassol, Cyprus, November 5-9, 2018, Proceedings, Part IV, Leveraging Applications of Formal Methods, Verification and Validation. Industrial Practice, 2018, Springer International Publishing, Cham [163]

Zunehmender Wettbewerb, stärkere Kundenorientierung, kürzere Produktlebenszyklen und eine beschleunigte technologische Entwicklung bedeuten, dass Unternehmen vor der Herausforderung stehen, die eigene Produktion in immer kürzeren Abständen an die

Marktgegebenheiten anzupassen. Das Fabrikplanungsprojekt wird immer komplexer; es steht jedoch immer weniger Zeit für Anpassungen zur Verfügung. Insbesondere in der ersten Planungsphase werden Ziele ohne verlässliche Planungsinformationen für den weiteren Verlauf festgelegt, was weitreichende Konsequenzen für das Ergebnis einer erfolgreichen Planung hat. Diese Publikation zeigt eine Möglichkeit, bereits in einer frühen Phase der Zielplanung sinnvolle Lösungsalternativen zu generieren, um einen effizienten Zeit- und Kostenplanungsprozess zu ermöglichen. Mit Hilfe von kombinatorischer Logik und SMT-Solving wird eine Möglichkeit zur Variantenkompilierung auf Basis von zuvor definierten Ziel- und Rahmenparametern erstellt. Durch Verwendung eines kombinatorischen Logikansatzes können automatisch erste grobe und plausible Lösungsvarianten generiert werden, auf deren Basis dann der detaillierte endgültige Planungsprozess zum Erreichen der gesetzten Ziele ermittelt und erstellt werden kann. So können Planungsengpässe aufgrund einer falschen Variantenauswahl und ein großer Zeitaufwand für die Erstellung von Lösungsvarianten vermieden werden.

1.4.6 Smart Factory Adaptation Planning by means of BIM in Combination of Constraint Solving Techniques

- **Autoren:** Lisa Theresa Lenz, Julian Graefenstein, Jan Winkels, Mike Gralla
- **Erschienen in:** Proceedings of the International Council for Research and Innovation in Building and Construction (CIB), World Building Congress 2019 – Constructing Smart Cities, Hongkong, 2019 [93]

Auch in dieser Veröffentlichung geht es um die Notwendigkeit für produzierende Unternehmen ihre Produktionssysteme stetig anzupassen. Damit einher gehen häufig entsprechende Umbaumaßnahmen an den Fabrikgebäuden und deren Infrastruktur. Ein Anpassungsprozess, bei dem es sich entweder um eine eher kleine Anpassung (zum Beispiel von nur einer Produktionszelle) oder aber um die Reorganisation einer gesamten Fabrik handelt, sollte so kurz und effektiv wie möglich sein. Diese Dynamik bringt immer mehr Modernisierungen in kürzerer Zeit mit sich. In Verbindung mit der damit einhergehenden Komplexität steigt die Schwierigkeit der Anpassungen.

In dem Moment, in dem das Planungsteam die aufgabenorientierte Planungsphase beendet und die Ergebnisse der Bauleitung übergibt, entsteht eine Lücke in den Ansätzen, die die Schnittstellen zwischen der Planungsphase und der Realisierungsphase eines Fabrikplanungsprojekts verwalten. In diesem Beitrag wird diese Lücke adressiert. Bereits in der Planungsphase werden automatisch vorverarbeitete Informationen und Module für die Realisierungsphase bereitgestellt, um den Entscheidungsprozess und insbesondere den Prozess der Erstellung eines Zeitplans für die Realisierung zu beschleunigen. Anhand eines

beispielhaften Szenarios wird gezeigt, wie der Ansatz verwendet wird, um einen Teil einer Montagelinie durch Roboter zu ersetzen. Es wird gezeigt, dass der Ansatz in der Lage ist, die Ergebnisse der Planungsphase in den Zeitplan der Realisierung zu übernehmen, was die Realisierungsphase beschleunigt, indem bestimmte Aufgaben automatisch orchestriert werden.

1.4.7 Trends In Automatic Composition Of Structures For Simulation Models In Production And Logistics

- **Autoren:** Sigrid Wenzel, Jakob Rehof, Jana Stolipin, Jan Winkels
- **Erschienen in:** Proceedings of the 2019 Winter Simulation Conference (CIB), World Building Congress 2019 – Maryland, 2019 [135]

Dieser Artikel stellt die Herausforderungen der automatischen Generierung von Simulationsmodellen in der Produktion und in der Logistik dar. Zu diesem Zweck werden zunächst bestehende Ansätze in diesem Bereich vorgestellt, analysiert und umgesetzt. Es werden Trends auf dem Gebiet der automatischen Zusammensetzung von Strukturen für Simulationsmodelle ermittelt und präsentiert.

Ziel ist es, die Anwendbarkeit der Synthese zur Erzeugung und zur automatisierten Zusammensetzung von Strukturen für Simulationsmodelle zu veranschaulichen. Der Fokus liegt insbesondere auf der Synthese logischer Strukturen mittels kombinatorischer Logik. Die Strategie, wie diese Logik die automatisierte Generierung von Strukturvarianten in Simulationsmodellen unterstützen kann, wird anhand eines Beispiels der Intralogistik dargestellt. Hierzu werden verschiedene Möglichkeiten von Strukturvarianten verdeutlicht und ihr Potenzial für eine automatisierte Zusammensetzung aufgezeigt, womit auch Anknüpfungspunkte an die Ergebnisse dieser Dissertation gegeben sind.

Kapitel 2

Fabrikplanung

2.1 Grundbegriffe der Fabrikplanung

Für eine solide Auseinandersetzung mit dem Themenfeld der Fabrikplanung ist es notwendig, zunächst die grundlegende Bedeutung des Begriffs selbst darzulegen. Der Begriff der *Fabrik* beschreibt „ein[en] Ort, an dem Wertschöpfung durch arbeitsteilige Produktion industrieller Güter durch den Einsatz von Produktionsfaktoren erfolgt“ [151]. Über diese Definition des VDI hinaus wird der Begriff der Fabrik in anderen Teilen der einschlägigen Literatur erweitert. In [164] wird der Wertschöpfungsaspekt zum Beispiel auch auf den Aspekt von Dienstleistungen ergänzt. Alles in allem existiert allerdings ein Konsens darüber, die Fabrik als einen Ort der Wertschöpfung zu betrachten.

Wenn die Definitionen der Fabrik noch insgesamt zumindest grundlegend konsistent sind, zeigt sich, dass der Begriff der *Planung* weitaus differenzierter betrachtet und definiert wird. Was genau Planung ausmacht und welche Aspekte in die Planung einfließen, wird je nach Anwendungsfall, Projekt, Domäne und Projektzeitpunkt unterschiedlich betrachtet. Eine allgemeine Definition bietet der REFA Verband, der Planung als „gedankliche Vorwegnahme einer zielgerichteten und aktiven Zukunftsgestaltung“ beschreibt [116]. Planung beinhalte „das systematische Suchen und Festlegen von Zielen sowie Aufgaben und Mitteln zum Erreichen dieser Ziele“[116]. Dieses grundlegende Verständnis von Planung lässt sich ebenso auf das Planen von Fabriken und Fabrikgebäuden anwenden. Allerdings finden sich in der Literatur viele unterschiedliche Auffassungen, wie *Fabrikplanung* zu verstehen ist. Der Begriff geht auf den Beginn der Industrialisierung im 18. Jahrhundert zurück. Es ist naheliegend, dass mit den ersten Fabriken auch erste praktische Ansätze entwickelt wurden, wie die neuartigen Produktionsstätten zu konzipieren sind. Die wissenschaftliche Auseinandersetzung mit der Fabrikplanung hingegen begann erst im 20. Jahrhundert und geht vor allem auf die Arbeiten von Kettner [79] und Rockstroh [123] zurück [127]. Später entwickelten zum Beispiel Grundig [57] und Aggtelelky [2] eigene Planungsansätze (siehe Kapitel 2.3) und entwickelten damit einhergehend auch das allgemeine Verständnis von Fabrikplanung weiter. Heute wird vor allem in der deutschen Industrie und Wissenschaft in der Regel auf die Definition des Fabrikplanungsausschusses des Vereins der deutschen Ingenieure (VDI) verwiesen, die besagt:

Definition 2.1

*„**Fabrikplanung** ist der systematische, zielorientierte, in aufeinander aufbauenden Phasen strukturierte und unter Zuhilfenahme von Methoden und Werkzeugen durchgeführte Prozess zur Planung einer Fabrik von der ersten Idee bis zum Hochlaufen der [geplanten] Produktion.“ [151]*

Diese Definition lässt sich sowohl auf einen kompletten Neubau einer Fabrik anwenden (erste Idee einer Fabrik auf dem Papier bis hin zum Start der Produktion) als auch auf

die Fabrikplanung, die im Rahmen eines Anpassungsprozesses notwendig wird. In einem solchen Fall ist die erste Idee das Aufkommen des Anpassungsbedarfs; das Hochlaufen der Produktion ist dann der Start der Produktion nach erfolgter Anpassung. Auf diesem Fabrikplanungsbegriff basiert auch diese Arbeit.

Analog zu dem Begriff *Fabrikplanung* lässt sich der Fabrikbetrieb definieren. Der Begriff Fabrikbetrieb beschreibt das Betreiben, Lenken und Steuern der Abläufe in einer Fabrik. Dies geschieht unter der Prämisse, die Zielvorgaben des Unternehmens durch eine sozial-ökonomische und ökologisch-ressourceneffiziente Aufbau- und Ablauforganisation im partizipativ-transparenten Zusammenwirken der Mensch-Technik-Organisation-Komponenten (MTO) sowie Kooperation innerhalb und außerhalb der Fabrik zu sichern [127]. Es lässt sich zusammenfassen, dass die Fabrikplanung aus personeller, technisch-organisatorischer, ökonomischer und ökologischer Sicht Gestaltungslösungen und Potenziale bereitstellt, die durch den Fabrikbetrieb für unterschiedliche, unternehmerische Ziele genutzt und ausgeschöpft werden [127].

2.1.1 Aufgaben im Fabriklebenszyklus

Da Fabrikplanung nach Definition 2.1 den gesamten Prozess von der ersten Idee und Absicht eine Fabrik zu errichten bis hin zum Betriebsbeginn nach Abschluss aller Bau- oder Anpassungsmaßnahmen umfasst, spielt sie in weiten Teilen des klassischen Fabriklebenszyklus eine entscheidende Rolle. Dieser Fabriklebenszyklus und der Einfluss der Fabrikplanung werden im folgenden Kapitel betrachtet.

Aufgrund der weitreichenden Einflüsse der Fabrikplanung auf den Bau, die Anpassung und den Betrieb einer Fabrik, umfasst sie sehr unterschiedliche Aufgabenfelder. Welche Aufgabenfelder dabei genau Bestandteil der Fabrikplanung darstellen, unterscheidet sich wieder je nach verwendeter Definition. Zum Teil werden einzelne Aufgaben in unterschiedlichen Definitionen auch verschiedenen Aufgabenfeldern zugeordnet. Grundig [57] und Wiendahl [158, 160] zum Beispiel betrachten die Generalbebauungsplanung als eigenen Aufgabenbereich. In anderen Quellen (wie [127, 112, 128]) hingegen ist diese Teil der Fabrikstrukturplanung. Die Aufgabenfelder nach der VDI Richtlinie 5200 sind in Abbildung 2.1 dargestellt.

Nach dem VDI [151] existieren darüber hinaus verschiedene Planungsebenen. Diese adressieren unterschiedliche Aspekte der Fabrikplanung und sind dementsprechend auch unterschiedlichen Hierarchiestufen im Unternehmen zugeordnet. Die Ebenen gliedern sich in Produktionsnetz, Gebäude, Segmente und Arbeitsplatz. Wie in Kapitel 1 erläutert, konzentriert sich diese Arbeit auf die Planungsaufgaben, deren bauliche Arbeiten an der

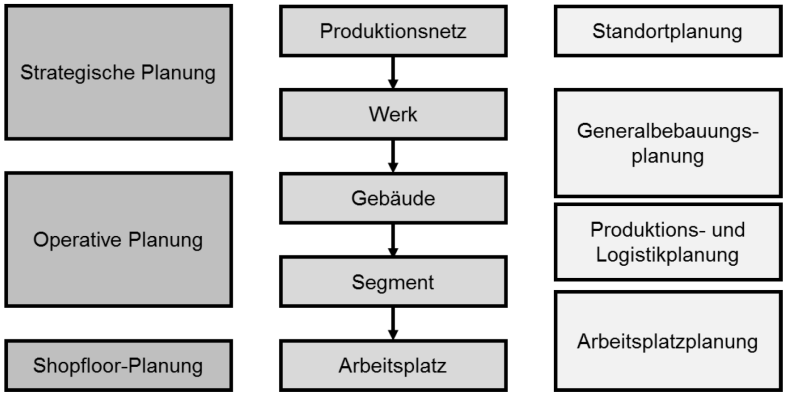


Abbildung 2.1: Planungsebenen der Fabrikplanung nach VDI 5200 [151]

Produktion (Segmente) oder dem Gebäude (Gebäude, Werk) zugeordnet sind. Abbildung 2.1 zeigt außerdem, wie sich die einzelnen Aufgabenbereiche der Fabrikplanung in die Planungsebenen einordnen. Daraus ist wiederum abzuleiten, dass sich diese Arbeit auf die Planungsebenen der strategischen und operativen Planung beschränkt.

Der Fabrikplanung und ihren Aufgaben steht der *Fabrikbetrieb* gegenüber. Im Fabrikbetrieb stehen Lenkungs- und Steuerungsaufgaben eines bestehenden Fabriksystems im Mittelpunkt. Darüber hinaus zählen zu diesem auch Instandhaltung und Ablauforganisation. Gerade im Hinblick auf die bereits im ersten Kapitel dieser Arbeit vorgestellten modernen Herausforderungen an Fabriksysteme und der damit verbundenen Notwendigkeit zur permanenten Systemanpassung, ist eine rein sequentielle Betrachtung der Phasen Fabrikplanung und Fabrikbetrieb nicht mehr zeitgemäß [127]. Vielmehr müssen beide Phasen als Einheit in ständiger Wechselwirkung betrachtet werden [57]. Dies spiegelt sich auch im Fabriklebenszyklus wider, welcher in Abbildung 2.2 dargestellt ist.

Der Lebenszyklus einer Fabrik gliedert sich in fünf verschiedene Phasen: Entwicklung, Aufbau, Anlauf, Betrieb und Abbau. Inhalt und zeitliche Ausdehnung der einzelnen Phasen unterscheiden sich dabei in der Praxis teils erheblich. Dies resultiert aus den unterschiedlich langen Lebenszyklen von Produkten und Prozessen auf der einen und den längeren Lebenszyklen der jeweiligen Fabrikanlagen (Gebäude, Anlagensysteme) auf der anderen Seite[57]. Der gesamte Fabriklebenszyklus stellt die Kombination von Produkt-, Prozess-, Flächen- und Gebäudelebenszyklus dar. Hierbei ist der Produktlebenszyklus in der Regel der kürzeste, der Gebäudelebenszyklus wiederum der längste. Die grundsätzliche Herausforderung bei der Kombination von Fabrikplanung und Fabrikbetrieb besteht in der Beherrschbarkeit und Synchronisation dieser unterschiedlichen Lebenszyklen. Eine Lösung bietet das Konzept der permanenten Fabrikanpassung [127].

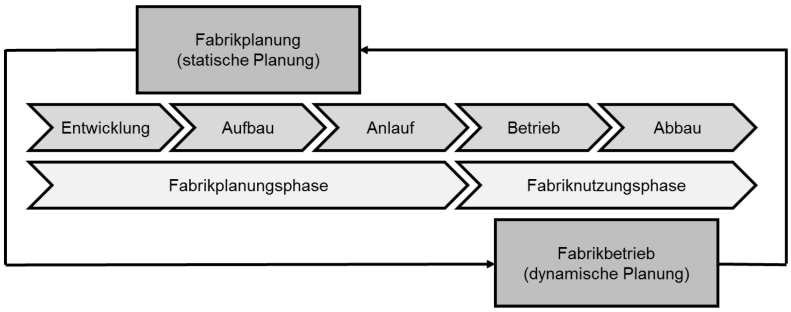


Abbildung 2.2: Fabriklebenszyklus nach Grundig [57]

Die Fabrikplanungsphase besteht aus den ersten Schritten des Fabriklebenszyklus (Entwicklung, Aufbau, Anlauf). In der Fabrikplanungsphase erfolgt unter Einsatz entsprechender Methoden und Werkzeuge (welche im weiteren Verlauf dieser Arbeit noch vorgestellt werden), die Vorausplanung der erforderlichen Fabrikanlagen sowie die Zeitplanung für deren Fertigstellung. Diese Phase wird als (quasi-)statische Planungsphase bezeichnet, da ein gegebenes Produktionsprogramm die Größe und Anordnung der Fabrik bestimmt [57]. Im Anschluss an die Planungsphase folgt die Fabriknutzungsphase, welche aus den Schritten Betrieb und Abbau besteht. In dieser Phase erfolgt die ständige Anpassung eines veränderlichen Produktionsprogramms an eine vorhandene Fabrikanlage. Diese Phase wird als dynamische Planungsphase bezeichnet.

Die Aufgabenbereiche der Fabrikplanung sind somit der Fabrikplanungsphase zuzuordnen, während sich die Fabriknutzungsphase durch die Aufgabenbereiche des Fabrikbetriebs auszeichnet. In Anbetracht der immer kürzeren Produktlebenszyklen stellen Fabrikplanungsaufgaben gewissermaßen Daueraufgaben in Unternehmen dar, da Produktionsprogrammveränderungen oftmals auch eine Anpassung der Fabrikanlagen an aktuelle Produktionserfordernisse bedingen [57, 160]. Dies verursacht allerdings einige Herausforderungen an die Fabrikplanung, welche im folgenden Kapitel beschrieben werden.

2.1.2 Merkmale von Fabrikplanungsaufgaben

In diesem Kapitel wird dargelegt, wodurch sich Fabrikplanung und Fabrikplanungsaufgaben überhaupt auszeichnen. Aus den vorangegangen Definitionen der Fabrikplanung geht bereits hervor, dass es sich um einen „kreativen Prozess zur Gestaltung einer Fabrik von der Idee über alle Planungsphasen und -strukturen unter Berücksichtigung verschiedener Anlässe und Planungsfälle“ [127] handelt. Dieser Prozess zeichnet sich besonders durch seinen interdisziplinären Charakter und die große Anzahl an involvierten Fachdisziplinen und Planungswerken aus. Abbildung 2.3 zeigt die beteiligten Planungswerke und ihre grundlegenden Aufgaben. Obwohl die Informatik als zentraler Innovationstreiber in vielen technologischen Bereichen und Disziplinen eine entscheidende Rolle spielt [69], findet sie

sich in der Literatur zur Fabrikplanung kaum wieder. Dennoch hat sie gerade im Kontext der Industrie 4.0 zentralen Einfluss auf die Gewerke der Fabrikplanung.

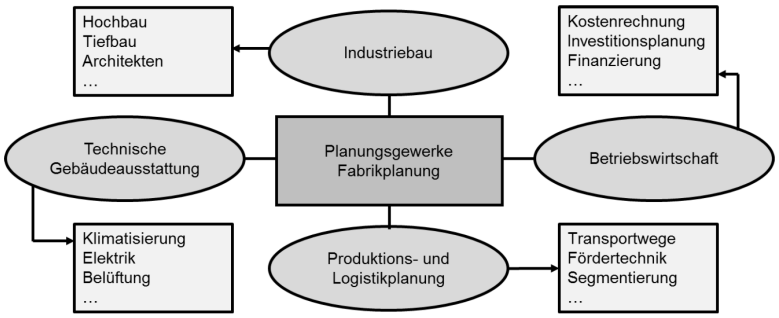


Abbildung 2.3: Beteiligte Planungsgewerke in der Fabrikplanung nach [57]

Allein durch die große Anzahl an beteiligten Fachdisziplinen ist bereits eine hohe Komplexität von Planungsprozessen in der Fabrikplanung gegeben. Dies hat zur Folge, dass viele verschiedene Schnittstellen in der Planung koordiniert werden müssen [57, 160]. Zudem zeichnen sich Fabrikplanungsaufgaben durch Merkmale wie Einmaligkeit, Neuartigkeit sowie durch konkrete Kosten- und Terminvorgaben mit zeitlichen Begrenzungen und Meilensteinen aus. Dadurch ergeben sich zwangsläufig große Ähnlichkeiten zu Merkmalen von Projekten, weshalb die Bearbeitung von Fabrikplanungsaufgaben durch ein umfangreiches Projektmanagement geleitet und mit Instrumenten der Projektplanung, -steuerung, und -kontrolle abgewickelt wird [57, 127]. Oftmals wird auch von *Fabrikplanungsprojekten* gesprochen. Die Rolle des Projektmanagements in der Fabrikplanung wird vertiefend in Kapitel 2.2 behandelt, während die Methoden und Werkzeuge zur Organisation einer Fabrikplanungsaufgabe in Kapitel 2.3 und Kapitel 2.4 aufgearbeitet werden.

Vor allem dem zeitlichen Aspekt kommt in den Planungsaufgaben eine zentrale Rolle zu; in der Regel herrscht ein hoher Termindruck. Eine strikte Terminierung sämtlicher Aufgaben ist insbesondere aufgrund der hohen Komplexität unumgänglich. Vor allem das Setzen von Meilensteinen ist von hoher Bedeutung, da beteiligte Planungsgewerke zum Teil erst dann ihre Arbeit aufnehmen können, wenn andere ihre erfolgreich abgeschlossen haben. Zum Beispiel kann mit der Planung der Elektrik erst begonnen werden, wenn die Gebäudeplanung bereits entsprechend weit fortgeschritten ist.

Kürzere Produkt- und Prozessinnovationszyklen bedeuten, dass die Projektlaufzeit von Fabrikplanungsvorhaben minimiert werden muss, um die neuen Produkte so schnell wie möglich am Markt platzieren und produzieren zu können. Allerdings führen bestehende Marktdynamiken und immer kürzere Produkt- und Prozessinnovationen zu ad-hoc Änderungen von Planungsvorhaben, Projektzielsetzungen oder notwendigen Zeitpunkten zum

Anlauf der Produktion und somit zu einem höheren Grad an Komplexität während der Planungsphase [57].

Vor allem bei Umbau- und Anpassungsmaßnamen ist der Spielraum bei der terminlichen Planung sehr gering. Da die laufende Produktion möglichst nicht beeinträchtigt werden soll, stehen für die konkrete Umsetzung von solchen Maßnahmen meist nur wenige Tage im Jahr (z.B. Feiertage oder Betriebsferien) zur Verfügung, was eine exakte Planung und Koordinierung der Beteiligten noch wichtiger macht. Es lässt sich also festhalten, dass die Komplexität, die erforderliche Genauigkeit und die Anzahl der abzuwickelnden Planungsprojekte immer weiter ansteigt, während gleichzeitig die Zeitfenster zur Planung selbst, aber auch zur Umsetzung der Pläne, immer kleiner werden. Dieser Umstand wird, wie bereits in Kapitel 1 angedeutet, als das *Dilemma der Fabrikplanung* (siehe Abbildung 2.4) bezeichnet.

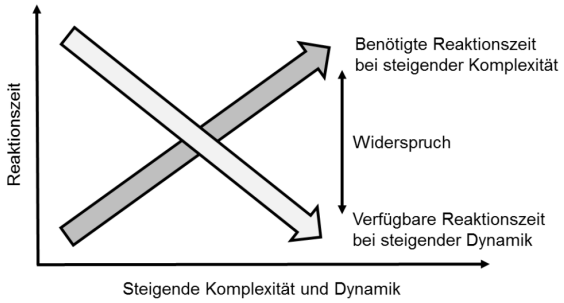


Abbildung 2.4: Das Dilemma der Fabrikplanung i. A. a. [25, 57]

Es wird gefordert, Fabrikplanungsprojekte immer schneller abzuschließen. Die zunehmende Komplexität und Dynamik der Planungsvorgänge erfordert jedoch mehr zeitliche Ressourcen während der Planungsphase. Erschwerend kommt hinzu, dass Fabrikplanungsaufgaben oft Widersprüchen hinsichtlich des betrachteten Planungszeitpunkts, der benötigten Genauigkeit erster Planungsergebnisse und der zu diesem Zeitpunkt verfügbaren Planungsdaten unterliegen. Das bedeutet, dass die Planungsergebnisse erst mit fortschreitender Bearbeitung der Planungsschritte präziser werden, diese Planungsergebnisse aber schon in früheren Planungsphasen benötigt werden, um bspw. Angaben über Investitionen oder terminliche Verschiebungen machen zu können.

Dies alles hat zur Folge, dass sich für Fabrikplanungsprojekte keine einfachen oder objektiv optimalen Lösungen generieren lassen. Die vorherrschende Dynamik der Randbedingungen (Marktdynamiken, kürzere Produkt- und Prozessinnovationszyklen), die Vielfältigkeit der beteiligten Fachdisziplinen sowie die Heterogenität der betrachteten Planungsfälle erfordern ein schrittweises Vorgehen von der Konzept- zur Detailplanung projektindividu-

elle Restriktionen und einen transparenten Dialog zwischen den beteiligten Planungsdisziplinen und Nutzern der zu planenden Fabrikanlagen [160]. Verschiedene Vorgehensmodelle zur Fabrikplanung werden in Kapitel 2.3 vertieft.

2.2 Planung und Projektmanagement als Instrument der Fabrikplanung

Aus der großen inhaltlichen Schnittmenge beim Projektmanagement und der Fabrikplanung ergeben sich zwangsläufig Anknüpfungspunkte zwischen den beiden Disziplinen. So sind Fabrikplanungsprojekte durch ein umfangreiches Projektmanagement geleitet und mit den Instrumenten der Projektplanung, -steuerung, und -kontrolle abzuwickeln. In den folgenden Abschnitten wird deshalb beschrieben, wie das Projektmanagement im Kontext der Fabrikplanung Anwendung findet.

2.2.1 Grundbegriffe

Wie bei der Fabrikplanung auch, existieren für Begriffe wie *Projekt* und *Management* zahlreiche, zum Teil stark voneinander abweichende Definitionen. In dieser Arbeit soll vor allem die in Deutschland hauptsächlich akzeptierte und verbreitete Definition der DIN 69901 [42, 3] als Grundlage dienen:

Definition 2.2

> *Ein **Projekt** ist ein Vorhaben, das im Wesentlichen durch die Einmaligkeit der Bedingungen (Zielvorgabe, zeitliche, finanzielle, personelle oder andere Begrenzungen und eine projektspezifische Organisation) in ihrer Gesamtheit gekennzeichnet ist. [42]*

Genau wie die Fabrikplanung zeichnen sich auch Projekte durch Merkmale wie einmalige, komplexe und innovative Aufgabenstellungen sowie interdisziplinäre und fächerübergreifende Zusammenarbeit von Projektbeteiligten aus [144]. Damit Projekte transparent und nachvollziehbar strukturiert und die Komplexität zumindest teilweise reduziert wird, um die Bearbeitung des Projektes zielgerichtet zu halten, ist die Erstellung von schematischen Projektabläufen unumgänglich. Der definierte Projektablauf stellt in der Regel eine schrittweise Vorgehensweise dar, in der Projektaufgaben abgearbeitet werden [13, 38, 109]. Diese Projektabläufe werden in der Regel von den verantwortlichen Planern im Projektmanagement vorgegeben. Sie vereinen nach DIN69901 [42] die Gesamtheit von Führungsaufgaben, -techniken, -organisation und -mitteln für die Initiierung, Definition, Planung, Steuerung und den Abschluss von Projekten. Das Projektmanagement umfasst somit alle relevanten Planungsaufgaben.

Generell lässt sich der Projektverlauf in mehrere Phasen unterteilen. Darüber, wie diese genau zu bezeichnen sind und welche der Planungsaufgaben dazugehören, gibt es unterschiedliche Auffassungen [30, 17, 38, 85, 13, 42]. Die unten stehende Abbildung 2.5 gibt einen Überblick über die gängigsten Definitionen von Projektphasen. Wie sich diese inhaltlich gestalten und welche Meilensteine wann zu erreichen sind, wird im eigentlichen Projektplan der jeweiligen Phase festgelegt und terminiert. Hierzu existieren verschiedene Werkzeuge, auf die in Kapitel 2.4.1 eingegangen wird.

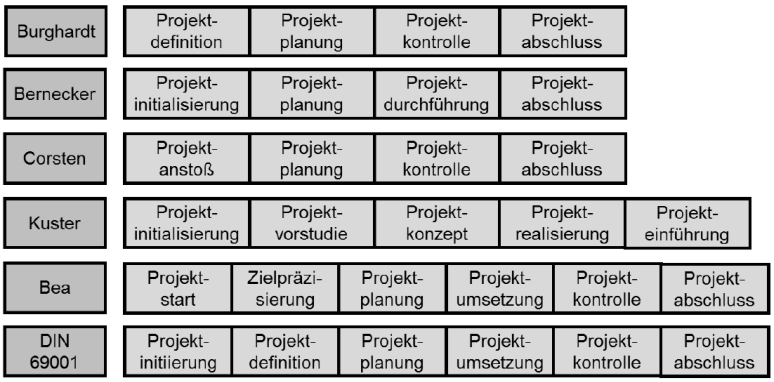


Abbildung 2.5: Übersicht über verschiedene Definitionen von Projektphasen. Darstellung nach [30, 17, 38, 85, 13, 42, 109]

Ist der Ablauf eines Projektes (auch innerhalb der einzelnen Phasen) definiert, gilt es als nächstes, sich mit der Organisation der einzelnen Aufgaben zu befassen. Die *Projektorganisation* lässt sich in zwei Dimensionen gliedern: die Ablauf - und die Aufbauorganisation [85]. In der Ablauforganisation werden zeitliche, räumliche, mengenmäßige und logische Beziehungen zur Erreichung eines Zieles definiert. Dies geschieht durch Beschreibung der notwendigen Tätigkeiten, der Abfolge der Aktivitäten sowie der Zuordnung von Aufgabenträgern entsprechend ihrer Kompetenzen und Verfügbarkeit zu einzelnen Tasks. Typische Hilfsmittel dabei sind Ablauf-, Netz- und Terminpläne (siehe Kapitel 2.4.1). Im Rahmen der Aufbauorganisation werden hingegen die Beziehungsstrukturen der Aufgabenträger während des Projektablaufs geregelt. Die Beziehungsstrukturen werden durch Weisungs- und Kommunikationsbeziehungen miteinander verknüpft und in ein funktionales Beziehungsgefüge mit definierten Rollen eingeordnet. Dies geschieht typischerweise durch die Verwendung von Organigrammen oder Stellenbeschreibungen [85].

Es lässt sich festhalten, dass die Ablauforganisation und ihre Werkzeuge und Hilfsmittel i.d.R. fest definierten logischen Strukturen folgen, die zum Teil auch von einem Projekt in ein anderes übertragbar sind. Die Aufbauorganisation hingegen gestaltet sich von Projekt zu Projekt und von Unternehmen zu Unternehmen sehr individuell und auf den jeweiligen

Anwendungsfall hin zugeschnitten. Wiederkehrende Strukturen oder wiederverwertbare Muster lassen sich hier eher selten erkennen. Aus diesem Grund konzentriert sich die vorliegende Arbeit auf die Ablauforganisation innerhalb der Projekt/Fabrikplanung, da ihr logischer Aufbau sich besser für Ansätze der automatisierten Erzeugung anbietet [160, 109].

Eine spezielle Sichtweise auf die Strukturen von Projektmanagement sind zudem noch die vier Projektdimensionen nach Kuster [85]. Abbildung 2.6 zeigt diese Dimensionen und die mit ihnen verbundenen Aufgabenfelder. Die funktionale Dimension regelt, welche Arbeitsschritte in den einzelnen Projektphasen zu durchlaufen sind. Die Aufgabenbereiche der funktionalen Dimension tragen daher zur Gestaltung der Ablauforganisation bei. Bei der personell-psychologisch-sozialen Dimension sind Aufgaben, die zur Verbesserung und Förderung der Zusammenarbeit und des Arbeitsklimas im Projekt beitragen, zusammengefasst. Die institutionelle Dimension regelt vor allem die interdisziplinären Aspekte der Projektorganisation und deren Integration ins Gesamtunternehmen. Hier steht folglich die Aufbauorganisation im Vordergrund. In der instrumentellen Dimension geht es um die Bereitstellung aller benötigten Arbeitsmittel für das Projekt.

Abbildung 2.6: Die vier Dimensionen des Projektmanagements i. A. a. [85]

2.2.2 Zieldimensionen in Fabrikplanungsprojekten

Entscheidend für eine gute Projektplanung ist die hinreichend gute Definition von Projektzielen. Nur wenn klar ist, wohin man möchte, ist es auch möglich, den besten Weg dorthin zu bestimmen. Eine unsaubere oder unvollständige Definition von Projektzielen kann dagegen schwerwiegende Folgen haben. Sind zum Beispiel einzelne Projektabschnitte oder Positionen fixiert, die nicht zu den Zielvorstellungen von Kunden oder Unternehmen passen, kann dies erhebliche Verzögerungen oder Kostensteigerungen verursachen [131].

Es kann und muss jedoch nicht immer zwangsläufig ein einziges Projektziel existieren. Vielmehr setzt sich das Projektziel aus mehreren Einzelzielen zusammen, die im Projektverlauf erreicht werden sollen. Oft korrespondieren diese Einzelziele mit zeitlichen Meilensteinen im Projektverlauf [42]. Bei der Definition dieser Ziele gibt es einige Aspekte zu beachten: Ziele sollten immer klar und eindeutig formuliert sein und so kommuniziert werden, dass sie von allen Beteiligten verstanden werden. Da auf Basis dieser Ziele sämtliche Projektaktivitäten ausgerichtet und Auswahlentscheidungen möglicher Alternativen getroffen werden, ist ein gemeinsames Verständnis über die Zielvorstellungen unvermeidbar. Mit einer gemeinsamen Zielvereinbarung können Projektverständnis und Akzeptanz bei allen Projektbeteiligten gefördert werden. Zusätzlich steigt die Motivation, diese Ziele auch umzusetzen. Es wird zudem ein Soll-Zustand definiert, der auch als Erfolgsmaßstab dienen kann [131]. Als anerkannte Methoden zur Zielfindung und -formulierung dienen häufig die *KREOL*- und die *SMART*-Methode, welche in Abbildung 2.7 dargestellt sind. Der Prozess der Zielformulierung sollte dabei im Projektteam erfolgen. Die definierten Projektziele müssen vom Auftraggeber als konkrete Soll-Vorgaben akzeptiert und deren Erfüllung durch ein Projekt-Controlling überwacht werden [131].

KREOL-Methode	
• K	Konkret
• R	Rahmen beschreiben
• E	Erreichbare/realistische Ziele
• O	Operationalisierbarkeit (Messbarkeit) der Ziele
• L	Lösungsneutral (Ziele geben keinen Lösungsweg vor)
SMART-Methode	
• S	Spezifisch
• M	Messbar
• A	Attraktiv/Motivierend (Positiv formuliert)
• R	Realistisch
• T	Terminiert

Abbildung 2.7: Zielformulierung nach KREOL und SMART i. A. a. [131]

Projektziele lassen sich zur besseren Strukturierung in verschiedene Kategorien einteilen. Dazu gehören zum Beispiel Sachziele (Projektergebnisse) oder Kosten- und Terminziele. Die Einteilung hilft bei der Spezifikation des globalen Projektziels. Andere verbreitete Varianten der Zieleinteilung nutzen die Differenzierung zwischen Ergebniszielen (Finanzziele, Funktionsziele, personelle Ziele, Sozialziele) und Vorgehenszielen (Terminziele, Budgetziele, Personalziele, politische Ziele). Innerhalb dieser Unterteilungen kann darüber hinaus noch zwischen individuellen Zielen der Projektbeteiligten und absoluten Muss-Zielen (Leistungsmerkmale, Meilensteine, Kostenrestriktionen) des Projekts unterschieden werden. Auf diese Weise lassen sich dann auch direkt Kontrollmechanismen in das Projektmanagement integrieren: Weichen individuelle Ziele zu sehr von den Muss-Zielen des Projektes ab, muss die Projektleitung eingreifen. [131, 166].

Sind alle Ziele eines Projektes definiert und strukturiert, ist es als nächstes erforderlich, Abhängigkeiten und Beziehungen zwischen den einzelnen Zielen zu finden und zu definieren. Daraus ergibt sich i.d.R. direkt eine Vorlage für den Ablaufplan eines Projektes. Generell gibt es fünf verschiedene Arten von Zielbeziehungen:

1. **Konflikthafte/konkurrierende Ziele:** Die Erfüllung eines Ziels beeinträchtigt die Erfüllung eines anderen Ziels.
2. **Komplementäre Ziele:** Die Erfüllung eines Ziels zieht automatisch die Erfüllung eines anderen nach sich.
3. **Indifferente Ziele:** Zwei Ziele sind voneinander unabhängig.
4. **Identische Ziele:** Zwei Ziele sind inhaltlich identisch.
5. **Sich gegenseitig ausschließende Ziele:** Die Erfüllung eines Ziels verhindert die Erfüllung eines anderen.

Liegt der fünfte Fall vor, muss vom Projektmanagement eine Priorisierung der Ziele vorgenommen und entschieden werden, welches der konkurrierenden Ziele zu erfüllen ist. Zielbeziehungen können sich im Verlauf eines Projektes aber stets auch ändern. Deswegen sollten sie im Projektverlauf regelmäßig neu evaluiert werden.

In der Fabrikplanung im speziellen, aber auch in der Projektplanung allgemein, lassen sich in der Regel drei Zielgrößen identifizieren, die in jedem Planungsfall eine wesentliche Rolle spielen: Kosten, Zeit und Qualität. Diese drei Faktoren stehen sehr häufig in konkurrierenden Beziehungen, welche allgemein auch als das *Spannungsdreieck* des Projektmanagements beschrieben werden [52]. Die Beziehung der drei Zielgrößen ist so eng miteinander verwoben, dass die Änderung eines Faktors automatisch auch Änderungen

an den beiden anderen Größen nach sich zieht. Zum Beispiel haben erhöhte Qualitätsanforderungen in der Regel steigende Kosten und eine längere Projektlaufzeit zur Folge. Vor dem Start des Projekts sollten daher die Interessen der Projektbeteiligten festgestellt werden, um eine Rangfolge der Zielgrößen festzulegen, da die Optimierung aller Größen gemeinsam nicht möglich ist.

Zusätzlich zu den drei genannten Zielgrößen nehmen zahlreiche Literaturquellen auch die eingesetzten Ressourcen und den Projektnutzen in die Reihe der Standardzielgrößen mit auf. Da auch diese in Abhängigkeit zu den Größen in dem beschriebenen Spannungsdreieck stehen, wird aus der klassischen Dreieckstruktur ein komplexeres System verschiedener sich beeinflussender Größen (vgl. Abbildung 2.8).

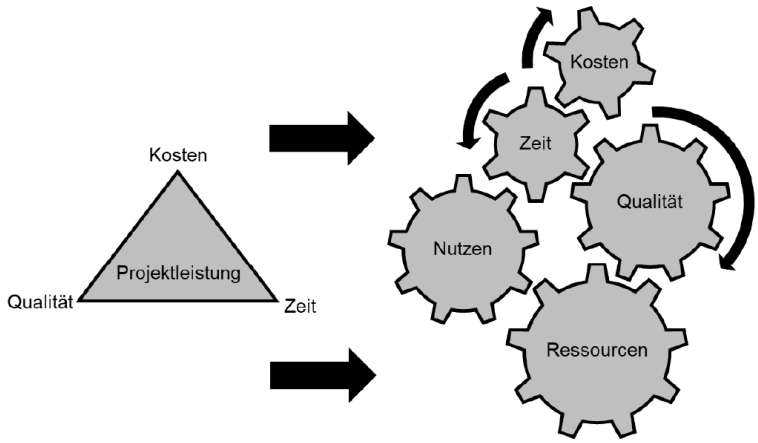


Abbildung 2.8: Wandel der Zielgrößen im Projektmanagement i. A. a. [52, 85]

2.2.3 Erfolgs- und Misserfolgsfaktoren

Auch wenn alle Grundlagen des Projektmanagements eingehalten, sämtliche Ziele klar definiert und der Projektverlauf klar strukturiert und dokumentiert wurde, können Planungsprojekte scheitern. Auslöser sind verschiedene Faktoren, die für den Erfolg bzw. den Misserfolg eines Projektes ausschlaggebend sind. Viele der Misserfolgsfaktoren lassen sich durch fachliche, organisatorische und soziale Kompetenzen des Projektteams verhindern [160, 133]. Typische Gründe für das Scheitern von Projekten sind:

- Ein überstürzter Planungsbeginn ohne vorherige Festlegung der Projektziele in konkreter und einheitlicher Form sowie die fehlende Klärung gegenseitiger Vorstellungen bezüglich des Projektverlaufs.
- Es wird ohne Methodenkonzept gearbeitet, da die Bedeutung eines systematischen Vorgehens unterschätzt wird („Ingenieure wollen technische Probleme lösen und sich nicht mit Formalien aufhalten“).

- Das Projekt wird nicht ganzheitlich geplant, sondern stattdessen werden Einzellösungen optimiert. Dadurch werden u.U. Zielkonflikte gar nicht oder viel zu spät erkannt.
- Kosten-, Termin- oder Leistungsziele werden aufgrund fehlender Kompetenz im Hinblick auf die Fabrikplanung von Beginn an unrealistisch gesetzt.
- Innerhalb des Projektteams werden Entscheidungen emotional getroffen. Machtpositionen werden zu Lasten von Sachargumenten ausgenutzt.
- Widerstände oder Vorbehalte gegen bestimmte Vorgehensweisen oder Lösungsoptionen werden nicht an alle Beteiligten kommuniziert.
- Bei Konflikten und deren Lösungserarbeitung wird nicht zwischen Sachkonflikten (unterschiedliche Fachsichten und stellenbedingte Interessen), Beziehungskonflikten (unterschiedliche persönliche Werte und Kulturen) und Scheinkonflikten (unklare Kommunikation und Begrifflichkeiten) unterschieden.

Insbesondere die Nicht-Beachtung von Interessenkonflikten und die fehlende Ganzheitlichkeit von Lösungen führen zum Entstehen von Spannungsfeldern zwischen Auftraggebern und -nehmern (vgl. Abbildung 2.9). Auftraggeber sind üblicherweise daran interessiert, mit einem minimalen Kosteneinsatz den größtmöglichen Ertrag aus dem Projekt zu ziehen. Dafür verfolgen sie meist die Einzelziele, die in der Abbildung 2.9 links aufgeführt sind. Ebenso ist es den Auftraggebern in der Regel wichtig, dass das Risiko für eine Budgetüberschreitung möglichst minimal gehalten wird. Dies ist auch meist dann noch der Fall, wenn die Gründe für solche Überschreitungen beim Auftraggeber selbst zu suchen sind (z.B. als Resultat von Planänderungen). Als spätere Nutzer der zu planenden Fabrikanlage sind Auftraggeber als solche natürlich insgesamt an einem guten Projektergebnis interessiert. Zu diesem Zweck sollen meist auch langjährige Missstände beseitigt werden, da Fabrikplanungsvorhaben dazu eine passende Gelegenheit bieten. Der Nutzer bzw. Auftraggeber bringt sukzessive stets neue Wünsche zur Umsetzung in die Planung ein, ohne diese mit der Geschäfts- oder übergeordneten Projektleitung ausreichend abzustimmen. Dies führt zu Kosten- und Terminüberschreitungen, die oft den verantwortlichen Planern bzw. Auftragsnehmern angelastet werden [160].

Die Planer respektive Auftragnehmer haben wiederum meist das Ziel, den Aufwand zu minimieren und den eigenen Gewinn zu optimieren. Um sich gegen etwaige Änderungswünsche der Auftragsgeberseite abzusichern, bestreben sie vertragliche Gewährleistungspflichten möglichst gering zu halten. Zusätzlich unterliegen Auftragsnehmer aufgrund von Bieterwettstreiten oder Nachlässen auf Angebotspreise nach Auftragserteilung einem besonderen Preisdruck. Das bedingt den Versuch, die entgangenen Einnahmen aus dem Ur-

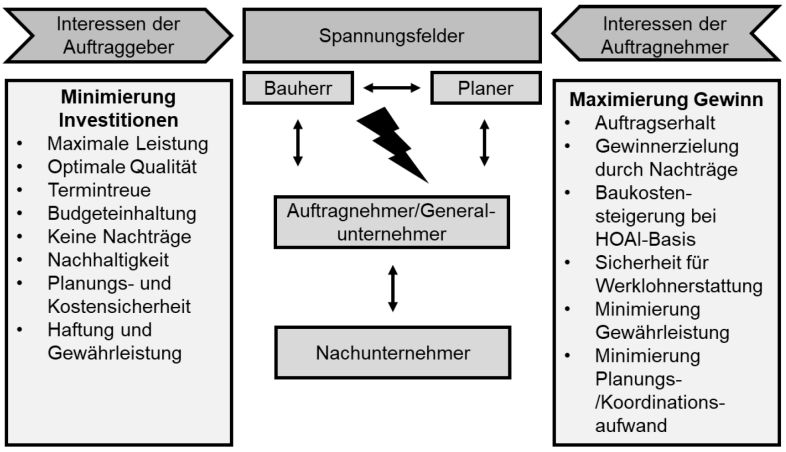


Abbildung 2.9: Spannungsfelder in Fabrikplanungsprojekten i. A. a. [160, 133]

sprungsangebot im Bieterwettstreit durch erhöhte Honorare, die bei nachträglichen Projektänderungen fällig werden, auszugleichen. Die Honorargestaltung ist zumindest in Deutschland allerdings rechtlich geregelt. In der Honorarordnung für Architekten und Ingenieure (HOAI) sind Stundensätze und mögliche Steigerungsraten eingegrenzt und festgehalten [159, 160, 44].

Da die Interessen von Auftragsgeber und -nehmerseite unterschiedlich sind, ist die konkrete Definition von Zielen und Bedingungen im Dialog zwischen beiden Seiten für den Projekterfolg unerlässlich. Genauso müssen sowohl für das Gesamt- als auch für einzelne Teilprojekte, die Ziele und der Projektverlauf für alle Beteiligten nachvollziehbar dokumentiert und aufbereitet werden [160].

Bei der Abwicklung von (Fabrikplanungs-)Projekten sind vor diesem Hintergrund einige Faktoren zu berücksichtigen, um einen erfolgreichen Projektabschluss gewährleisten zu können. Zunächst ist anzumerken, dass Projekte keine Routineaufgaben darstellen und i.d.R. nicht nach einem allgemeingültigen Prozess abgearbeitet werden können. Jedes Projekt ist einzigartig und der Abwicklungsprozess muss im Projektteam für jedes Projekt einzeln erarbeitet werden. Zudem müssen folgende Grundsätze beachtet werden:

1. Dem Projektleiter müssen ausreichende Kompetenzen und Verantwortlichkeiten übertragen werden. Dies vereinfacht den Prozess der Entscheidungsfindung. Allerdings sollte der Projektleiter immer auch Punkt 3 und 4 beachten.

2. Im Projektteam muss ausreichend Know-how hinsichtlich der Zieldefinition sowie des Projektcontrollings und -steuerung vorhanden sein. Dies ist notwendig, um bei Abweichungen und unvorhergesehenen Ereignissen während des Projektverlaufs gegensteuern zu können.

3. Projektarbeit ist immer Teamarbeit.

4. Mangelnde Konfliktlösungskompetenzen erschweren die Zusammenarbeit im Projektteam. Durch unzureichende Zusammenarbeit werden oft falsche Entscheidungen getroffen, sollten die unterschiedlichen Interessen der Projektbeteiligten falsch oder gar nicht kommuniziert werden.

Werden diese Grundsätze befolgt, existiert bereits eine Grundlage für die erfolgreiche Umsetzung des Projektes. Sind sich alle Beteiligten einig darüber, was im Projekt erreicht werden soll und wie die Kompetenzen und Zuständigkeiten verteilt sind, sind die Grundlagen geschaffen, die notwendigen Schritte zu den jeweiligen Projektzielen hin zu unternehmen. Im folgenden Kapitel wird der Weg zum Ziel hin näher betrachtet, indem verschiedene Modelle und Vorgehen der Fabrikplanung vorgestellt und klassifiziert werden.

2.3 Bestehende Planungsmodelle und -verfahren

Im vorherigen Kapitel wurden insbesondere die eindeutige Formulierung von Projektzielen und ihre Bedeutung für den Erfolg von (Fabrik-) Planungsprojekten erörtert. Ebenso wurde erwähnt, das eine sinnvolle, verständliche und klare Strukturierung des Ablaufs von Projekten (und damit die Art der Zielerreichung) von zentraler Bedeutung ist. Dieses Kapitel gibt einen Überblick über verschiedene Ansätze zur Strukturierung von Planungsworkflows und stellt gängige Konzepte aus diesem Bereich vor. Dabei wird sowohl auf klassische, eher sequentiell ausgelegte, Vorgehen wie auch auf moderne Methoden eingegangen. Ziel ist es, passende Methodiken zu finden, die eine weitestgehend automatisierte Erzeugung von projektspezifischen Planungsworkflows ermöglichen. Fabrikplanungsvorhaben zeichnen sich durch eine Vielzahl komplexer Entscheidungsprobleme aus, deren Lösung ein systematisches und iteratives Vorgehen erfordert [109, 79].

Vor allem seit den 1980er Jahren wurden eine Vielzahl von Methoden und Modellen entwickelt und publiziert, die ein systematisches Vorgehen abbilden. Diese Modelle lassen sich hinsichtlich ihrer Strukturierung und Fokussierung klassifizieren [109]. Die Strukturierung eines Fabrikplanungsmodells kann dabei inhaltlicher oder zeitlicher Natur sein. Inhaltlich strukturierte Modelle definieren einzelne Aufgabenbereiche und stellen Lösungsverfahren für diese Aufgabenbereiche in den Mittelpunkt. Zeitlich strukturierte Modelle konzentrieren sich eher auf den zeitlichen Ablauf der Planungsschritte und den Abhängigkeiten, die zwischen diesen Schritten bestehen. Zeitlich strukturierte Fabrikplanungsmodelle sind i.d.R. logisch besser zu beschreiben [109]. Das Kriterium der *Fokussierung* beschreibt, wie allgemeingültig ein Fabrikplanungsmodell jeweils ist. Generelle Planungsmodelle verfolgen einen allgemeinen Anspruch auf Einsetzbarkeit in verschiedenen Branchen, Unternehmensbereichen oder hinsichtlich der Lösung bestimmter Teilaspekte der Fabrikplanung. Stark

fokussierte Planungsmodelle hingegen beschränken sich in der Regel auf einzelne Branchen oder Planungsaspekte. Eine Klassifizierung der gängigsten Fabrikplanungsmodelle nach *Nöcker* [109] ist in Abbildung 2.10 dargestellt.

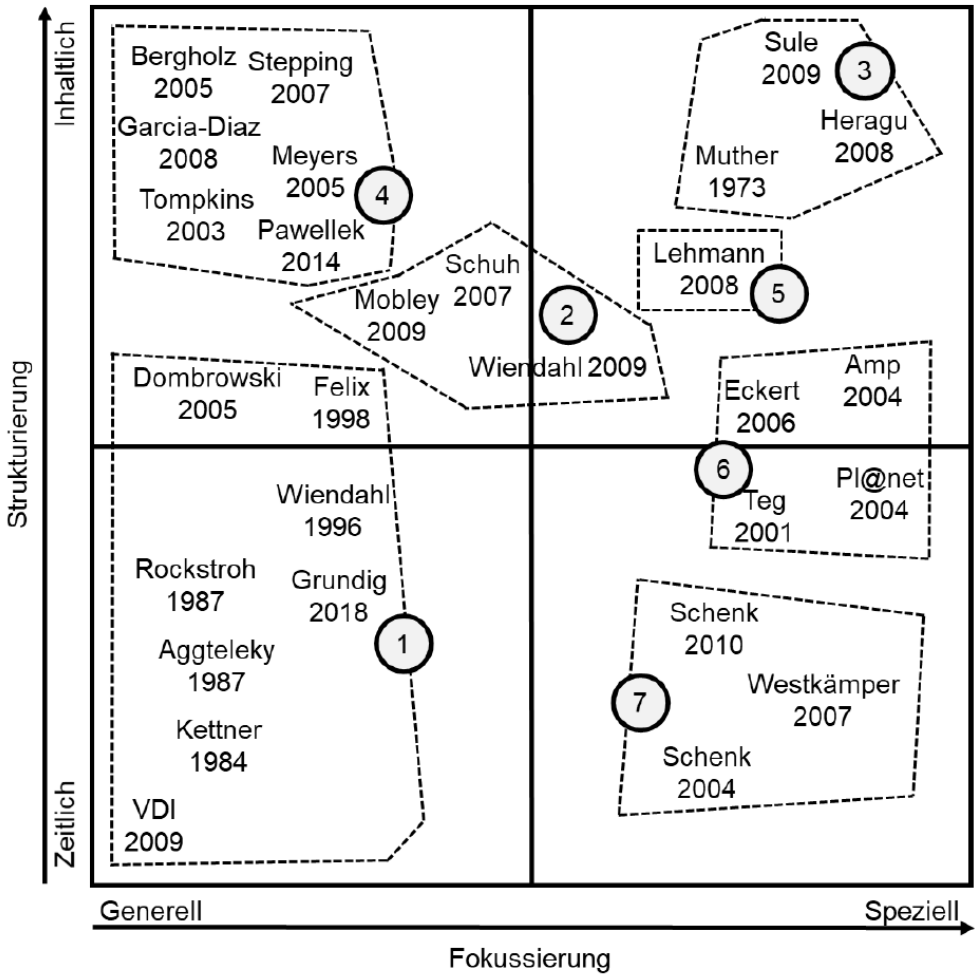


Abbildung 2.10: Klassifizierung von Planungsmodellen der Fabrikplanung, i. A. a. [109, 16]

Fabrikplanungsmodelle lassen sich demnach in sieben Klassen unterteilen. Jede dieser Klassen repräsentiert eine andere Strömung innerhalb der Fabrikplanung. (die Nummerierung in Abbildung 2.10 entspricht den Nummern der folgenden Auflistung) [109]):

1. Klassische konsekutive/sequentielle Phasenmodelle
2. Planungsmodelle mit Fokus auf das Projektmanagement
3. Planungsmodelle mit Fokus auf Teilaufgaben
4. Konfigurierbare Ansätze
5. Planungsmodelle mit Fokus auf die digitale Fabrik

6. Planungsmodelle mit Fokus auf mittelständische Unternehmen

7. Planungsmodelle mit Fokus auf Wandlungsfähigkeit

Aufgrund der Zielstellung dieser Arbeit, einen möglichst allgemeingültigen und automatisierbaren Ansatz der Planung zu realisieren, werden stark fokussierte Modelle nicht weiter betrachtet. Im Fokus des weiteren Verlaufs stehen vor allem die klassischen konsekutiven und sequentiellen Ansätze (Klasse 1), Planungsmodelle, welche vor allem das Projektmanagement betrachten (Klasse 2), und frei konfigurierbare Ansätze (Klasse 4). Diese werden in den folgenden Abschnitten vorgestellt.

2.3.1 Sequentielle Planungsmodelle

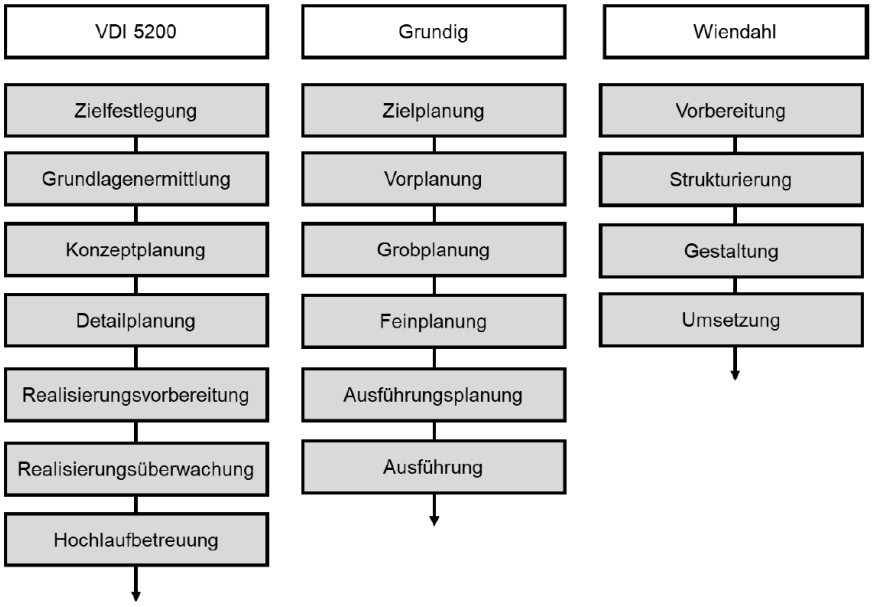


Abbildung 2.11: Beispiele für sequentielle Planungsmodelle [109]

Mit dem Aufkommen der wissenschaftlichen Auseinandersetzung mit dem Thema Fabrikplanung und der damit verbundenen Einführung systematischer Planungsansätze und -modelle wurden ab den ca. 1980er Jahren zunächst sogenannte sequentielle und phasenorientierte Planungsmodelle vorgestellt. Diese zeichnen sich dadurch aus, dass Planungsprojekte in einzelne Elemente zerlegt und dann vom Groben zum Feinen hin ausgeplant werden [109, 132]. Dabei wird der Fabrikplanungsprozess als solcher in einzelne inhaltlich-methodisch abgrenzbare und logisch strukturierte Phasen gegliedert, welche zeitlich abgestuft abgearbeitet werden [57, 109]. Wie genau die Phasen benannt und gestaltet sind, unterscheidet sich je nach Modell. So definiert Grundig [57] insgesamt sechs Phasen, Wiendahl hingegen nutzt nur vier [158]. Die VDI-Norm 5200 hingegen kennt sogar sieben Phasen der Fabrikplanung (vgl. Abbildung 2.11). Die Bedeutung der Planungsphasen sind

am Beispiel Grundig in Abbildung 2.12 erläutert.

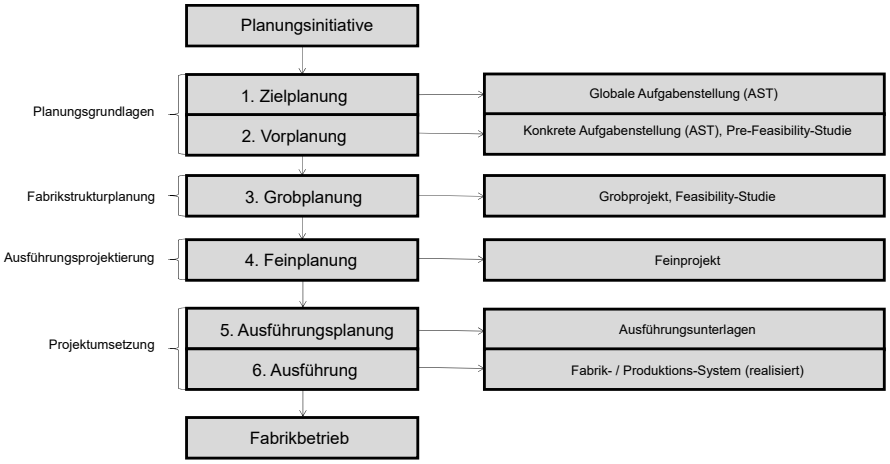


Abbildung 2.12: Planungsphasen nach Grundig (eigene Darstellung i.A.a. [57])

Gemein haben die genannten Ansätze eine Ablauflogik, nach der eine Schrittfolge der Form Vorbereitung, Grobplanung, Detailplanung, Ausführungsplanung und Ausführung zu durchlaufen sind [16, 109]. Der Durchlauf bzw. die Abarbeitung der Phasen, erfolgt in der Regel rein linear unter Einbeziehung von Meilensteinen. In einigen Fällen kann es möglich sein, dass mehrere Iterationen einer Planungsphase durchlaufen werden müssen, bevor mit der nächsten Phase begonnen werden kann. In diesem Fall spricht man von einer *rekursiven Phase*. Die generelle Sequenz an Planungsphasen wird dabei aber nicht verändert. Phasenorientierte Meilensteine können als das Ende eines jeden Iterationsschrittes im Planungsablauf angesehen werden, anhand derer entschieden wird, ob der Planungsablauf in die nächste Phase voranschreitet oder ein Schritt in der Iteration zurückgesprungen werden muss, um Plan-Anpassungen vorzunehmen. Somit zeichnet alle Ansätze ein schrittweiser Weg zum Planungsziel hin aus, der durch ein systematisch-rationales Vorgehen gekennzeichnet ist. Die Vorgehen selbst sind allerdings starr definiert. Plan-Anpassungen in bereits abgeschlossenen Planungsphasen (der Rücksprung in andere Phasen) oder ein je nach Situation gerichtetes „Springen“ zwischen Planungsphasen ist nicht vorgesehen. Treten also während des Prozesses unerwartete Ereignisse auf, kann dort nur über Plan-Anpassungen innerhalb der aktuellen Planungsphase reagiert werden, weswegen die Eignung solcher Vorgehen bei sich stetig verändernden Rahmenbedingungen oder häufig verändernden Planungsverläufen zunehmend kritisch hinterfragt wird [109, 16].

2.3.2 Planungsmodelle mit Fokus auf dem Projektmanagement

Wie bereits erwähnt, zeichnen sich Fabrikplanungsprojekte auch durch ihre Interdisziplinarität aus. Teilaspekte der Fabrikplanung greifen zum Teil sehr tief in bestehende Unternehmensstrukturen und Abläufe ein; entsprechend müssen Mitarbeiter aus verschiedensten Fachdisziplinen in die Planung miteinbezogen werden [158, 109]. Planungsmodelle mit dem Fokus auf dem Projektmanagement betrachten weniger die Rolle des Planungsobjekts. Fabrikplanungsvorhaben werden hier als komplexe Planungsprojekte verstanden. Instrumente des Projektmanagements werden daher nicht nur als Hilfsmittel bei der Projektabwicklung genutzt, sondern sind feste und elementare Bestandteile des Planungsprozesses [158, 109]. Planungsmodelle, die den Fokus auf solche Projektmanagementaspekte legen, folgen demnach folgendem Schema: Zunächst wird der Planungsprozess in Teilaufgaben aufgeteilt und diese den korrespondierenden Fachabteilungen und -disziplinen zugeteilt. Mit dieser Festlegung entsteht auch direkt ein Schema, wie die Zusammenarbeit organisiert werden soll. Es werden Projektleiter und Prozessbegleiter zur Koordination der Planungsaktivitäten bestimmt, die mit ihren Teilgruppen die Definition und Einhaltung der jeweiligen Projektziele überwachen. Die Gesamtleitung des Projektes koordiniert hauptsächlich die Zusammenführung der Teilprojekte zur Erreichung des globalen Projektziels. Die Planungsmodelle mit Fokus auf das Projektmanagement erweitern die sequentiellen Planungsmodelle daher um einige wesentliche Aspekte. Insbesondere stehen bei ihnen das Nutzen des Fachwissens der verschiedenen Bereiche und die sozialen und koordinativen Aspekte des Projektmanagements im Vordergrund.

2.3.3 Modulare Planungsmodelle

Klassische Fabrikplanungsmodelle haben das Problem, dass sie aufgrund ihrer starren Vorgehensweise nur bedingt auf Unsicherheiten und sich verändernde Begebenheiten reagieren können. Projektmanagement-basierte Ansätze geben diesbezüglich keine konkreten Prozesse vor, sondern vertrauen auf die Fachkompetenz der Planungsbeteiligten. Beiden Ansätzen ist also gemein, dass es kein konkretes Verfahren gibt, das bei plötzlich auftretenden Ereignissen, die Planänderungen notwendig machen, greifen könnte. Diesen Umstand möchten modulare Planungsmodelle adressieren [114]. Bei modularen Ansätzen werden Arbeitsabläufe aus zuvor definierten Modulen (also einzelnen, unabhängigen Bausteinen) zusammengesetzt. Ist eine Neuplanung notwendig, wird der betroffene Planabschnitt durch Neuanordnung der Module oder durch Verwendung anderer Module neu komponiert und somit ein neuer Workflow erzeugt. Modulare Ansätze setzen bei der Plangestaltung insbesondere auf eine Parallelisierung und Synchronisierung einzelner Planungsabschnitte, was sie zusätzlich von anderen Ansätzen abhebt [109, 155, 114].

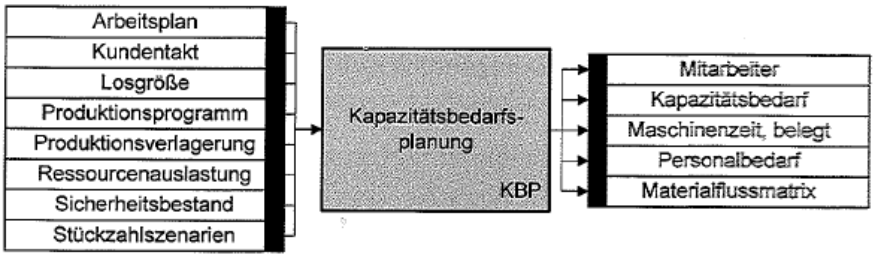


Abbildung 2.13: Modul Kapazitätsbedarfsplanung aus [109]

Eines der ersten und führenden Konzepte zur modularen Fabrikplanung ist das Verfahren nach *Nöcker* [109], welches im Rahmen einer Dissertation im Jahr 2012 an der RWTH Aachen entstanden ist. Nöcker entwickelt in seiner Arbeit ein Konzept zur zustandsbasierten Fabrikplanung. Dieses Planungsmodell soll die Interaktion der Fabrikplanungsaufgaben erhöhen, um Planungsunsicherheiten und dynamischen Veränderungen von Planungsinformationen im Projektverlauf entgegenzuwirken und durch möglichst maximale Parallelisierung von Aufgaben die Projektlaufzeit zu verkürzen [109]. Dabei werden Aufgaben in Module unterteilt, die jeweils unterschiedliche Eingabeinformationen benötigen und unterschiedliche Ausgabeinformationen produzieren. Ein Beispiel für solch ein Modul zeigt Abbildung 2.13 anhand des Moduls „Kapazitätsbedarfsplanung“. Dieses Modul benötigt, um erfolgreich absolviert werden zu können, unter anderem das Produktionsprogramm und liefert als Ausgabe wiederum den Personalbedarf. Soll ein vollständiger Projektplan erzeugt werden, ist es notwendig, alle Eingangsinformationen eines Moduls abzudecken. Dies kann entweder dadurch geschehen, dass Informationen bereits bekannt sind oder dass ein weiteres Modul, welches im vorliegenden Beispiel das Produktionsprogramm als Ausgabeinformation liefert, vorgeschaltet wird (siehe Abbildung 2.14). Jedes Modul enthält darüber hinaus Arbeitspakete, die in seinem Verlauf abgearbeitet werden müssen sowie Informationen bezüglich Umsetzungskosten und -dauer. Gesammelt werden alle Module in einer projektneutralen Sammlung, die der Autor selbst als „Modullandkarte“ bezeichnet. Dabei handelt es sich um ein strukturiertes Repository, welches 22 von Nöcker selbst definierte oder aus der Literatur abgeleiteten Module enthält. Aufgrund der Tatsache, dass Ein- und Ausgangsinformationen jedes Moduls bekannt sind, wird eine höchstmögliche Transparenz zwischen den Planungsaufgaben erzeugt und Abhängigkeiten zwischen einzelnen Modulen sind sofort ersichtlich [109].

Seit dem Erscheinen des Verfahrens wurde das Planungsmodell von Nöcker in der Literatur immer wieder aufgegriffen und erweitert. Im Planungsmodell nach *Meckelnborg* [97] zum Beispiel wurde die Modullandkarte um Module aus dem Bereich des Industriebaus erweitert. Es kamen zu den 22 Modulen von Nöcker noch 15 weitere hinzu. Andere Module wurden erweitert oder ersetzt, so dass Meckelnborg auf insgesamt 35 Module kommt.

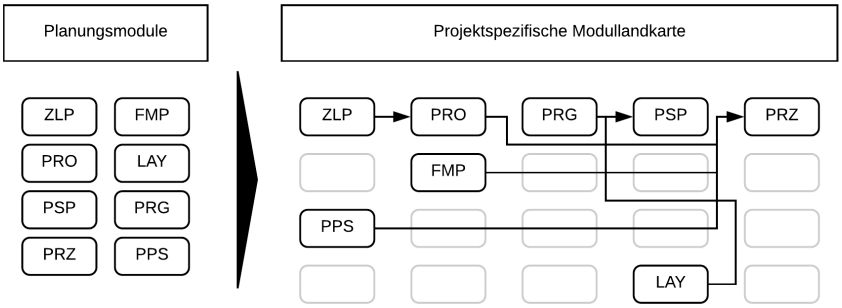


Abbildung 2.14: Erzeugung eines Ablaufplans durch eine projektspezifische Modullandkarte nach [109]

Außerdem führte [97] ein Koordinationsmodell zur Orchestrierung und Steuerung der Planungsverläufe ein. Dazu werden modulübergreifend Planungstakte festgelegt, in denen die einzelnen Projektteams für eine festgelegte Zeitspanne weitgehend isoliert von den anderen Planungsteams an ihren modulindividuellen Zielen arbeiten. Nach Beendigung eines Planungstaktes werden die Ergebnisse zwischen den Planungsmodulen synchronisiert. Vor jedem Takt werden die Teilergebnisse für den Folgetakt geplant und alle benötigten Informationen entsprechend der Modulsignatur weitergegeben. Bei akuten Störfällen wird der Planungstakt unterbrochen und ggf. andere Module ausgewählt, die die Zielinformationen liefern [97].

Trotz der eigentlich hochflexiblen Einsetzbarkeit und der klaren logischen Struktur haben sich die modularen Ansätze in der Praxis bislang kaum durchgesetzt. Dies liegt vor allem daran, dass vielen Planungsverantwortlichen die Komposition der Projektpläne aus einer Modullandkarte (oder ähnlich strukturierten Sammlungen von Modulen) als eine zu komplexe Aufgabe erscheint. In der Praxis erscheint es einfacher direkt einen Plan zu erstellen, als umständlich passende Module zu ermitteln. Plananpassungen werden so eher pragmatisch als systematisch vorgenommen. Daher besteht Bedarf nach Technologien, welche den Planer bei der Auswahl und der Komposition der Module unterstützen, um so einen systematischen Ansatz für Plananpassungen zu ermöglichen [31]. Eine solche Technologie ist Gegenstand dieser Arbeit und wurde bereits in Grundzügen in [54] publiziert (vgl. Kapitel 1.4.2).

Die Ansätze von Nöcker und Meckelnborg zeichnet vor allem ihre Allgemeingültigkeit aus. Die Module sind so gestaltet, dass sie wiederverwertbar sind und ihnen kein bestimmter Anwendungsfall zugrunde liegt. Die vordefinierten Planungsmodule sind als ein Planungsbaukasten zu betrachten, der inhaltlich und ablauforganisatorisch an ein spezielles Planungsobjekt angepasst werden kann, was bei entsprechender technologischer Unter-

stützung bei der Zusammenstellung der Module ein gut geeignetes Werkzeug ist, um der zunehmenden Dynamik und Ungewissheit bei Fabrikplanungsworkflows gerecht zu werden [109].

2.3.4 Zusammenfassung Planungsmodelle

Sequentielle und phasenorientierte Modelle unterliegen der Prämisse, dass sich an grundsätzlichen Abläufen und Zielen, die während der Initialphase festgelegt wurden, während des Projektverlaufes nichts mehr ändern lässt. Der modernen Dynamik von realen Planungsvorhaben werden diese Vorgehen durch ihre starre Architektur somit nicht mehr gerecht [109, 54].

Bei Fabrikplanungsmodellen, die den Fokus auf das Projektmanagement legen, ist eine grundlegende Flexibilität bereits erkennbar. Allerdings sind sie eher auf die Projektorganisation als auf den Projektablauf hin ausgerichtet und stehen somit in Abhängigkeit der Kompetenzen des Planungsteams. Bei konfigurierbaren Ansätzen erfolgt eine Koordination inhaltlich vorgegebener Teilprozesse. Sie basieren auf Modulbibliotheken, welche je nach Situation zu vollständigen Workflows zusammengefügt werden können. Diese Modulbibliotheken sind hinsichtlich ihres Inhalts, Methoden und Instrumenten bereits ausgereift, lediglich ihre Zusammenstellung erscheint noch zu komplex und benötigt technologische Unterstützung [109, 97, 54, 163]. Es lässt sich feststellen, dass modulare Ansätze in der Lage scheinen, Schwächen sequentieller Planungsmodelle hinsichtlich der Anpassbarkeit von Planungsverläufen auszugleichen. Auch aufgrund der klaren logischen Struktur der Modulbibliotheken erscheinen sie als die naheliegende Wahl zur Umsetzung einer automatischen und situativen Plangenerierung. Demnach bilden sie im weiteren Verlauf dieser Arbeit auch die Grundlage für die Umsetzung der Zielstellung dieser Arbeit.

Nachdem passende, etablierte Planungsvorgehen ermittelt werden konnten, ist zu prüfen, in wie weit etablierte Planungswerkzeuge, mit denen die vorgestellten Verfahren in der Praxis üblicherweise ausgeführt werden, geeignet sind, auch solche strukturell flexiblen Pläne abzubilden und umzusetzen. Im folgenden Kapitel werden gängige Planungstools vorgestellt und hinsichtlich ihrer Eignung zur Umsetzung modularer Planung hin untersucht.

2.4 Planungswerkzeuge

Wurde zuvor auf die generellen Aufgaben, Merkmale und Verfahren zur Planung von Groß- und Bauprojekten in der Fabrikplanung eingegangen, werden im Folgenden die Werkzeuge und Methoden vorgestellt, mit deren Hilfe sich konkrete Pläne und Arbeitsabläufe erstellen, strukturieren und visualisieren lassen. Dabei wird insbesondere auf die klassische Netzplantechnik eingegangen, aber auch alternative Planungsmethoden aufgezeigt.

2.4.1 Netzplantechnik

Bei der Netzplantechnik (englisch z.T. Schedule Network Analysis, Precedence Diagrams, oder auch CPM-Graphs [4]) handelt es sich um eine Methode zur zeitlichen und organisatorischen Planung von Projekten. Die Technik in ihrer heutigen Form findet ihren Ursprung im Jahre 1955, als das Chemieunternehmen Du Pont de Nemours & Co. sie für die Revision und Instandhaltung von Chemieanlagen und die US Navy sie als Organisationshilfsmittel und zur Kostenplanung verwendeten. Seitdem wird sie als eine der am weitesten verbreiteten Methode für die Projektplanung in zahlreichen Disziplinen eingesetzt [18]. Die Technik gilt als leicht zu handhaben und zu erlernen. Aufgrund ihrer weiten Verbreitung wird sie von zahlreichen Software-Lösungen zur Projektplanung unterstützt. Besonders beliebt ist die Netzplantechnik in Projekten aus den Bereichen Beschaffung, Bau und Produktion. Sie wird verwendet, um die einzelnen Vorgänge oder Prozessschritte zu staffeln und zu strukturieren und einen optimalen Prozessablauf zu planen [4, 18].

Netzplantechnik basiert auf der graphischen Darstellung eines Projektes in Form eines Netzes, in dem die einzelnen Projektschritte wie Knoten entsprechend ihrer Relation zueinander verbunden werden. Dadurch wird ersichtlich, wie einzelne Projektabschnitte voneinander abhängen, welche Arbeiten in welcher Reihenfolge verrichtet werden müssen und welche Arbeitsschritte unabhängig voneinander und somit parallelisierbar sind. Auf Basis des Graphen lässt sich eine zeitliche Planung und Überwachung des Projektes durchführen. Die Netzplantechnik bietet Werkzeuge zur Berechnung von Start und Endzeitpunkten für einzelne Arbeitsschritte, sowie zur Berechnung von etwaigen Pufferzeiten. Außerdem lässt sich durch die Technik der sogenannte *„Kritische Pfad“* ermitteln, welcher Vorgänge und Projektabschnitte kennzeichnet, deren Fortschritt für das Gesamtprojekt von größter Bedeutung sind. Es lässt sich genau erkennen, an welchen Punkten eine zeitliche Verzögerung dazu führen würde, dass der Endtermin des Projekts nicht mehr eingehalten werden kann. Dadurch können schon zu einem frühen Zeitpunkt der Planung Maßnahmen zur Risikovermeidung in das Projekt miteinbezogen werden. Durch die kontinuierliche Überwachung von Terminen sämtlicher Projektbestandteile lassen sich Verzögerungen frühzeitig auch in Teilbereichen erkennen. Aus diesem Grund hat sich die Netzplantechnik vor allem zur Planung komplexerer Projekte (wie dem Bau von Industrieanlagen) als Standard durch-

Bezeichnung	Erläuterung
CPM	Critical Path Method. Es handelt sich um einen Vorgangspfeil-Netzplan, 1955 entwickelt bei Du Pont de Nemours & Co.
MPM	Metra Potential Methode. Entwickelt 1958 in Frankreich durch die Gruppe Metra. Es ist ein Vorgangsknotennetzplan mit nur wenigen Anordnungsbeziehungen. Häufig Synonym für Vorgangsknotennetzpläne.
PDM	Precedence Diagramming Method. Eine Abwandlung von MPM, die ursprünglich stark durch die Integration in Rechenprogrammen von IBM verbreitet wurde.
PPS	Projekt-Planungs- und Steuerungssystem, ebenfalls von MPM abgeleitet. Entwickelt im Auftrag des Bundesverteidigungsministeriums durch Dornier, veröffentlicht 1968.
PERT	Program Evaluation and Review Technique, im ursprünglichen Begriffssinn Vorgangspfeilnetz, heute im angloamerikanischen Sprachraum Synonym für Netzplantechnik (auch Vorgangsknotennetze). Ursprünglich im Jahre 1958 unter Leitung der US Navy für das Polaris-Raketenprogramm entwickelt. Mit der PERT-Theorie lassen sich auch stochastische Netzplanberechnungen durchführen.

Tabelle 2.1: Varianten von Netzplantechniken. In Anlehnung an [18]

gesetzt [18, 63].

Allerdings existiert für die Netzplantechnik kein weltweit einheitlicher Standard. Vielmehr haben sich je nach Region und Anwendungsdomäne verschiedene Typen der Netzplantechnik entwickelt. Eine Übersicht über die gängigsten Varianten findet sich in Tabelle 2.1. Auch die graphischen Notationen unterscheiden sich zum Teil je nach eingesetztem Tool. Grundlegend kann zwischen Vorgangsknoten-Netzplänen und Vorgangspfeilnetzplänen entschieden werden. Bei ersterem werden Vorgänge als Knoten dargestellt, die Pfeile repräsentieren Anordnungs- und Reihenfolgebeziehungen (Beispiel: MPM). Bei einem Vorgangspfeilnetzplan hingegen werden die Planungsschritte als Pfeile dargestellt, deren logische Reihenfolge aus der Anordnung der Knoten hervorgeht (CPM). Auf einzelne Ausprägungen der Netzplantechnik wird in diesem Kapitel nicht eingegangen. Stattdessen wird im Folgenden die generelle Funktionalität und der Aufbau von Netzplänen erläutert.

Grundbegriffe der Netzplantechnik

Damit ein Projekt überhaupt mit Hilfe der Netzplantechnik sinnvoll geplant werden kann, müssen einige Vorarbeiten durchgeführt werden. So sind das Projektziel, die einzelnen Schritte oder auch einzusetzende Ressourcen im Vorfeld zu definieren. Auch müssen be-

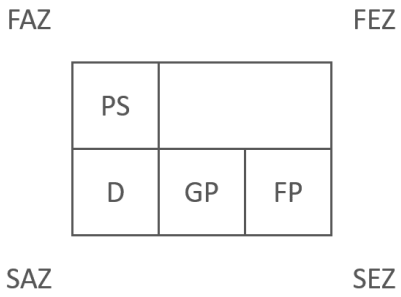


Abbildung 2.15: Vorgang in der Netzplantechnik

reits Kennzahlen der einzelnen Vorgänge vorliegen (z.B. Dauer eines Vorgangs oder der Ressourcenbedarf). Das bedingt, dass vor einer Terminplanung des Projekts mit Hilfe der Netzplantechnik mindestens die Phase der Zielplanung abgeschlossen und der Projektplan in Form einer Grobplanung vorhanden sein müssen. Sind diese Voraussetzungen gegeben, kann mit der Terminplanung begonnen werden. Um das generelle Vorgehen zu erläutern, werden deren Grundbegriffe definiert.

Vorgang (V): Ein einzelner in sich abgegrenzter Projekt- oder Prozessschritt (PS), welcher im Rahmen des Projekts abzuarbeiten ist, wird als Vorgang bezeichnet. Jeder Vorgang verfügt über verschiedene Attribute. Dazu gehören unter anderem die Vorgangsdauer (D), der Kostensatz, der Ressourcenbedarf sowie eine Liste logischer Vorgänger. Ein Vorgang kann erst abgearbeitet werden, wenn *alle* seine logischen Vorgänger abgearbeitet wurden. Es ist zwar grundsätzlich möglich auch andere Abhängigkeiten zwischen Vorgängen zu modellieren (zum Beispiel eine *Anfang-zu-Anfang*-Beziehung, bei der ein Vorgang beginnen kann, sobald mit der Bearbeitung seines Vorgängers begonnen wurde), die klassische *Ende-zu-Anfang*-Beziehung stellt aber den Regelfall dar. Darüber hinaus verfügt jeder Vorgang noch über weitere Attribute wie den Start- bzw. End-Zeitpunkt oder Pufferzeiten. Diese werden im Kontext des Gesamtplans berechnet (siehe Abb. 2.15). Dabei wird in der Regel zwischen der Berechnung der Zeitpunkte vom Projektstart (*Vorwärtsplanung*) und der Berechnung vom Projektende aus (*Rückwärtsplanung*) unterschieden. Zu berechnen sind im Detail [78]:

- **Frühester Anfangszeitpunkt (FAZ):** Der FAZ ergibt sich aus den frühesten Endzeitpunkten seiner Vorgänger und ist der früheste Zeitpunkt, zu dem ein Vorgang im Kontext eines Gesamtplans begonnen werden kann. Handelt es sich bei dem Vorgang um den Startvorgang im Projekt, so ist der FAZ gleichzeitig auch der Starttermin des Projektes.

FAZ_i: Frühester Anfangszeitpunkt eines Vorgangs $V_i = FEZ_j$, wobei hier und im Folgenden gilt: i = Index eines Vorgangs, j = Indizes der unmittelbaren Vorgänger von Vorgang i und $i, j \in \mathbb{N}$.

- **Frühester Endzeitpunkt (FEZ):** Dieser ist das Resultat der Addition des FAZ mit der Dauer des Vorgangs im Rahmen der Vorwärtsplanung. Es handelt sich dabei um den Zeitpunkt, zu dem ein Vorgang frühestmöglich beendet werden kann.

  FEZ_i : Frühester Endzeitpunkt eines Vorgangs $V_i = FAZ_i + D_i$

- **Spätester Endzeitpunkt (SEZ):** Der SEZ wird im Rahmen der Rückwärtsplanung bestimmt und bezeichnet den Zeitpunkt, zu dem ein Vorgang spätestens beendet sein muss, um das Gesamtprojektende nicht zu gefährden.

  SEZ_i : Spätester Endzeitpunkt eines Vorgangs $V_i = SAZ_k$ (unter Einhaltung des Projektendtermins), wobei hier und im Folgenden gilt: k = Indizes der unmittelbaren Nachfolger von Vorgang i und $k \in \mathbb{N}$.

- **Spätester Anfangszeitpunkt (SAZ):** Der SAZ wird durch Subtraktion der Vorgangsdauer vom SEZ im Rahmen der Rückwärtsplanung bestimmt. Der SAZ markiert den Zeitpunkt, zu dem ein Vorgang spätestens begonnen werden muss, um den Plan einhalten zu können.

  SAZ_i : Spätester Anfangszeitpunkt eines Vorgangs $V_i = SEZ_i - D_i$ (unter Einhaltung des Projektendtermins)

Netz und Netzknoten: In Vorgangsknotennetzen werden die einzelnen Projektschritte innerhalb eines Netzplans wie erwähnt als Prozessknoten dargestellt. Das Netz entsteht durch die Verbindung zwischen den Knoten untereinander durch gerichtete Kanten. Jede Kante verweist dabei von einem Vorgänger auf seinen Nachfolger. Um eine Berechnung der Projektzeiten zu ermöglichen, muss ein Netz in jedem Fall *zyklenfrei* sein.

Zusätzlich zu den frühesten (bzw. spätesten) Start- und Endzeitpunkten der Vorgänge, ergeben sich aus der Netzstruktur und den Zeitpunkten der Vorgänge noch für jeden Netzknoten sogenannte Pufferzeiten.

Pufferzeiten: Bei der Planung der einzelnen Vorgänge können Pufferzeiten entstehen. Diese bieten Spielraum bei der Ausführung einzelner Projektschritte und bilden somit Zeitreserven. Es wird zwischen vier unterschiedlichen Zeitpuffern unterschieden [78].

1. **Gesamtpuffer (GP):** Der GP definiert, um wie viele Zeiteinheiten ein Vorgang verzögert werden kann, ohne dass der SEZ des letzten Vorgangs (und somit der Termin des geplanten Projektendes) verletzt wird.

$GP_i = SAZ_i - FAZ_i$

2. **Freier Puffer (FP):** Ein FP beschreibt die Zeitspanne zwischen dem spätesten FEZ aller Vorgänger eines Knotens und dem Knoten mit dem frühesten FAZ aus der Menge der Nachfolger (S) desselben Knotens. Der freie Puffer definiert somit den Zeitraum, in dem ein Vorgang verzögert werden kann, ohne dass ein anderer Vorgang davon beeinflusst wird.

   $FP_i = min_{k \in S(i)}(FAZ_k) - FEZ_i$

3. **Freier Rückwärtspuffer (FRP):** Er bildet das Gegenstück zum freien Puffer. Hierbei handelt es sich um die Differenz zwischen dem spätesten SEZ der Vorgänger und dem frühesten SAZ der Nachfolger. Er umreißt dieselbe Zeitspanne wie der FP, aber aus der Perspektive der Rückwärtsplanung heraus betrachtet.

   $FRP_i = SAZ_k - SEZ_j$

4. **Unabhängiger Puffer UP:** Der sogenannte unabhängige Puffer gibt die maximale Zeitspanne an, um welche ein Vorgang noch verschoben werden kann, wenn seine Vorgänger alle zum spätesten Zeitpunkt enden und die Nachfolger zum frühesten Zeitpunkt starten sollen.

   $UP_{i,j} = max\{0, FAZ_j - SEZ_i - D_{i,j}\}$

Kritischer Pfad: Summiert man die Gesamtpuffer aller Vorgänge auf einem Pfad im Netz, erhält man die Gesamtpufferzeiten für diesen Pfad. Berechnet man dies für alle Pfade im Netz, wird sich ein Pfad ergeben, für den dieser Gesamtpuffer gleich Null ist. Das bedeutet, dass in der Abfolge der Vorgänge auf diesem *kritischen Pfad* genannten Pfad keine Verzögerungen eintreten dürfen. Ansonsten kann die Zeitplanung des Gesamtprojektes nicht gehalten werden.

Beispiel: Planung mit der Netzplantechnik

Zur Verdeutlichung des Vorgehens bei der Planung eines Projektes mittels eines Netzplans, wird das Verfahren an einem einfachen Beispiel erläutert. Wie bereits dargelegt, müssen die einzelnen Prozessschritte bereits vor der Erstellung definiert worden sein. Tabelle 2.2 zeigt eine Reihe von Prozessschritten, ihre jeweilige Dauer und die zur Bearbeitung der Prozessschritte erforderlichen Vorarbeiten.

Wichtig ist bei der Planung, dass die Aufstellung dieser einzelnen Vorgänge sorgfältig geplant und vollständig ist. Eine spätere Anpassung des Plans ist nur unter großem Aufwand möglich [4]. Anhand der Listen von Vorgängen, die vor einem Vorgang zu bearbeiten sind, lässt sich die Liste in einen Netz-Graphen mit gerichteten Kanten überführen. Dabei wird jeder Vorgang als Knoten gemäß der Darstellung in Abbildung 2.15 in den Graphen aufgenommen. Für jeden Eintrag in der Spalte „Vorher zu Bearbeiten“ wird anschließend

Arbeitsschritt	Dauer in Stunden	Vorher zu Bearbeiten
A	2	
B	4	A
C	3	B
D	2	B
E	1	C, D
F	4	C
G	5	E, F

Tabelle 2.2: Tabellarische Vorgangsliste

eine gerichtete Kante, die vom Vorgänger auf den jeweiligen Knoten zeigt, eingefügt. Für die Prozessschritte aus Tabelle 2.2 ergibt dies den Graphen, der in Abbildung 2.16 zu sehen ist.

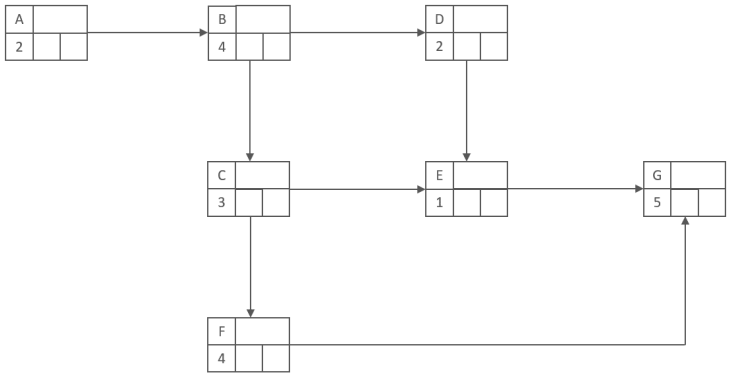


Abbildung 2.16: Vorgänge als Netzplan

Als nächstes erfolgt die Terminierung der FAZ und FEZ der Vorgänge mittels der Vorwärtsplanung. Dabei wird ausgehend von einem Startzeitpunkt (im vorliegenden Beispiel $t = 0$) der FAZ des ersten Vorgangs A bestimmt. Da der frühestmögliche Startzeitpunkt des ersten Vorgangs immer dem Projektstart gleichzusetzen ist, ergibt sich für $FAZ_A = 0$. Entsprechend des Vorgehens aus Kapitel 2.4.1 lassen sich so die FAZ und FEZ der übrigen Vorgänge berechnen. Es ergeben sich die Zeiten, die in Abbildung 2.17 zu sehen sind.

Analog zu diesem Vorgehen erfolgt anschließend die Rückwärtsplanung. Basierend auf der Annahme, dass der SEZ des letzten Vorgangs (G) immer seinem FEZ entspricht, ist dies folglich der Ausgangspunkt für die Terminierung mittels Rückwärtsplanung. Das Ergebnis dieser Planung ist in Abbildung 2.18 dargestellt.

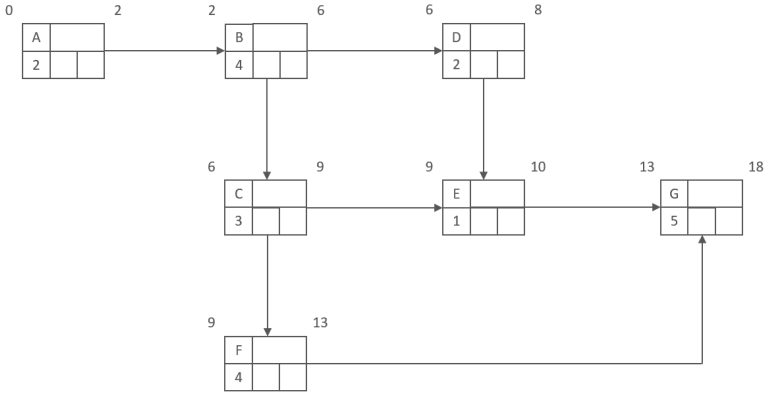


Abbildung 2.17: Netzplan nach abgeschlossener Vorwärtsplanung

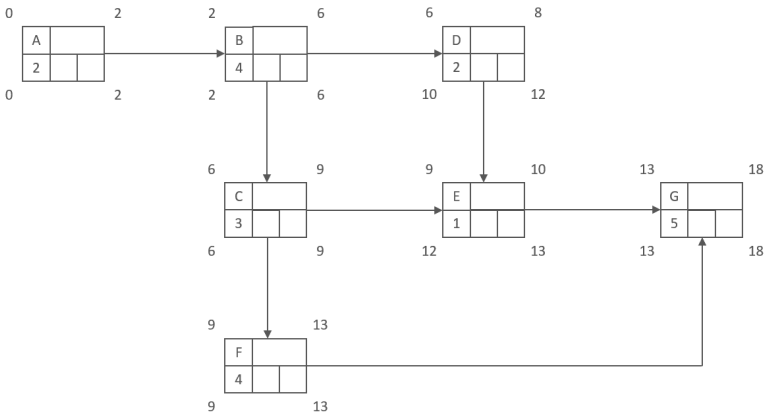


Abbildung 2.18: Netzplan nach abgeschlossener Rückwärtsplanung

Stehen die terminlichen Zeiträume fest, in denen Vorgänge bearbeitet werden können, ist es möglich, dass darauf aufbauend (entsprechend dem Vorgehen aus Kapitel 2.4.1) die Pufferzeiten errechnet werden. Der Gesamtpuffer ergibt sich aus der Differenz zwischen SAZ und FAZ für einen Vorgang. Für den Vorgang D bedeutet das: $SAZ_D = 10, FAZ_D = 6; GP_D = SAZ_D - FAZ_D = 10 - 6 = 4$. Der Gesamtpuffer sowie der freie Puffer für das Beispiel sind in Abbildung 2.19 zu finden. Der freie Rückwärtspuffer sowie der unabhängige Puffer wurden in dem Beispiel ausgelassen, funktionieren aber analog. Wie zu erkennen ist, ergibt sich für den Pfad $A \rightarrow B \rightarrow C \rightarrow F \rightarrow G$ ein Gesamtpuffer von Null je Knoten. Das bedeutet, dass auf genau diesem Pfad keine Verzögerungen auftreten dürfen. Der Pfad ist somit der *kritische Pfad*.

Kritische Würdigung der Netzplantechnik

Die Netzplantechnik ist ein seit vielen Jahren etabliertes, einfach zu erlernendes Werkzeug zur Terminplanung [84]. Sie ermöglicht eine graphische Repräsentation von Planstruk-

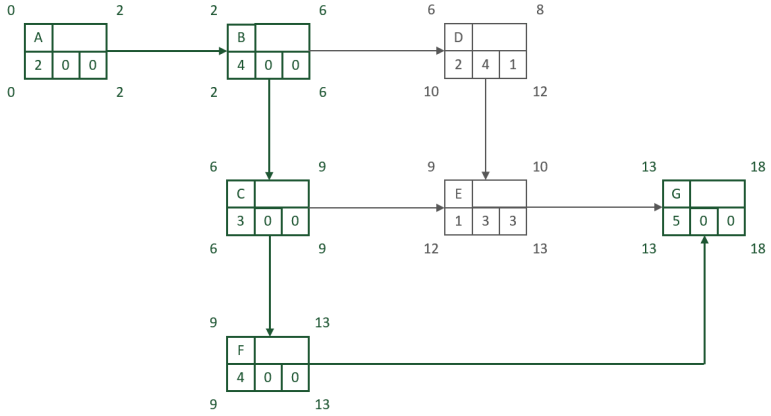


Abbildung 2.19: Netzplan mit Puffer und kritischem Pfad

turen und wird als solches seit vielen Jahren in Wissenschaft und Praxis angewendet [125, 107]. Die Verwendung der Technik bietet darüberhinaus noch weitere Vorteile. So werden Projekte, für die ein Netzplan erstellt wird, bereits vorab durchdacht und übersichtlich visualisiert. Durch die in der Technik vorhandenen Berechnungsmöglichkeiten, sind kritische Vorgänge und Prozessschritte sowie etwaige Zeitpuffer direkt identifizierbar. Damit kann Planungssicherheit geschaffen werden [84].

Dadurch, dass Abhängigkeiten zwischen einzelnen Vorgängen unmittelbar durch Kanten ersichtlich sind, lässt sich direkt erkennen, wo Vorgänge parallel bearbeitet werden können. Dies erleichtert die Zuordnung von Ressourcen zu den Vorgängen, da zeitliche Überschneidungen aus dem Plan heraus abgelesen werden können [53].

Auch zur Projektsteuerung und -überwachung eignet sich das Werkzeug. Werden Pufferzeiten überschritten oder sonstige Planabweichungen festgestellt, können durch Neuberechnung der Termine im Plan schnell neue Fristen definiert und die Auswirkungen der Abweichungen eingesehen werden. Durch diese Möglichkeit der Neuberechnung im Plan kann mit einem Netzplan effektiv auf unvorhergesehene Ereignisse reagiert werden, was den Einsatz der Technik im Industrie 4.0 empfehlenswert macht. Eine weitere Bedeutung resultiert letztlich aus der Vielzahl der vorhandenen Softwarelösungen, welche die Netzplantechnik implementiert haben (siehe Kapitel 4.2.1).

Durch die Größe und die Komplexität, die Netzplangraphen schnell erlangen können, besteht allerdings die Gefahr, dass sie leicht unübersichtlich werden. Außerdem werden viele der Funktionen, die dieses Werkzeug liefert, auch nicht für alle Projekttypen benötigt. Aus diesem Grund haben sich gerade für kleinere Projekte andere Werkzeuge, wie zum Beispiel die Gantt- (siehe Kapitel 2.4.2) oder die PLANNET-Technik (eine Weiterent-

wicklung der Gantt-Technik) mit einer kompakteren visuellen Darstellung durchgesetzt. Im Bauwesen existieren für bestimmte Klassen von Bauprojekten zudem spezialisierte Formen der Netzplantechnik. Beispielsweise werden für sogenannte Linienbaustellen *Weg-Zeit-Diagramme* (Kapitel 2.4.2) eingesetzt.

Da Netzpläne vor allem bei großen Projekten zum Einsatz kommen, sind auch die modellierten Netze folglich sehr groß. Ein Netzplan für ein durchschnittliches Bauprojekt im industriellen Kontext weist beispielsweise etwa 200 Knoten auf. Eine steigende Detaillierung des Netzplanes ist daher unmittelbar mit einem steigenden Kontroll- und Revisionsaufwand verbunden, um Fehler durch Abweichungen zwischen Plan- und Istzustand eines Projekts zu vermeiden. Auch steigt der Modellierungsaufwand für einen Plan erheblich. Der Planer bzw. die Planerin muss über ein tiefes Verständnis über die Zusammenhänge aller im Plan vorkommenden Vorgänge verfügen, um einen validen Plan erstellen zu können. Die Größe und Komplexität sowie fehlende Kontrollmechanismen für Netz-/ Planstrukturen beim Modellieren machen die Pläne somit anfällig für Fehler. Ist der Netzplan außerdem zu abstrakt bzw. praxisfremd aufgebaut, ist die Wahrscheinlichkeit hoch, dass dieser von den Anwendern nicht verstanden oder umgesetzt wird.

Kritisch ist außerdem anzumerken, dass es mit einem Netzplan zwar möglich ist, durch Neuterminierung einzelner Vorgänge auf unvorhergesehene Ereignisse zu reagieren, allerdings ist man dabei stets an die Abfolge der Aufgaben und Vorgänge in dem Plan gebunden. Eine Änderung der Arbeitsabläufe und Strukturen im Plan ist nicht vorgesehen. Dies wäre höchstens durch eine Neumodellierung des Netzes möglich, was gerade bei größeren und komplexeren Bauprojekten einen großen Aufwand zur Folge hätte [157]. Für Projekte mit sich stark ändernden und anpassungsbedürftigen Planungsverläufen ist die Netzplantechnik demnach nur eingeschränkt geeignet.

2.4.2 Alternativen und Erweiterungen zur Netzplantechnik

Die Netzplantechnik ist das in der praktischen und akademischen Anwendung am weitesten verbreitete Planungswerkzeug. Einen Großteil der etablierten Alternativen stellen außerdem Weiterentwicklungen dieser Technik dar. Da die Netzplantechnik jedoch bei der Anpassung der Planstrukturen einen Mangel an Flexibilität aufweist, werden deshalb die gängigsten der Alternativen vorgestellt und bezüglich ihrer Eignung für das modulare Ad-Hoc-Planen bewertet.

PERT und GERT-Diagramme

PERT steht für *program evaluation and review technique* und stellt eine ereignisorientierte Variante der Netzplantechnik dar. Genau wie die „herkömmliche“ Netzplantechnik hat

PERT seinen Ursprung in den 1950er Jahren. Entwickelt wurde die Technik im Rahmen des *Polaris*[1]-Projektes. Dieses Projekt zeichnete sich durch besondere Herausforderungen aus. Insbesondere mussten Termine und Fristen für die Forschung, Entwicklung und Fertigung von völlig neuartigen Komponenten geplant werden [148, 60]. Da es keine Erfahrungswerte für den Zeitbedarf der einzelnen Vorgänge gab, konnten diese, ebenso wie die Kosten, nicht im Vorfeld seriös abgeschätzt und geplant werden. Dementsprechend konnte die Terminplanung nur auf Basis von Wahrscheinlichkeiten erfolgen. Dabei sollte jeder Lieferant und jede sonstige, an einzelnen Vorgängen beteiligte Partei den Zeitbedarf für einzelne Aufgaben abschätzen. Mit Hilfe der PERT-Methode konnte daraus ein zuverlässiger Terminplan erzeugt werden [60].

Die Methode weist grundsätzlich starke Ähnlichkeit zur Kritischen-Pfad-Methode der klassischen Netzplantechnik auf. Sie verwendet als Berechnungsgrundlage allerdings den erwarteten Mittelwert der Vorgangsdauer. Das Zeitfenster für die Durchführung eines Vorgangs ist bei PERT keine skalare Größe, sondern es wird von einer Wahrscheinlichkeitsverteilung ausgegangen, wobei die Beta-Verteilung als Grundlage genommen wird. Der erwartete Mittelwert der Vorgangsdauer setzt sich zusammen aus der minimalen, optimistisch geschätzten Dauer d_min, der häufigsten (nach „bestem Wissen“ geschätzten) Dauer d_norm, und der maximalen, pessimistisch geschätzten Dauer d_max. Aus diesen einzelnen Werten ergibt sich dann folgender Mittelwert der geschätzten Vorgangsdauer:

$$d_{mittel} = \frac{d_{min}+4 \cdot d_{norm}+d_{max}}{6}$$

Eine weitere Besonderheit von PERT im Vergleich zur klassischen Netzplantechnik sind sogenannte *Entscheidungsknoten*. Diese stellen *UND-, ODER-* oder *XOR* Verzweigungen dar und können sowohl deterministisch als auch stochastisch belegt werden. Das bedeutet, dass Folgepfade sowohl durch festgelegte Bedingungen als auch durch Wahrscheinlichkeiten gewählt werden können. Diese Entscheidungsknoten ermöglichen es, auch nicht eindeutige Planstrukturen zu modellieren. Dadurch, dass alternative Pfade im Plan modelliert werden können, ist es auch möglich, auf unterschiedliche Situationen während des Projektverlaufs mit unterschiedlichen Abfolgen von Vorgängen zu reagieren. Allerdings müssen diese schon bei der Planerstellung antizipiert werden. Treten unvorhergesehene Verzögerungen auf, die durch die Variationsmöglichkeiten im Plan nicht abgedeckt sind, muss auch ein PERT-Netzplan aufwändig neu gestaltet werden. PERT-Pläne sind somit zwar deutlich flexibler als klassische Netzpläne, aber auch genau so wenig wandlungsfähig (vgl. Kapitel 1.1f). Die klassische Netzplantechnik ist demnach für Projekte prädestiniert, deren Vor-

[1] **Polaris-Projekt:** strategische Mittelstreckenraketen-Projekt der US-Armee aus den 1950er Jahren. Im Rahmen des Projektes wurden Rakten entwickelt, die von U-Booten aus abgefeuert werden konnten [148].

gänge aus Erfahrung anderer Projekte weitgehend bekannt sind und deren Zeitbedarfe verlässlich prognostiziert werden können. PERT bietet sich hingegen für die Planung von Projekten mit größerer Unsicherheit bezüglich der Zeitabschätzung an [60]. Für eine anpassungsfähige Planung „on-demand“ eigenen sich beide Ansätze gleich gut.

Ein zu PERT stark verwandtes Verfahren stellt die so genannte *GERT*-Technik (*Graphical Evaluation and Review Technique*) dar. Sie ist PERT sehr ähnlich, ermöglicht es aber zusätzlich stochastische Ausführungswahrscheinlichkeiten für einzelne Vorgänge abzubilden.

Gantt-Charts und PLANNET-Technik

Das ursprüngliche Konzept von Gantt-Diagrammen geht auf den polnischen Ingenieur Karol Adamiecki zurück, der bereits in den 1890er Jahren ein dem heutigen Gantt-Chart sehr ähnliches Werkzeug zur Strukturierung von Managementaufgaben in einem von ihm betriebenen Stahlwerk entwarf. Um das Jahr 1910 herum entwickelte Henry Gantt, amerikanischer Ingenieur und Managementberater, die heute bekannte Version des Diagramms. Ursprünglich waren die Diagramme nicht sonderlich beliebt und verbreitet [161]. Dies lag vor allem daran, dass sie zur damaligen Zeit aufwändig von Hand gezeichnet werden mussten und selbst kleinste Änderungen am Projektplan ein komplettes Neuanfertigen des Diagramms notwendig machten. Durch den modernen Software-Support erfreut sich diese Form der Darstellung von Terminplänen heutzutage aber gerade für kleinere und kompaktere Projekte großer Beliebtheit.

Genau wie die Netzplantechnik werden Gantt-Charts üblicherweise verwendet, um Terminpläne für Projekte zu visualisieren und zu verfolgen. Der Aufbau eines Diagramms ist dabei dem eines Netzplangraphen ähnlich. Einzelne Projektvorgänge werden hier als Balken dargestellt, Abhängigkeiten zwischen den Vorgängen werden (genau wie bei Netzplänen auch) durch direkte Kanten von zwei Vorgängen repräsentiert. Auch bei Gantt-Charts kann zwischen verschiedenen Anordnungsbeziehungen von Vorgängen unterschieden werden (*Ende-zu-Anfang*, aber auch *Anfang-zu-Anfang*, *Anfang-zu-Ende* und *Ende-zu-Ende*). Es variiert jeweils der Startzeitpunkt eines Nachfolgevorgangs in Abhängigkeit zu seinem Vorgänger [157].

Eine Besonderheit von Gantt-Diagrammen ist die direkte Visualisierung von Vorgangsdauern. Diese wird direkt durch die Länge des jeweiligen Balkens im Diagramm abgebildet. Gerade besonders zeitaufwändige Vorgänge sind sofort ersichtlich. (Vgl. Abbildung 2.20)

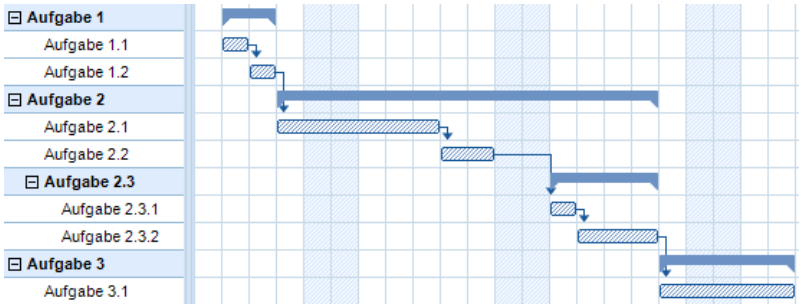


Abbildung 2.20: Beispiel eines Gantt-Diagrammes

Darüber hinaus bieten Gantt-Diagramme allerdings keine wesentlichen Features mehr. Es ist (wie bei Netzplänen auch) sofort abzulesen, welche Vorgänge wann und in welcher Reihenfolge abzuarbeiten sind und wie diese zusammenhängen. Die Errechnung von Pufferzeiten oder kritischen Vorgängen sieht das Planungswerkzeug allerdings nicht vor. Durch die strukturelle Ähnlichkeit von Netzplänen und Gantt-Diagrammen bieten die meisten Softwarelösungen allerdings auch beide Visualisierungswerkzeuge für die Pläne an. In der Regel kann man permanent zwischen den Darstellungsformen wechseln, weswegen die beiden Planungstechniken fast synonym zueinander zu betrachten sind [100, 157].

Verstärken lässt sich dieser Effekt noch durch Hinzuziehung der so genannten *PLANNET-Technik*, welche in den Gantt-Darstellungen der meisten Softwarelösungen standardmäßig integriert ist. Bei der PLANNET-Technik handelt es sich um eine Weiterentwicklung der GANTT-Technik. Sie ermöglicht die Ausweisung von Pufferzeiten. Das geschieht durch das Hinzufügen von verbindenden Linien, die terminliche Abhängigkeiten darstellen. Dadurch ergibt sich automatisch der Ausweis von Pufferzeiten, welche üblicherweise gestrichelt dargestellt werden. Die PLANNET-Technik wird angewendet, um die Projektterminierung für wenige Projektteile durchzuführen. Es ergibt sich eine optisch überzeugende Darstellungsform von Abhängigkeiten und Pufferzeiten.

Durch das Hinzuziehen der Plannet-Erweiterung sind die Netzplantechnik und Gantt-Charts von ihrem Funktionsumfang her fast als äquivalent zu bewerten. Lediglich die Zuordnung von Ressourcen zu einzelnen Vorgängen funktioniert in der Netzplantechnik noch direkt im Graphen. Dies ist in den Balkendiagrammen des erweiterten Gantt-Charts nicht vorgesehen[157].

Im Bezug auf Anpassungen der Planstruktur weisen die Balkendiagramme ähnliche Probleme wie Netzpläne auf. Zwar lassen sich zeitliche Verzögerungen und ihre Konsequenzen durch das Verschieben einzelner Balken (in entsprechender Software) relativ ein-

fach abbilden, der grundsätzliche Ablauf der einzelnen Vorgänge sowie die Struktur des Plans können aber nur über einen Neuentwurf des Diagramms visualisiert werden.

Weg-Zeit-Diagramme

Weg-Zeit-Diagramme haben als Planungstool einen der Netzplantechnik ähnlichen Aufbau [7]. Sie werden insbesondere zur Planung und Darstellung von linearen Bauprojekten wie dem Brückenbau verwendet. Dabei werden der zeitliche Ablauf sowie die örtlichen Gegebenheiten der Baustelle abgebildet. Das Weg-Zeit-Diagramm zeigt, wann und wo und in welcher Geschwindigkeit einzelne Aufgaben zu erledigen sind.

Die Darstellung des Zusammenhangs zwischen örtlichen und zeitlichen Parametern erfolgt dabei über zwei Achsen. Die vertikale Achse repräsentiert die fortlaufende Zeit, die horizontale Achse den jeweiligen Wegabschnitt. Jeder Vorgang bzw. jede Aufgabe wird dabei in Linienform im Diagramm vermerkt. Durch die Position und die Länge der Linie im Koordinatensystem des Diagramms wird dann sowohl das Start-Datum und der Start-Ort als auch das End-Datum und der End-Ort bestimmt [35, 37] (siehe Beispiel in Abbildung 2.21).

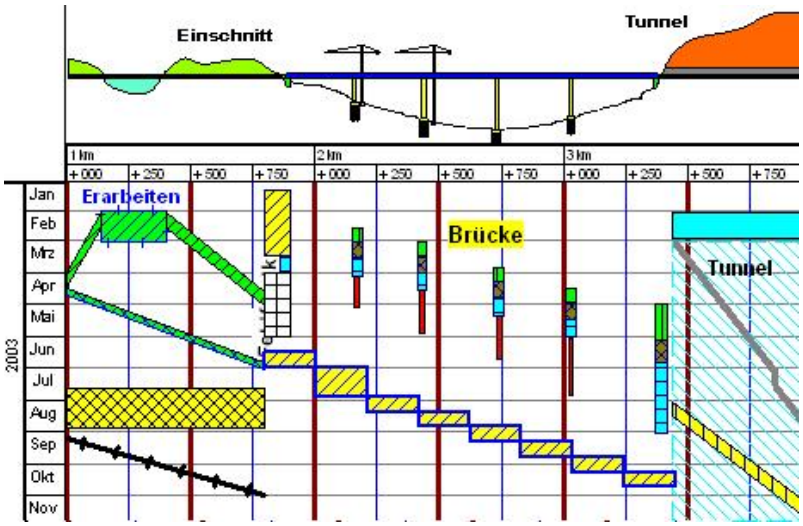


Abbildung 2.21: Beispiel für ein Weg-Zeit-Diagramm. Von Linear project GmbH - Linear project GmbH, CC-by-sa 3.0/de, https://bit.ly/2FZ0NMw

Die Geschwindigkeit, in der ein Vorgang gemäß der Planung bearbeitet werden muss, ergibt sich aus seiner Steigung im Diagramm. Je schwächer die Neigung, desto schneller muss der Vorgang absolviert werden. Auch lassen sich Konflikte zwischen einzelnen Vorgängen direkt im Diagramm ablesen. Überschneiden sich zwei Linien, würde das bedeuten, dass zwei Vorgänge zur gleichen Zeit am gleichen Ort bearbeitet werden müssen.

Gerade auf Linienbaustellen ist das nicht möglich [7]. Solche Konflikte sind im klassischen Gantt-Diagramm, der Plannet-Technik oder auch Netzplänen nicht zu erkennen[61]. Weg-Zeit-Diagramme dagegen können eine direkte Verbindung zum Lageplan der Baustelle erzeugen und vermitteln, an welcher Stelle gearbeitet werden muss. Allerdings sind sie allein auf die besondere Anwendungsdomäne der Linienbaustelle beschränkt. Netzpläne und Gantt-Diagramme dagegen sind flexibel einsetzbar [106, 61].

DCR Graphen

Dynamic Condition Response Graphs (DCR-Graphen) stellen eine constraintbasierte grafische Prozessnotation für das **Adaptive Case Management**. Diese Technik ermöglicht das Erfassen von Arbeitsabläufen und Geschäftsprozessen samt ihrer jeweiligen Nebenbedingungen (Constraints). Im Gegensatz zu anderen Techniken aus diesem Bereich handelt es sich bei DCR nicht um ein direktes Derivat der Netzplantechnik, sondern um eine vergleichsweise neue und eigenständige Entwicklung. Anders als zu diesen flussdiagrammbasierte Notationen, die Übergänge zwischen Zuständen beschreiben, die durch Ereignisse und Aktionen ausgelöst werden, konzentrieren sich DCR-Diagramme stattdessen auf die Erfassung der Geschäftsregeln und den Informationsfluss. DCR Graphen sind auf eine kontinuierliche Anpassung von Prozessen ausgelegt und unterstützen zusätzlich die sofortige Simulation und Umsetzung von Case Management Tools. Dabei erfassen DCR-Grafiken die Logik hinter dem Prozess [136, 64].

Entwickelt wurden DCR Graphen an der IT University of Denmark in Zusammenarbeit mit der Firma Exformatics A/S. Ein Spin-Off Unternehmen von Exformatics vertreibt das Werkzeug seit 2018 kommerziell.

DCR versteht sich als Weiterentwicklung klassischer Prozessmodellierungstechniken wie eben Netzplänen oder auch BPMN. Die Graphen verfügen dabei über verschiedene Arten von Kanten, die verschiedene Arten von Informationsflüssen (z.B. eine Request- und Response-Kante (zweite Kante in Abbildung 2.22)) oder Regeln (Condition-Kante (oberste Kante in Abbildung 2.22)) repräsentieren. Dadurch ergeben sich weitreichende Modellierungsmöglichkeiten. Beispielsweise lässt sich durch die Verwendung der entsprechenden Kante modellieren, dass eine Einschränkung zwischen Aktivitäten verlangt, dass die Ausführung einer Aufgabe (A) der Ausführung einer anderen Aufgabe (B) folgt. Die Reihenfolge, in welcher die Aufgaben insgesamt ausgeführt werden müssen, bleibt aber frei. Denkbare Abfolgen wären also B, AB, BAB, AAB...[136].

Ähnlich wie BPMN (welches nicht Teil der Betrachtung dieser Arbeit ist) zielt DCR insbesondere auf die Darstellung wiederkehrender Geschäftsprozesse ab. Für Planung und

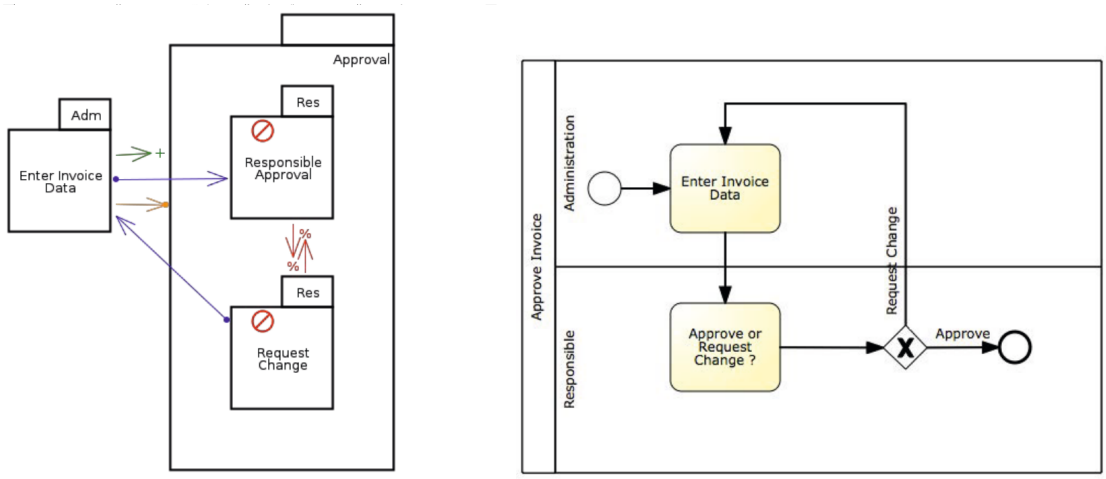


Abbildung 2.22: Beispiel eines DCR Graphen inklusive äquivalentem BPMN-Prozess, entnommen aus [136]

Projektmanagement ist das Tool hingegen nicht vordergründig designt. Vielmehr stellt die Erstellung eines DCR Graphen selbst die Planung der Geschäftsprozesse dar. Dementsprechend finden sich auch keine Anwendungen von DCR im Bereich der Fabrikplanung. Obgleich DCR Graphen leichter strukturell anzupassen sind als z.B. Netzplangraphen, existiert auch noch kein Verfahren für deren automatische Anpassung. Aufgrund der hohen Variabilität, die innerhalb eines solchen Graphen abgebildet werden kann, gestaltet sich deren automatische Generierung darüberhinaus auch potentiell komplexer.

DCR Graphen eignen sich sehr gut zur Abbildung bestehender Fabriksysteme mit hoher systemimmanenter Flexibilität. Zur Planung von Systemanpassungen erscheinen sie nur bedingt geeignet.

2.4.3 Fazit Planungswerkzeuge

Mit der Netzplantechnik existiert ein weit verbreitetes und etabliertes Planungswerkzeug. Durch den umfangreichen Softwaresupport ist es möglich, auch in kurzer Zeit verlässliche Terminplanungen durchzuführen. Allerdings weist diese Technik Defizite bei der Anpassungsfähigkeit von Plänen auf. Änderungen im Arbeitsablauf können nur durch zeitaufwändiges Neumodellieren realisiert werden [125, 107]. Demnach muss der Arbeitsplan eines Projektes bereits im Vorfeld feststehen und kann bei unerwarteten Ereignissen nur unzureichend angepasst werden. Es gibt zwar Ansätze wie die PERT- und GERT-Technik, die diese Problematik adressieren, allerdings sind diese aufgrund schwieriger Abschätzungen von Eintrittswahrscheinlichkeiten und dem nochmals deutlich erhöhten Modellierungsaufwand in der Praxis nicht sonderlich verbreitet und akzeptiert [125]. Auch fehlen gängige

Software-Tools, die diese Techniken unterstützen.

Weitere Alternativen wie Gantt-Charts oder Weg-Zeit-Diagramme fügen zwar zum Teil weitere Funktionen ein (wie beispielsweise die räumliche Dimension in Weg-Zeit-Diagrammen, Variabilität in DCR Graphen), allerdings sind sie streng genommen nur andere Visualisierungskonzepte des selben Vorgehens oder spezialisierte Techniken für dezidierte Anwendungsfälle [157, 106]. Eine bessere Alternative im Hinblick auf die Anpassungsfähigkeit von Plänen stellen sie ebenfalls nicht dar.

Insgesamt ist festzuhalten, dass es bezüglich der strukturellen Anpassbarkeit von Plänen eine Forschungslücke gibt. Zwar existieren Methoden, die im Falle von Verzögerungen innerhalb eines Planes die Terminierung anpassen können, allerdings fehlen Ansätze, die im Falle eines Anpassungsbedarfs die Struktur des Plans neu planen können. Die Entwicklung einer solchen Methodik ist aus diesen Gründen deshalb zentraler Bestandteil dieser Arbeit.

2.5 Zusammenfassung Planung

Das Planungsproblem beschreibt allgemein, die Herausforderung unter Beachtung aller gegebenen Einschränkungen zu ermitteln, welche Arbeitsschritte in welcher Reihenfolge zu erbringen sind, um ein gegebenes Planungsziel zu erreichen. Die Fabrikplanung unterliegt dabei allgemein dem Dilemma, dass immer komplexere Aufgaben in kürzerer Zeit exakter koordiniert und terminiert werden müssen. Auch nimmt die Anzahl an durchzuführenden Planungsaufgaben durch immer kürzere Produktlebenszyklen stets weiter zu. Vor allem durch die große Anzahl an beteiligten Fachdisziplinen und der gegenseitigen Abhängigkeiten zwischen diesen steigt die Bedeutung einer genauen und projektspezifischen Koordination der Aufgaben innerhalb eines Planungsprojektes [57, 160]. Kapitel 2 hat gezeigt, dass Fabrikplanungsmodelle in der Vergangenheit häufig als sequentielle Vorgehensmodelle aufgefasst und realisiert wurden [2, 79]. Im Kontext der Industrie-4.0-Bewegung und dem damit verbundenen erhöhten Bedarf an anpassungsfähigen Systemen stoßen die bislang etablierten Vorgehen allerdings an ihre Grenzen. Nöcker [109] und andere versuchten dieser Herausforderung durch ein modulares Vorgehen, bei dem Planungsverläufe individueller und auf den Planungsfall zugeschnitten zusammengestellt werden können, gerecht zu werden. Diese Modelle stellten sich aber als sehr aufwändig und komplex heraus. Die Komposition von Plänen mit der Komplexität von Fabrikplanungsaufgaben aus einzelnen Planungsmodulen erscheint als eine Aufgabe, die Planer*innen oft scheuen [109]. Ein Ansatz ist die Automatisierung der Komposition, um flexiblere Planungsverläufe zu ermöglichen, ohne die Komplexität und Schwierigkeit der Planungsaufgaben für die Pro-

jektbeteiligten weiter zu erhöhen.

Unabhängig vom verwendeten Planungsmodell stellt ein Plan stets eine Abfolge (oder Sequenz) von Aufgaben dar, die zur Erreichung eines Planungsziels durchgeführt werden müssen. Es stellt sich folglich die Frage, ob sich Technologien anbieten, die die Ermittlung der für das Planungsziel relevanten Planungsaufgaben sowie deren Zusammenstellung zu einem Gesamtplan auf Basis von Modularen Verfahren unterstützen können. Dadurch ließe sich die Komplexität der Aufgabe für Planer*innen einschränken. Durch Anwendung der Typinhabitation mittels kombinatorischer Logik und anderen Methoden und Techniken aus der Informatik ließe sich diese Automatisierung umsetzen. Deshalb sollen im Folgenden die in [109] und anderen darauf aufbauenden Arbeiten vorgestellten Module als Ausgangspunkt für eine Technik zur automatischen Modellierung und Generierung von Planungsworkflows benutzt werden. Die generierten Pläne sollen darüber hinaus ausführbar, benutzbar und verständlich aufgearbeitet sein, weshalb sich die Generierung eines etablierten Planungswerkzeugs empfiehlt. Rückblickend auf die Erkenntnisse aus Kapitel 2.4.1 bietet sich die Verwendung der Netzplantechnik an.

Vor dem Hintergrund des in Kapitel 2 dargestellten Standes der Forschung ergibt sich für die Zielsetzung dieser Arbeit folgendes Vorgehen:

1. Ermittlung von Prozess-Bausteinen basierend auf den modularen Vorgehen aus [109], die die Aufgaben der beteiligten Planungsgewerke abdecken.

2. Aufbereitung und Sammlung der Bausteine in einem einheitlichen aufgebauten Repository mit einheitlichen Schnittstellen.

3. Erstellung eines Vorgehens zur Zusammenstellung von Planungsverläufen aus den Bausteinen im Repository.

4. Überführung und Darstellung der erstellten Planungsworkflows in einen Netzplan.

Das Verfahren soll weitestgehend automatisch ablaufen, damit im Falle eines Anpassungsbedarfes schnell neue Lösungen und Planungsverläufe „on demand“ generiert werden können. Im Idealfall soll das Verfahren auch noch in der Lage sein, mehrere strukturell unterschiedliche Varianten eines Plans zu generieren, sodass Planer*innen nicht nur innerhalb der Netzplantechnik Entscheidungs- und Optimierungsmöglichkeiten haben, sondern auch zwischen verschiedenen alternativen Abfolgeplänen für das jeweilige Projekt auswählen können. Dies ist im Kontext der Fabrikplanung insofern erforderlich, da es für Projektpläne dieser Art keine optimalen Lösungen gibt [160]. Durch die Komplexität und zum Teil Widersprüchlichkeit der Einflussgrößen handelt es sich bei der Erzeugung von Plänen stets um ein multi-kriterielles Optimierungsproblem, deren Kriterien sich selten

exakt gewichten lassen. Deshalb ist die Auswahl der jeweils geeignetsten Lösung dem verantwortlichen Planer/der verantwortlichen Planerin zu überlassen [160, 57].

Das folgende Kapitel gibt einen Überblick über Möglichkeiten zur automatischen Komposition von Planungsworkflows auf Basis einer Modulsammlung. Dabei wird insbesondere auf die Synthese und das Constraintsolving eingegangen. Ziel ist es, eine geeignete Auswahl an Methoden und Technologien zu treffen, um die praktische Umsetzung der Zielstellung dieser Arbeit durchführen zu können.

Kapitel 3

Synthese

Softwaresynthese ermöglicht die Komposition eines Programms aus einer logischen Spezifikation heraus [26] und kann zudem als eine automatisierte Suche über alle möglichen Programme in einer Programmiersprache betrachtet werden. Der Anwender schränkt die Suche dabei durch Angabe von Spezifikationen und Bedingungen ein. Besonders beliebt ist die Synthese von Software bei der Automatisierung einfacher Programmieraufgaben in Softwareprojekten, bei denen kleinere oder wiederkehrende Codefragmente automatisch generiert werden [73]. Obgleich die Synthese bei der Erzeugung von Programmcode ihren (informatischen) Ursprung hat, ist sie auch auf andere Gebiete anwendbar. So wurden zum Beispiel Syntheseverfahren zur Erzeugung von BPMN-Abläufen oder sonstigen nebenläufigen Prozessen eingesetzt [21, 26]. Dieses Kapitel gibt einen Überblick über bestehende Arbeiten im Bereich der Synthese und Generierung sowohl von Software als auch von Prozessen und Workflows. Im Anschluss wird das dieser Arbeit zugrunde liegende Konzept der Synthese mittels kombinatorischer Logik mit Intersektionstypen sowie das verwendete Framework *(CL)S* vorgestellt.

3.1 Bestehende Arbeiten

Im Bereich der Synthese lassen sich unterschiedliche Ansätze in der Spezifikation und in der Berechnung des jeweiligen Syntheseergebnisses finden. Einige basieren dabei auf typtheoretischen Ansätzen und verwenden deduktive Methoden, andere bauen hingegen auf Automatentheorie und temporaler Logik auf [21]. Außerdem lassen sich die Ansätze in die Kategorien „komponentenbasiert“ und „from-Scratch“ aufteilen. Während letztere das zu synthetisierende Programm nahezu vollständig aus der Spezifikation und der Logik des Frameworks heraus erzeugen, greifen komponentenbasierte Ansätze auf vorgefertigte Bestandteile zurück und setzen das Syntheseergebnis in Form einer Komposition aus diesen Einzelteilen zusammen. (Software-)Komponenten werden dabei als eine architektonische Einheit bezeichnet, die

1. eine Menge der Systemfunktionalität oder -daten kapselt,

2. den Zugang zu dieser Menge über eine explizit definierte Schnittstelle einschränkt und

3. über explizit definierte Abhängigkeiten im erforderlichen Ausführungskontext verfügt [98].

Darüberhinaus existieren Frameworks, welche sich nach *Bodik und Jobstmann* [26] der funktionalen Synthese mit einem semantischen Suchraum zuordnen lassen. Bei der funktionalen Synthese sind die Programme im Suchraum nicht auf Programme mit einer endlichen Anzahl von Programmstati beschränkt. Vielmehr ist die Suche von beliebigen

Programmen in Allzwecksprachen (engl. General Purpose Language) möglich. Der Suchraum wird semantisch durch Axiome eingeschränkt [26]. Ebenso kann er auch syntaktisch eingeschränkt werden [21]. Ein gutes Beispiel ist die sogenannte syntaxgeführte Softwaresynthese (engl. syntax-guided software synthesis, kurz: SyGuS). In dieser Variante der Softwaresynthese gibt es neben den Korrektheitsspezifikationen (in Form einer logischen Formel und einer Hintergrundtheorie) auch eine syntaktische Vorlage, welche den Suchraum einschränkt [33]. Verifiziert werden können diese semantischen Korrektheitsspezifikationen unter anderem durch einen SMT-Solver (siehe Kapitel 3.5), mit welchem der Wahrheitswert einer logischen Formel bestimmt werden kann [5]. Die Verbindung zwischen SyGuS und SMT-Solving wurde bereits in zahlreichen Projekten zur Synthese u.A. von schleifenfreien Programmen [59], Excel-Makros [58] oder Protokollen eingesetzt [5]. In diesen Projekten erfolgte die syntaktische Einschränkung der Synthese durch die Angabe von Ein- und Ausgabebeispielen [59, 58]. Nach *Zdancewic et al.* sind solche Ein- und Ausgabebeispiele als Verfeinerungstypen (englisch:. Refinement Types) zu interpretieren, wodurch die Synthese als Suche eines Inhabitanten dieses Typs betrachtet werden kann [49]. Diese Sichtweise deckt sich mit der von *Rehof* und dem in seiner Arbeitsgruppe entstandenen funktionalen Syntheseframework (CL)S [20]. Das Konzept der taxonomisch unterstützten (Prozess-) Synthese mit semantischen Typen, dem das (CL)S folgt, wurde 1997 von Steffen et al. [141] vorgestellt und stützt sich dabei auf eine Publikation von Steffen et al. aus dem Jahre 1993 [142]. Aus dieser Entwicklung entstanden auch Frameworks wie die DyWA [74] und DIME [27], ebenso ETI [96] und Prophets [103], die eine semantische, beschreibungsorientierte Entwicklung von Programmen und Prozessen ermöglichen.

3.1.1 Plan- und Prozesssynthese

Auch im Kontext der Planung existieren bereits einige Ansätze, die Synthese einsetzen (z.B. [14, 24]). Es gibt nur wenige Ansätze, die geeignet wären, Planung und damit die Entwicklung von Lösungsvarianten zu generieren. Die Arbeit von *Ilghami* in [71] demonstriert die Synthese einer Planungssoftware für domänenspezifische hierarchische Task-Netzwerke (HTN). Planware [14, 24] ist ein System zur Synthese von Algorithmen, die domänenspezifische Zeitpläne erstellen. Ebenso kann die Synthese von Workflows als Teil der Planungssynthese betrachtet werden. Hier existieren arbeiten z.B. von *Lamprecht* [87], aber auch aus der Semantic Web Services Challenge [72], in deren Rahmen Konzepte und Frameworks zur Generierung von Workflows aus gegebenen Spezifikationen heraus vorgestellt wurden. Das von *Awad* in [9] vorgeschlagene Konzept ermöglicht die iterative Verfeinerung der Domänenspezifikation, bis ein geeigneter Prozess generiert werden kann. Nach *Grambow* [56] wird eine automatisierte Laufzeitflexibilität für Software-Engineering-Prozesse durch In-Corporations eines semantischen Reasoner und Ad-hoc-Änderungen von Prozessinstanzen bereitgestellt. Diese können dann in AristaFlow dargestellt werden [119].

Fernandes stellte wiederum das Konzept der inkrementellen Planung [46] vor, welches auf einer Bibliothek von vordefinierten einfachen und szenariospezifischen Workflows basiert. Aus dieser Bibliothek wurden Workflows für bestimmte komplexere Szenarien generiert. Allerdings war dieser Ansatz stark auf dezidierte Planungsfälle zugeschnitten und verfolgte keine allgemeingültige Anwendbarkeit [46]. Modellbasierte Ansätze zur choreographischen Umsetzbarkeit wurden ebenfalls untersucht [8].

Für die Workflow-Generierung existiert bereits eine Auswahl an Ansätzen. Diese Ansätze decken dabei verschiedene Aspekte der Thematik ab. So lassen sich Ansätze für die Synthese und Anpassung von Prozessen, die bereits bestehen (z.B.[40, 56]), genauso identifizieren, wie Ansätze, die Prozessskizzen vervollständigen [88], ausführbare Workflows bestimmter Prozesse generieren [124] oder aber auch die Synthese von kompletten Prozessen ermöglichen. Die Art und Weise wie diese Ansätze gestaltet sind, ist ebenso vielfältig. *Zhang et al.* verwenden zum Beispiel in [167] maschinelles Lernen, um Prozesse iterativ zu verfeinern. Dieser Ansatz stützt sich stark auf Aufgaben der menschlichen Intelligenz [167]. Einzelne Aufgaben werden in einem Workflow nach ontologiebasierten Koordinationsregeln [34, 56] ausgerichtet, andere (eher deduktive) Herangehensweisen nutzen wiederum Theoremprover [23]. Darüberhinaus kann die Synthese gemäß einer Reihe von Input-/Output-Beispielen durchgeführt werden [90, 47]. Die Suchtechnik bestimmt, wie der Synthesealgorithmus nach dem Zielprogramm im Suchraum sucht. Es gibt verschiedene Ansätze, die das Lösen von Bedingungen [138], die Verwendung von semantic Reasonern (z.B. graphenbasierte Suchtechniken) [56] oder neuronale Netze [111] beinhalten.

3.1.2 Komponentenbasierte Softwaresynthese

Im Bereich der komponentenbasierten Softwaresynthese wurden Ansätze und Techniken vorgestellt, mit denen Variabilität in Produktlinien unter Verwendung von Merkmalsmodellen formalisiert werden kann. Feature Modeling und Software Product Lines (SPL) [113, 6, 149] sind eng verwandte Themen, die die Individualisierung und Standardisierung von Software repräsentieren. Beide tragen dazu bei, die Entwicklungszyklen von Softwareprodukten zu verkürzen, indem sie die Wiederverwendung von Software erleichtern und ein systematisches Management von Softwareproduktfamilien nachvollziehen. Durch die Suche wurde das Programmierparadigma mit der Feature-orientierten Softwareentwicklung entworfen, das durch eine umfassende Entwicklungsumgebung unterstützt wird [146]. Variabilitätsmodellierung ist ein weit ausgearbeitetes Forschungsthema im Kontext der Geschäftsprozessmodellierung und findet auch weiterhin viel Beachtung.

Variabilität wird in vielen Kompositionsansätzen durch Gestaltung von einzelnen Features oder Komponenten nach dem sogenannten Opt-In-Prinzip erzielt. Das bedeutet, dass

diese Komponenten dem zu erzeugenden System optional und beliebig hinzugefügt werden können. Das System wird durch das Vorhandensein der Komponente in der Lösung durch die Funktionen der Komponente erweitert, ist aber auch ohne diese voll funktionsfähig. Das Konzept ist vergleichbar mit Feature-Diagrammen und den in ihnen Darstellbaren Konfigurationen [77]. Nach diesem Prinzip arbeitet unter anderem die aspektorientierte Programmierung (z.B. in [80]), aber auch Dependency Injection Frameworks wie Google Guice [150] oder die generative Programmierung von Czarnecki [39]. Auch *Rehof's et al's* „Combinatory Logic Syntehsizer Framework“ arbeitet nach diesem Prinzip. Hier wird die Komposition in einer Metasprache durchgeführt [43], in der Features hinzugefügt werden können. Darüberhinaus können komplexe Komponenten im (CL)S eine Vielzahl von Code-Transformationen durchführen. Dies ist charakteristisch für transformatorische Ansätze wie die Delta-Modellierung [36].

3.1.3 Verwandte Themenfelder

Starke Anknüpfungspunkte gibt es zu dem Forschungsfeld der Domänenmodellierung. Ein zu synthetisierendes Programm enthält üblicherweise domänenrelevanten Code. Genauso sind Constraints oder Abhängigkeiten, die bei der Synthese beachtet werden müssen, oft domänenabhängig. Dementsprechend kann die Spezifikation des Suchraums durch Verwendung eines domänenspezifischen Modells erreicht werden [124]. Die Entwicklung dieser Modelle wird in der Regel durch Modellierungswerkzeuge unterstützt, die zum Teil eigene Codegenerierungsfunktionalitäten aufweisen. Diese sind der Synthese ähnlich. Der zusätzliche Konstruktionsaufwand für diese Tools kann durch Meta-Modellierungs-Frameworks wie CINCO [143, 105, 104] kompensiert werden. Bei Ansätzen wie diesen wird dem Nutzer die Beschreibung und Spezifikation des gewünschten Programmes in einer grafischen Oberfläche mit Hilfe von Formalismen wie der temporalen Logik [9, 165, 88], Logik erster Ordnung [5] oder Logik höherer Ordnung [139] ermöglicht.

Nach *Bodik und Jobstmann* [26] gibt es darüber hinaus noch zahlreiche weitere Anwendungsfelder und Ansätze aus dem Bereich der Synthese. So gibt es das Themenfeld der *reaktiven Synthese*, die darauf abzielt, automatisch ein reaktives System aus einer formalen Spezifikation zu konstruieren. Die Spezifikation ist dabei in Linear-Temporaler-Logik (LTL) formuliert und beschreibt die logischen Eigenschaften des Systems [26, 29, 115]. Andere Ansätze sind zum Beispiel die *quantitative Synthese* (Belohnungs- und Kostenerweiterungen werden für verschiedene Arten von Systemen berücksichtigt (z.B. [28, 41].), *Induktive Programmierung* (Synthese von funktionalen sowie logischen Programmen aus unvollständigen Spezifikationen (z.B. [48, 81])), oder die *Synthese für zeitgesteuerte und hybride Systeme* (Syntheseprobleme der Steuerung für Erreichbarkeitsspezifikationen in Hybridsystemen, die auf der Spieltheorie basieren (z.B. [94, 95])). Außerdem zeigen *Bodik*

et al. [26] noch eine Reihe weiterer Anwendungsgebiete der Synthese auf. Hierzu zählen die Synthese von Client Code (z.B. [51]), die Synthese von Funktionen zur String-Manipulation [89, 59] sowie die Synthese von Garbage-Collectoren [152, 153] oder Daten Strukturen [62, 137].

Wie zu erkennen ist, ist das Themenfeld der Synthese allgemein und der Workflow-Generierung im speziellen ein sehr vielfältiges Forschungsfeld. Für die technische Umsetzung und Unterstützung von modularen Planungsverfahren (wie das von Nöcker) scheinen komponentenbasierte Ansätze am besten geeignet, da sich ihr grundlegendes Konzept der automatischen Komposition verschiedener Komponenten zu einer spezifizierten Lösung hin direkt auf den modularen Charakter der Planungsworkflows dieser Verfahren übertragen lässt. Ebenso erfüllen Planungsmodule nach Nöcker die drei Anforderungen an Komponenten, welche in *Medvidovic et al.* [98] definiert wurden. Sie kapseln einzelne Teilabläufe eines Plans in sich ab. Durch die definierten Ein- und Ausgabe-Informationen verfügen sie über eine klar definierte Schnittstelle und über klar definierte Abhängigkeiten im Ausführungskontext. Es ist deshalb naheliegend, einen komponentenbasierten Syntheseansatz zur Umsetzung dieser Arbeit zu wählen. Darüber hinaus ermöglicht die Variabilität, die durch solche Ansätze abgedeckt wird, das Erstellen mehrerer strukturell unterschiedlicher Workflows, die verschiedene Handlungsalternativen darstellen. Dadurch ist das flexible Erstellen von Handlungsoptionen gewährleistet. Das (CL)S-Framework erscheint demnach als folgerichtiger Ansatz.

Für die automatische Erzeugung von Netzplänen wird in diesem Kapitel die dem (CL)S zugrundeliegende komponentenbasierte und funktionale Synthese basierend auf kombinatorischer Logik mit Intersektionstypen vorgestellt. Außerdem wird das Framework und der Umgang damit selbst dargelegt.

3.2 Kombinatorische Logik

Das Prinzip der kombinatorischen Logik ist auf Moses Schönfinkel zurückzuführen, der bereits in den 1920 Jahren in diesem Bereich geforscht hat. Er publizierte 1924 das Ursprungswerk der kombinatorischen Logik „Über die Bausteine der mathematischen Logik“[129]. Im gleichen Zeitraum entstand auch das λ-Kalkül [66], bei dem es sich ebenfalls um ein logisches System zur Wiedergabe der Idee von zu berechnenden Funktionen und Algorithmen handelt. Im Unterschied zum λ-Kalkül werden in der kombinatorischen Logik keine gebundenen Variablen betrachtet. Dennoch sind die beiden Systeme gleich ausdrucksstark [66]. Kombinatorische Logik ist die Logik der Funktionen, welche auf zwei Arten betrachtet werden können. So lässt sich eine Funktion zum einen als Menge ihrer Ein- und Ausgabeparameter betrachten. Eine Funktion, die die Zahlen 4 und 5 addiert und als Ergebnis

folgerichtig die 9 liefert, kann somit auch als Triple (4,5,9) gesehen werden. Andererseits lässt sich eine Funktion auch als Regelwerk auffassen, mit deren Hilfe man von der Ein- zur Ausgabe kommt. Beide Ansichten stehen im ständigen Wechselspiel und spielen auch bei der Betrachtung von Planungsmodulen (siehe Kapitel 4.4.1) eine Rolle [22].

Funktionen werden in der kombinatorischen Logik als *Terme* bezeichnet. Terme bestehen wiederum aus weiteren Elementen: Konstanten und Variablen. Die Kombinatoren S und K zählen typischerweise zu den Konstanten, ebenso wie die Kombinatoren B, C, W, M oder I. Variablen werden als Kleinbuchstaben dargestellt, die Großbuchstaben M, N, P und Q kennzeichnen Meta-Variablen. Diese können durch Konstanten oder Variablen ersetzt werden. Terme, die solche Meta-Variablen enthalten, werden auch als Meta-Terme bezeichnet. Zusätzlich existiert ein binärer Operator, welcher durch die Nebeneinanderstellung von Termen die Anwendung von Funktionen darstellt. Aus dieser Menge an Begriffsdefinitionen, definiert sich die Menge der Terme der kombinatorischen Logik wie folgt:

Definition 3.1

Die Menge der Terme wird induktiv definiert durch:

1. *Wenn Z eine Konstante ist, dann ist Z ein Term.*
2. *Wenn x eine Variable ist, dann ist x ein Term.*
3. *Wenn M und N Terme sind, dann ist (MN) ein Term.*

$e ::== Z \mid x \mid (ee)$

Terme können gekürzt werden ($MNP \equiv ((MN)P)$), da diese grundsätzlich linksassoziativ sind. Ist ein Term Teil einer Applikation von Termen oder Teil eines größeren Terms, so spricht man auch von Subtermen.

Definition 3.2

Die Relation ist Subterm auf der Menge der Terme und wird definiert durch:

1. *Z ist ein Subterm von Z*
2. *x ist ein Subterm von x*
3. *M ist ein Subterm der Terme (NM) und (MN)*
4. *(NP) ist ein Subterm von (NP)*
5. *Wenn M ein Subterm von N ist und N ein Subterm von P, dann ist M ein Subterm von P*

3.2.1 Kombinatoren

Kombinatoren sind ein Werkzeug zur Modifikation der Argumente einer Funktion, die durch ihre Stelligkeit[2] charakterisiert werden. Der einfachste Kombinator ist der Identitätskombinator I, welcher über die Stelligkeit 1 verfügt (unärer Kombinator). Der Kombinator S verfügt über eine Stelligkeit von 3. Wie genau Kombinatoren Argumente einer Funktion modifizieren, wird durch Axiome verdeutlicht. Ein Axiom besteht aus zwei Termen, die durch ein Separator-Symbol getrennt sind. Der Term auf der linken Seite zeigt die Stelligkeit des Kombinators und der Term auf der rechten Seite zeigt das Ergebnis der Anwendung des Kombinators [22].

Name	**Effekt**	**Axiom**
Identitätskombinator	Gibt die Eingabe als Ausgabe zurück	$Ix.x$
Assoziatorkombinator	Fügt die Assoziation des rechten Terms hinzu	$Bxyz.x(yz)$
Stornierungskombinator	Storniert das zweite Argument und streicht es in der Ausgabe	$Kxy.x$
Permutationskombinator	Permutiert die Reihenfolge der Argumente	$Sxyz.xz(yz)$
Duplikationskombinator	Duplikation eines Teils der Eingabe	$Wxy.xyy$ $Mx.xx$

Tabelle 3.1: Kombinatoren und dazugehörige Axiome

Die in Tabelle 3.1 aufgelisteten Kombinatoren stellen sogenannte primitive Kombinatoren dar. Weitere Kombinatoren können dadurch erzeugt werden, dass Terme erstellt werden, die nur aus primitiven Kombinatoren bestehen [22]. Solche zusätzlichen komplexeren Kombinatoren lassen sich allerdings auch durch ihre primitiven Gegenstücke charakterisieren. Ein freier Kombinator Z mit dem Axiom $Zxyz.Zx$ zählt ebenfalls zu den Stornierungskombinatoren. Zusätzlich besitzen Kombinatoren die Eigenschaft der *Regularität*. Ein Kombinator ist regulär, sofern das erste Argument auch nach Anwendung des Kombinators an erster Stelle steht.

Für die Verwendung von kombinatorischer Logik und der Kombinatoren existiert ein mit dem Lambda-Kalkül vergleichbares Kalkül. Das sogenannte SKI-Kombinator-Kalkül ist eine reduzierte Version des ungetypten Lambda-Kalküls. Obgleich es nicht zur Implementierung von Software geeignet ist, kann es dennoch als rudimentäre Programmiersprache angesehen werden. Das ebenfalls von Moses Schönfinkel [129] und Haskell Curry vorgestellte Kalkül ist einfach gehalten und ist Turing-vollständig. Dabei kommt es nur mit einer Basis der Kombinatoren SKI aus, wobei sich I als redundant herausgestellt hat [118].

[2]Bezeichnet die Anzahl der Argumente einer Verknüpfung, einer Abbildung bzw. eines Operators oder die Parameteranzahl von Funktionen, Prozeduren oder Methoden.

Definition 3.3

Basis: *Eine endliche, nicht-leere Menge $\mathfrak{B}$, bestehend aus Kombinatoren wird als ~~Basis~~ bezeichnet.*

Das SKI- (oder SK-) Kalkül gilt als kombinatorisch abgeschlossen, sodass aus den Kombinatoren S und K jeder beliebige weitere Kombinator und Term konstruiert werden kann. Es konnte gezeigt werden, dass das Inhabitationsproblem der kombinatorischen Logik mit dieser Basis PSPACE-vollständig ist [118].

Definition 3.4

Kombinatorische Abgeschlossenheit: *Eine Basis $\mathfrak{B}$ ist kombinatorisch abgeschlossen, wenn für jede Funktion f mit $f_{x_1 \ldots x_n} \rhd_w M$, wobei M aus den Variablen $x_1 \ldots x_n$ besteht, ein Kombinator Z existiert, der in der Basis enthalten ist oder aus den Konstanten der Basis definiert werden kann, sodass $Zx_1 \ldots x_n \rhd_w M$ gilt.*

Lemma 3.1

Die Basen $\{S, K\}, \{I, B, C, W, K\}$, sowie $\{I, J, K\}$ sind kombinatorisch abgeschlossen [26].

Die Konstruktion einer kombinatorisch abgeschlossenen Basis ermöglicht folglich die Konstruktion beliebiger Terme aus dieser Basis heraus. Für die Konstruktion und die Ableitung von Termen können Hilfsmittel wie Variablen, aber auch Operationen wie die *Substitution* und die *Reduktion* von Termen verwendet werden. Die grundlegende Definition dieser Operationen und die Regeln, nach denen sie verwendet werden, werden in den folgenden beiden Abschnitten dargelegt.

3.2.2 Substitution

In den vorgestellten Kombinatoren wurden Variablen für die Beschreibung der Axiome genutzt. Da die Kombinatoren nicht ausschließlich mit Variablen, sondern auch mit beliebigen Termen genutzt werden können, können die Variablen substituiert werden [22]. Zur Darstellung von syntaktischer Gleichheit von Termen wird das Symbol $\equiv$ verwendet [26]. Die Substitution definiert sich wie folgt:

Definition 3.5

*Die **Substitution** einer Variable x durch den Term M im Term N wird durch N_x^M dargestellt und ist wie folgt definiert:*

1. *Wenn $N \equiv x$, dann ist $N_x^M \equiv M$*
2. *Wenn $N \equiv y$ mit $x \neq y$, dann ist $N_x^M \equiv N$*
3. *Wenn $N \equiv Z$, dann ist $N_x^M \equiv N$*
4. *Wenn $N \equiv (P1P2)$, dann ist $N_x^M \equiv (P_{1_x}^M P_{2_x}^M)$*

3.2.3 Reduktion

Kombinatoren, Terme und Axiome können reduziert werden. Die Reduktion formalisiert das Konzept der Funktionsanwendung und soll die Auswirkungen der Ausführung eines Kombinators darlegen. Zur Verdeutlichung der Funktionsweise der Reduktion muss zunächst der Begriff des Redexes eingeführt werden.

Definition 3.6

Redex: *Sei Z ein n-stelliger Kombinator, dann ist ein Term der Form $ZM_1 ; \ldots M_n$ ein Redex. Z bezeichnet den Kopf und $M_1 \ \ldots \ M_n$ die Argumente des Redex.*

Terme, welche die Meta-Variablen $M_1 \ \ldots \ M_n$ instanziieren, können selbst Kombinatoren oder Redexe enthalten. Aus diesem Grund kann auch ein Redex wiederum einen oder mehrere Kombinatoren enthalten. Dies hat auch zur Folge, dass der Term Z nicht nur im Kopf, sondern auch in den Termen des Redexes vorkommen kann [22].

Definition 3.7

Ein-Schritt-Reduktion: *Sei Z ein primitiver Kombinator mit dem Axiom $Z_{x_1 \ldots x_n} \rhd P$. Wenn N ein Term mit einem Subterm der Form $ZM_1 \ldots M_n$ ist, und N' ist N mit dem Subterm ersetzt durch $P_{x_1}^{M_1} \ldots P_{x_n}^{M_n}$, dann ist N ein-Schritt-reduziert zu N', was durch die Schreibweise $(N \rhd_1 N_0)$ dargestellt wird.*

Redexe und die Ein-Schritt-Reduktion sind Hilfsmittel, mit denen die Auswirkungen eines Kombinators in Form eines Axioms formal beschrieben werden können.

Name	**Axiom**	**Erläuterung**
S: Permutationskombinator	$Sxyz \rhd xy(zy)$	Permutiert die Reihenfolge der Argumente
K: Stornierungskombinator	$xy \rhd x$	Entfernt das zweite Argument der Eingabe

Tabelle 3.2: Axiome der Kombinatoren S und K

Tabelle 3.2 zeigt erneut die Axiome der Basiskombinatoren S und K. Vor allem anhand des K-Kombinators lässt sich die Ein-Schritt-Reduktion gut verdeutlichen. Der Teilterm auf der linken Seite des Axioms (xy) ist gleichzusetzen mit M aus Definition 3.7, der rechte Teil wiederum mit P. Folglich gilt $N = Kxy$ sowie $N' = [x/x, y/y]x$ und $P = x$. Gemäß den Substitutionsregeln aus Definition 3.5 lässt sich ableiten, dass gilt $[x/x, y/y]x \equiv x$, woraus folgt, dass $N' \equiv P \Rightarrow N \rhd_1 P$ ist.

Durch Erweiterung der Ein-Schritt Reduktion lassen sich noch weitere Reduktionsformen (wie die schwache Reduktion) ableiten.

Definition 3.8

Schwache Reduktion $(\triangleright_w)$***:*** *Reflexiver und transitiver Abschluss der Ein-Schritt-Reduktion.*

Für die schwache Reduktion bezogen auf einen Term M gilt, dass sich dieser Term auch selbst schwach reduziert ($M \triangleright_w M$). Lässt sich M zusätzlich über mehrere Schritte auf P reduzieren ($M \triangleright_1 N \triangleright \ldots \triangleright_1 P$), so gilt auch, dass $M \triangleright_w P$ (M reduziert schwach auf P). Man spricht von einer schwachen Reduktion, da sie grundsätzlich an die Anzahl der Argumente in den Axiomen der Kombinatoren gebunden ist. Ein Kombinator wie zum Beispiel S erwartet immer drei Argumente. Folglich könnte SZM auch nicht reduziert werden [26].

Durch Anwendung dieser Reduktionsregeln und der gezeigten Kombinatoren, ist die Ableitung und Konstruktion beliebiger Terme mittels der kombinatorischen Logik möglich [26]. Die kombinatorische Logik ist demnach ein ausdrucksstarkes, wohldefiniertes Werkzeug zur Komposition mathematischer Ausdrücke. Im folgenden Abschnitt werden nun die notwendigen Erweiterungen dieser theoretischen Grundlage erläutert, die zur praktischen Anwendung der kombinatorischen Logik im Hinblick auf die Synthese von Software oder Planstrukturen notwendig sind.

3.3 Getypte Kombinatorische Logik mit Intersection Types

Die kombinatorische Logik sowie die dazu korrespondierenden Kalküle wie das SK-Kalkül sind mathematisch wohldefinierte Werkzeuge, um komponentenbasierte Kompositionen durchzuführen. Damit diese aber zur Synthese verwendet werden können, muss das Konzept noch um Typen erweitert werden. Typen waren bereits Teil der ersten höheren Programmiersprachen wie *Fortran*. Ein Typ ist eine Klassifizierung von Daten, durch die der Entwickler dem Compiler die beabsichtigte Verwendung von Daten und Variablen mitteilen kann. Ein Datentyp schränkt dabei Werte ein, die ein Ausdruck (beispielsweise eine Variable oder eine Funktion) annehmen darf. Dadurch werden in der Regel auch die Operationen definiert, die mit den Daten durchgeführt werden können. Ebenso wird die Bedeutung der Daten und die Art und Weise, wie Werte dieses Typs gespeichert werden können, klassifiziert und beschrieben. Durch Typen können Programmierfehler schon bei der Kompilierung eines Programms erkannt werden [121, 118].

Seit den Anfängen der höheren Programmiersprachen entwickelten sich die Typsysteme dieser Sprachen stetig weiter. Viele dieser Typsysteme waren jedoch oft inkonsistent, fehlerbehaftet und erforderten einen hohen Aufwand bei der Typisierung. Daraus resultierten Forschungen, die Typsysteme auf theoretischer Basis untersuchten und z.B. Zusammenhänge zwischen Typsystemen und der intuitionistischen Logik erkannten. Es zeigte sich die Notwendigkeit von Logiken zur Spezifikation von exakten und konsistenten Typsystemen

[121]. Durch die Typisierung von Variablen ist zudem auch eine genauere Spezifikation von Axiomen und Kombinatoren im Kontext der kombinatorischen Logik möglich. Durch Angabe von Typen ist nicht nur auszuweisen, wie viele Argumente ein Axiom benötigt, sondern es ist auch eine genaue Angabe über die Art des Arguments möglich. Zum Beispiel lassen sich unterschiedliche Axiome für unterschiedliche Arten von Zahlen (ganze Zahlen, natürliche Zahlen) formulieren.

In diesem Abschnitt wird das Konzept der Typen in der Kombinatorischen Logik vorgestellt. Diese bildet mit einigen Erweiterungen wie dem Subtyping und den Intersektionstypen die Grundlage für die Spezifikation und Synthese von getypten Komponenten im (CL)S-Framework. Dazu sind zunächst drei Definitionen notwendig.

Definition 3.9

Simple Typen: *Sei $\alpha \in P$ und P eine nicht-leere Menge an grundlegenden Typen und $\alpha \in P$. Außerdem ist der Typkonstruktor $\rightarrow$ eine binäre Funktion auf Typen. Die Menge der Typen wird induktiv definiert durch:*

1. *Wenn $\alpha \in P$ gilt, dann ist α ein Typ.*
2. *Wenn σ und τ Typen sind, dann ist auch der Ausdruck $(\sigma \rightarrow \tau)$ ein Typ.*

Ein grundlegender Typ aus der Menge P beschreibt dabei Typen, die Mengen von Elementen mit bestimmten Eigenschaften repräsentieren. Ein Beispiel stellt der grundlegende Typ N dar, welcher die Menge der natürlichen Zahlen angibt. H wiederum steht für die Menge der aussagenlogischen Variablen. Der Ausdruck $(\sigma \rightarrow \tau)$ formalisiert, dass eine Funktion einen Typen σ als Argument akzeptiert und auf einen Typen τ abbildet [65].

Definition 3.10

Getypte Terme: *Die Menge der getypten Terme wird induktiv definiert:*

1. *Wenn M ein atomarer Term und τ ein Typ ist, dann ist $M : \tau$ ein getypter Term.*
2. *Wenn $M : \tau \rightarrow \sigma$ und $N : \tau$ getypte Terme sind, dann ist $(M : \tau \rightarrow \sigma\ N : \tau) : \sigma$ ein getypter Term.*

In der kombinatorischen Logik mit simplen Typen ist jedem Term exakt ein Typ zugeordnet. In der kombinatorischen Logik mit Intersektionstypen können Terme auch einer Intersektion von Typen zugeordnet sein [22].

Definition 3.11

Intersektionstypen: *Sei $\alpha \in P$ und P eine nicht-leere Menge an Typvariablen, ω eine Typkonstante und $\rightarrow$ und $\cap$ Typkonstruktoren, dann ist τ ein Intersektionstyp und wie folgt definiert:* $\sigma,\ \tau ::= \omega \mid \alpha \mid (\tau \rightarrow \sigma) \mid (\tau \cap \sigma)$

Intersektionstypen erlauben einfache Spezifikationen von Typen, die mit simplen Typen nicht möglich wären. Beispielsweise lässt sich der Typ Funktion f mit $f : D \rightarrow D$ durch den Intersektionstypen $\tau_f = \cap_{d \in D} d \rightarrow f(d)$ repräsentieren [118].

3.3.1 Subtyping

Im Rahmen der Typtheorie stellt das Subtyping (auch Subtyp-Polymorphismus) eine spezielle Form des Polymorphismus[3] dar. Ein Subtyp ist ein Datentyp, der eine oft spezialisierte Form eines anderen, übergeordneten Datentyps darstellt. In der Regel zeichnen sich Typen, die zueinander in einer Subtyp-Relation stehen, durch eine Form der Sustituierbarkeit aus[120]. Ein klassisches Beispiel hierfür ist die Typisierung von Zahlenwerten. Erlaubt man Ganzzahlenwerte überall dort, wo Fließkommazahlen erwartet werden, gilt, dass Integer (Ganzzahlwerte) ein Subtyp von Float (Fließkommazahlen) ist.

Alternativ lässt sich in diesem Beispiel auch ein generischer Typ als gemeinsamer Supertyp (Zahl) von Ganzzahlen und den Realwerten definieren. In diesem zweiten Fall gilt, dass Integer und Float jeweils Subtypen des Typs Zahl darstellen, aber keine Subtypen voneinander sind.

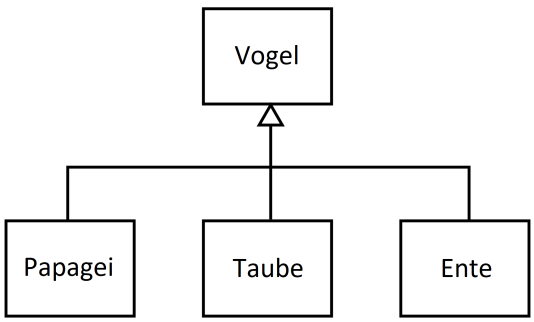


Abbildung 3.1: Beispiel für eine Taxonomie

Durch diese Ausdrucksfähigkeit von Beziehungen erhöht sich die Ausdrucksstärke von Typsystemen ungemein [117]. Auf diese Weise ist es zum Beispiel möglich, taxonomische Zusammenhänge sowie Vererbung mithilfe eines Typsystems zu modellieren. Ein Beispiel

[3] **Polymorphismus (Typtheorie):** Ermöglicht einem Typsystem die Verwendung ein und derselben Typbezeichnug / -Signatur für Entitäten verschiedener Typen oder die Verwendung eines einzigen Symbols zur Darstellung mehrerer verschiedener Typen [32].

dafür ist in Abbildung 3.1 zu sehen. Der Typ „Vogel“ hat drei Subtypen „Ente“, „Taube“ und „Papagei“. Konzeptionell ist jeder von ihnen eine Variante des Grundtyps „Vogel“, der viele „Vogel“-Merkmale erbt, aber einige spezifische Unterschiede aufweist. Innerhalb des Typsystems kann jedoch jeder Subtyp durch den Supertypen „Vogel“ substituiert werden. Im Diagramm in Abbildung 3.1 wird die UML-Notation verwendet, wobei die Pfeile mit offenem Kopf die Richtung und Art der Beziehung zwischen dem Supertyp und seinen Subtypen anzeigen.

Durch das Hinzuziehen von Intersektionstypen wird das Subtyping in Typsystemen wiederum komplexer, da mehrere Subtypbeziehungen definiert werden müssen. Durch die Einführung der reflexiven und transitiven Relation $\leq$ auf Typen ist das Subtyping wie folgt definiert:

Definition 3.12

Axiome des Subtypings mit Intersektionstypen:

- $\sigma \leq \omega$, $\omega \leq \omega \rightarrow \omega$
  Die Typkonstante ω ist allgemeiner Supertyp.
- $\sigma \cap \tau \leq \sigma$, $\sigma \cap \tau \leq \tau$
  Eine Intersektion von zwei Typen ist Subtyp beider Einzeltypen.
- $\sigma \leq \sigma \cap \sigma$
  Jeder Typ ist Subtyp (oder gleich) der Intersektion des jeweiligen Typs mit sich selbst.

  Außerdem gilt:
- $(\sigma \rightarrow \tau) \cap (\sigma \rightarrow \phi) \leq \sigma \rightarrow \tau \cap \phi$
- $\sigma \leq \sigma' \wedge \tau \leq \tau' \Rightarrow \sigma \cap \tau \leq \sigma' \cap \tau'$
- $\sigma \leq \sigma' \wedge \tau \leq \tau' \Rightarrow \sigma' \rightarrow \tau \leq \sigma \rightarrow \tau'$

Stehen zwei Typen in einer Beziehung zueinander, in der beide jeweils Subtyp des anderen sind ($\sigma \leq \tau$ und $\tau \leq \sigma$), gelten beide Typen als gleich. Aus den Axiomen des Subtypings in Definition 3.12 können darüberhinaus folgende Eigenschaften abgeleitet werden [117].

Definition 3.13

Eigenschaften des Subtypings mit Intersektionstypen:

- $(\sigma \rightarrow \tau) \cap (\sigma \rightarrow \phi) = \sigma \rightarrow (\tau \cap \phi)$
- $(\sigma \rightarrow \tau) \cap (\sigma' \rightarrow \tau') \leq (\sigma \cap \sigma') \rightarrow (\tau \cap \tau')$

Durch Subtyping und Intersektionstypen ist das Typsystem der kombinatorischen Logik mit Intersektionstypen sehr ausdrucksstark und bietet vielfältige Möglichkeiten der Spezifizierung von Daten und Komponenten. Diese werden im (CL)S zur Inhabitation von Typen benutzt, welche im nächsten Abschnitt thematisiert wird.

3.3.2 Inhabitation

Eine zentrale Fragestellung der Softwaresynthese stellt die sogenannte Inhabitationsfrage dar, welche ein Entscheidungsproblem in der kombinatorischen Logik ist. Inhabitation stellt die Frage, ob sich aus einer gegebenen Menge an (getypten) Komponenten (oder Kombinatoren) Γ ein Term $\mathfrak{e}$ mit dem gewünschten Zieltypen τ erzeugen lässt ($\exists \mathfrak{e}.\Gamma \vdash \mathfrak{e} : \tau$). $\mathfrak{e}$ ist in diesem Fall der so genannte *Inhabitant* [118]. Sofern die Menge der Kombinatoren in Γ nicht fest ist, wird von einer relativisierten Inhabitation gesprochen. Wird die Basis Γ durch propositionale Logik gemäß dem Curry-Howard-Isomorphismus[4] formalisiert, ist die Inhabitation mit dem Beweisbarkeitsproblem dieser Logik gleichzusetzen. Ein Inhabitant stellt in diesem Fall einen konkreten Beweis dar [118]. Dies gilt vor allem für das in Kapitel 3.2 vorgestellte SK-Kalkül, welches auf einer Basis mit den Kombinatoren S und K aufsetzt. Mit dieser Basis kann ein System gemäß der Propositionen-als-Typen Korrespondenz formuliert werden, welches zu einer propositionalen, intuition-istischen (minimalen) Logik im Hilbert-Stil korrespondiert [118]. Diese Logik basiert auf zwei Prinzipien der Deduktion, Modus Ponens und der Substitution von Formeln durch festgelegte Axiome. Nach dem Statmans Theorem ist die Beweisbarkeit in einer propositionalen, intuitionistischen Logik PSPACE-vollständig [118]. Gemäß dem Curry-Howard-Isomorphismus ist das Inhabitationsproblem im λ-Kalkül mit simplen Typen ebenfalls PSPACE-vollständig. Aufgrund der Äquivalenz des λ-Kalkül mit simplen Typen und des SK-Kalküls, gilt dies auch für das Inhabitationsproblem des SK-Kalküls.

Wie in Kapitel 3.2.1 bereits angedeutet, existieren eine Reihe Algorithmen und Techniken, die das Inhabitationsproblem auf verschiedenen Typsystemen mit verschiedensten Basen lösen können. Auch im (CL)S Framework ist für die Durchführung der Synthese ein solcher Algorithmus implementiert. Nachdem das vorangegangene Kapitel einen kurzen Überblick über die zugrundeliegende Theorie des Frameworks vermittelt hat, wird nun das Framework selbst mit seinen Funktionalitäten und Möglichkeiten zur Komponenten-Modellierung und Komposition dieser Komponenten vorgestellt. Vor allem das Spezifizieren von Repositories steht dabei im Vordergrund.

[4] Als Curry-Howard-Isomorphismus (auch Curry-Howard-Korrespondenz) bezeichnet man die Interpretation von Typen als logische Aussagen und von Termen eines bestimmten Typs als Beweise der zum Typ gehörenden Aussage; und umgekehrt. Benannt ist er nach den Mathematikern Haskell Brooks Curry und William Alvin Howard.

3.4 Combinatory Logic Synthesizer - (CL)S

Beim Combinatory Logic Synthesizer ((CL)S) handelt es sich um ein typbasiertes Synthetisierungs-Framework für komponentenbasierte Repositories. Nach der Klassifikation von Schaefer et al. [126] eignet es sich insbesondere zur Handhabe von unvorhersehbaren Variabilitäten, was es besonders geeignet für die Synthese von Planungsworkflows im Kontext der Fabrikanpassung macht. Im Vergleich zu ähnlich operierenden Frameworks wie *Google Guice* [150] ist es darüber hinaus besonders ausdrucksstark, da es die Verwendung von Annotationen, Intersektionstypen und Typvariablen unterstützt. Der Kern des Synthese-Algorithmus' basiert auf dem Inhabitations-Verfahren mittels kombinatorischer Logik zur Softwaresynthese, welches in Kapitel 3.3.2 erläutert wurde.

Einem bestehenden (Typ-)System können neue Variabilitätspunkte in Form neuer Komponenten und Typen jederzeit hinzugefügt werden, wobei der vorherige Lösungsraum erhalten bleibt. Im Extremfall kann jedes bereits existierende System ohne Variabilitätspunkte als eine einzige Komponente enthalten sein, die nur die Implementierung dieses Systems darstellt. Die Verwendung von Intersektionstypen (Kapitel 3.3 und [43]) ermöglicht die Kombination von Problem- (Spezifikation des zu inhabitierenden Zieltyps) und Lösungsräumen (Spezifikation der Komponenten) innerhalb derselben typbasierten Beschreibung. Dadurch wird die Formulierung einer nachvollziehbaren Abbildung zwischen den beiden Lösungsräumen stark erleichtert [15]. (CL)S ermöglicht somit die Spezifizierung von Komponeten, deren Implementierung sowie die Modellierung von Variabilität und die automatische Komposition von Komponenten unter Beachtung der modellierten Variabilitätsregeln. All dies geschieht innerhalb des Frameworks auf eine einheitliche Weise.

Vergleichbar der constraintbasierten Variabilitätsmodellierung [75, 140] stellt (CL)S die Variabilität dar, indem es wiederverwendbare Artefakte individuell klassifiziert. Durch Subtyping und die semantische Beschreibung von Komponeten mittels der Intersektionstypen, entsteht eine domänenspezifische Taxonomie, die diese Klassifzierung vornimmt. Anstatt boolesche [45] oder temporale Logik [141] für die Klassifizierung zu verwenden, verlässt sich (CL)S auf Typinformationen, die oft bereits als API-Spezifikation vorhanden sind. Nach [43] sind Typinformationen aussagekräftiger als eine einfache taxonomische oder boolesche Logikklassifikation und stellen per se schon eine Turing-vollständige, logische Programmiersprache dar.

Die theoretische Grundlage des Frameworks für die Synthese bildet kombinatorische Logik mit Intersektionstypen [20]. Das Framework wurde so konzipiert, dass es die Ausführung der Inhabitanten ermöglicht. Dadurch ergibt sich die Anforderung, dass jeder

Inhabitant ein syntaktisch korrektes Programm der nativen (Ziel-) Programmiersprache darstellen muss. Die native Programmiersprache ergibt sich durch die verwendete Sprache, die für die Implementierung der Komponenten verwendet wurde. Diese ist dabei allerdings nicht an die Implementierungssprache des Frameworks (SCALA) gebunden. Vielmehr können die Komponenten Codefragmente in jeder beliebigen Zielsprache in Form sogenannter *String Templates* enthalten, also Textbausteine, aus denen nach der Inhabitation ausführbarer Programmcode erzeugt wird.

Jede Komponente, die innerhalb des Frameworks definiert wird, verfügt über zwei Typen: Zum einen dem *nativen Typen*, der dem Typ der Implementierung der Komponente in der nativen Sprache entspricht (z.B. ein String oder Integer). Zum anderen kann noch ein weiterer Typ zur Darstellung von konzeptionellen oder semantischen Strukturen vergeben werden. Dieser sogenannte *semantische Typ* spezifiziert die Bedeutung einer Komponente innerhalb des Gesamtsystems. So zeigt das Beispiel $x : int \cap Kontostand$ eine Variable x mit dem nativen Typen int, was impliziert, dass x eine Zahl beinhaltet. Durch den zusätzlichen semantischen Typen $Kontostand$ wird zudem deutlich, dass der Zahlenwert, der in x gespeichert wird, einen Kontostand repräsentiert. Intersektionstypen dienen folglich der Verknüpfung der Informationen des nativen und des semantischen Typs.

3.4.1 Spezifikation von Komponenten und Repositories

(CL)S nutzt für die Synthese eine strukturierte Sammlung an Komponenten sowie einen Zieltypen. Im Kontext des (CL)S werden die Komponenten als *Kombinatoren* bezeichnet. Um Komponenten für die Synthese nutzbar zu machen, müssen diese entsprechend der Vorgaben des Frameworks spezifiziert werden. Dies wird in den folgenden Abschnitten an einem Beispiel erläutert.

Im Beispiel soll ein Programm synthetisiert werden, das zur Berechnung von Führungsgebühren und Zinsen auf einem Bankkonto genutzt werden kann. Es existieren verschiedene Kontomodelle, die sich durch Merkmale wie den Zinssatz, die Gebühren, die Einräumung eines Dispo-Kredits oder die Limitierung von Transaktionen pro Tag unterscheiden (siehe Tabelle 3.3). Um diese Kontovarianten mit dem (CL)S generieren zu können, müssen diese zunächst in Komponenten zerlegt werden. Die einzelnen Merkmale stellen dabei die Bestandteile (also die Komponenten) eines Kontos dar.

Es gilt die Annahme, dass jedes Kontomodell aus den Komponenten *Zinssatz*, *Kontoführungsgebühren*, *Einräumung Dispokredit* und *Transaktionslimit* bestehen kann, wobei nicht jedes Modell über alle Komponenten verfügen muss. Dies wird im (CL)S durch Signaturen in Form von Funktionen dargestellt. Die Signaturen der einzelnen Kontomodelle

sind in Abbildung 3.2 dargestellt.

	Studentenkonto	Standard-Konto	Premium-Konto
Zinssatz	0,01% p.A.	0,5% p.A.	1,5% p.A.
Kontoführungsgebühren	Nein	30 Euro / Jahr	60 Euro / Jahr
Einräumung Dispokredit	Nein	bis max. 1000 Euro	bis max. 10.000 Euro
Transaktionslimit	bis max. 1.000 Euro / Tag	bis max. 5.000 Euro / Tag	Nein

Tabelle 3.3: Konto-Modelle

Im Beispiel stellt der letzte Parameter den nativen (String) und den semantischen (Konto(Student)) Typ der Komponente dar. Bei den vorangestellten Parametern handelt es sich um Eingabeargumente der jeweiligen Kombinatoren. Das bedeutet, dass für die Inhabitation einer Komponente vom Typen $String \cap Studentenkonto$ zwingend auch Komponenten mit den semantischen Typen $Zinssatz(0.01)$ und $Transaktionslimit(1000)$ benutzt werden müssen. Dies kann durch Verwendung von Kombinatoren mit entsprechenden Typsignaturen geschehen. In diesem Fall würde das System folglich auf die Kombinatoren **WaehleTransaktionslimit** und **WaehleZinsatz** zurückgreifen, um die Parameter des Studentenkonto-Kombinators zu belegen. Das Beispiel zeigt auch, dass durch den Einsatz von Variablen Komponenten individuell spezifiziert werden können.

Konto: String ∩ Konto(a)

Studentenkonto: (Float → Int → String) ∩ (Zinssatz(0.01) → Transaktionslimit(1000) → Studentenkonto)

Standardkonto: (Float → Int → Int → Int → String) ∩ (Zinssatz(0.5) → Gebuehren(30) → Dispo(1000) → Transaktionslimit(5000) → Standardkonto)

Standardkonto: (Float → Int → Int → String) ∩ (Zinssatz (1.5) → Gebuehren(60) → Dispo(10000) → Premiumkonto)

WaehleZinssatz: Float ∩ Zinssatz(a)

WaehleTransaktionslimit: Int ∩ Transaktionslimit (b)

WaehleGebuehren: Int ∩ Gebuehren(c)

WahleDispo: Int ∩ Dispo(d)

WF: {
{a → 0.01}, {a → 0.5}, {a → 1.5}, {b → 1000}, {b → 5000}, {c → 30}, {c → 60}, {d → 1000}, {d → 10000}
}

Abbildung 3.2: Beispiel Kombinator Repository

Wie bereits erwähnt, existiert eine ausgereifte Implementierung des Frameworks in der Programmiersprache *SCALA*[5].

```
1  @combinator object Studentenkonto {
2      def apply(Zins: Float, Limit: Int) : String =
3      "Stringtemplate ..."
4
5      val semanticType : Type = 'Zinssatz(0.01) =>: 'Transaktionslimit =>: 'Studentenkonto
6  }
```

Abbildung 3.3: Beispiel einer Implementierung einer Komponente in (CL)S

Abbildung 3.3 zeigt, wie ein Kombinator innerhalb des Frameworks in SCALA implementiert wird. Die Annotation `@combinator` kennzeichnet eine Funktion als Kombinator, sodass diese später bei der Inhabitation verwendet werden können. Der Aufbau folgt dabei einem festen Schema: In der `apply`-Methode werden die Parameter der Komponente inklusive ihrer (nativen) Typen definiert. Direkt dahinter erfolgt die Definition des nativen Zieltyps der jeweiligen Komponente. Dabei entspricht Codezeile 2 im wesentlichen dem linken Teil der Signatur aus Abbildung 3.2. Es folgt ein Block, in dem beliebig zu synthetisierender und zu generierender Programmcode formuliert werden kann. Dies kann in Form von String-Templates geschehen, aber auch in Form konkreter Berechnungen. Die Codebausteine aus anderen Komponenten können über die in Zeile 2 definierten Parameter verwendet werden. Am Ende werden die semantischen Typen sowohl der Parameter als auch der Komponente selbst definiert. Es können Teile der Definition durch die Verwendung von Variablen noch offen gelassen, aber auch für die jeweilige Komponente bereits fest spezifiziert werden. In diesem Fall wird für das Beispiel ein spezifischer Zinssatz, der zur Komponente passt, vorgeschrieben.

```
1  val semanticTaxonomy = Taxonomy("Konto")
2          .addSubtype("Studentenkonto")
3          .addSubtype("Standardkonto")
4          .addSubtype("Preiumkonto")
```

Abbildung 3.4: Taxonomie

Die einzelnen Kontomodelle sind spezialisierte Formen des „normalen“ Kontos. Diese Beziehung ist durch Subtyping in einer Taxonomie repräsentiert. In Abbildung 3.4 wird die Taxonomie für das Beispiel implementiert. Durch den Aufruf von `Taxonomy` in der ersten Zeile mit dem Zieltypen `Konto` wird dieser als Supertyp markiert. Der Aufruf der Methode

[5] **SCALA**: Funktionale, objektorientierte Programmiersprache, die im Jahr 2004 unter anderem für die JAVA-Plattform veröffentlicht wurde. Der Name leitet sich vom englischen Ausdruck *„scalable language“* (skalierbare Sprache) ab. Da es explizit für die JAVA-Plattform veröffentlicht wurde, ist SCALA auch sehr eng mit JAVA verbunden. Es können nahezu alle bestehenden Java-Bibliotheken, -Frameworks und Werkzeuge wie Entwicklungsumgebungen auch mit SCALA genutzt werden. Anders als JAVA, ist SCALA allerdings eine rein objektorientierte Sprache[162, 19]

`addSubtype` in der zweiten und dritten Zeile stellt die Subtypen Beziehung zwischen dem Supertypen `Konto` und `Studentenkonto` sowie `Standardkonto` und `Premiumkonto` her.

Die einzelnen Komponenten werden in einer Klasse namens `Repository` implementiert. Diese Klasse bildet die Komponentenbibliothek, die bei der Synthese verwendet wird. Es sei zudem erwähnt, dass das Framework die Substitution von Variablen unterstützt. So kann im semantischen Typ einer Komponente statt eines konkreten Typs auch eine Variable angegeben werden, die (CL)S anhand einer Substitionsmap (WF) ersetzt. Dies ist beispielsweise beim Zinssatz der Fall, der für ein generelles Konto variabel gestaltet ist, bei konkreten Derivaten aber durch feste Werte substituiert wird.

3.4.2 Inhabitation

Mit den spezifizierten und implementierten Komponenten kann (CL)S Kontomodelle synthetisieren. Hierfür muss zunächst ein Zieltyp festgelegt werden. Da die Komponente mit dem Zieltyp `Konto` alle weiteren Komponenten umfasst, wird dieser Typ als Zieltyp der Synthese ausgewählt. Sollen nur spezielle Kontomodelle erzeugt werden, ließe sich dies durch Angabe eines spezielleren Zieltyps in Zeile 3 in Abbildung 3.5 durchführen.

```
1  lazy val Gamma = new Repository {}
2  lazy val reflectedGamma = ReflectedRepository(Gamma, semanticTaxonomy)
3  lazy val inhabitationResult = reflectedGamma.inhabit[String]('Konto)
```

Abbildung 3.5: Start der Synthese im (CL)S Framework

Abbildung 3.5 zeigt den Quellcode für die Durchführung einer Synthese mit dem Framework (CL)S. Zunächst wird in der Variable `Gamma` die Klasse Repository instanziiert, welche die Komponentenbibliothek enthält. In der zweiten Zeile wird ein sogenanntes ReflectedRepository erzeugt, welches die definierte Taxonomie auf die Komponentensammlung anwendet. In diesem Schritt werden die Komponenten, die in der Komponentenbibliothek Gamma implementiert sind, durch Reflexion ermittelt. Anschließend wird die Synthese durch den Aufruf der Methode `inhabit` gestartet. Eine genaue Beschreibung des Inhabitationsalgorithmus und seiner Implementierung im (CL)S findet sich in der Dissertation von Jan Bessai [19] und ist nicht Gegenstand dieser Arbeit. Das Ergebnis der Synthese ist eine kontextfreie Grammatik, die vom Framework (CL)S für die Erzeugung von Inhabitanten verwendet wird.

Die Grammatik besteht aus einer Menge von Schlüssel-Wert-Paaren. Die Schlüssel repräsentieren die Intersektionstypen der Komponentenbibliothek. Der zugehörige Wert ist eine Menge von Komponenten, deren Zieltypen einem Intersektionstypen aus dem

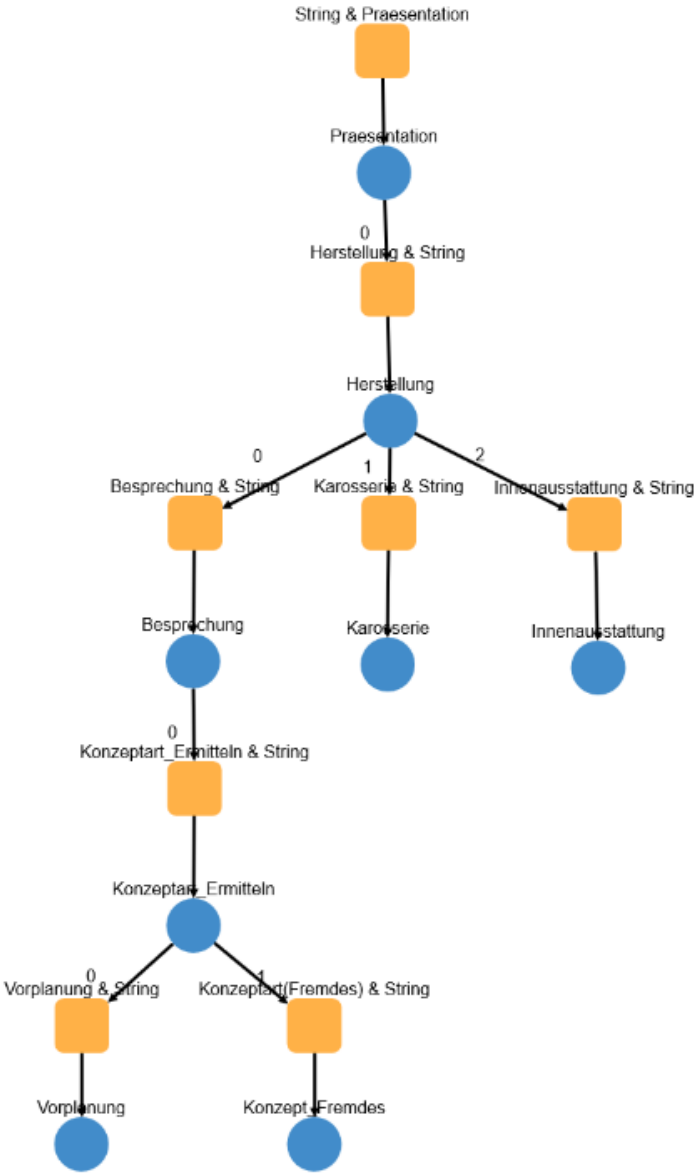


Abbildung 3.6: Beispiel eines Inhabitationsbaumes

Schlüssel entsprechen. Aufgrund des Subtypings entspricht der Typ aus dem Schlüssel nicht immer dem Zieltypen, welcher in den Komponenten spezifiziert wurde. So sind die Subtyp-Komponenten ihrem Supertypen zugewiesen. Weiterhin verfügt jede Komponente über eine Liste von Typen der Argumente, die bei der Synthese der Komponente ebenfalls synthetisiert werden müssen.

Konkrete Inhabitanten stellen immer einzelne Wörter dieser Grammatik dar. Inhabitanten sind *Sequenzen* von Aufrufen der verwendeten Komponenten, die den jeweiligen Zieltypen erfüllen. Zwischen den Argumenten einer einzelnen Komponente besteht aller-

dings kein temporaler Zusammenhang. Eine Typsignatur $a \to b \to c$ besagt nur, dass Komponenten vom Typen a und b vorliegen müssen um c verwenden zu können. Betrachtet man die Signatur also als Graphen mit temporaler Ordnung, lässt sich diese auch als Baum mit c als Wurzel sowie a und b als Kindknoten betrachten. Selbiges gilt für die gesamten Sequenzen, die das (CL)S erzeugt. Inhabitanten lassen sich somit auch als sogenannte *Inhabitationsbäume* in einer Baum-Struktur darstellen (siehe Abbildung 3.6). Ein Knoten im Baum repräsentiert dabei stets eine verwendete Komponente. Der Wurzelknoten entspricht der Komponente mit dem Zieltyp der Inhabitation. Die Liste seiner Kindknoten stellt die Liste der Komponenten dar, welche die Argumente der jeweiligen Komponente bedienen. Das Framework synthetisiert dabei alle möglichen Varianten von Kompositionen, die aus dem gegebenen Repository erzeugbar sind und die den angegebenen Zieltypen erfüllen [19]. Entscheidet sich der Nutzer für eine (oder auch mehrere) Variante, wird aus dem Code in den Stringtemplates der einzelnen Komponenten (entsprechend der Kompositionsstruktur des Inhabitationsbaums) ausführbarer Programmcode erzeugt. Der Codegenerator ist dabei fest in das Framework integriert und muss nicht manuell vom Nutzer angestoßen werden [19].

3.4.3 Einsetzbarkeit des (CL)S im Kontext der Plan-Synthese

Aus dem vorgestellten Beispiel ist zu schließen, dass das (CL)S Framework sich sehr gut für die komponentenbasierte Komposition eignet. Es ist in der Lage, Komponenten unter Beachtung ihrer Abhängigkeiten gemäß einer Zielspezifikation zu komponieren. Es liefert darüberhinaus die gesamte Bandbreite an möglichen Varianten dieser Komposition. Bezogen auf die Erzeugung von Ablaufplänen zeigt sich (CL)S insbesondere für die Komposition von Planungsworkflows aus zuvor definierten Modulen geeignet. Module (wie jene nach [109]) lassen sich durch Intersektionstypen gemäß ihrer Ausgabeinformationen spezifizieren und unter Angabe eines Planungsziels (dem Zieltyp) zusammensetzen. Einschränkungen und Abhängigkeiten der Module werden beim Kompositionsprozess berücksichtigt. Durch Abdeckung des gesamten Variabilitätsbereichs, können dem Planer darüberhinaus auch sämtliche zur Verfügung stehende Handlungsoptionen aufgezeigt werden. Das (CL)S kann somit zur Entscheidungsfindung im Anpassungsprozess beitragen.

Allerdings finden sich bezogen auf Fabrikanpassungsprojekte noch weitere Einschränkungen, die sich teils auf einzelne Module, teils auf das Gesamtprojekt beziehen. Dazu gehören beispielsweise Budget- oder Ressourcen-Limitierungen. Da (CL)S sämtliche Spezifikationen auf der Ebene der Komponenten betrachtet, finden solche globalen Einschränkungen beim Synthese-Prozess keine Beachtung. Um deren Einhaltung sicherzustellen, sind weitere Techniken hinzuzuziehen. Diese werden im folgenden Kapitel behandelt.

3.5 SMT Constraint-Solving

Neben der reinen Zusammensetzung von Planungsprozessen soll die hier entwickelte Methodik auch dazu dienen, dass die generierten Lösungen für die Rahmenbedingungen des jeweiligen Planungsfalls geeignet sind. Ein Prozess muss daher bestimmte Auflagen erfüllen, weshalb Techniken benötigt werden, die sicherstellen, dass Prozesse und Lösungen diese Auflagen einhalten. Dieses Kapitel gibt einen Überblick über solche Techniken und Strategien zur Findung von Lösungen unter Berücksichtigung sogenannter Constraints.

Constraints können als Zwänge, Nebenbedingungen oder Einschränkungen auf Wertebereichen verstanden werden. Im jeweiligen Anwendungskontext definieren sie Bedingungen auf Objekten oder Beziehungen zwischen ihnen. Mit ihnen können die Eigenschaften von Problemen und deren Lösungen beschrieben werden. Sie werden dabei als Gleichungen, boolesche Werte oder eine beliebige Menge von Symbolen modelliert. Nachdem ein Problem durch eine Menge an Constraints beschrieben ist kann herausgefunden werden, ob eine Zielfunktion unter den Constraints erfüllbar ist und wie eine solche Lösung zu gestalten ist [67].

Constraint-Systeme beschreiben Syntax und Semantik ihrer jeweiligen Constraints. Über sie kann eine Auswahl der verwendeten Lösungsverfahren und Heuristiken getroffen werden. Beispiele für solche Systeme sind unter anderem:

- **Lineare Gleichungssysteme:** Sie dienen zur Behandlung arithmetischer Ausdrücke und Relationen über rationale Zahlen.
- **Lineare Optimierung**
- **Boolesche Constraints:** Die Variablen stellen die Gültigkeit einer Aussage binär dar.
- **Finite-Domain-Constraints:** Erzwingt, dass den Variablen immer endliche Wertebereiche zugeordnet sind [67].

In vielen Bereichen der Informatik lassen sich wichtige Probleme auf Formeln einer bestimmten Logik und die Frage nach ihrer Erfüllbarkeit reduzieren. Einige dieser Probleme können beispielsweise in Aussagenlogik formuliert und mit modernen SAT-Solvern auf ihre Erfüllbarkeit getestet werden. Andererseits können andere Probleme in der klassischen Logik intuitiver und kompakter gestaltet werden. Dazu gehört z. B. die Prädikatenlogik, da diese durch Verwendung von nicht-booleschen Variablen, Funktionen, Prädikaten (mit positiver Arität) und Quantifizierern ausdrucksvoller ist als die Aussagenlogik. Dies führt jedoch zu einem Konflikt zwischen der Ausdruckskraft einer Logik und der Fähigkeit, die

Erfüllbarkeit von Formeln in dieser Logik automatisch zu bestimmen.

Als ein praktischer Kompromiss werden Fragmente der Prädikatenlogik 1. Stufe entweder syntaktisch eingeschränkt (in dem zum Beispiel nur bestimmte Klassen von Formeln zulässig sind) oder semantisch eingeschränkt (durch Einschränkung der Interpretation bestimmter Funktionen oder Prädikate). Eine Kombination aus syntaktischen und semantischen Einschränkungen ist ebenfalls möglich. Durch diese Einschränkungen wird das Erfüllbarkeitsproblem in der Prädikatenlogik entscheidbar. Kern der Fragestellung dieses Problems ist es, für eine konkret vorgegebene Fragestellung eine Lösung zu finden, die alle Vorbedingungen dieser Fragestellung erfüllt. *Constraint Satisfaction Programming (CSP)* kommt vor allem in Bereichen des Operation Research, allerdings auch in der künstlichen Intelligenz, zum Einsatz. Die Lösung stellt in der Regel eine konkrete Belegung von Zielvariablen dar. Als Constraint werden die Bedingungen und Einschränkungen bezeichnet, die für diese Variablen gelten. Dies können zum Beispiel vorgegebene Wertebereiche, allerdings auch Beziehungen zu anderen Variablen sein. So sei exemplarisch zu zwei Zielvariablen x und y die Constraintmenge $\{x + y = 5, x > 1, y > 1, x > y\}$ gegeben. Mit Hilfe von Constraintsolving-Methoden lässt sich daraus die Zielbelegung $x = 3$ und $y = 2$ errechnen, die alle vorgegebenen Bedingungen erfüllt. In diesem Beispiel handelt es sich dabei um die einzige Lösung. In vielen Fällen ist es aber auch denkbar, dass mehrere Lösungen zu einem Problem existieren. Im Gegensatz zu anderen Heuristiken oder Optimierungsverfahren werden außer der Frage, ob die Lösung alle Constraints erfüllt, zunächst keine weiteren Aussagen über die Lösungsqualität getroffen. Es soll nicht eine möglichst gute Lösung gefunden, sondern vielmehr ermittelt werden, ob überhaupt eine Lösung unter den bestehenden Einschränkungen existiert. Ist dies nicht der Fall, sind die Constraints widersprüchlich. Für die rechnergestützte Lösung von Fragestellungen dieser Art steht bereits Software (so genannte SMT- oder Constraintsolver) zur Verfügung. Beispiel sind hier Alt-Ergo, Beaver, Boolector, CVC4, iSAT, MathSAT5, openSMT, SMTInterpol, Sonolar, STP, veriT, Yices und Z3 [11, 50].

Ein *Constraint-Solver* kann, nachdem ein Problem durch Constraints und ihre Systeme beschrieben wurde, herausfinden, ob eine Erfüllbarkeit der Constraint-Konjunktionen existiert und wie eine konkrete Variablenbelegung aussehen kann, um die Gleichungen zu erfüllen. Viele moderne Solver (wie zum Beispiel Microsofts Z3) vereinen verschiedene Logiken und Heuristiken zur Lösungsfindung und wählen je nachdem, wie das Problem formuliert wurde, die entsprechend passende Logik aus [102]. Abhängig von dieser Auswahl ergeben sich teilweise bereits große Performance-Unterschiede darin, wie schnell ein Solver ein bestimmtes Problem lösen kann. Auch können unterschiedliche Solver durch die Auswahl unterschiedlicher Lösungsstrategien ebenfalls eine große Diskrepanz in der Laufzeit zueinander aufweisen. Eine Auswahl des „richtigen“ Solvers fällt demnach schwer. Die

Forschung bewegt sich in diesem Zusammenhang in die Richtung, auch die Auswahl des Solvers zu abstrahieren. [68]. Aus diesem Grund stellt auch der folgende Abschnitt mit *SMT-LIB* eine Sprache vor, die das maschinennahe Codieren von Constraints ermöglicht, und die von fast allen gängigen Solvern interpretiert werden kann [67].

3.5.1 SMT-LIB

Der SMT-LIB Standard beschreibt ein Format zur Formulierung von Entscheidungsproblemen [82] und besteht aus drei Bestandteilen: Theorie-, Logik-Deklarationen sowie Skripte [10]. Die Syntax dieses Formats orientiert sich an der Sprache Common Lisp, da diese nicht nur ein einfaches Parsing erlaubt, sondern auch für den Menschen lesbar ist.

Innerhalb der SMT-LIB Sprache wird darüber hinaus zwischen grundlegenden und kombinierten Theorien unterschieden. Als grundlegende Theorien werden zum Beispiel die Arithmetik oder Array-Theorien aufgefasst. Diese Theorien sind expliziter Bestandteil von SMT-LIB und werden auch von allen Solvern, die den SMT-Standard befolgen, implementiert [10]. Auskunft, ob es sich bei einer Theorie um eine grundlegende Theorie handelt und ob diese Bestandteil des Standards ist, gibt der sogenannte Katalog von SMT-LIB. Zentraler Bestandteil dieses Kataloges ist außerdem die Kern-Theorie, welche die aussagenlogischen Konnektoren (siehe Tabelle 3.4), die Theorie der Gleichheit sowie die Theorie der uninterpretierten Funktionen enthält. Durch sie können Funktionen deklariert werden, deren Interpretation durch Aufstellung von Randbedingungen eingeschränkt und definiert werden kann.

Ein Problem wird in SMT-LIB zunächst durch die Deklaration von Symbolen formuliert. Symbole können dabei entweder Konstanten oder Funktionen sein. Dies geschieht mit einem Aufruf der Form $declare - fun\ f(\sigma 1; \ldots; \sigma n)\sigma)$. Die Symbole $\sigma 1; \ldots; \sigma n$ bezeichnen dabei die Typen der Funktionsargumente, σ den Typen der Rückgabe. Im Kontext von SMT-LIB werden Typen auch als Sorten bezeichnet. Die Deklaration von Konstanten erfolgt entweder analog (wobei allerdings dort die Menge der Funktionsargumente leer gelassen wird) oder aber durch den Aufruf $(declare - const\ f\sigma)$ [82, 10]. Sowohl bei Funktionen, als auch bei Konstanten, die auf diese Weise deklariert werden, handelt es sich um sogenannte uninterpretierte (oder auch freie) Symbole. Das bedeutet, dass für sie keine Interpretation durch eine grundlegende Theorie im Katalog vorgegeben ist. Funktionen, wie zum Beispiel die Addition $(+ab)$, sind durch die Arithmetik bereits interpretiert [99].

Constraints kommen in diesem Zusammenhang bei der Interpretation von uninterpretierten Funktionen ins Spiel. Sie werden durch den Aufruf $assert\ \varphi$ deklariert, wobei φ eine Formel mit einem dezidierten Rückgabetyp darstellt. Zur Konstruktion dieser Formeln

werden sämtliche (durch grundlegende Theorien interpretierte) Funktionen und Formeln sowie die in Tabelle 3.4 dargestellten booleschen Konnektoren genutzt.

Konnektor	Bezeichnung in SMT-LIB
$\neg$	$not\ a$
$\Rightarrow$	$(=>\ a\ b)$
$\bigotimes$	$(xor\ a\ b)$
$\bigvee$	$(or\ a\ b)$
$\bigwedge$	$(and\ a\ b)$

Tabelle 3.4: SMT-LIB Konnektoren

Variablen innerhalb einer Formel φ können zudem noch über All- oder Existenzquantoren gebunden werden. Das unten stehende Beispiel zeigt wie der Allquantor in ein Constraint eingebunden wird.

$$(assert(forall((x_1\sigma_1))(forall((x_2\sigma_2))\dots(forall((x_n\sigma_n))\psi))))$$

Das Beispiel besagt, dass die Formel ψ aus bis zu n Variablen mit jeweils eigener Sorte (bzw. Typen) σ bestehen und gelten kann. ψ steht für eine Formel, welche die Variablen der Quantoren verwendet und einen aussagenlogischen Wert als Rückgabe hat. Die Schreibweise kann durch „Ausklammern“ des $forall$ noch vereinfacht werden ($(assert(forall((x_1\sigma_1)\ (x_2\sigma_2)\dots(x_n\sigma_n n))\psi))$). Für einen Existenzquantor wird analog das Schlüsselwort $exists$ verwendet.

Die Deklaration einer Funktion und die Verwendung des Aufrufs $(define-fun\ f\ ((x_1\sigma_1);\dots;(x_n\sigma_n))\sigma\ \psi)$ können das Hinzufügen von Randbedingungen und die Funktionsdeklaration abkürzen. Die Argumentenliste besteht dabei nicht mehr ausschließlich aus Sorten, sondern aus getypten Variablen. In der Formel wird das Constraint formuliert.

Sind alle gewünschten Funktionen, Konstanten und Constraints deklariert, kann das SMT-Skript an einen SMT-Solver übergeben werden. Dieser konjugiert alle im Skript deklarierten Randbedingungen. Die Konjunktion wird anschließend durch den Solver auf Erfüllbarkeit hin untersucht. Dies geschieht durch den Aufruf **check-sat** im Skript. Ein Solver kann darauf verschiedene Antworten liefern:

1. **sat:** Für das Ergebnis erfüllbarer Formelmengen. In diesem Fall kann durch den Befehl **get-model** ein Modell mit einer gültigen Variablenbelegung zurückgegeben werden.

2. **unsat:** Für das Ergebnis unerfüllbarer Formelmengen.

3. **unknown:** Der Solver findet kein Ergebnis (zum Beispiel durch Abbruch der Berechnung aufgrund einer Überschreitung der maximalen Rechenzeit).

Das folgende Beispiel soll dies noch einmal verdeutlichen: Die Formel $(assert\ (and\ (or\ a\ b)\ (not\ a)))$ ist für die Belegung $a = False$ und $b = True$ offenkundig gültig. Dies lässt sich selbstverständlich auch mit Hilfe eines SMT-Solvers berechnen. Abbildung 3.7 zeigt auf der linken Seite das dazugehörige SMT-Skript. Zunächst werden in den ersten beiden Zeilen die aussagenlogischen Variablen a und b als Konstanten mit dem Typen *bool* deklariert. Durch das Schlüsselwort **assert** werden Constraints hinzugefügt. In diesem Fall besagt die Formel, dass a oder b wahr sein müssen, und a wiederum `false` um die Gesamtformel zu erfüllen. In Zeile 6 und 7 wird der Solver aufgefordert die Erfüllbarkeit der Formel zu überprüfen und (wenn möglich) eine Belegung zu liefern, die die Formel erfüllt. Die Antwort des Solvers findet sich in Abbildung 3.7 auf der rechten Seite und zeigt die naheliegende Belegung $a = False$ und $b = True$.

```
1 (declare-fun a () Bool)
2 (declare-fun b () Bool)
3
4 (assert (and (or a b) (not a)))
5
6 (check-sat)
7 (get-model)
```

```
1 (model
2     (define-fun a () Bool
3         false)
4     (define-fun b () Bool
5         true)
6 )
```

Abbildung 3.7: Beispiel für ein SMT-Skript und Modell

Auf diese Weise lassen sich auch Constraints formulieren, die den Wertebereich von Variablen einschränken. In dem Beispiel in Abbildung 3.8 darf eine Variable nur einen Wert zwischen 4 und 9 annehmen. Dies ist in Zeile 2 des Beispiels formuliert. Für dieses Beispiel antwortet der SMT-Solver mit **sat** und erzeugt ein Modell, welches die erfüllende Belegung $x = 4$ liefert. An dieser Stelle sei noch einmal daran erinnert, dass dies nicht zwangsläufig eine optimale Lösung darstellen muss. Der Solver liefert nur eine beliebige, aber gültige Lösung.

```
1 (declare-const x () Int)
2 (assert (and (=< 4 x) (<= x 9)))
```

Abbildung 3.8: Beispiel für ein SMT-Skript mit einschränkendem Wertebereich

3.5.2 Einsetzbarkeit von SMT-Solving im Bereich der Plan-Generierung

Gerade die zuletzt vorgestellte Funktionalität, bestimmte Zielvorgaben bezüglich der Wertebereiche von Variablen zu machen und diese auf eine Realisierbarkeit hin überprüfen zu können, macht SMT-Solving zu einer sehr nützlichen Technologie im Kontext der Automatisierung von Planung. So ist die Technik gut dafür geeignet, Pläne zu parametrisieren oder

um zu überprüfen, ob Pläne Zielvorgaben einhalten können (was auch bereits in mehreren Veröffentlichungen realisiert wurde [55, 91, 122]). Durch die beliebige Komplexität und die große Freiheit beim Formulieren von Termen, eignet sich SMT-Solving darüberhinaus auch zur Findung von Planstrukturen. Allerdings zeigen Untersuchungen [110, 122], dass gerade bei größeren und komplexeren Plan-Strukturen die Rechenzeit der Solver stark zunimmt. [110, 122] empfehlen daher die Zuhilfenahme weiterer Technologien, um die Solver entlasten zu können.

Insgesamt lässt sich allerdings festhalten, dass SMT-Solving einen wesentlichen Beitrag zur Erfüllung der Zielstellung dieser Arbeit leisten kann.

3.6 Zusammenfassung Synthese

Das Kapitel hat einen Überblick über bestehende Ansätze im Bereich der automatischen Komposition und Synthese von Software und Workflows gegeben. Es wurden das (CL)S-Framework sowie die dem Framework zugrundeliegende Theorie der kombinatorischen Logik ebenso vorgestellt, wie Grundlagen des Constraintsolvings. Die Verwendung von komponentenbasierten Syntheseframeworks (wie dem (CL)S) zur Komposition von Planungsmodulen erscheint naheliegend, bedarf aber den Einsatz von weiteren, ergänzenden Technologien (wie dem SMT-Solving), um die Komplexität von Fabrikplanungsprojekten vollständig abbilden und berücksichtigen zu können. Kombiniert man beide Technologien, lassen sich die in Kapitel 2.5 dargestellten Schritte zur automatischen Komposition von Fabrikplanungsabläufen vollständig abdecken:

1. Aus [109] und den darauf aufbauenden Arbeiten lässt sich eine hinreichend große Anzahl an Modulen oder Komponenten ableiten, die den gesamten Fabrikplanungsverlauf widerspiegeln.

2. Die Schnittstellen der Komponenten aus [109] lassen sich mit Hilfe von Intersektionstypen spezifizieren, wodurch sie die Plaunungsmodule im (CL)S Framework sammeln lassen.

3. Durch die Synthesefähigkeit des (CL)S lassen sich unter Angabe eines Planungsziels in Form eines Zieltyps sämtliche Optionen für einen Ablaufplan komponieren. Das (CL)S ist in der Lage die zur Erfüllung eines Planungsziels zu bewältigenden Aufgaben zu ermitteln und die entsprechenden Komponenten in einer Abfolge zusammenzustellen, die das Planungsziel erfüllt. Durch SMT-Techniken lässt sich die Einhaltung weiterer Einschränkungen durch die Inhabitanten erzwingen.

4. Durch die Codegenerierungsoptionen des (CL)S lassen sich die Inhabitanten in eine Zielsprache überführen, die zur Darstellung von Netzplangraphen geeignet ist.

Das folgende Kapitel stellt eine prototypische Implementierung vor, welche das Zusammenspiel von Synthese und SMT-Solving mit dem Ziel der Plan-Generierung umsetzt. Dabei wird insbesondere auf die Implementierung der Komponentensammlung, die Umsetzung der Netzplantechnik sowie das Verfahren bei der Synthese von Anpassungen an laufenden Plänen eingegangen.

Kapitel 4

Software Architektur

In den vorangegangenen Kapiteln wurden diverse Möglichkeiten zur automatisierten Konfiguration und Komposition von Software, Prozessen oder Planung vorgestellt. Ebenso wurden Ansätze zur Modularisierung von Planung und Planungsansätzen sowie Anforderungen an die Planung von Fabriken aufgezeigt. Mit dem Ziel Möglichkeiten zur Automatisierung von Planung aufzuzeigen, soll in diesem Kapitel eine prototypische Softwarearchitektur erarbeitet werden, die (unter Verwendung der in den vorherigen Kapiteln vorgestellten Ansätzen) eine automatisierte Planung realisieren soll. Geschehen soll dies in Form der automatischen Generierung von Netzplänen.

4.1 Allgemeiner Überblick

Die Architektur besteht aus mehreren Komponenten, die über einen dateibasierten Datenaustausch mittels JSON (siehe Abschnitt 4.2.2) lose miteinander gekoppelt sind[6]. Der Aufbau der Architektur ist Abbildung 4.1 zu entnehmen. Auf die einzelnen Komponenten, ihr Zusammenwirken sowie die Rollen, die diese einzelnen Komponenten einnehmen, wird an dieser Stelle zunächst kurz, im weiteren Verlauf des Kapitels dann detaillierter eingegangen.

Kern der Architektur bildet eine serverbasierte Applikation, die aus den vom Nutzer (dem verantwortlichen Projektplaner) über eine Client-Anwendung bereitgestellten Daten Netzpläne generiert. Diese bestehen aus mehreren Einzelkomponenten, die jeweils eine bestimmte Funktion während der Generierung übernehmen (siehe 4.1).

- **JSON-Parser:** Übersetzt die aus der Client-Software im JSON-Dateiformat an den Server übertragenen Daten in ein für die Folgekomponente verarbeitbares Format.
- **(CL)S:** Das (CL)S ist für die Erzeugung der Planstrukturen verantwortlich. Es nimmt eine Sammlung von Plan-Komponenten entgegen, die durch die Verwendung von Intersection Types getypt sind. Darüber hinaus benötigt das (CL)S noch einen Zieltypen, welcher das Planungsziel repräsentiert. Durch Beantwortung der Inhabitationsfrage generiert das Framework daraus parallelisierbare Sequenzen von Planungsaufgaben, die in einem Planungsworkflow abgearbeitet werden müssen. Diese

[6] **Lose Kopplung:** Bezeichnet einen geringen Abhängigkeitsgrad zwischen mehreren Software-Komponenten. Durch diesen Ansatz lassen sich Änderungen einzelner Komponenten oftmals einfacher durchführen, da diese dann lediglich lokale Auswirkungen mit sich bringen. Dem Gegenüber steht eine enge Kopplung, in der Änderungen nicht lokal bleiben, sondern in der Regel zusätzliche Anpassungen in allen gekoppelten Komponenten nach sich ziehen. Allerdings weisen lose gekoppelte Systeme, welche meistens durch einheitliche Datenschnittstellen oder -formate gekennzeichnet sind, oft einen Nachteil bezüglich der Performance gegenüber eng gekoppelten Systemen auf.

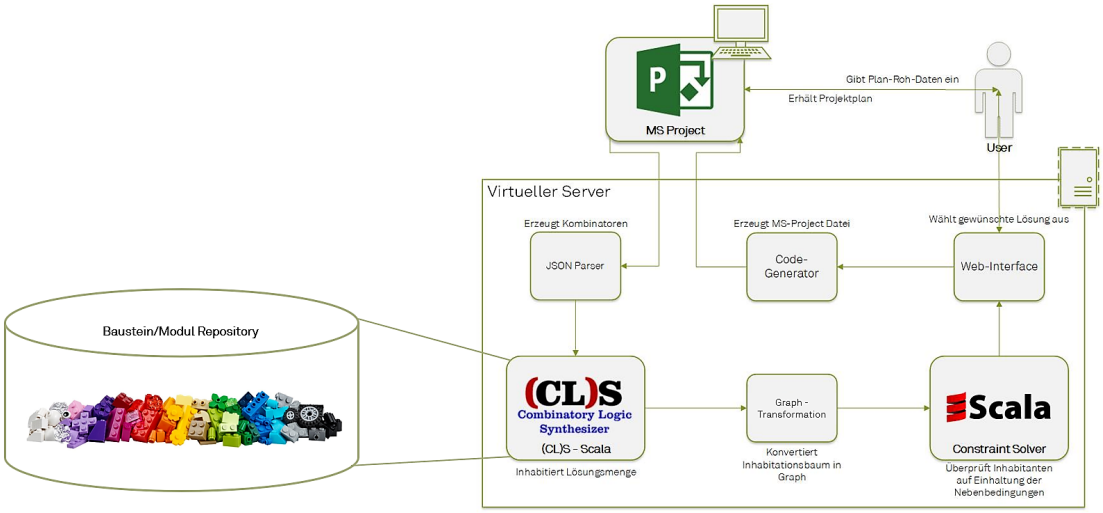


Abbildung 4.1: Schematische Darstellung der Architektur des Softwareprototyps

erfüllen das gewünschte Planungsziel und berücksichtigen die jeweiligen Abhängigkeiten der verwendeten Module (siehe Kapitel 3.4).

  - **Input:** Menge von getypten Kompontenten, Planungsziel als Zieltyp.
  - **Output:** Menge von Sequenzen von Planungsaufgaben, die das Planungsziel erfüllen.

- **Merge-Algorithmus:** Im Merge-Alogirthmus finden eine Reihe von Graphtransformationen statt, welche die generierten Sequenzen des (CL)S in gültige Netzplanstrukturen überführen. Zunächst werden innerhalb der Sequenzen voneinander unabhängige Aufgaben in einzelne Pfade aufgeteilt und parallelisiert. Auf diese Weise entstehen Baumstrukturen. Durch weitere Transformationen, wie dem Hinzufügen eines allgemeinen Startknotens, oder dem Zusammenführen von redundanten Zweigen des Baumes entsteht für jede generierte Lösung des (CL)S ein gerichteter, zyklenfreier Graph. Dieser stellt die maximal parallelisierte Abfolge von Planungsaufgaben zur Erreichung des Planungsziels dar.

  - **Input:** Menge von Sequenzen von Planungsaufgaben, die das Planungsziel erfüllen.
  - **Output:** Menge von gerichteten, zyklenfreien Graphen mit Knoten von maximal parallelisierten Planungsaufgaben, die das Planungsziel erfüllen.

- **SMT Constraint-Solver:** Hier werden die generierten Graphstrukturen auf Einhaltung der durch den Planer vorgegebenen Nebenbedingungen (Constraints) überprüft. Dabei werden Lösungen aussortiert, die zum Beispiel ein vorgegebenes Projekt-Budget oder ein Zeit-Limit überschreiten. Ebenso wird anhand des globalen Modells

überprüft, ob die Lösungen gegebenenfalls Module verwenden, die sich gegenseitig ausschließen.

  - **Input:** Menge von gerichteten, zyklenfreien Graphen, die das Planungsziel erfüllen.
  - **Output:** Menge von gerichteten, zyklenfreien Graphen, die das Planungsziel unter Berücksichtigung der Projekt-Constraints erfüllen.

- **Web-Interface:** Die nach der Überprüfung durch den Constraint-Solver noch gültigen Lösungen werden dem Planer auf einer Web-Oberfläche präsentiert. Der Nutzer hat die Möglichkeit, sich für einen Netzplan zu entscheiden und diesen auszuwählen.

- **Code-Generator:** Die ausgewählte Lösung wird an einen Code-Generator übergeben, der diese zurück in ein für die Client-Software nutzbares Format überführt und an den Client überträgt, damit der Nutzer mit dem generierten Plan weiterarbeiten kann.

  - **Input:** Menge von gerichteten, zyklenfreien Graphen, die das Planungsziel unter Berücksichtigung der Projekt-Constraints erfüllen.
  - **Output:** Repräsentation eines (oder mehrerer) Graphen aus der Menge in zur Clientsoftware kompatibler Form.

Da besonderes Augenmerk auf die Praxistauglichkeit des Ansatzes gelegt werden soll, ist es wünschenswert, dass die Techniken und Methoden auch in Verbindung mit bereits etablierten Planungstools und in bestehenden Software-Ökosystemen einsetzbar sind. Dazu soll die Nutzerinteraktion der Lösung maßgeblich auch mit einem weit verbreiteten Projektmanagement-Tool als Client-Software stattfinden. Dazu wurde im Oktober 2016 eine anonyme Umfrage unter insgesamt 32 Personen durchgeführt, die entweder in der Industrie oder an Universitäten im Bereich der Bauprojekt- oder Fabrikplanung beschäftigt sind. Ziel der nicht repräsentativen Umfrage war es herauszufinden, welche Tools für die Erstellung von Terminplänen und Netzplänen sowie zu deren Pflege eingesetzt werden. Die Teilnehmer*innen sollten dabei die in ihrem Unternehmen/an ihrer Universität im Einsatz befindliche Software nennen. Mehrfachnennungen waren möglich. Die Auswertung (siehe Abbildung 4.2) ergab, dass hauptsächlich Produkte der Firma Microsoft im Einsatz sind. Jeder der Befragten gab an, Planungsaufgaben in Excel zu absolvieren. 29 von 32 gaben zudem an, das Projektplanungs-Tool „Microsoft Project“ zu nutzen. Auch Statistiken anderer Untersuchungen zeigen das Bild einer monopolartigen Vormachtstellung von Microsoft in diesem Bereich [145]. Sowohl Excel als auch MS Project verfügen über ein ähnliches Dateiformat und sind als Teil der Microsoft Office Produktfamilie vollständig zueinander kompatibel [100]. Da MS Project als dezidiertes Projektmanagement-Tool bessere Möglichkeiten zur Visualisierung von Netzplänen bietet, wurde dieses Tool als Teil der

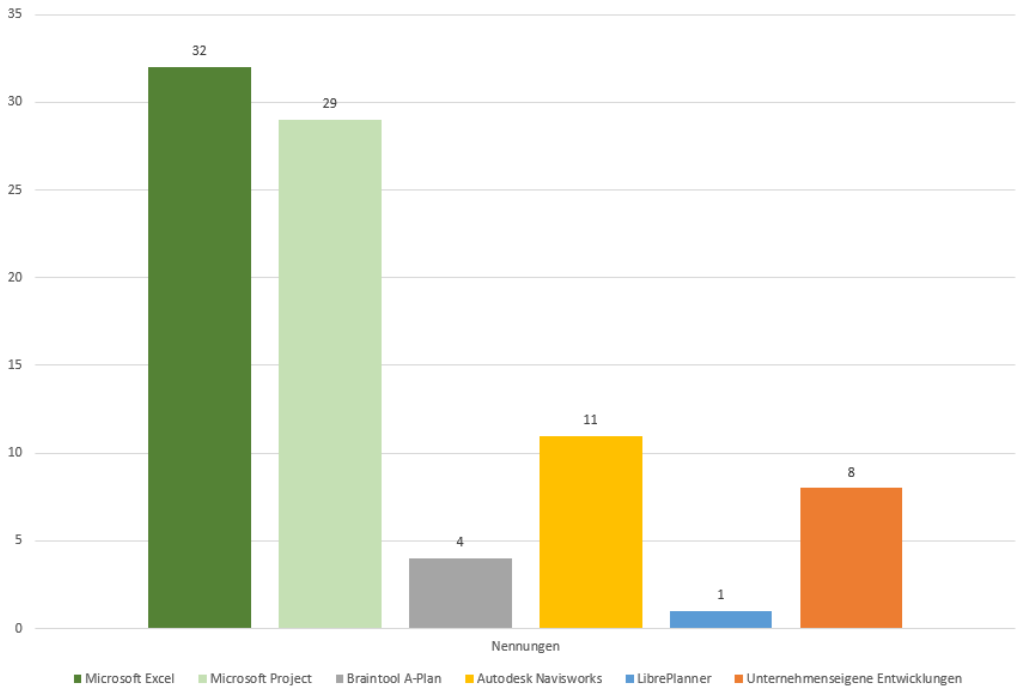


Abbildung 4.2: Verwendete Projektplanungs-Software

hier vorgestellten Architektur ausgewählt. Es soll im Folgenden näher beleuchtet werden.

4.2 Userinteraktion und Datenaustausch

4.2.1 Microsoft Project

Microsoft Project ist eine Software zum Planen, Steuern und Überwachen von Projekten, welches derzeit[7] in der deutschsprachigen Version Microsoft Project 2016 vorliegt. Das Tool ist Bestandteil der Office-Familie und lässt sich sowohl server- als auch clientseitig mit Software von Drittanbietern in andere IT-Systeme integrieren, zum Beispiel in ERP-Software wie SAP PS oder Groupware wie Lotus Domino oder Exchange. Für den Servereinsatz werden allerdings noch der Microsoft-SQL-Server und die Windows SharePoint-Services benötigt, um Project Web-Access zu ermöglichen.

Im Microsoft Project Client werden die Projekte dezentral verwaltet. Für eine zentrale Verwaltung der Projekte und Ressourcen ist Microsoft Project Server oder eine andere Integrationsplattform erforderlich. Eine zentrale Verwaltung hat den Vorteil, dass Abhängigkeiten und Auswertungen über mehrere Projekte zur Verfügung stehen. Ein weiterer Vorteil ist, dass die Projektteams eingebunden werden können. Damit vereinfacht sich die

[7] im Herbst 2018

Informationsverteilung und das Rückmelden wie Fortschrittsstatus und Ist-Zeiten. Für die Teameinbindung können Web Access oder andere Tools verwendet werden.

Microsoft Project ist außerdem eng mit den anderen Tools aus der Microsoft-Office Familie verzahnt. So besteht unter anderem eine enge Verbindung zu Excel, was sich sowohl in der Oberflächen-Gestaltung als auch in der Strukturierung der Datei-Formate niederschlägt. Daten lassen sich somit sehr einfach von einem Tool in das andere überführen. Die Planungsaufgaben eines Projektes können somit sowohl in Excel als auch in Project durchgeführt werden. Dies ist sofern von Bedeutung, da „Excel [...] das einzige PM-Tool im Client-Bereich mit größerem Marktanteil als Microsoft Project [ist].“ [145].

In der Architektur dieser Arbeit fungieren die MS-Office-Tools als Userinterface. Der Planer benutzt nach wie vor die ihm bekannten Oberflächen und Funktionen. Allerdings werden einige Aufgaben, die zuvor vom Planer manuell durchgeführt wurden, nun automatisiert. Dazu gehören vor allem das Erzeugen einer Planstruktur sowie das Ermitteln verschiedener Lösungs- und Strukturvarianten eines Netzplans. Anstatt einen gesamten Projekt- bzw. Netzplan per Hand zu modellieren, gibt der Planer in dem hier dargestellten Prototyp nur noch die Rahmendaten des zu erzeugenden Plans ein. Dabei handelt es sich unter anderem um:

- Die maximale Gesamtdauer des zu planenden Projektes.
- Das Projektbudget.
- Projekt-Ressourcen (z.B. Mitarbeiter oder Maschinen).
- Einzelne das Projekt betreffende Vorgänge bzw. Planungsmodule, sofern diese nicht bereits von der bestehenden Modulsammlung aus Kapitel 4.3 abgedeckt sind.
- Das Planungsziel.

Um die Rahmendaten des Plans und alle weiteren relevanten Informationen an die Server-Archtitektur zu übertragen, wurde eigens im Rahmen dieser Arbeit ein Microsoft Office Plugin entwickelt, welches es ermöglicht, MS Project und MS-Excel Dateien in ein entsprechend aufgearbeitetes JSON-Datei-Format zu überführen und an den Webservice des Servers zu übertragen. Ferner ist es möglich, auch Daten vom Server zu empfangen und zu verarbeiten. Dies ist notwendig, um die generierten Netzpläne später in Empfang nehmen zu können[8].

[8]Das PlugIn ist unter `https://james.cs.tu-dortmund.de/smjawink/MS-Project-CLS` zum Download verfügbar

4.2.2 Datenaustausch mit JSON

Durch die Einbindung von MS Office Software in die Software Architektur ist eine generelle Praxistauglichkeit und -relevanz gewährleistet. Um aber auch die Anbindung zu anderen Tools zu ermöglichen, wurde ein Dateiaustauschformat gewählt, welches weit verbreitet und einfach anwendbar ist: JSON.

Die JavaScript Object Notation, kurz JSON, ist ein kompaktes Datenformat in einer einfach lesbaren Textform zum Zweck des Datenaustauschs zwischen Anwendungen. JSON wird zur Übertragung und zum Speichern von strukturierten Daten eingesetzt. Es dient als Datenformat bei der Serialisierung. Zwar gibt es mit XML (Extensible Markup Language) noch eine weit verbreitete Methode für die API-Integration und den offenen Datenaustausch, jedoch bietet die Verwendung der JavaScript Object Notation (JSON) einige Vorteile. So hat JSON einen kompakteren Stil als XML und ist oft lesbarer. Der einfache Ansatz von JSON kann bei APIs und Schnittstellen, die mit komplexen Systemen arbeiten, erhebliche Verbesserungen bezüglich der Performance bewirken. Der XML-Software-Analyseprozess dagegen kann sehr lange dauern. Ein Grund für dieses Problem sind die DOM-Manipulationsbibliotheken. JSON verwendet insgesamt weniger Daten, so dass es die Analysegeschwindigkeit erhöhen kann.

Darüber hinaus verfügt die JSON-Struktur eine leichtere Zuordnung zu Domänenobjekten von der eingesetzten Programmiersprache. JSON verwendet dafür im Gegensatz zum XML-Baum eine Kartendatenstruktur. In einigen Situationen können Schlüssel / Wert-Paare die Möglichkeiten einschränken, aber insgesamt ergibt sich ein vorhersagbares und leicht verständliches Datenmodell. Denn JSON-Objekte und Code-Objekte stimmen überein. Das ist beim Erstellen von Domänenobjekten in dynamischen Sprachen von Vorteil.

Die Restriktionen in JSON sind einer der größten Vorteile. Ein weit verbreiteter Gedanke unter Entwicklern ist, dass XML die Obergrenze darstellt, da es die Modellierung von einer größeren Anzahl von Objekten unterstützt. Die Einschränkungen von JSON vereinfachen jedoch den Code, erhöhen die Vorhersagbarkeit sowie die Lesbarkeit. In klar definierten Anwendungsfällen und entsprechend schon während der Planung definierbaren Datenobjekten bietet das schlankere JSON demnach zahlreiche Vorteile. Insbesondere bei Webanwendungen und mobilen Apps wird es in Verbindung mit JavaScript, Ajax oder WebSockets zum Übertragen von Daten zwischen dem Client und dem Server häufig genutzt. Dadurch ist es prädestiniert, als Austauschformat zwischen den clientbasierten Planungstools (wie Microsoft Project) und den serverbasierten Komponenten zur automa-

```
{
  "Gesamtdauer": "9999 Tage",
  "Gesamtkosten": "9999 €",
  "AllTasks": [
    {
      "Nummer": "1",
      "Aktiv": null,
      "Vorgangsmodus": "Automatisch geplant",
      "Name": "Abrufverhalten",
      "Dauer": "0 Tage",
      "Anfang": "24 Oktober 2018 8:00",
      "Ende": "25 Oktober 2018 8:00",
      "Vorgänger": [
     "Produktionsprogrammplanung",
        "Produktionsstrukturplanung"}
      ],
      "Sammelvorgang": null,
      "Gliederungsebene": "1",
      "Kosten": null,
      "Ressourcenname": null,
      "Wahrscheinlichkeit": null,
      "Vorraussetzung": null
    },
    {
      "Nummer": "2",
      "Aktiv": null,
      "Vorgangsmodus": "Automatisch geplant",
      "Name": "Absatzprognosen",
      "Dauer": "0 Tage",
      "Anfang": "24 Oktober 2018 8:00",
      "Ende": "25 Oktober 2018 8:00",
```


Abbildung 4.3: Beispiel eines Netzplans repräsentiert in JSON

tischen Planerzeugung zu fungieren.

Jede einzelne Komponente in der Architektur nimmt Daten als JSON Datei entgegen und gibt sie als solche auch wieder aus. Dies ermöglicht, jederzeit weitere Komponenten hinzuzufügen oder auszutauschen, da für jedes neue Element lediglich eine JSON-Schnittstelle vorhanden sein muss, die in der Lage ist die strukturierten Daten auszulesen und zu verarbeiten. Die JSON-Datei beinhaltet dabei jeweils den aktuellen Zustand der generierten Pläne. Zu Beginn des Prozesses befindet sich zunächst lediglich eine Liste mit zu verknüpfenden einzelnen Vorgängen in der Datei. Später wird daraus eine Liste mit aus diesen Vorgängen komponierten Plänen. Am Ende des Prozesses steht dann exakt der gesamte Plan (siehe Beispiel in Abbildung 4.3). Eine detaillierte Beschreibung der JSON-Formatvorlage dieser Architektur befindet sich im Anhang A der Arbeit.

4.3 Globales Planungsmodell und Modulsammlung

Grundlage der automatischen Komposition im Rahmen der Netzplansysnthese bietet das Repository aus taxonomisch geordneten Bausteinen, aus denen die Pläne zusammenge-

fügt werden. Wie bereits in Kapitel 2 erläutert, existieren bereits Ansätze der Fabrikplanung, die auf Modulen basieren und demnach solche Repositorys bereithalten. Neben den (bereits vorgestellten) Ansätzen von Nöcker [109] und Meckelnborg [97] gibt es weitere Arbeiten (wie die von Krunke [83]) oder Arbeiten, die beispielsweise Module für den Baubereich definieren [93]. Ein wesentlicher Kern dieser Arbeit stellt das Zusammentragen und Strukturieren dieser Module sowie das Vereinheitlichen ihres Aufbaus, sodass die Bausteinsammlung möglichst universell eingesetzt werden kann.

Auf Basis der Literatur konnten insgesamt 22 Module nach Nöcker, 19 nach Meckelnborg, sowie weitere 19 Module nach Krunke [83] aus der Fabrikplanung und 15 aus dem Baubereich identifiziert und aufgearbeitet werden (siehe Tabelle 4.1). Darüber hinaus konnten noch 21 weitere Planungsschritte aus vorhandenen Plänen von realen Projekten extrapoliert werden. Somit besteht das Basis-Repository aus insgesamt 96 Modulen, die insgesamt 250 Informationen, die im Planungsverlauf verarbeitet werden müssen enthalten. Eine Übersicht über das gesamte Repository befindet sich in Anhang B.

Die Planungsmodule kapseln Planungsaufgaben in abgeschlossenen Einheiten. Jede Einheit ist für die Erfüllung einer Planungsaufgabe verantwortlich und kann über eine klar definierte Schnittstelle mit anderen Modulen verbunden werden. Als Schnittstelle fungieren Informationen, die in einem Modul zum Bewältigen einer Planungsaufgabe benötigt werden. Diese bilden die Eingabeobjekte des jeweiligen Moduls. Gleichzeitig werden durch das Erfüllen der jeweiligen Planungsaufgabe Informationen generiert, die wiederum als Ausgabeobjekte des Moduls dienen. Zusätzlich können in einem Planungsmodul geeignete Methoden bzw. Werkzeuge hinterlegt werden, welche der Planer während des Planungsprozesses nutzen kann. Diese inhaltliche Ausgestaltung der Module ist allerdings nicht Gegenstand dieser Arbeit. Vielmehr sei an dieser Stelle auf die Dissertation von *Julian Graefenstein* verweisen, die ebenfalls im Rahmen des Graduiertenkollegs entstand und die sich speziell mit der Gestaltung von Planungsmodulen auseinandersetzt [76]. Hinzuweisen ist, dass ein Planungsmodul mehrere alternative Methoden anbieten kann, die sich in Planungsumfang und Detailtiefe der generierten Ergebnisse unterscheiden. Ist dies der Fall, befinden sich mehrere Module des selben Typs im Repository, welche sich in ihren Attributen, aber nicht in ihrer Schnittstelle unterscheiden. Bei der Synthese trägt dies zur Findung von alternativen Lösungswegen bei.

Um nicht nur auf einen Teilaspekt der Fabrikplanung zu fokussieren, sondern eine Methodik bereitzustellen, die den Projektumfang vollständig umfasst, sollten die wichtigsten fabrikplanerischen Fragestellungen abgedeckt werden. Einschränkend ist anzumerken, dass aufgrund der Vielzahl an fabrikplanerischen Fragestellungen die dokumentierte Auswahl an Modulen nicht umfassend sein kann. Es besteht die Möglichkeit, zusätzliche Planungs-

Nöcker 2012	**Meckelnborg 2015**	**Krunke 2017**
Zielplanung	Zielplanung	Zielplanung
Produktplanung	Produktanalyse	Produktplanung
Produktionsprogrammplanung	Produktionsprogrammanalyse	Produktionsprogrammplanung
Produktionsstrukturplanung	Produktionsstrukturplanung	Produktionsstrukturplanung
	Prozessanalyse	Prozessanalyse
Prozessplanung	Technologiekettenplanung	Prozessplanung
Kapazitätsangebotsplanung	Kapazitätsplanung	Kapazitätsangebotsplanung
Kapazitätsbedarfsplanung		Kapazitätsbedarfsplanung
Fertigungsmittelplanung	Produktionsmittelplanung	Fertigungsmittelplanung
Materialflussplanung		
Informationsplanung	IT-Planung	
Personalplanung	Personal- und Organisationsplanung	Personal- und Organisationsplanung
Arbeitsvorbereitung	Arbeitsvorbereitung	Arbeitsvorbereitung
Produktionsplanung und -steuerung	Produktionssteuerungsplanung	Produktionsplanung und -steuerung
Transportplanung	Produktionslogistikplanung	Logistikplanung
Flächen- und Raumplanung	Werksstrukturplanung	General- und Werksstrukturplanung
Layoutplanung	Layoutplanung	Layoutplanung
Arbeitsplatzgestaltung	Arbeitsplatzgestaltung	Arbeitsplatzgestaltung
Gebäudeplanung		
TGA-Planung		TGA-Planung
Produktionsanlaufplanung		
Umsetzungsplanung	Umsetzungsplanung	Umsetzungsplanung
Investitionsplanung	Wirtschaftlichkeitsberechnung	Wirtschaftlichkeitsbewertung
	Standortauswahl	
22 Module	19 Module	19 Module
	+ 15 Module Industriebauplanung	

Tabelle 4.1: Module im Basis-Repository (Fabrikplanung)

module zu definieren und so den inhaltlichen Umfang entsprechend zu erweitern. Diese Möglichkeit ist als eigener Schritt explizit in der Methodik vorgesehen und muss projektspezifisch durchgeführt werden (siehe ach die Anwendungsbeispiele in Kapitel 5).

Wie bereits erwähnt, kapseln Module Aufgaben der Fabrikplanung in logisch abgegrenzte Einheiten. Die Abhängigkeiten der einzelnen Planungsmodule sind durch Ein- und Ausgangsinformationen, den Planungsinformationen, dargestellt. Allerdings ergeben sich je nach Ursprung der Module auch unterschiedliche Ein- und Ausgabe-Informationen für ein inhaltlich identisches Modul. Auch teilen einige Quellen Module in einzelne Teilbausteine auf. Ein Beispiel liefert das Modul *Layoutplanung*, dass nach Meckelnborg in 2 unterschiedliche Teilmodule aufgeteilt wurde (Abbildung 4.4), nach Nöcker allerdings ein einheitliches Modul darstellt (Abbildung 4.5). Auch dies führt zum Vorhandensein von unterschiedlichen Modulen mit dem selben Typen im Repository und somit zur möglichen Variabilität in generierten Lösungen.

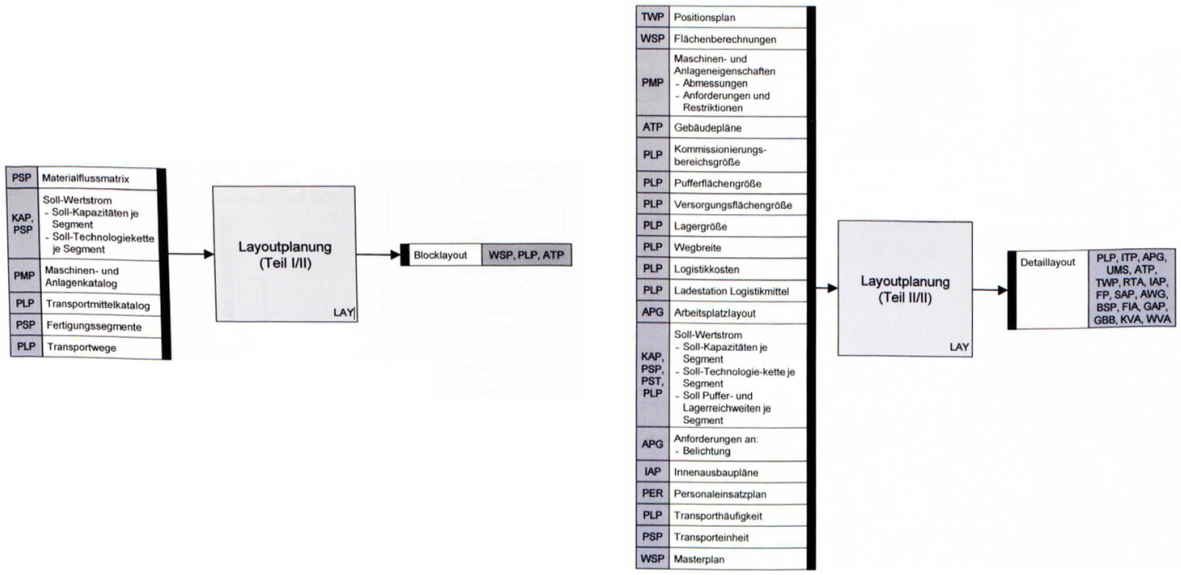


Abbildung 4.4: Modul Layoutplanung aus Meckelnborg [97]

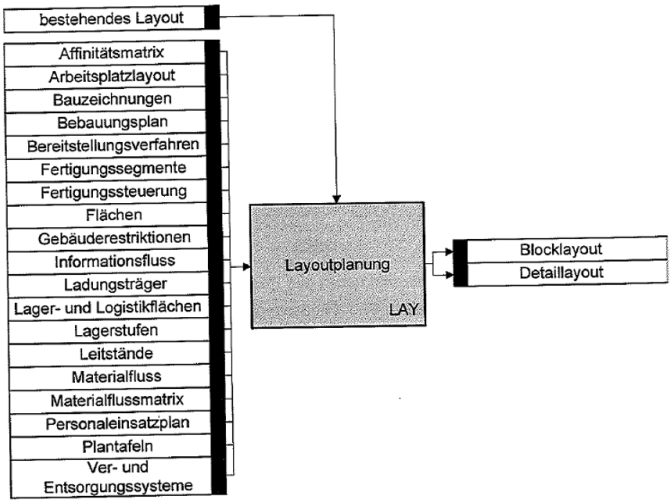


Abbildung 4.5: Modul Layoutplanung aus Nöcker [109]

Die Bezeichnung der Planungsaufgaben ist in den zugrundeliegenden Arbeiten allerdings meist einheitlich. Auf dieser Basis lässt sich die Sammlung an Modulen strukturieren und anhand der Planungsaufgaben taxonomisch ordnen. Dies ist ebenfalls der Tabelle in Anhang B zu entnehmen.

Die Erstellung von Planungsverläufen erfolgt über die Auswahl und Zusammenstellung der Module gemäß ihrer Schnittstellen, einem Planungsziel und unter Berücksichtigung von Projektbedingungen (Projektconstraints). Dabei wird ein (oder mehrere) Zielmodul ermittelt, welches das Planungsziel erfüllt. Anschließend werden solange passende Module (also Module, die die benötigten Inputinformationen als Ausgabeinformationen liefern) vor das

jeweilige Modul geschaltet, bis alle Eingabeinformationen erfüllt sind und alle Inputinformationen vorliegen (siehe Abbildung 4.6).

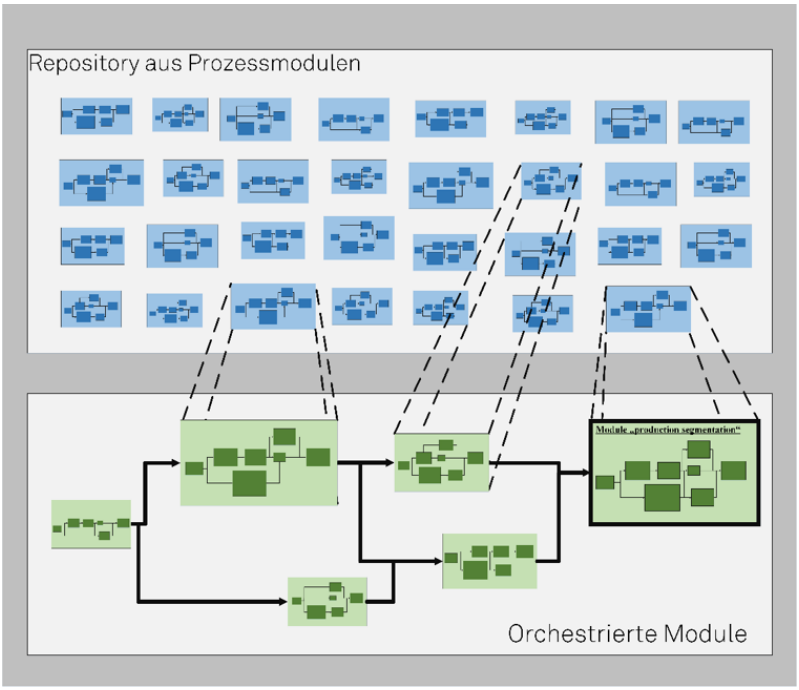


Abbildung 4.6: Vorgehen für das Erstellen eines Planungsverlaufs

Beim Erstellen des Plans müssen dabei diverse Constraints beachtet werden. Zum einen existieren in der Regel sogenannte *globale Projektconstraints*, also Einschränkungen, die für das gesamte Projekt gelten. Dies sind in der Regel Budget- und Zeitrestriktionen. Zum anderen kann es *lokale Constraints* auf Modulebene geben. Diese treten nur bei Auswahl eines entsprechenden Moduls in Kraft und beinhalten in der Regel Anforderungen an andere Module im Planverlauf (z.B. wechselseitiger Ausschluss zweier Module).

Nachdem die Sammlung von Modulen und das Vorgehen zum Erstellen von Planungsverläufen konzeptionell erklärt wurde, setzt sich das folgende Unterkapitel mit der Implementierung des Vorgehens auseinander.

4.4 Erzeugen von Planvarianten mit (CL)S und SMT-Solving

Sind sämtliche für das jeweilige Planungsprojekt relevanten Informationen, Module, Einschränkungen und Ziele durch die jeweiligen Tools erfasst worden, werden daraus Planungsworkflows in Form von Netzplänen generiert. Dabei sollen diese sowohl strukturell als auch semantisch korrekt sowie unter den gegebenen Constraints gültig sein. *Strukturelle Korrektheit* bedeutet, dass keine der in Kapitel 2.4.1 vorgestellten Modellierungs-Regeln für Netzpläne verletzt werden dürfen. Die generierten Workflows dürfen also beispielsweise keine Zyklen aufweisen. *Semantische Korrektheit* beschreibt, dass die generierte Abfolge von Modulen innerhalb des Workflows mit Blick auf die Anwendungsdomäne sinnvoll sein muss. Es sollen Arbeitsabläufe entstehen, die in der Praxis auch umsetzbar sind. Die Korrektheit unter den gegebenen Constraints besagt, dass die generierten Pläne keine Einschränkungen wie zum Beispiel Budget-Schranken verletzen dürfen.

Von den in Kapitel 2 vorgestellten Techniken und Methoden scheint keine für sich genommen in der Lage zu sein, alle Anforderungen an die zu berechnenden Lösungen zu erfüllen. Verfahren zur Synthese mittels kombinatorischer Logik wie das (CL)S (siehe Kapitel 3.4) ermöglichen zwar semantisch korrekte Lösungen anhand einer vorgegebenen Spezifikation (Planungsziel, Modulsammlung etc.), allerdings ist es nach derzeitigem Stand nicht möglich, Lösungen auf die Einhaltung numerischer Constraints wie beispielsweise das Projekt-Budget zu überprüfen.

Demgegenüber stehen Techniken wie das SMT-Solving, welches sich zwar zur Beachtung numerischer Constraints eignet, aber bei der Erzeugung von komplexen Lösungsstrukturen zu erheblichen Performance-Probleme führen kann (siehe Kapitel 3.5). Um den beschriebenen Bedingungen dennoch gerecht werden zu können, werden daher in der vorliegenden Lösung beide Ansätze miteinander kombiniert. Zur Erzeugung von Planvarianten wird die kombinatorische Synthese durch das (CL)S verwendet. Die so erzeugten Planvarianten werden anschließend von einem SMT-Solver überprüft. Das sequentielle Vorgehen, das zunächst eine Synthese-Anfrage bearbeitet und anschließend die Lösungen per SMT Solver überprüft, stellt sich für einen Großteil der Fälle als die performanteste Kombination dieser beiden Ansätze heraus. Die Details dazu werden in Kapitel 4.4.4 näher erläutert.

4.4.1 Plansynthese mit (CL)S Scala

Wie in Kapitel 3.4 bereits beschrieben, ist das (CL)S in der Lage, aus einer strukturierten Sammlung von Komponenten Lösungen gemäß eines gegebenen Zieltyps zu erzeugen. Dieses Verfahren soll zur Erzeugung der Netzpläne verwendet werden. Eine strukturierte Komponentensammlung liegt durch das Modul-Repository aus Abschnitt 4.3 sowie aus

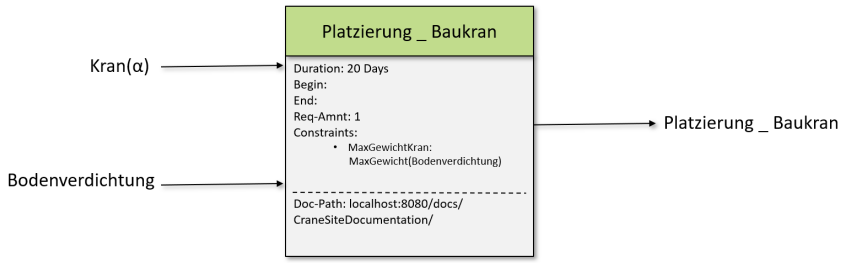


Abbildung 4.7: Beispiel eines Planungsmoduls

den vom Nutzer übertragenen Daten bereits vor. Damit das (CL)S diese nutzen kann, müssen die Daten in Kombinatoren überführt werden. Zu diesem Zweck werden die einzelnen Module zusammen mit ihren Eingangs- und Ausgangsinformationen betrachtet und als Funktionen angesehen. Aus dem Modul *Platzierung-Baukran* aus Abbildung 4.7 ist beispielsweise erkennbar, dass das Modul zwei Eingangsinformationen benötigt und eine Ausgangsinformation erzeugt. Dies wiederum lässt sich als eine Funktion auffassen, welche die Objekte des Typen „Kran“ und vom Typen „Bodenverdichtung“ als Parameter entgegennimmt und als Objekt vom Typen „Platzierung-Baukran“ zurückgibt. Die Funktion (bzw. das Modul) stellt also sicher, dass Kranplätze immer verdichtet sein müssen, bevor ein Kran platziert werden kann.

Durch eine solche funktionale Interpretation eines Planungsmoduls lässt sich das Konzept einfach auf die Kombinatoren des (CL)S anwenden. Die Typen „Kran“,„Bodenverdichtung“ und „Platzierung-Baukran“ werden als semantische Typen übernommen und die Funktion als Kombinator abgebildet. Daraus ergibt sich für das Beispielmodul ein Kombinator mit folgender Signatur:

Platzierung-Baukran: (String $\rightarrow$ String $\rightarrow$ String) $\cap$ (Kran(α) $\rightarrow$ Bodenverdichtung $\rightarrow$ Platzierung-Baukran)

Die weiteren Informationen und Attribute dieses Moduls, die in Abbildung 4.7 dargestellt sind (z.B. die Dauer des Vorgangs), werden funktionsintern in entsprechenden Variablen gesichert. Sie spielen an dieser Stelle noch keine Rolle, werden aber für den später eingesetzten SMT-Solver benötigt (siehe Kapitel 4.4.3). Bei der Darstellung z.B. verschiedener Typen von Baukränen spielt die Fähigkeit des (CL)S mit *Subtyping* umzugehen eine entscheidende Rolle. Analog zum Beispiel aus Kapitel 3.3.1 können so verschiedene Varianten von Kränen taxonomisch geordnet in das Repository aufgenommen werden. Zwar enthält das Basis-Repository von sich aus keine verschiedenen Modelle von Baukränen und macht auch keinen Gebrauch von dieser Funktion, zur weiteren Spezifizierung und

Erweiterung des Repositorys ist Subtyping aber ein sehr probates Mittel.

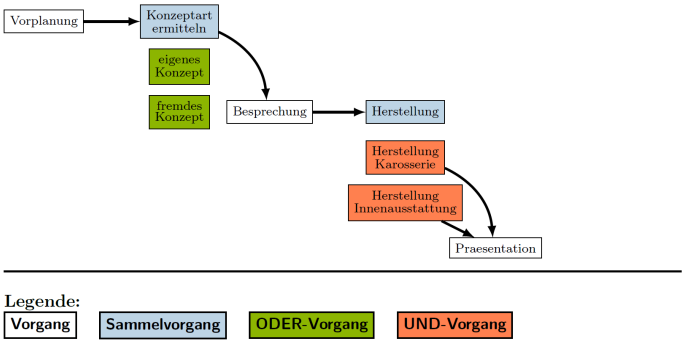
Das (CL)S nimmt die Module als JSON-String entgegen und wandelt sie intern in eine Liste von Kombinatorfunktionen um. Damit das (CL)S die möglichen Varianten eines Netzplanes berechnen kann, braucht es ein Repository mit getypten Kombinatoren und einen Zieltypen[9]. Ein Modul wird durch einen einzelnen Vorgang des Netzplans repräsentiert. Bevor die Vorgänge in getypte (CL)S Kombinatoren transformiert werden können, müssen abstrakte Kombinatoren definiert werden, die es ermöglichen, MS Project-Netzpläne abzubilden. Dadurch können die getypten Kombinatoren automatisch aus den Vorgängen der JSON-Datei generiert werden.

Im Anschluss berechnet das (CL)S mögliche Varianten des Netzplans und gibt die einzelnen Ergebnisse (Inhabitanten) in einer Baumstruktur (Inhabitationsbäume) mit einer Auflistung der verwendeten Kombinatoren aus. Das Planungsziel wird somit als Syntheseanfrage bzw. als Zieltyp für den Inhabitationsalgorithmus aufgefasst. In der Folge versucht der Algorithmus Ketten von Funktionsaufrufen zu generieren, die final als Output den gewünschten Zieltypen und damit das gewünschte Planungsziel enthalten. Da die einzelnen Funktionsaufrufe deckungsgleich mit der Verwendung des zugehörigen Planungsmoduls sind, sind diese Ketten semantisch mit den gewünschten Planungsworkflows gleichzusetzen.

Den Ablauf verdeutlicht folgendes Beispiel: Abbildung 4.8 zeigt den fiktiven und sehr vereinfachten Prozess zum Bau eines Fahrzeuges. Nach der Vorplanung wird das Konzept des Fahrzeuges festgelegt. Es besteht die Möglichkeit, ein eigenes Konzept zu erstellen und zu verfolgen oder ein fremdes einzukaufen. Nach weiteren Abstimmungsterminen erfolgt die Herstellung des Fahrzeugs, zu der naturgemäß zwingend die Karosserie und Innenausstattung gehört. Eine Fahrzeugpräsentation bildet den Abschluss des Prozesses.

Für jeden Vorgang ist das Attribut *Nummer* eindeutig. Zugleich ist die Nummer aber auch abhängig von der Reihenfolge innerhalb des jeweiligen Plans. Da sie stets neu vergeben wird, kann die Vorgangsnummer nicht zur Identifizierung eines Vorgangs dienen. Relevante Attribute sind *Vorgänger* und *Gliederungsebene*. Bei dem Vorgängerattribut werden die Vorbedingungen (Inputs) des jeweiligen Vorganges aufgelistet. Da es sich bei dem dargestellten Knoten um einen Startvorgang handelt, ist die Vorgängerliste allerdings leer. Die Gliederungsebene beschreibt, auf welcher Ebene sich ein Vorgang in einem Sammelvorgang befindet. Der Vorgang *Herstellung Karosserie* liegt beispielsweise in der ersten tieferen Ebene eines Sammelvorganges und befindet sich somit auf Gliederungsebene 2. Das Attribut „Name“ wird hier zur Erstellung der semantischen Typen in

[9] siehe Kapitel 3.4




```
{
  "Nummer": "1",
  "Name": "Vorplanung",
  "Dauer": "5 Tage",
  "Anfang": "14 Mai 2018 08:00",
  "Ende": "18 Mai 2018 17:00",
  "Vorgänger": null,
  "Gliederungsebene": "1",
  "Sammelvorgang": "false",
},
```

Abbildung 4.8: Netzplan- und JSON-Beispiel - Fahrzeugbau

den CLS-Kombinatoren verwendet und dient dazu, die Abhängigkeiten der semantischen Typen verständlich darzustellen. Bei der Überführung der Vorgänge (entsprechend der Prozessmodule) aus der JSON in Kombinatoren wird das in Abbildung 4.9 dargestellte Kombinator-Template verwendet.

```
@combinator object combinator_objectname {
  def apply (TypePredecessor: predecessor_scala_data_type):
    obj_scala_data_type = data_value
  val semanticType: Type = 'TypeVorgaenger =>: 'TypeObject
}
```

Abbildung 4.9: Kombinator-Template zur Generierung von Kombinatoren aus einzelnen Vorgängen

Die generischen Kombinatoren verwenden die Vorgangsnummer als semantischen Typen. Der Objektname wird durch `combinator_objectname` repräsentiert. In der apply Methode werden die Typen der Vorgänger des jeweiligen Vorganges (inklusive ihres Scala-Datentypen) mit einem Komma separiert aufgelistet. Der Scala-Datentyp des Kombinatorobjektes wird dabei inklusive eines Wertes mit `obj_scala_data_type=data_value` angegeben. Beim Wert des semantischen Typen des Kombinatorobjektes `semanticType:Type` wird die Abhängigkeit der Vorgangstypen von links nach rechts mit `=>:` definiert. So stehen die Typen der Vorläufer immer links in der Abhängigkeitskette und bilden auf den rechts stehenden Ausgangstypen ab. Beispiele von generierten Kombinatoren zeigt Abbildung 4.10.

Da letztlich sämtliche Vorgänge/Module in einer für das (CL)S verwendbarer Syntax vorliegen, kann die Synthese der Planungsworkflows durch das Stellen einer Synthese-Anfrage gemäß Kapitel 3.4 erfolgen. In diesen Fällen wird das gewünschte Planungsziel als Zieltyp angegebenen. Für das in Abbildung 4.8 gezeigte Beispiel ergibt sich das Repository gemäß Abbildung 4.11). Für den Zieltyp `String&Praesentation` erhält man zwei

```
@combinator object obj_Vorplanung {
  def apply: String = "str_Vorplanung"
  val semanticType: Type = 'Vorplanung
}
```

Abbildung 4.10: Beispiele für generierte Kombinatoren

Inhabitanten als mögliche Lösungen, die in ihrer Grundstruktur identisch erscheinen und sich lediglich in der gewählten Konzeptart unterscheiden (eine mit eigenem, eine mit eingekauftem Fahrzeugkonzept). In Abbildung 4.12 sind die Inhabitationsbäume dargestellt. Werden die Baumstrukturen von unten nach oben gelesen, entsprechen sie der Abfolge der Arbeitsschritten, die notwendig sind, um das jeweilige Planungsziel im Wurzelknoten zu erreichen. Hierbei erhalten die Kanten im Inhabitationsbaum eine *temporale* Bedeutung, die in der reinen Softwaresynthese des (CL)S nicht vorhanden war. Es gilt, dass ein Vorgang immer dann durchgeführt werden kann, wenn für den entsprechenden Knoten im Baum gilt, dass alle seine Kindknoten bereits abgeschlossen sind.

Γ = {

Vorplanung: String ∩ Vorplanung

Konzeptart_Ermitteln (String → String → String) ∩
(Vorplanung → Konzeptart(a) → Konzeptart_Ermitteln)

Konzept_Eigenes: String ∩ Konzeptart(Eigenes)

Konzept_Fremdes: String ∩ Konzeptart(Fremdes)

Besprechung: (String → String) ∩ (Konzeptart_Ermitteln → Besprechung)

Karosserie: String ∩ Karosserie

Innenausstattung: String ∩ Innenausstattung

Herstellung: (String → String → String → String) ∩
(Besprechung → Karosserie → Innenausstattung → Herstellung)

Praesentation: (String → String) ∩
(Herstellung → Präsentation)

}

WF = { {a → Eigenes}, {a → Fremdes}
}

Abbildung 4.11: Generiertes Repository

Durch die temporale Interpretation der erzeugten Inhabitationsbäume lässt sich eine solche Darstellung bzw. ein solches Vorgehen als gute Vorlage für die Erstellung von Netzplänen nutzen. Auch in einem Netzplan repräsentiert ein Knoten im Graph einen Vorgang, welcher erst bearbeitet werden kann, wenn alle seine Vorgänger bereits vollständig bearbeitet wurden. Dreht man die Bäume also um 90 Grad im Uhrzeigersinn und invertiert die Richtung der Kanten, liegt die grundsätzliche Struktur eines gewünschten Netzplans bereits vor. Da das (CL)S die Inhabitationsbäume so konstruiert, dass alle Voränger einer Komponente auf der selben Ebene im Baum repäsentiert sind, ist der koresspondierende Netzplan dazu auch maximal parallelisiert. Jede Vorbedingung eines Knotens wird in

einem eigenen unabhängigen Pfad erfüllt und erst an der Stelle mit anderen Pfaden zusammengeführt, an der es unbedingt notwendig ist. Bevor die Inhabitationsäume allerdings in vollständige Netzpläne überführt werden können, müssen noch einige Modifikationen an ihnen vorgenommen werden, die im nächsten Abschnitt aufgegriffen werden.

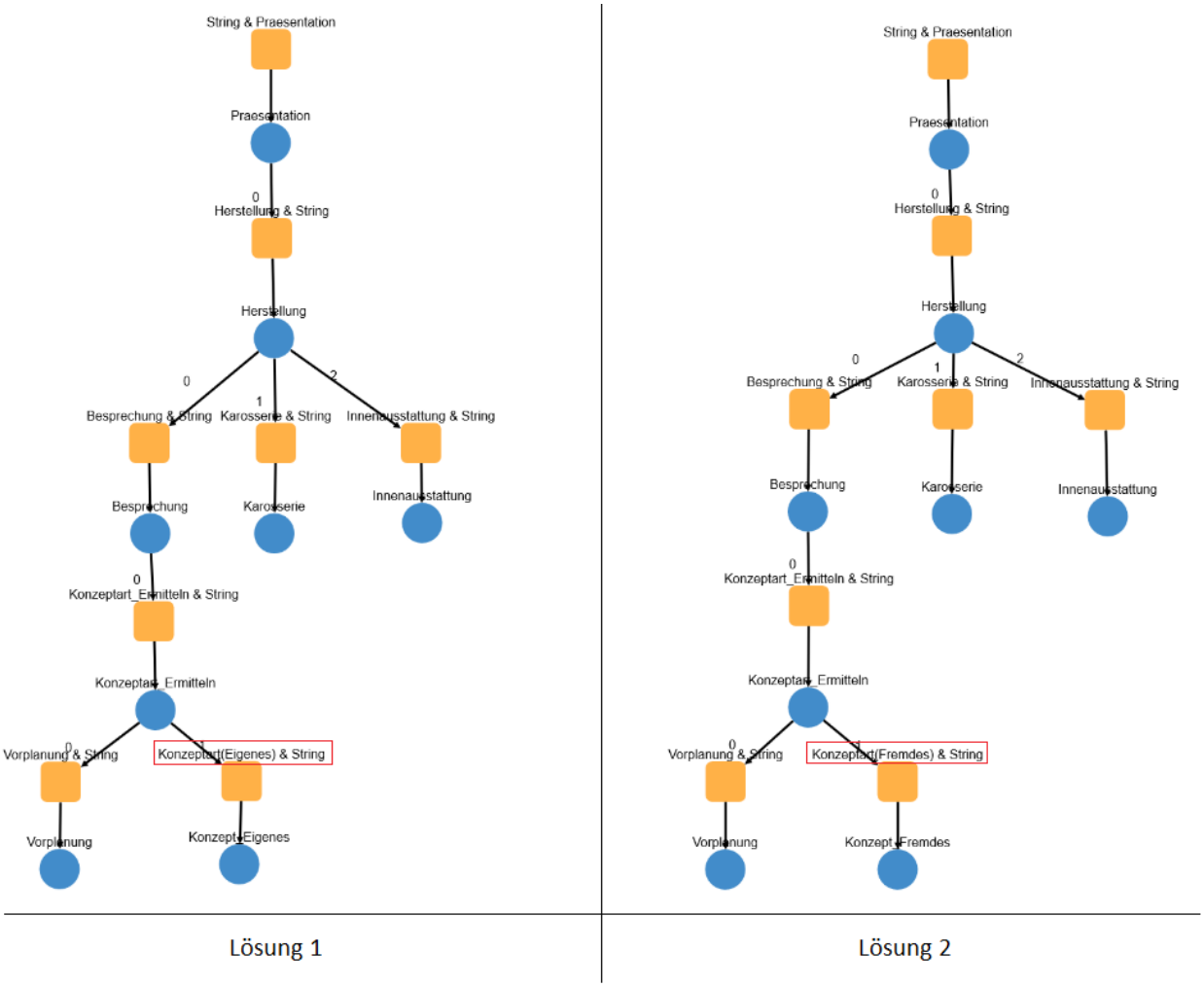


Abbildung 4.12: Inhabitationsbäume zur Anfrage String und Praesentation aus dem Repository aus Abbildung 3.8

4.4.2 Graphtransformation vom Inhabitationsbaum zum Netzplan

Wie beschrieben, werden die Inhabitanten in einer Baumstruktur dargestellt, deren Wurzelknoten das Planungsziel darstellt. Damit diese Bäume zu validen Netzplänen werden können, müssen sie erweitert werden. Insbesondere sind folgende Operationen auf den Baumgraphen notwendig.

1. Es wird ein einheitlicher Startpunkt benötigt. Da die Wurzel das Ziel des Plans (also das zeitlich gesehene Ende) darstellt, muss unterhalb der Blätter des Baumes jeweils ein weiterer Knoten hinzugefügt werden. Dieser dient als Startpunkt für den Plan und vereint sämtliche Zweige des Baumes.

2. Bei der Synthese durch das (CL)S wird jeder Zweig des Baumes unabhängig von den anderen weiter konstruiert. Das kann zu dem Problem führen, dass parallele Pfade im Netzplan durch die Synthese nicht mehr zusammengeführt werden und somit zwei redundante Zweige entstehen. Diese müssen dann nachträglich zusammengeführt werden.

Das erste Problem ist vergleichsweise einfach zu lösen, indem ein zusätzlicher Startknoten, der keine Kosten oder einen Zeitverzug verursacht sowie Kanten zu allen Blättern des Baumes aufweist und unterhalb der Blätter in den Baum eingefügt wird. Das zweite Problem wird mit Hilfe von Graph-Operationen angegangen. Das Verfahren ist dabei in einen Algorithmus eingebettet, welches den Baum durchläuft und nach Redundanzen durchsucht. Dabei wird zunächst für jeden Teilbaum ermittelt, ob dort mehrfach dieselbe Komponente verwendet wurde. Ist dies der Fall, muss geprüft werden, ob die Pfade zusammengelegt werden können. Hierbei ist der Umstand hilfreich, dass ein Großteil der Planungsaufgaben in einem Fabrikplanungsprojekt singulärer Natur sind [2]. Das bedeutet, dass die Komponenten in der Regel nur einmal im fertigen Planungsverlauf vorkommen sollen. Wird dieselbe Komponente also in verschiedenen Zweigen des Inhabitationsbaumes verwendet, ist dies ein Indikator dafür, dass an dieser Stelle die Komponenten zu einem Knoten verschmolzen und die Zweige wieder zusammengeführt werden müssen (siehe Abbildung 4.13). Dies ist deshalb möglich, weil die Knoten Aufgaben repräsentieren, die im Verlauf der Planung abgearbeitet werden müssen. Wurde eine Aufgabe in einem Planunszweig erledigt, gilt dies auch für alle anderen Bereiche des Planungsverlaufs. Sollte es in einigen Fällen dennoch gewünscht sein die Mehrfachnutzung des Moduls (des Kombinators) zu erlauben, kann dies durch Angabe eines entsprechenden Modul-Constraints (siehe 4.4.3) erreicht werden.

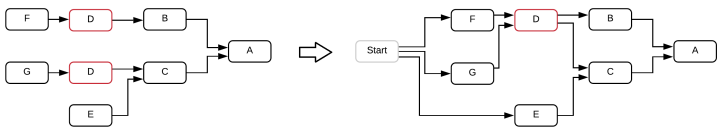


Abbildung 4.13: Graphtransformation zum Umwandeln eines Inhabitationsbaumes in einen Netzplan

4.4.3 SMT Solving mittels Scala Graph

Durch die vorangegangene Synthese liegen eine Reihe von Varianten möglicher Planungsworkflows vor, die das vorgegebene Planungsziel erfüllen und sich aus für den Planer bekannten Arbeitsschritten und Vorgängen (Modulen) zusammensetzen. Zusätzlich ist jedoch eine Überprüfung notwendig, die sicherstellt, dass die Lösungsvarianten, die dem Planer am Ende wieder zur Verfügung gestellt werden sollen, auch projektspezifische numerische Constraints (Budgetschranken, Zeitlimit, Ressourcenbeschränkungen) sowie modulspezifische Contraints (z.B. gegenseitiger Ausschluss bestimmter Module) einhalten. Diese Überprüfungen finden durch einen SMT Solver statt.

In der vorliegenden Architektur ist dies mittels *Scala Graph*[10] realisiert. Dabei handelt es sich nicht um einen klassischen SMT-Solver wie beispielsweise *Microsoft Z3* oder *CVC4*. Vielmehr ist Scala Graph eine Scala Library, welche Funktionalitäten für die Erzeugung von Graphen bereitstellt. Allerdings verfügt diese Library auch über das Paket *Constrained*. Mit diesem Paket können Benutzer von Graph vordefinierte oder benutzerdefinierte Constraints nahtlos in Graphen integrieren. Constraints dienen hier dazu, bestimmte Eigenschaften während der gesamten Lebensdauer von Graphinstanzen sicherzustellen, die nicht durch Typ-Parameter durchsetzbar sind. Allgemeine Einschränkungen sind zum Beispiel, dass Graphen zusammenhängend, azyklisch, in einer Baumstruktur oder planar sein müssen. Darüber hinaus kann der Benutzer eine benutzerdefinierte Einschränkungsimplementierung bereitstellen, die alle domänenspezifischen Anforderungen abdeckt.

Vordefinierte Einschränkungen werden in `scalax.collection.constrained.constraints` bereitgestellt. Graphen können dynamisch oder statisch eingeschränkt werden. Ein Graph wird als dynamisch eingeschränkt bezeichnet, wenn bei der Erstellung ein Constraint an seine Factory-Methode übergeben wurde oder wenn er das Ergebnis einer Operation ist, die auf einem dynamisch gebundenen Graphen basiert. Ein statisch eingeschränkter Graph hingegen enthält alle Validierungen, die erforderlich sind, damit alle Operationen die gewünschten Constraints erfüllen bereits in seiner Implementierung. Somit muss bei der Erstellung kein Constraint mehr übergeben werden. Natürlich könnte jeder dynamisch beschränkte Graph auch statisch implementiert werden, sobald die Constraints bekannt sind. Eine statische Implementierung kann sich aber immer dann lohnen, wenn eine breite Verwendung vorliegt oder Typparameter eingeschränkt werden müssen.

Der Einsatz von Scala Graph bietet sich aus mehreren Gründen an:

- Bei Netz- und Terminplänen handelt es sich um Graphstrukturen. Scala Graph ist genau auf diesen Anwendungsfall zugeschnitten.

[10] `http://www.scala-graph.org/`

- Durch die Spezialisierung auf Contraints auf Graphstrukturen ist die Implementierung deutlich schlanker und einfacher als bei einem universal SMT-Solver wie Z3. Dies kommt auch der Performance zugute.
- Scala Graph ist ebenso wie das (CL)S in der Programmiersprache SCALA implementiert. Dies erleichtert die Zusammenführung beider Technologien.
- Scala Graph verfügt von Haus aus eine JSON Schnittstelle und lässt sich direkt in die Architektur übernehmen.
- Manipulationen am Graphen, die sich aus den Contraints ergeben, lassen sich direkt mit Bordmitteln von Scala Graph durchführen.

Für die Durchsetzung der globalen Constraints, wie die maximale Dauer oder das maximale Budget, lassen sich einfache Funktionen aufstellen. Für den Zeitconstraint wurde zum Beispiel die Formel zur Berechnung des kritischen Pfades implementiert, mit deren Hilfe sich der Constraint `assert(<=lengthCP(Graph)limit)` durchsetzen lässt. Für die Berechnung der Kosten werden die Kostensätze aller im Graph vorhandenen Knoten summiert und anschließend mit dem Budget Limit verglichen. Wird einer der beiden Constraints verletzt, wird der jeweilige Graph verworfen.

Für lokale Constraint wird die `refuseNode`-Funktion von Scala-Graph verwendet. Diese erstellt eine Art „Blacklist“ von Knoten, die nicht im Zusammenhang mit dem jeweiligen Knoten verwendet werden dürfen. Sie kann regulär im Kombinator implementiert werden. Abbildung 4.14 zeigt die Verwendung der Funktion im Beispiel aus Abbildung 4.7.

```
1  @combinator object Platzierung_Baukran {
2    def apply(Kran: String, Bodenverdichtung:String):String = "Platzierung Baukran"
3
4    val Duration = 20
5    if (Graph.getNode("Kran").getWeight() > Graph.getNode("Bodenverdichtung").getCompressionRate())
6      {Graph.refuseNode(Graph.getNode("Kran"))}
7
8    val semanticType: Type = 'Kran(alpha)  =>: 'Bodenverdichtung  =>: 'Platzierung_Baukran
9  }
```

Abbildung 4.14: Beispiel für die Implementierung eines lokalen Constraints

4.4.4 Kombination von SMT Solving und Synthese

In der prototypischen Implementierung, die im Rahmen der Arbeit entstanden ist, erfolgte die Kombination zwischen Synthese und SMT-Solving durch ein sequentielles „Hintereinanderschalten“ der beiden Verfahren. Zunächst wurden Lösungsvarianten mittels Synthese erzeugt, anschließend wurden Varianten, die nicht den Constraints entsprachen, wieder entfernt. Dies ist aber nicht die einzige Möglichkeit, beide Ansätze miteinander zu kombinieren. Insgesamt sind drei Kombinationsmöglichkeiten denkbar, die schon bei der theoretischen Betrachtung spezifische Vor- und Nachteile mit sich bringen.

1. *Vorgelagertes SMT-Solving (Abb. 4.15):* Hierbei werden die SMT Bereichnungen auf dem Repository durchgeführt, um sie so zu gestalten, dass keine unter den Constraints illegalen Lösungen durch die Synthese erzeugt werden können. Das geschieht zum Beispiel durch Entfernen von Modulen, deren Nutzung sehr zeit- oder kostenintensiv wären.

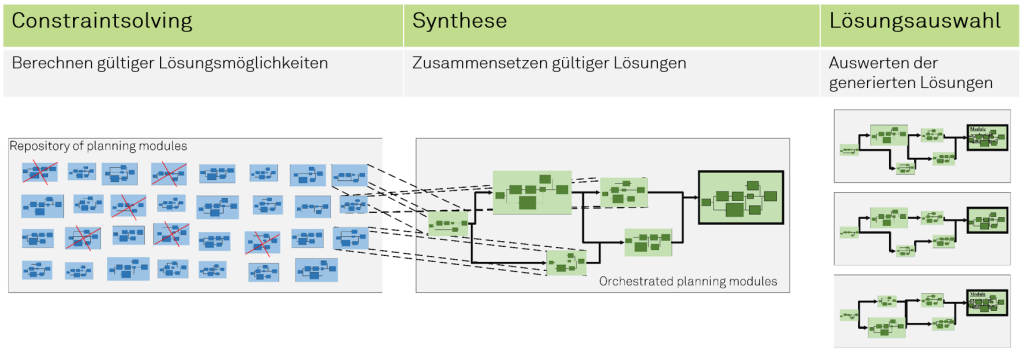


Abbildung 4.15: Vorgelagertes SMT-Solving

- Vorteile dieser Lösung:
  - Durch das Vorselektieren von Modulen und Komponenten wird die Erzeugung von Blind-Lösungen, also solche, die von vorne herein ungültig sind, verhindert. Durch die geringere Anzahl der zu erzeugenden Lösungs-Varianten wird die Rechenzeit des Synthese-Algorithmus verringert.
- Nachteile dieser Lösung:
  - Um seriös vorselektieren zu können, müssen mögliche ungültige Lösungen im Vorhinein durch den SMT-Solver berechnet werden. Da dieser nun aber nicht nur bereits „fertige“ Lösungen überprüfen, sondern zunächst selbst Lösungen berechnen muss, steigt die Rechenzeit.
  - Unter Umständen ergeben sich bei der Komposition der Lösungen durch das (CL)S trotz guter Vorselektion dennoch Lösungen, die einzelne Constraints verletzen. Gegebenenfalls ist deshalb eine weitere Überprüfung nach dem Generieren von Lösungen durch das (CL)S notwendig um sicherzustellen, dass wirklich nur gültige Lösungen an den Nutzer weitergegeben werden.

2. *Integriertes SMT-Solving (Abb. 4.16):* Hier werden die Lösungen bereits vom SMT Solver überprüft, während sie vom (CL)S erzeugt werden. Lösungen, die schon während des Konstruktionsprozesses Constraints verletzen, werden so direkt verworfen.

- Vorteile dieser Lösung:

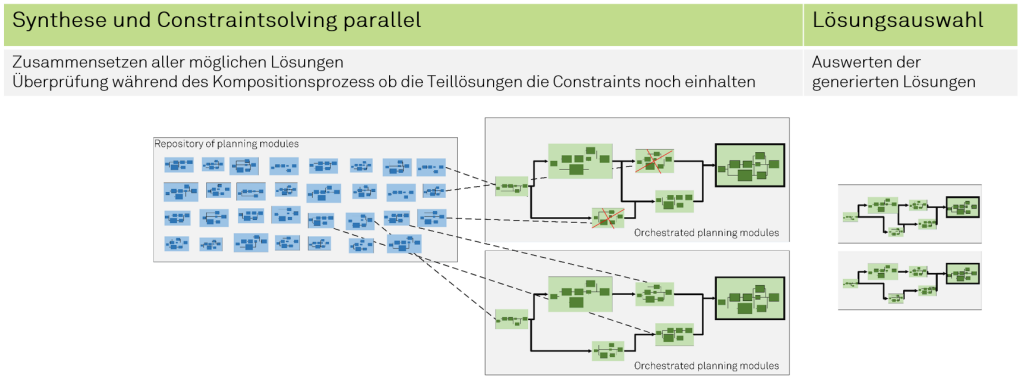


Abbildung 4.16: Integriertes SMT-Solving

  - Es wird direkt bei der Konstruktion eines Lösungsansatzes durch den Inhabitationsalgorithmus des (CL)S erkannt, ob er verwendbar ist. Sobald der Ansatz bzw. die Lösung den ersten Constraint verletzt, kann die Konstruktion abgebrochen werden. Somit müssen nicht zahlreiche Lösungsvarianten erst vollständig generiert und dann wieder verworfen werden.

- Nachteile dieser Lösung:
  - Der SMT-Solver muss tief in das (CL)S integriert werden, da er parallel zur Lösungsgenerierung arbeiten muss. Das Konzept der losen Kopplung und die damit verbundene Freiheit in der Wahl des Solvers und des Syntheseframeworks fallen somit weg.
  - In diesem Verfahren müsste jede Lösung nach jedem Konstruktionsschritt überprüft werden. Die Anzahl der notwendigen Anfragen an den SMT-Solver wäre demnach beträchtlich höher. Es ist möglich, dass die Einsparungen an Rechenzeit auf Seite des (CL)S bei weitem nicht ausreichen, den Mehraufwand auf Seiten des SMT-Solvers zu kompensieren.

3. *Nachgelagertes SMT-Solving (Abb. 4.17):* Bei diesem Verfahren handelt es sich um die bereits beschriebene sequentielle Verkettung, in der erst Lösungen synthetisiert und anschließend an den SMT-Solver übergeben werden.

   - Vorteile dieser Lösung:
     - Der SMT-Solver muss nur die generierten Lösungen betrachten und jede Lösung lediglich ein einziges Mal überprüfen. Daraus ergibt sich mutmaßlich ein Performancegewinn auf Seiten des Solvers.
   - Nachteile dieser Lösung:
     - Das (CL)S kann beim Generieren nicht beurteilen, ob einzelne Inhabitanten (Lösungen) unter den numerischen Constraints überhaupt verwendbar sind. Es wird folglich Rechenzeit dafür verwendet, Blind-Lösungen zu generieren.

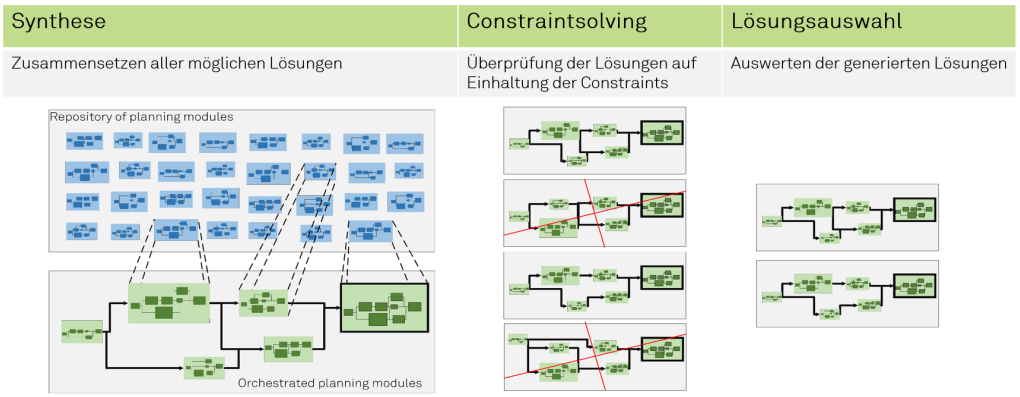


Abbildung 4.17: Nachgelagertes SMT-Solving

– Die Menge an Lösungen, die das (CL)S für eine Inhabitations-Anfrage generieren kann, ist potenziell unendlich groß. Die Überprüfung jedes Elements einer theoretisch unendlich großen Menge dauert folglich ebenfalls unendlich lange. Dies wird in der Regel jedoch dadurch abgefangen, dass das (CL)S nach einer bestimmten (hinreichend großen) Anzahl gefundener Lösungen abbricht.

Ermittlung und Evaluation der optimalen Kombinationsstrategie

Es wurden Experimente durchgeführt, um die optimale Kombinationsstrategie zwischen den beiden Technologien zu ermitteln. Während sich die Strategie des integrierten Constraintsolvings aufgrund der Arbeitsweise des Inhabitationsalgorithmus des (CL)S nicht realisieren ließ, wurden die anderen beiden Strategien prototypisch implementiert. Die Implementierungen wurden anschließend mit identischen Synthese-Anfragen auf dem selben Repository und Constraints gestellt. Als Repository diente das unmodifizierte Basis-Repository aus Abschnitt 4.3. Die Anfragen wurden dann nach komplexeren Ziel-Modulen (*Innenausbaupläne* mit 6 bzw. *Produktionslogistikplanung* mit 25 direkten Abhängigkeiten) gestellt (siehe Abbildung 4.18). Es wurden jeweils die Projektdauer und -Kosten als numerische Constraints angegeben. Die Module und Constraints wurden so gewählt, dass jeweils eine der durch die Synthese generierte semantisch korrekte Lösung die Constraints nicht einhält und somit prinzipiell entfernt werden müsste. Es sollten für die Innenausbaupläne eine und und für die Produktionslogistikplanung wiederum zwei Lösungen am Ende des Verfahrens erzeugt worden sein. Alle Lösungen sollten unter Beachtung der Constraints korrekt sein.

Durchgeführt wurde das Experiment auf einem Windows-Rechner mit einem Intel i7-5500U (2,4 GHZ) Prozessor und 16 GB Arbeitsspeicher, welcher sowohl den Client (MS-Project 2013, 64-Bit) als auch die Server-Seite der Architektur durch einen virtuellen Ser-

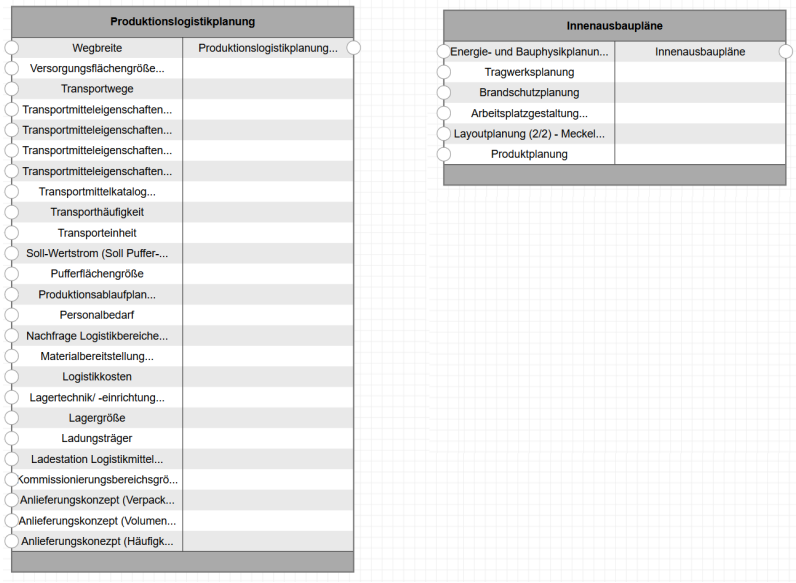


Abbildung 4.18: Zielmodule des Experiments

ver ausführt. Jede Anfrage an das System wurde exakt 10 Mal gestellt. Gemessen wurde jeweils die Zeitspanne zwischen dem Versand der Anfrage in MS-Project bis zum Eintreffen der JSON-Datei mit dem Lösungsset. Die Ergebnisse sind in Tabelle 4.2 dargestellt.

	Constraints		Durchschnittl. Anzahl	Anzahl gefundene Lösungen		Laufzeit in Sekunden	
Zielmodul	max. Dauer	max. Kosten in Euro	Knoten in Lösung	Vorgelagert	Nachgelagert	Vorgelagert	Nachgelagert
Innenausbaupläne	3 Monate	125.000	21	1	1	10,3	9,9
Produktionslogistikplanung	8 Monate	840.000	47	1	2	42	38,5

Tabelle 4.2: Laufzeitmessungen der Kombinationsvarianten

Tabelle 4.2 zeigt, dass sich die Variante des *nachgelagerten SMT-Solvings* als die performanteste Strategie herausgestellt hat. Ungültige Lösungen, also solche, die nicht zu den Spezifikationen der Anfrage passten, wurden bei keinem der Ansätze erzeugt, sogenannte *False Negatives*, also Lösungen die fälschlicherweise verworfen wurden traten ebenfalls nicht auf. Im Vergleich zwischen den beiden Ziel-Modulen zeigt sich, dass bei entsprechend größer werdenden Lösungen, sich die geringe Belastung des SMT-Solvers auszahlt. Aus diesem Grund wurde in dem vorliegenden Software-Prototypen diese Strategie verwendet.

Am Ende dieses Verfahrens ergibt sich eine Reihe von Lösungen, von denen jede einzelne eine eigene Variante eines Planungs-Workflows (repräsentiert als Netzplan) darstellt. Dabei ist unter Berücksichtigung der korrekten Arbeitsweise des SMT-Solvers und des (CL)S (siehe Kapitel 3.4 und 3.5) anzunehmen, dass jede dieser Varianten

1. das gewünschte Planungsziel erfüllt,

2. sich aus Prozessschritten, Vorgängen und Modulen zusammensetzt, die dem Planer bekannt und für das Projekt durchführbar sind und

3. alle erfassten Constraints und Einschränkungen, die das Projekt betreffen, berücksichtigen.

4.4.5 Auswahl der Lösungen und Übertragung in Client-Software

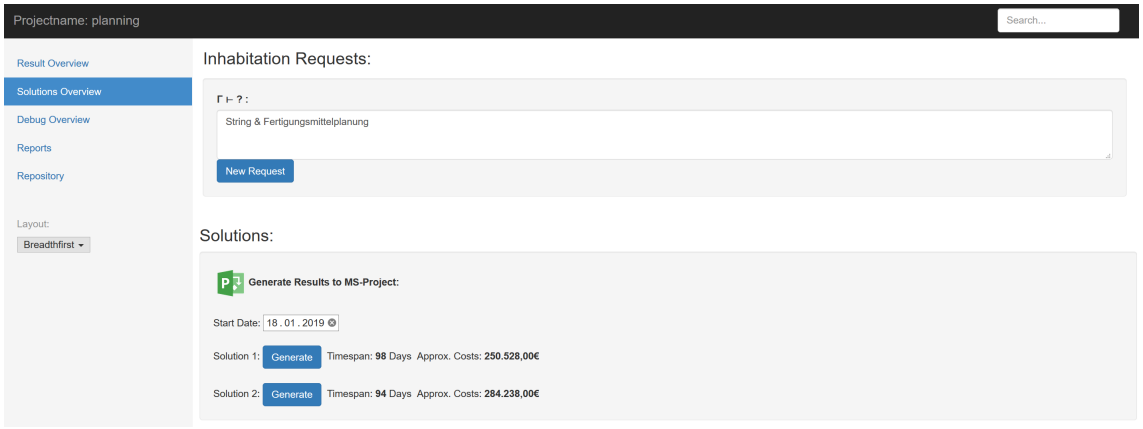


Abbildung 4.19: Web-Interface zur Darstellung der Plan-Varianten

Der folgende Schritt für den verantwortlichen Planer beinhaltet, den für ihn passenden Planungsworkflow auszuwählen. Bei dieser Entscheidung handelt es sich um ein *multikriterielles Optimierungsproblem*. Wie in Kapitel 2.4.1 erläutert, zeichnet sich ein Projektplan durch verschiedene Kennzahlen, die sich nicht allgemein gewichten lassen, aus. Aus diesem Grund empfiehlt es sich, es dem Planer selbst zu überlassen, den für die jeweilige Situation passenden Plan individuell auszuwählen. Um die jeweilige Entscheidung bestmöglich zu unterstützen, werden die generierten Lösungen in einer Weboberfläche aufgearbeitet. Dabei sollen dem Planer alle relevanten Kennzahlen und Informationen der jeweiligen Variante zur Verfügung gestellt werden (siehe Abbildung 4.19).

4.5 Zusammenfassung

Nach Auswahl der präferierten Lösung erzeugt der Code-Generator aus den Daten erneut eine JSON-Datei, welche anschließend an die Client-Software des Planers zurück übertragen und dort in das entsprechende Dateiformat transformiert wird. In diesem Fall wird eine Microsoft-Project-Datei erzeugt. Der Planer kann mit einem fertigen Projektplan seine Arbeit fortsetzen.

Dieses Kapitel hat anhand einer beispielhaften Implementierung gezeigt, wie sich Netzpläne auf Basis einer Komponentensammlung generieren lassen können und wie sicherge-

stellt werden kann, dass die generierten Pläne zu den Gegebenheiten des jeweiligen Projektes passen und praktisch umsetzbar sind. Der allgemeine Workflow ist in Abbildung 4.20 noch einmal illustriert. Dabei werden folgende Schritte in folgenden Bestandteilen der Architektur durchlaufen:

1. Erfassen von projektspezifischen Daten (Zeit- und Kosten Budget, projektspezifische Vorgangskomponenten, bereits absolvierte Vorgänge, Planungsziel etc.). Dies geschieht in der Projektmanagement Software **MS Project**, die es dem Planer/der Planerin erlaubt, in einer etablierten Software zu arbeiten. Dadurch fügt sich das Verfahren einfach in bestehende Arbeitsabläufe in der Fabrikplanung ein.

2. Übertragen der Informationen per JSON an das (CL)S.

3. Durch das als Zieltyp vorliegende Planungsziel sowie den getypten Komponenten inhabitiert das **(CL)S** mögliche Planvarianten. Es ermittelt insbesondere, welche Aufgaben in welcher Reihenfolge zum Erreichen des Planungsziels zu bewältigen sind. Dadurch entstehen Sequenzen von Planungsaufgaben, die das Planungsziel erfüllen.

4. Die Sequenzen lassen sich durch Parallelisierung von unabhängigen Planungsaufgaben in Inhabitationsbäume überführen. Durch weiteres **transformieren der Inhabitationsbäume** werden gerichtete, zyklenfreie Graphen erzeugt, deren Struktur mit denen von Netzplänen übereinstimmt.

5. Mittels der Constraint-Funktionalitäten von **Scala-Graph** werden die vorhandenen Graphen auf Einhaltung der Projekt-Constraints überprüft. Graphen, die diese Einschränkungen nicht einhalten, werden aussortiert.

6. Übertragung der verbliebenen Lösungen an die Client-Software des Planers.

7. Der Planer/die Planerin kann mit dem generierten Plan seine Arbeit fortsetzen.

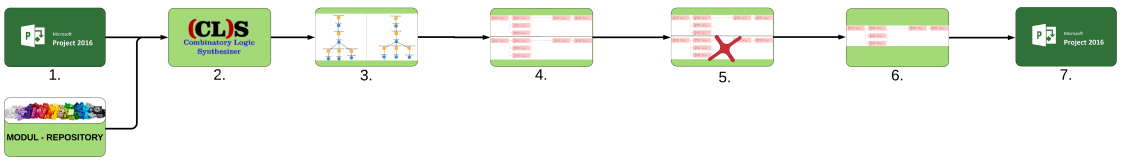


Abbildung 4.20: Workflow zur Netzplan-Generierung

Im folgenden Kapitel wird dieses Vorgehen sowie die Implementierung anhand von Experimenten mit realen Planungsdaten evaluiert.

Kapitel 5

Anwendung und Evaluation

Der Anspruch des vorgestellten Vorgehens ist es, ein Werkzeug zum Einsatz in Planungsprojekten der Realität zu entwerfen. Der Forschungsprozess beginnt und endet daher im Anwendungszusammenhang. Dazu werden in diesem Kapitel drei Fallbeispiele vorgestellt, welche die in Kapitel 4 dargelegte Architektur im Hinblick auf Performance, Validität und Qualität der Lösungen untersuchen sollen. Die Fallbeispiele sollen stellvertretend für verschiedene Anwendungsfälle und Varianten von Fabrikplanungsprojekten sowie verschiedene Typen von Experimenten stehen, die im Rahmen dieser Arbeit durchgeführt wurden.

Im ersten Experiment wird die Intralogistik einer bestehenden Fabrik angepasst, ohne dass bauliche Veränderungen an der Fabrik selbst durchgeführt werden sollen. Dabei wird auf einem sehr feinen Detaillevel geplant. Zudem müssen zahlreiche Rahmenbedingungen des bestehenden Fabriksystems berücksichtigt werden.

Im zweiten Fallbeispiel werden Anlagen aus zwei bestehenden Werken an einem neu zu errichtenden Standort zusammengeführt. In diesem Projekt steht die terminliche Planung von Baumaßnahmen im Vordergrund. Entsprechend wird auf einem gröberen Detaillevel geplant, was einen kompakteren Netzplan zur Folge hat. Die Herausforderung besteht darin, bestehende Anlagen aus den alten Fabriken mit in die Planung zu integrieren und eine strikte Zeitvorgabe einhalten zu können.

Im dritten Experiment geht es um einen kompletten Neubau einer Fabrik mit angeschlossenem Warenlager und Logistikzentrum. Da sich die Planung und Durchführung des Projektes immer wieder veränderten Rahmenbedingungen stellen musste, war der Netzplan stetig anzupassen zu verfeinern. Die Folge ist ein sehr detailliert ausgeplanter und entsprechend umfangreicher Netzplan. In diesem Experiment steht die Dynamik von modernen Planungsaufgaben im Mittelpunkt. Es soll zum einen die Fähigkeit des Systems die Anpassung von Plänen im laufenden Projekt durchführen zu können, überprüft werden, zum anderen soll die Performanz im Umgang mit sehr großen Plänen untersucht werden.

Den Experimenten liegt jeweils das Basis-Repository an Planungsmodulen (Kapitel 4.3) zugrunde, welches für den jeweiligen Use Case mit fallspezifischen Modulen erweitert wurde. Anwendungsfälle, Rahmenbedingungen, Constraints und der jeweilige Referenzplan, mit dem die generierten Lösungen verglichen werden, entstammen dabei realen Projekten, die bei (im Rahmen dieser Arbeit) akquirierten Industrie-Partnern durchgeführt wurden. Diese sind allerdings anonymisiert.

Zur Evaluation stehen jeweils Pläne und Planungsdaten der realen Projekte bereit. Die generierten Ergebnisse werden dann mit den Real-Plänen anhand diverser Kennzah-

len (Projekt Dauer, -Kosten, Ressourcen-Einsatz etc.) bewertet. Auf diese Weise soll die Praxistauglichkeit des Ansatzes dieser Arbeit ermittelt werden.

5.1 Planung einer Fabrikkonfiguration

Das Experiment wurde ursprünglich in *Winkels et al.* [163] veröffentlicht und basiert als einziges nicht auf dem in Kapitel 4.3 vorgestellten Basis Repository. Ziel des vorgestelltem Experiments war es, eine Möglichkeit aufzuzeigen, sinnvolle Lösungsalternativen einer Fabrikkonfiguration zu generieren, um einen effizienten Zeit- und Kostenplanungsprozess zu ermöglichen um damit den *Proof of Concept* für die Planung mittels komponentenorientierter Synthese zu erbringen. Mit Hilfe zuvor definierter Ziel- und Rahmenparametern sowie vorhandenen Informationen zum aktuellen Fabriksystem sollten verschiedene Lösungsvarianten für die Zielplanung erstellt werden. Durch den Vergleich von Kombinationen unterschiedlichster Lösungsmöglichkeiten sollten automatisch erste grobe und plausible Lösungen generiert werden, auf deren Basis der detaillierte Planungsprozess zum Erreichen der ermittelten Lösungsvariante erstellt werden kann.

5.1.1 Szenario

Als Grundlage des Experimentes diente ein reales Planungsszenario eines Unternehmens in der Fertigungsindustrie. Der Planungsimpuls wurde von der Unternehmensleitung aufgrund einer veränderten Kundenstruktur mit entsprechend unterschiedlichen Produkten und Verkaufszahlen ausgelöst, woraufhin das Produktionssystem geändert werden musste. Zunächst wurden auf der Ebene der Unternehmensführung Rahmenbedingungen und Ziele formuliert, um die Richtung für die endgültige Planung festzulegen. Die wichtigsten Punkte waren:

- Keine Änderungen an bestehenden Produktionslinien.
- Mitarbeiter werden nicht entlassen oder eingestellt.
- Die Eigenschaft oder der vorhandene Bebauungsbereich werden nicht geändert.
- Die Fabrikhalle wird nicht erweitert.
- Mit den vorhandenen Fertigungsmitteln muss ein bestimmter Umsatz produziert werden.

Weitere Informationen zu diesen Rahmenbedingungen und Zielen wurden bereitgestellt, um grobe Lösungsvarianten nach diesen Vorgaben entwickeln zu können. Erstens konnte ein Verkaufsvolumen ermittelt werden, das sich vom vorherigen unterschied. Die Anforderungen des Kunden an jedes einzelne Produkt sollen volatiler als zuvor sein, was

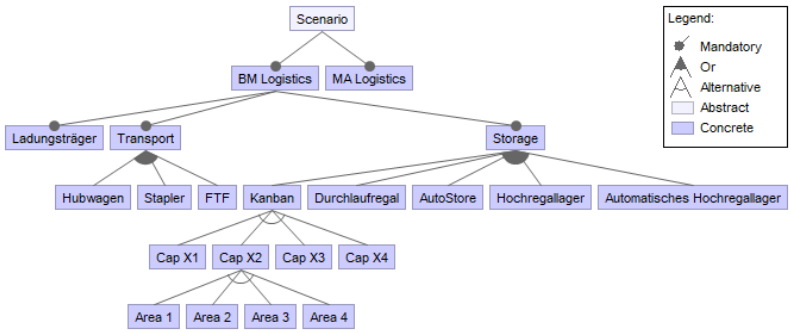


Abbildung 5.1: Feature-Diagramm des Szenarios. Entnommen aus [163]

die Flexibilität des neuen Systems erhöht. Ebenso führen die geänderten Produktvarianten zu einem erhöhten Lagerbedarf, um die entsprechend größere Produktvielfalt abdecken zu können.

Die von der Unternehmensleitung gesetzten Ziele und Rahmenbedingungen führten zu der Entscheidung, dass die bedeutendste Variable, die die Anforderungen und Ziele erfüllen kann, die Planung eines neuen Logistikkonzepts darstellte. Die Anzahl der Mitarbeiter sowie die bestehende Struktur der Gebäude- und Produktionslinien sollten unverändert bleiben. Es wurde geprüft, ob die Fertigungslinien und die jeweilige Ausrüstung die neuen Produkte technisch herstellen können. Da dies garantiert werden konnte, lag der Fokus auf der zu verändernden Logistik. Die Lagertypen sowie die Fördermittel einschließlich der entsprechenden Verbindungen zu den jeweiligen Betriebsmitteln der Produktionslinien mussten neu gestaltet werden. Nachdem beschlossen wurde, ein neues Groblogistikkonzept zu entwickeln, wurden alle relevanten Parameter ermittelt, um die Ziele mit dem neuen Konzept erreichen zu können (siehe Abb. 5.1). Die Hauptparameter waren die Kapazität der Speichertypen, der benötigte Platz im Layout und die Kosten für jedes Logistikelement. Basierend auf diesen Parametern konnten drei grobe Lösungsvarianten generiert werden, die sich im finanziellen Aufwand neben anderen Besonderheiten unterschieden.

Für den Algorithmus wurden individuelle Parameter wie der Lagertyp (z.B. Kanban-Regal oder Hochregallager) sowie Transportmittel (z.B. klassische Gabelstapler oder fahrerlose Transportfahrzeuge) identifiziert und analysiert. Diese Parameter wurden in kleine Gruppen mit ihren individuellen Attributen wie Kostensätzen, Flächenbedarf oder Kapazitäten unterteilt, so dass diese Werte in eine Form übertragen werden konnten, die vom Algorithmus verarbeitet werden kann (siehe Abb. 5.1). Durch die Aufgliederung der einzelnen Lösungselemente können einzelne Handlungsoptionen in viele kleinere Elemente zerlegt werden. Dies sollte eine vielfältigere Generierung von Lösungsvarianten ermöglichen, ohne durch den Planer und den Lösungsraum seines Teilnehmers eingeschränkt zu werden. Darüber hinaus sollten die jeweiligen Attribute jedes Elements eine detailliertere

Zusammenstellung nach bestimmten Kriterien wie Kosten oder belegtem Platz in der Fabrik ermöglichen.

5.1.2 Implementierung

In diesem Abschnitt wird der zuvor vorgestellte Anwendungsfall als praktischer Rahmen für die Bewertung des Ansatzes zur automatisierten Zusammenstellung von Fabrikkonfigurationen verwendet. Das Szenario eignet sich gut zur Veranschaulichung der Prozesssynthese, da es gut strukturiert ist und inhärente Variabilität aufweist. Die Eingänge für die (CL)S stehen als begleitender Download zur Verfügung[11]. Ziel ist es, automatisiert zu zeigen, welche Konfigurationsoptionen für den Anwendungsfall verwendet werden können und unter welchen Bedingungen sie implementiert werden können. Um die Variabilität und die zahlreichen unterschiedlichen Konfigurationsmöglichkeiten im vorliegenden Szenario abbilden zu können, wurde das Szenario in ein Feature-Modell umgewandelt (Abbildung 5.1). Feature-Modelle werden durch Feature-Diagramme visualisiert und im gesamten Produktlinien-Entwicklungsprozess verwendet. Das Modell definiert die Merkmale, ihre Eigenschaften sowie ihre Abhängigkeiten, die sich im Diagramm widerspiegeln.

Das Modell zeigt die vorhandenen Variationsmöglichkeiten. Beispielsweise können verschiedene Transportsysteme kombiniert werden. Jedes System verfügt über individuelle Merkmale wie spezifische Kosten, Durchsatz oder Platzbedarf. Im nächsten Schritt wurde das Feature-Modell zur Synthese in Kombinatoren umgewandelt. Die einzelnen Kombinatoren sind in Abbildung 5.2 dargestellt. Der Name des Kombinators folgt der Struktur „Präfix“ + „Name des auszuwählenden Merkmals“. Mögliche Präfixe sind „storageSelector“ und „transportSelector“, um den entsprechenden Zweig des Feature Model Tree anzuzeigen. Der Kombinierer für die Auswahl eines AutoStorage-Systems wird daher als „Storage SelectorAutoStore“ bezeichnet. Grundsätzlich gibt es für jeden Eintrag im Feature-Tree einen Kombinator, dessen Signatur durch verschiedene Formen seiner Kindknoten ausgefüllt werden kann. Daher benötigt der vorhandene Kombinator „Configuration“ einen Ausdruck des Transport- und Speicherselektors, um ausgeführt werden zu können. Dies entspricht dem Eintrag BAConfiguration im Feature Model.

Exemplarisch kann man auf die beiden Kombinatoren *addTransport* und *addStorage* hinweisen, die die Anzahl der verwendeten Transport- oder Speichersysteme erhöhen sollen. Wenn beispielsweise zwei Gabelstapler verwendet werden sollen, wird der Kombinator *selectTransport(ForkLift)* zweimal ausgeführt und dann mit dem Kombinator *addStorage* zu einem Transportsystem kombiniert. Bei dieser Konstruktion gibt es jedoch ein Problem:

[11] `https://james.cs.tu-dortmund.de/smjawink/CLS-FactoryConfig`

Abbildung 5.2: Kombinator Repository

Da die Beschränkungen erst nach Abschluss des Synthesealgorithmus geprüft werden, kann es theoretisch vorkommen, dass der *addStorage*-Kombinator unendlich oft benutzt wird, was zu einer unendlichen Anzahl von Lösungen führt. Um dies zu verhindern, wurde die maximale Tiefe der resultierenden Baumgrammatik in der Implementierung begrenzt.

Die Inhabitation wird mit einem Aufruf der Form ausgeführt:
$lazy\ val\ resultsFromRequests: Results =$
$Results.add(Gamma.inhabit[Form](`FactoryConfig\ (`AutoStore))$

Das bedeutet, dass der Algorithmus gefragt wird, ob es möglich ist, aus dem angegebenen Repository C eine Lösung zu generieren, die die erforderlichen Spezifikationen erfüllt. Im obigen Beispiel wird die Verwendung eines AutoStore-Systems explizit angegeben. Für das Experiment wurden im vorliegenden Szenario keine Einschränkungen dieser Art festgelegt, um möglichst viele mögliche Lösungen zu ermöglichen. Alle anderen Einschränkungen wurden direkt aus dem gegebenen Szenario übernommen, um die Lösungen auf dieses Szenario anwendbar zu machen.

Das Lösungsset wird dann in der Weboberfläche angezeigt, auf der der Benutzer die Lösungen anzeigen und bewerten kann. Das Web-Interface mit den Lösungen des Versuchslaufs ist unten zu sehen.

Im realen Anwendungsfallszenario wurde die folgende Konfiguration als Lösung ausgewählt: Als Transportsystem wurde ein FTS-System festgelegt, das Lagersystem war ein

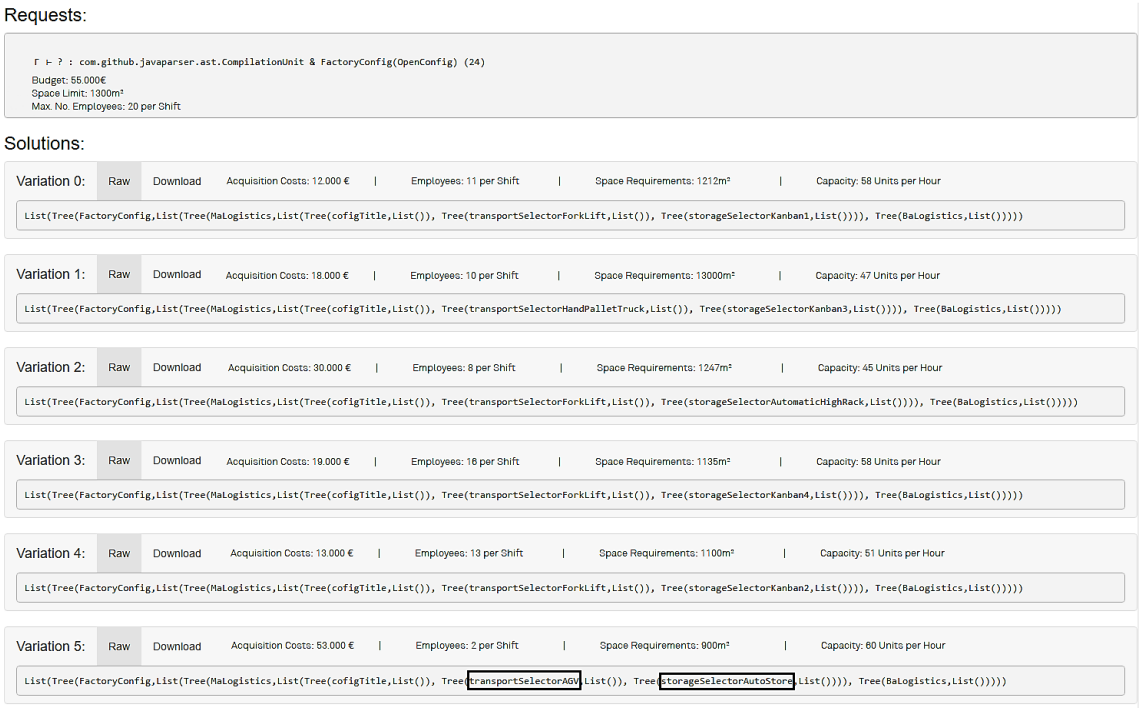


Abbildung 5.3: Liste der generierten möglichen Fabrikkonfigurationen

AutoStore-Rack. Wie Abb. 5.3 zeigt, erscheint diese Lösung auch im Lösungssatz des Algorithmus (Lösungsnummer 5 von 24). Darüber hinaus sind weitere alternative Konfigurationen zu sehen. Diese generierten Lösungen unterscheiden sich in Bezug auf verschiedene Maßnahmen, beispielsweise in Bezug auf die Kapazität oder die Anzahl der Mitarbeiter pro Schicht. Im Anwendungsfall scheint die Verwendung eines Gabelstaplers als Transportmittel billiger zu sein, bietet aber auch eine geringere Transportkapazität pro Stunde. Welche Konfiguration am Ende ausgewählt wird, hängt auch davon ab, wie die Verantwortlichen die einzelnen Parameter gewichten, welche persönlichen Präferenzen sie haben und welche am besten zu dem jeweiligen Fall passen. Insgesamt zeigt sich, dass der Ansatz nicht nur alle möglichen Lösungen bietet, sondern sie auch direkt in Bezug auf wichtige Kennzahlen bewertet. Auf diese Weise sehen die Verantwortlichen nicht nur direkt, welche Optionen zur Verfügung stehen, sondern werden auch direkt auf die jeweiligen Vor- und Nachteile hingewiesen. Eine schnelle und fundierte Entscheidung findet so eine qualifizierte Grundlage.

5.1.3 Zusammenfassung

Es hat sich gezeigt, dass mit dem beschriebenen Ansatz plausible und nützliche Lösungen für die Zielplanung gefunden werden können. Durch das Angebot einer Vielzahl unterschiedlicher Lösungen und einer großen Variabilität verschiedener Lösungsaspekte können

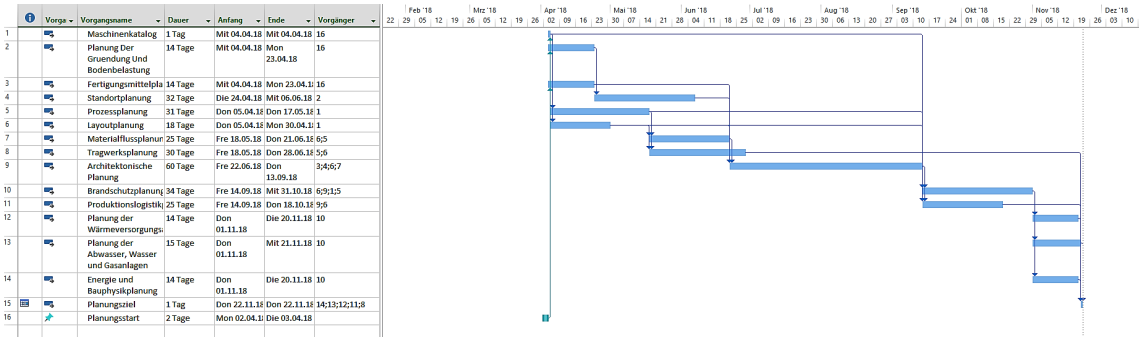
Planer auf der Grundlage der generierten Vorschläge schnelle und zuverlässige Entscheidungen treffen und so den Planungsprozess vorantreiben. Darüber hinaus ist es möglich, den folgenden detaillierteren Planungsprozess zu unterstützen, indem auf der gewählten Lösungsvariante bestimmte Anweisungen gegeben werden. Dadurch kann auch der potenzielle Lösungsraum für den gesamten Detailplanungsprozess reduziert werden, so dass anschließend ein genauerer Planungsprozess generiert werden kann.

Es zeigt sich aber auch, dass es verschiedene Anhaltspunkte gibt, die Technologie weiter zu verbessern. Zu nennen sind zum Beispiel die Unterstützung komplexerer Entscheidungen und anderer Aspekte des Planungsprozesses, wie die Ressourcenplanung oder die Layoutplanung durch das Verfahren. Darüber hinaus können andere Bereiche, in denen Planungsworkflows zur Verwaltung komplexer Planungsprojekte erforderlich sind, mit dem vorgestellten Ansatz behandelt werden. Dies wird in den folgenden Experimenten in diesem Kapitel weiter evaluiert.

5.2 Neubau eines Produktionswerkes für eine Werkszusammenlegung

In diesem Abschnitt soll das Verfahren eingesetzt werden, um einen Netzplan für ein konkretes Planungsprojekt aus der Praxis zu erzeugen. Grundlage des Experiments ist dabei ein Planungsprojekt eines Fertigungsunternehmens aus Süddeutschland, in dessen Rahmen zwei bestehende Werke in einer neu zu bauenden Fabrik an einem neuen Standort zusammengeführt werden sollen. Für den Bau der neuen Fabrik müssen insbesondere die Standortwahl, die Gestaltung des Layouts der neuen Anlage sowie diverse bauliche Aspekte berücksichtigt werden. Diese Planungsschritte mussten insbesondere durchgeführt werden, um die für die Baugenehmigung benötigten Unterlagen und Informationen zu generieren. Als Zeitrum für die Planungsaufgaben wurden 33 Wochen veranschlagt; zu finanziellen Rahmenbedingungen konnten keine Angaben gemacht werden.

Das mit der Durchführung der Planung beauftragte Planungsbüro hat einen Projektplan erstellt, der in Abbildung 5.4 dargestellt ist und den zeitlichen Ablauf und die Terminierung der einzelnen Planungsphasen und Aufgaben veranschaulicht. Dieser Plan dient im weiteren Verlauf als Referenzplan für die generierten Lösungen. Das Ziel des Experiments ist es, einen für das Projekt passenden Plan mit Hilfe des in Kapitel 4 Tools zu erzeugen. Die generierten Pläne werden dann mit dem Referenzplan verglichen. Als Vergleichskriterien dienen dabei:



	Vorga	Vorgangsname	Dauer	Anfang	Ende	Vorgänger
1		Maschinenkatalog	1 Tag	Mit 04.04.18	Mit 04.04.18	16
2		Planung Der Gruendung Und Bodenbelastung	14 Tage	Mit 04.04.18	Mon 23.04.18	16
3		Fertigungsmittelpla	14 Tage	Mit 04.04.18	Mon 23.04.1	16
4		Standortplanung	32 Tage	Die 24.04.18	Mit 06.06.18	2
5		Prozessplanung	31 Tage	Don 05.04.18	Don 17.05.18	1
6		Layoutplanung	18 Tage	Don 05.04.18	Mon 30.04.1	1
7		Materialflussplanun	25 Tage	Fre 18.05.18	Don 21.06.18	6;5
8		Tragwerksplanung	30 Tage	Fre 18.05.18	Don 28.06.18	5;6
9		Architektonische Planung	60 Tage	Fre 22.06.18	Don 13.09.18	3;4;6;7
10		Brandschutzplanung	34 Tage	Fre 14.09.18	Mit 31.10.18	6;9;1;5
11		Produktionslogistik	25 Tage	Fre 14.09.18	Don 18.10.18	9;6
12		Planung der Wärmeversorgungs	14 Tage	Don 01.11.18	Die 20.11.18	10
13		Planung der Abwasser, Wasser und Gasanlagen	15 Tage	Don 01.11.18	Mit 21.11.18	10
14		Energie und Bauphysikplanung	14 Tage	Don 01.11.18	Die 20.11.18	10
15		Planungsziel	1 Tag	Don 22.11.18	Don 22.11.18	14;13;12;11;8
16		Planungsstart	2 Tage	Mon 02.04.18	Die 03.04.18	

Abbildung 5.4: Referenzplan des Anwendungsfalls

1. Die **Planungszeit:** Es soll ermittelt werden, ob durch den automatischen Ansatz Pläne erzeugt werden können, die die Planungsaufgaben (z.B. durch Paralellisierung) in weniger als den im Referenzplan benötigten 33 Kalenderwochen durchführen können.

2. Die **inhaltliche Vollständigkeit und Korrektheit:** Es wird überprüft, ob die generierten Lösungen alle zuvor spezifizierten Planungsziele erfüllen.

Die Arbeitsschritte, die in diesem Referenzplan dargestellt sind, lassen sich alle durch Module aus dem Basis-Repository aus Abschnitt 4.3 darstellen. Dadurch ist die inhaltliche Vergleichbarkeit der generierten Lösungen zum Referenzplan gegeben. Eine Aufstellung der verwendeten Module im Referenzplan zeigt Tabelle 5.1. Es sei allerdings erwähnt, dass die Erzeugung von Plänen auf Basis des gesamten Repositories erfolgt. Die Auswahl der in den generierten Plänen verwendeten Module erfolgt durch den Inhabitationsalgorithmus, wodurch es später theoretisch zu Abweichungen zur Modulauswahl des Referenzplans kommen kann.

Wie bereits zuvor erwähnt, müssen vor der automatischen Erzeugung von Planungsworkflows die Projektrahmenbedingungen spezifiziert werden. Als konkrete Planungsziele wurden im Projekt benannt:

1. Abgeschlossene Tragwerksplanung des neuen Gebäudes

2. Abgeschlossene Produktionslogistikplanung des neuen Standortes auf Basis des Maschineneinkatalogs der aufzulösenden Standorte

3. Abgeschlossene Planung der Abwasser, Wasser und Gasanlagen

4. Abgeschlossene Energie und Bauphysikplanung

5. Abgeschlossene Planung der Wärmeversorgungsanlagen

Insbesondere die letzten drei Planungsziele waren für eine erfolgreiche Einholung der Baugenehmigung unerlässlich. Weiterhin soll berücksichtigt werden, dass am neuen Standort die Produktionsmaschinen der alten Werke verwendet werden. Der Maschinenkatalog

Nr.	Modulbezeichnung
1.	Prozessanalyse
2.	Logistikplanung
3.	Layoutplanung
4.	Standortauswahl
5.	Planung der Gründung und Bodenbelastung
6.	Architektonische Planung
7.	Produktionsmittelplanung
8.	Logistikplanung
9.	Layoutplanung
10.	Tragwerksplanung
11.	Brandschutzplanung
12.	Planung der Wärmeversorgungsanlagen
13.	Planung der Abwasser, Wasser und Gasanlagen
14.	Energie und Bauphysikplanung
15.	Architektonische Planung

Tabelle 5.1: Verwendete Module im Referenzplan

muss demnach nicht durch Abarbeitung eines Moduls erzeugt werden, sondern liegt bereits im Projekt vor. Das Basis-Repository aus Abschnitt 4.3 sieht bereits für jedes der Planungsziele ein Modul vor. Dadurch ist es nicht notwendig, das Repository um inhaltliche Module zu erweitern, um die Planungsziele abbilden zu können. Für das Experiment wurden dennoch zwei Module hinzugefügt, um die Projektrahmenbedingungen korrekt abbilden zu können. Zum einen wurde ein Modul *Planungsziel* hinzugefügt, der alle Module, die Planungsziele erfüllen (zum Beispiel das Modul Tragwerksplanung), als Input vereint. Wird eine Syntheseanfrage mit dem Zielmodul *Planungsziel* gestellt, ist gesichert, dass alle Planungsziele berücksichtigt werden. Die Scala-Implementierung des dazugehörigen Kombinators zeigt Abbildung 5.5. Außerdem wurde ein Modul *Maschinenkatalog* hinzugefügt, welches über keinerlei Vorbedingungen verfügt. Damit wird dem Umstand Rechnung getragen, dass diese Information bereits vorhanden ist.

```
@combinator object Planungsziel{
  def apply(EnergieUndBauphysikplanung: String,
            PlanungDerAbwasserWasserUndGasanlagen: String,
            PlanungDerWaermeversorgungsanlagen: String,
            Produktionslogistikplanung: String,
            Tragwerksplanung: String): String = "Planungsziel"

  //insert Attributes and Graphcode Here
  val Duration = 0
  val Costs = 0
  val Res = 0
  val semanticType: Type = 'EnergieUndBauphysikplanung =>: 'PlanungDerAbwasserWasserUndGasanlagen =>: 'PlanungDerWaermeversorgungsanlagen =>: 'Produktionslogistikplanung =>: 'Tragwerksplanung =>: 'Planungsziel

}
```

Abbildung 5.5: Implementierung des Planungsziel-Kombinators

Auf Basis dieses modifizierten Repositorys wurde anschließend eine Syntheseanfrage mit dem Zieltypen *Planungsziel* gestellt. Als Constraints wurde lediglich die maximale Laufzeit von 33 Wochen angegeben. Wie Abbildung 5.6 zeigt, konnte eine Lösung erzeugt werden, die den gewünschten Anforderungen entsprach. Auf dem eingesetzten Rechner[12] nahm der Generierungsprozess 46 Sekunden in Anspruch. Der Netzplan dieser Lösung ist in Abbildung 5.7 dargestellt. Wie zu erkennen ist, werden auch für die Bearbeitung dieses Plans insgesamt 33 Wochen benötigt. Ebenso ist zu sehen, dass die generierte Lösung die selben Module (bzw. Arbeitsschritte) wie der Referenzplan verwendet. Auch sind die Arbeitsabläufe weitestgehend identisch.

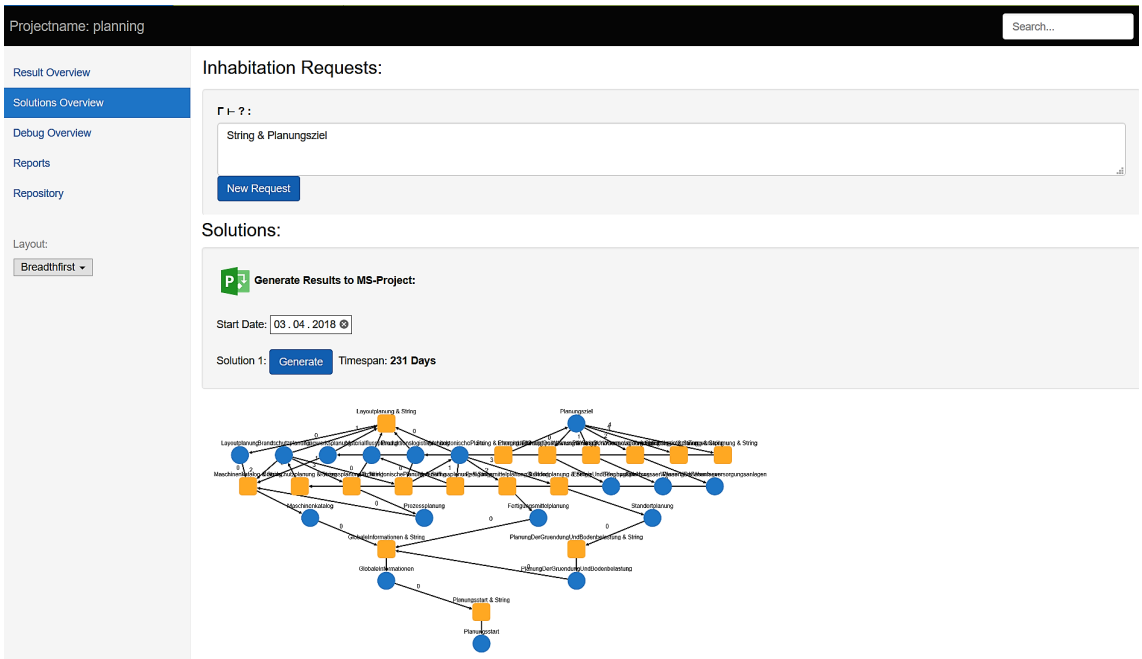


Abbildung 5.6: Generierte Planvarianten im (CL)S nach dem Constraintsolving

Alle Planungsziele finden im Planungsverlauf Berücksichtigung und werden erfüllt. Etwaige Vorarbeiten (wie die Brandschutzplanung) wurden ebenfalls durchgeführt, sodass inhaltliche Abhängigkeiten verschiedener Planungsaufgaben in den Plan mit eingeflossen sind. Im Zuge der Planung des beschriebenen Projekts bietet die automatisierte Fabrikplanung folglich die Möglichkeit einer einfachen und vor allem schnelleren Zusammenstellung und übersichtlichen Darstellung von Planungsaufgaben sowie ihrer Wirkbeziehungen. Die Visualisierung des Projekts in Form eines Netzplans führt zu einer ganzheitlichen und übersichtlichen Darstellung sämtlicher relevanter Projektinhalte.

[12] Dell Latitude 5491, Intel Core i7-8850H 2,6GHZ, 8GB RAM

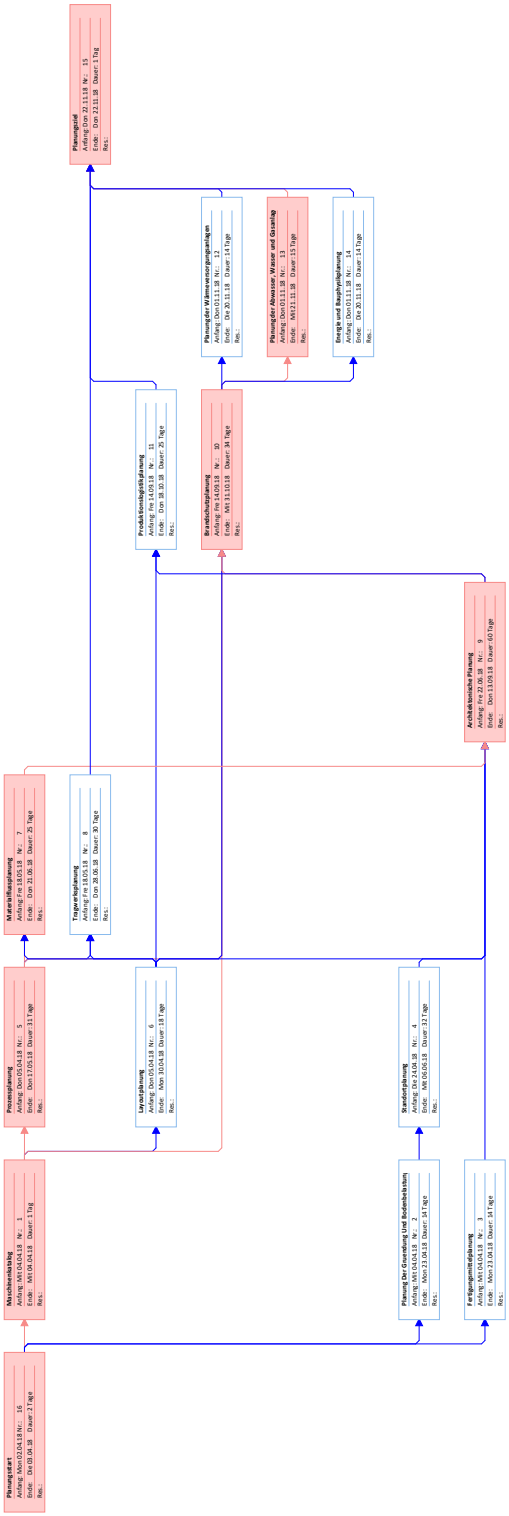


Abbildung 5.7: Generierter Netzplan

Dass die generierte Lösung zudem identisch mit dem Referenzplan ist, zeigt, dass das System in der Lage ist, auf Basis der Modulsammlung praxistaugliche Lösungen zu erzeugen.

5.3 Neubau einer Fabrik mit Umplanung während des Projektes

In diesem Fallbeispiel stellt der Neubau einer Fabrik in Sachsen, durchgeführt von einem deutschen Technologieunternehmen, die Grundlage dar. An dem neu erschlossenen und errichteten Standort entstand im Zeitraum von Juli 2013 bis September 2014 eine Produktionseinrichtung mit angeschlossenem Logistikzentrum und Lagerhallen. Innerhalb des Fallbeispiels wird der Fokus auf die Herausforderungen „Turbulenz und Dynamik“ während der Planung und Durchführung des Projektes gelegt. Zu Startzeitpunkt des Referenzplans wurde bereits eine von zwei Produktionshallen errichtet. Die Planungsgrundlagen, wie bspw. das zugrundeliegende Produktionsprogramm, waren zu Projektbeginn jedoch noch unklar. Darüber hinaus konnten zu Beginn der Planung noch nicht alle erforderlichen Planungsaufgaben vorhergesagt werden, da insbesondere die Bodenbeschaffenheit auf Teilen des Baugrundstücks im Vorfeld nicht ausreichend untersucht werden konnte. Dies hatte zur Folge, dass während der Projektdurchführung Planungsaktivitäten mehrfach angepasst und die Modulreihenfolge, verändert werden mussten und dass der finale Netzplan des Projektes nicht nur zahlreiche Iterationsstufen durchlaufen hatte, sondern dass sich dieser auch sehr umfangreich und komplex gestaltete.

Das Experiment verfolgt im Wesentlichen zwei Ziele: Zum einen soll die Mächtigkeit des Planungsansatzes dieser Arbeit auch im Hinblick auf entsprechend große und komplexe Pläne nachgewiesen werden. Es wird demnach versucht, Pläne von der Größe des Referenzplans (insgesamt 115 Knoten) automatisch zu erzeugen. Zum anderen soll die Fähigkeit des Systems, Pläne während des Projektverlaufes anpassen zu können evaluiert werden.

Zur Erreichung des ersten Ziels wird zunächst (analog zur Vorgehensweise des Fallbeispiels aus Kapitel 5.2) versucht, gültige Varianten des Netzplans des Projektes zu generieren und diese anschließend mit dem Referenzplan zu vergleichen. Der Referenzplan ist in Abbildung 5.8 dargestellt. Im zweiten Schritt wird ein Umplanungsfall des Projektes nachgestellt: Im realen Projektverlauf kam es während der Bearbeitung des Arbeitsschrittes *Geländeprofilierung Cut & Fill* zu Problemen, da die Beschaffenheit des Bodens eine plangemäße Geländeprofilierung nicht möglich machte. Dies führte dazu, dass die Geländeprofilierungstechnik geändert und die Modulreihenfolge angepasst werden musste um

mit möglichst wenig Zeitverzug weiterarbeiten zu können. Dieser Umstand soll mit dem Planungssystem nachgebildet werden.

5.3.1 Phase 1: Erzeugung von Planvarianten

Wie bereits erwähnt besteht das Projektziel in der Errichtung einer Produktionshalle mit angeschlossener Lagerfläche und Logistikzentrum. Zu Beginn der Planung durch das Planungsteam, welches den Referenzplan erstellt hat, war eine der beiden Produktionshallen bereits fertigstellt. Ein Layoutplan für das Gelände existierte ebenfalls bereits, ebenso wurden, bedingt durch den schon abgeschlossenen Bauabschnitt bereits einige Erd- und Entwässerungsarbeiten durchgeführt. Der Projektzeitraum (inkl. Bauzeit) sollte 36 Monate nicht überschreiten. Folgende Planungsziele wurden formuliert:

1. Vollständige Planung und Realisierung der technischen Gebäudeausrüstung (TGA).
2. Übergabefertiger Innenausbau der neuen Halle.
3. Erfolgreiche Behördliche Abnahme der Brandschutzplanung.

Bei Betrachtung des Referenzplans ist zu erkennen, dass für die einzelnen groben Bauphasen Arbeitsschritte formuliert wurden, für die sich im Basis-Repository passende Module finden. Allerdings wurden zwei davon zur genaueren Planung mit weiteren Arbeitsschritten präzisiert. Dies macht es notwendig, Submodule für die im Plan verwendeten Module zu definieren. Die Aufstellung der im Plan hinzugefügten Submodule zeigt Abbildung C.1 in Anhang C. Die Anbindung der Submodule an die jeweiligen Module erfolgte durch Modifikation der „Obermodule“. Deren Vorbedingungen wurden so erweitert, dass jedes Modul nun sämtliche Submodule als Vorbedingung aufweist. Ein Modul kann demnach erst abgeschlossen werden, wenn all seine Submodule ebenfalls abgeschlossen wurden. Abbildung 5.9 zeigt die Formulierung dieses Vorgehens in der Typsignatur des jeweiligen Kombinators.

Analog zum Vorgehen in Kapitel 5.2 wurde auch hier ein Modul zur Repräsentation der Planungsziele sowie mehrere Module zur Darstellung der bereits vorhandenen Informationen erstellt und dem Repository hinzugefügt. Erneut wurde eine Syntheseanfrage mit dem Zieltypen *Planungsziel* (siehe Abbildung 5.10) gestellt. Als Projektconstraints wurde die fixierte Maximallaufzeit des Projektes von 36 Monaten angegeben. Für die Berechnung der Lösungen benötigte der Testrechner 4 Minuten und 14 Sekunden. Insgesamt konnten drei Planvarianten erzeugt werden, die in Anhang C dargestellt sind. Anders als in Kapitel 5.2 konnte dieses Mal allerdings keine exakte Kopie des Referenzplans generiert werden.

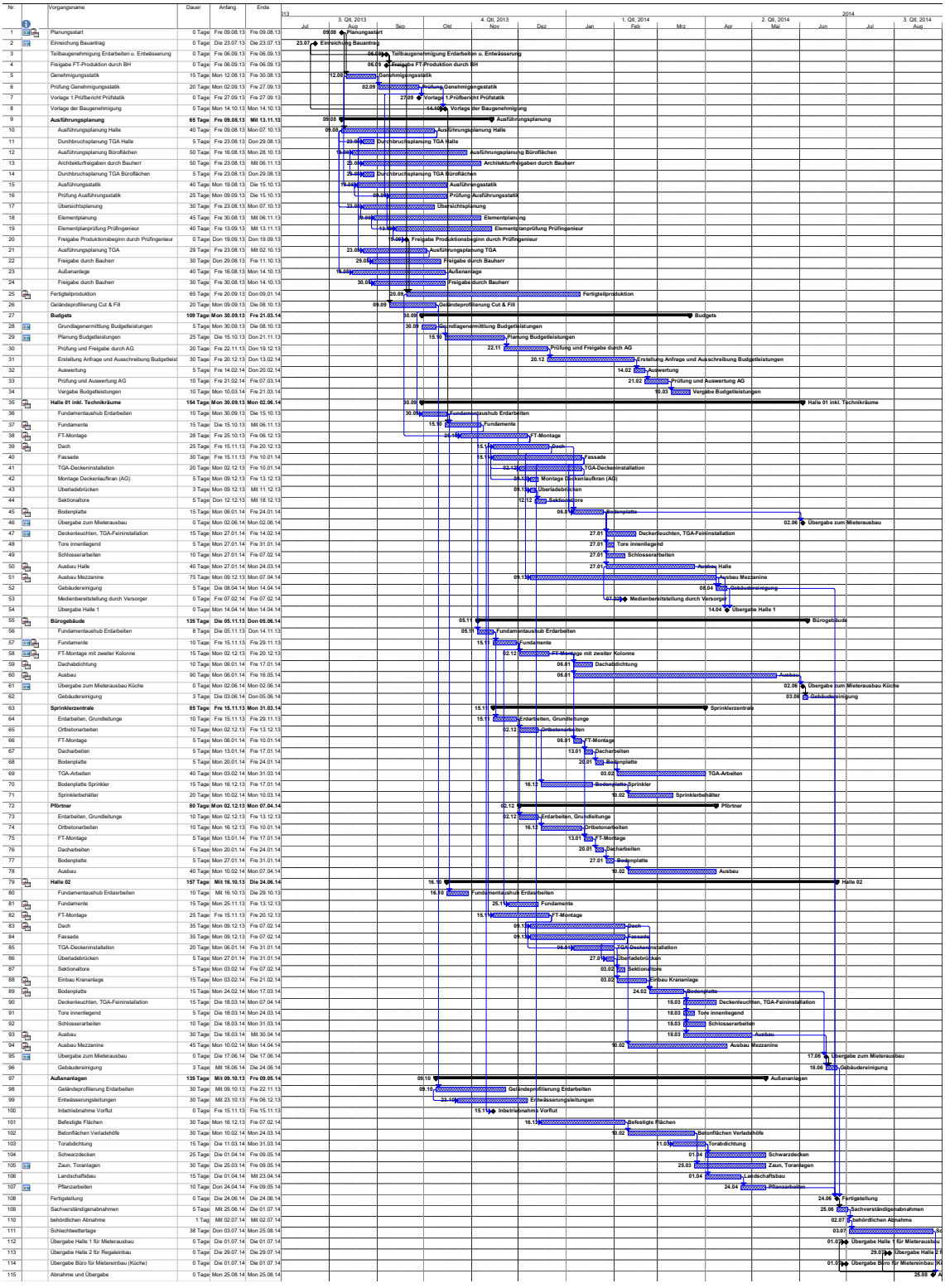


Abbildung 5.8: Referenzplan des Anwendungsfalls

Keine der erzeugten Lösungen überschreitet die 36 Monate Projektlaufzeit, Variante 2 kommt sogar mit nur 34 Monaten aus. Dies ist in sofern bemerkenswert, als dass in

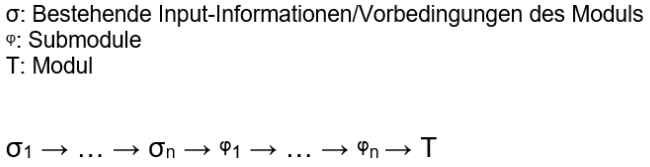


Abbildung 5.9: Integration von Submodulen in ein Obermodul

```
@combinator object Planungsziel{
def apply(Innenausbau: String,
          AbnahmeBehoerde: String,
          Pruefstatik: String
         ):String = "Planungsziel"

//insert Atrributes and Graphcode Here
//Duration = 0
//Costs = 0
//Res = Null
val semanticType: Type = 'Innenausbau =>: 'AbnhameBehoerde =>: 'Pruefstatik =>: 'Planungsziel
}
```

Abbildung 5.10: Implementierung des Planungsziel-Kombinators

dieser Variante die gleichen Module und Submodule verwendet wurden wie im Referenzplan. Allerdings wurden vor allem im Abschnitt der Budgetplanung mehrere Vorgänge parallel durchgeführt, was eine Zeitersparnis zur Folge hat (siehe Abbildung 5.11). Diese Parallelisierung findet in Variante 3 ebenfalls statt, allerdings wurden hier im Abschnitt der Ausführungsplanung andere, zeitaufwändigere Module gewählt, sodass der Zeitgewinn wieder egalisiert wurde.

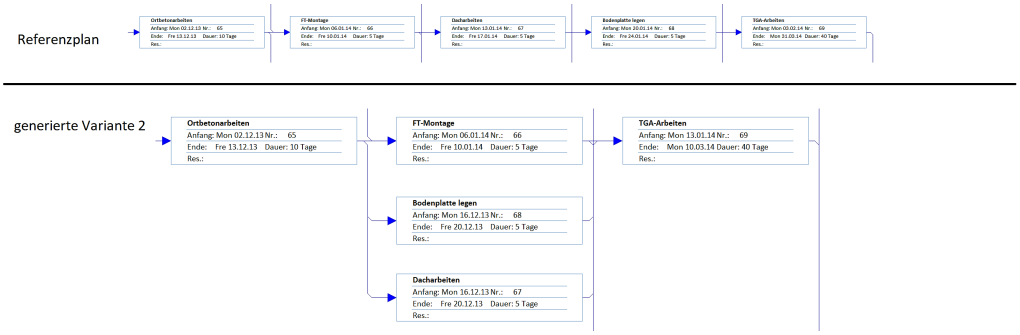


Abbildung 5.11: Paralellisierung im Netzplan

Zwar ist keiner der drei Varianten zu 100 Prozent identisch mit dem Referenzplan, weite Teile der Vorgangsabfolge sind allerdings dennoch gleich. Unterschiede finden sich in der Modulauswahl einzelner Vorgänge sowie der Anordnung und Parallelisierung von Modulen. Aus diesem Grund sind die generierten Lösungen durchaus als plausibel und anwendbar zu betrachten.

Dass keine exakte Kopie des Referenzplans erzeugt werden konnte, kann auch durch die zahlreichen Plan-Anpassungen, die in diesem Projekt durchgeführt werden mussten,

begründet werden. Alle drei generierten Varianten liefern jedoch plausible Netzpläne, die die Planungsziele erfüllen und innerhalb der gewünschten Projektlaufzeit bleiben. Im Falle der zweiten Lösung konnten durch eine alternative Anordnung der Arbeitsschritte sogar zwei Monate eingespart werden. Somit war diese Phase des Experiments erfolgreich.

5.3.2 Phase 2: Umplanung im laufenden Projekt

In der zweiten Phase des Experiments soll die Fähigkeit des Systems, Pläne im laufenden Projekt umzustrukturieren, verifiziert werden. Dafür wurde ein im realen Projekt eingetretener Anlass zur Umplanung dargestellt. Wie bereits dargelegt, traten während der Bearbeitung des Moduls *Geländeprofilierung Cut & Fill* Probleme auf. Die Beschaffenheit des Bodens des Baugrundstücks ließ sich aufgrund geologischer Besonderheiten im Vorfeld vollständig analysieren. Cut and Fill bezeichnet eine Technik zur großflächigen Erdumlagerung um z.B. Unebenheiten im Gelände auszugleichen. Dabei wird Erdreich an einer Stelle großflächig abgetragen oder ausgehoben (Cut) und an anderer Stelle verteilt (Fill). Bei den Aushubarbeiten des Erdreiches im realen Projekt stellte sich der abzutragende Boden als zu sandig heraus, als dass dieser an anderer Stelle auf dem Baugrundstück als Grundlage hätte dienen können. Die Tragfähigkeit des Bodens wäre nicht mehr in ausreichendem Maße gegeben gewesen.

Im Experiment soll dieser Umstand nun dargestellt werden, indem der Plan neu generiert werden soll, ohne die Struktur des Plans bis zum Modul *Geländeprofilierung Cut & Fill* zu verändern und ohne dass der Fertigstellungstermin des Projektes verändert wird. Als Grundlage des Versuchs wurde, zur besseren Vergleichbarkeit, der Referenzplan gewählt.

Gemäß dem Vorgehen aus Kapitel 4 wurden demnach folgende Modifikationen im Vergleich zu Abschnitt 5.3.1 am Repository vorgenommen:

1. Für sämtliche zum Zeitpunkt der Umplanung in Bearbeitung befindlichen Vorgänge wurden neue Module hinzugefügt, die die jeweiligen Module als Zieltypen aufweisen und über keinerlei Eingabetypen verfügen. Dadurch wird sichergestellt, dass der bereits abgeschlossene Plan nicht erneut berechnet werden muss. Die neuen Module weisen jeweils die Dauer und Kosten auf, die der Plan auf dem Pfad bis zum jeweiligen Modul bereits verbraucht hat, damit die Kennzahlen nicht verfälscht werden.

2. Das Modul *Geländeprofilierung Cut & Fill* wird entfernt, um neue Lösungen dazu zu zwingen, ohne diese Technik auszukommen.

Anschließend wird die Syntheseanfrage aus Abschnitt 5.3.1 bei gleichbleibenden Constraints erneut gestellt. Nach einer Berechnungszeit von 5 Minuten und 23 Sekunden konnte

das System eine Lösung generieren. Die generierte Lösung verwendet anstelle des Moduls *Geländeprofilierung Cut & Fill* das Modul *RC-Fill*, bei dem der Boden mit Reststoffen aus anderen Bauprojekten und Abschnitten verfüllt wird. Dieses Modul benötigt eine 4 Tage längere Bearbeitungszeit, die allerdings durch Parallelisierungen im weiteren Planverlauf wieder ausgeglichen wurden. Diese Parallelisierungen entsprechen dabei denen aus Lösung 2 aus Abschnitt 5.3.1.

Die Ergebnisse zeigen, dass der Ansatz in der Lage ist, auch bestehende Pläne in kurzer Zeit zu modifizieren und somit dem Planer alternative Lösungen zu präsentieren. Es sei erwähnt, dass noch sechs weitere Planvarianten gefunden werden, sobald man den terminlichen Constraint außer Kraft setzt. Diese Varianten benötigen allerdings bis zu 7,5 Monate mehr Bearbeitungszeit.

5.4 Zusammenfassung

Die in diesem Kapitel präsentierten Experimente stehen exemplarisch für verschiedene Planungsfälle, die im Bereich der Fabrikplanung auftreten können. Vorgestellt wurden der Fall einer internen Umplanung mit starken Limitierungen und einem hohen Detaillierungsgrad vorgestellt, der nicht über die generischen Bausteine abgebildet werden konnte (Abschnitt 5.1), ein klassisch kompakter Netzplan zur Neuplanung eines Standorts (Abschnitt 5.2) und eine Umplanung während eines laufenden Projektes (Abschnitt 5.3). Für alle drei Klassen von Anwendungsfällen produzierte der in dieser Arbeit entwickelte Ansatz zufriedenstellende und auch in der Praxis anwendbare Ergebnisse.

Kapitel 6

Zusammenfassung und Ausblick

Die gestiegene Komplexität von Planungsaufgaben, die Zunahme von Turbulenz und Dynamik im Projektverlauf sowie der Zeitdruck durch verkürzte Produkt- und Prozesslebenszyklen stellen die Herausforderungen des modernen Fabrikbetriebs dar, der sich die Fabrikplanung in Zusammenarbeit mit anderen Fachdisziplinen stellen muss. Die Fähigkeit Fabriken schnell planen zu können, wird zu einem immens wichtigen Erfolgsfaktor vor allem für Unternehmen in Hochlohnländern. Im Zuge der Industrie 4.0 und der damit einhergehenden Zunahme der Komplexität von zu planenden Systemen auf der einen Seite sowie der stets komplexeren der Planungsverläufe selbst auf der anderen Seite, benötigen (Fabrik-) Planer immer mehr Zeit, um die Anpassungs- und Neubau-Maßnahmen an Fabriken ausreichend planen zu können. Durch die wachsende Dynamik des Produktionsumfeldes, steht aber gleichzeitig immer weniger Zeit für Planung und Umsetzung zur Verfügung. Die Informatik kann helfen, dieses *Dilemma der Fabrikplanung* zu lösen, indem es intelligente Softwarelösungen zur Beschleunigung und (Teil-)Automatisierung von Planungsaufgaben bereitstellt.

Davon motiviert, war das Ziel dieser Arbeit die Konzeption und Implementierung eines Verfahrens zur automatischen Generierung von Planungsworkflows in Form von Netzplänen auf Basis einer taxonomisch strukturierten Sammlung von Bausteinen. Um dieses Ziel zu erreichen, wurde zunächst die Disziplin der Fabrikplanung vorgestellt. Dazu wurden die grundlegenden Begriffe definiert sowie etablierte Planungsmethodiken, Techniken und Werkzeuge vorgestellt. Es zeigte sich, dass klassische Planungsansätze und Werkzeuge oft zu unflexibel sind, um sich stetig ändernden Anforderungen anzupassen. Modernere, modulare Ansätze tragen diesen Herausforderungen hingegen bereits Rechnung. Nachdem mit dem modularen Fabrikplanungsansatz nach *Nöcker* [109] sowie den darauf aufbauenden Arbeiten z.B. von *Meckelnborg* [97] eine geeignete Planungssystematik ausgemacht und mit der Netzplantechnik ein passendes Planungswerkzeug identifiziert wurden, widmete sich die Arbeit der Ermittlung von Ansätzen aus der Informatik, die die Softwareseitige Implementierung dieser Ansätze unterstützen können.

Hierzu wurde insbesondere auf Ansätze aus der Software-Synthese und des Constraintsolvings eingegangen. Für eine prototypische Umsetzung erwies sich eine Kombination aus dem Synthese-Framework (CL)S sowie eines SMT-Solvers als geeignete Lösung. Die Arbeit hat daraufhin ein System vorgestellt, mit dem sich Planungsworkflows automatisch auf Basis eines Repositories aus wiederverwertbaren Bausteinen erzeugen lassen. Dabei wurde auf bestehende Konzepte der Fabrikplanung (modulare Planung nach Nöcker [109]) und des Software Engineerings (Synthese und SMT-Solving) zurückgegriffen. Als Ergebnis wurde eine prototypische Software vorgestellt, die das Verfahren implementiert und anwendbar macht.

In Anbetracht der Ergebnisse aus Kapitel 5 lässt sich festhalten, dass ein solches Vorgehen in der Lage ist, geeignete Workflows in Form von Netzplänen zu erzeugen. Trotz der standardisierten Basis von generischen Bausteinen, sind die erzeugten Pläne in konkreten (Praxis-) Projekten durchführbar. Sie sind in der Lage, die Gegebenheiten des jeweils einzigartigen Planungsprojektes zu berücksichtigten. Durch gezieltes Hinzufügen oder Entfernen einzelner Module sowie durch Codieren entsprechender Constraints, können beliebige projektspezifische Situationen abgebildet werden.

6.1 Kritische Reflexion

Das Ziel der Arbeit war es, die Fabrikplanung mit einer automatisierten Lösung bei Identifikation, Orchestrierung und Terminierung von Planungsaufgaben zu unterstützen. Durch den modularen Ansatz sowie die einheitliche Formulierung von Interaktionsschnittstellen zwischen den Modulen innerhalb eines Typsystems, ist ein flexibles und in einem breiten Spektrum von Anwendungsfällen einsetzbares System zur Zusammenstellung von Planungsworkflows in Form von Netzplänen entstanden. Das etablierte Planungstool ist praxisnah in zahlreichen Projekten einzusetzen. Verzögerungen im Projektablauf und Probleme beim Informationsaustausch zwischen den Planungsmodulen können in der Netzplantechnik frühzeitig erkannt werden. Diese Arbeit bietet die Technologie an, schnell auf solche Probleme zu reagieren.

Die Entscheidungsfindung und Modellierung von Plänen (auch im turbulenten und dynamischen Produktionsumfeld) wurde verbessert. Sich verändernde Zielvorgaben während der Projektdurchführung können ebenso aufgefangen werden wie andere Parameter, die eine Neuplanung notwendig machen. Zudem wurde dargelegt, dass weitere Module in den Planungsprozess integriert werden können (Kapitel 5.1) und somit auch Planungsfälle abdecken, für dass das Standard Repository keine (oder nur wenige) Bausteine vorsieht. Damit erfüllt die modulare Fabrikplanung auf Basis einer taxonomisch strukturierten Modulsammlung die Anforderung, dass Planungsinhalte kontinuierlich angepasst und ergänzt werden können. Die Planungstiefe einzelner Module kann durch das Hinzufügen von Submodulen zusätzlich verändert werden.

Dadurch, dass der Ansatz Pläne maximal parallelisiert, ist es ebenfalls möglich, etwaige Optimierungsmöglichkeiten in bestehenden Plänen zu erkennen. Darüberhinaus ist auch das Neuberechnen bzw. das Anpassen bestehender Pläne möglich. Dadurch zeigt sich, dass die Automatisierung von Planungsaufgaben auf Basis generischer Module durchaus praktikabel ist.

Der Einzigartigkeit von Fabrikplanungsaufgaben sowie der stark unterschiedlichen Ausgangslagen bei Planungsanlässen, wurde ebenfalls Rechnung getragen. Die Kapselung einzelner Planungsschritte in Module mit definierten Schnittstellen trugen zur Unterstützung bei der Konfiguration der dargestellten Fallbeispiele bei. Die Unabhängigkeit der einzelnen Module ermöglichte es neben den primär ausgewählten Basismodulen, weitere für das Projekt relevante Sub- und Assistenzmodule (z.B. Module zur Formulierung der Planungsziele) zu identifizieren und in die Synthese mit einzubeziehen. Die automatisierte, modulare Fabrikplanung lässt sich somit grundsätzlich auf jeden Planungsanlass anwenden.

Durch das verlässliche System wird es dem Projektplaner ermöglicht, aufwändige Modellierungen beim Erstellen von Netz- und Terminplänen zu automatisieren. Dies verkürzt die Reaktionszeit, die bei einem Anpassungsbedarf in einem Fabriksystem benötigt wird, um eine Anpassungsmaßnahme zu planen und durchzuführen, deutlich. Zum Gesamtziel des Graduiertenkollegs, die Anpassungsintelligenz von Fabriken zu erhöhen, leistet diese Arbeit demnach einen entscheidenden Anteil.

Bezogen auf die software-seitige Ebene ist anzumerken, dass die in dieser Arbeit vorgestellte Umsetzung der automatisierten Komposition von Planungsworkflows zwar sehr gut funktioniert, dennoch aber nur prototypischen Charakter hat. Die Auswahl und Kombination der verwendeten Frameworks und Technologien wurde aus einer rein pragmatisch, funktionsorientierten Sichtweise heraus getroffen. Sicherlich ließe sich das Ziel der Arbeit auch mit anderen Softwareansätzen erreichen. Die Generierung von Plaungsworkflows ließe sich z.B. auch mit Frameworks wie *CINCO* [104] oder *DIME* [27] umsetzen. Auch die vollständige Berechnung von Plänen durch einen Constraintsolver ist denkbar, gleich wenn ein solcher Ansatz als nicht performant beschreiben wurde [110].

6.2 Weiterführende Arbeiten

Selbstverständlich lässt sich das Verfahren auch auf andere Bereiche der Fabrikplanung sowie auch auf andere Domänen übertragen. Denn Kern der vorgestellten Ergebnisse ist eine Architektur, die die Beantwortung der Inhabitationsfrage unter Contstraints ermöglicht. Vereinfacht handelt es sich um eine automatische Zusammenstellung von Modulen unter Einhaltung von sowohl semantischen als auch numerischen oder syntaktischen Einschränkungen, welche sich sehr gut in anderen Bereichen einsetzen lassen. Themen, welche auch im Rahmen des Graduiertenkollegs und der Anpaasungsintelligenz von Fabriken denkbar sind, ist die automatisierte Genereriung von Simulationsmodellen sowie die automatisierte Evaluation von Fabrikplanungskonzepten.

Die Kombination aus (CL)S und SMT-Solving ist dazu geeignet, Produktlinien von Simulationsmodellen zu generieren. Hierbei werden einzelne Prozessbausteine als Bestandteil des Repositorys aufgefasst, deren Abhängigkeiten und Zusammenhänge in der zugehörigen Taxonomie beschrieben werden. Der Inhabitationsalgorithmus fügt die einzelnen Bausteine anschließend dann zu einem gewünschten Prozess zusammen. Die Anwendbarkeit des Ansatzes auf Prozesse wurde in [21] bereits nachgewiesen.

Bei der folgenden Codegenerierung wird aus den inhabitierten Modellen ausführbarer Programmcode erzeugt, ohne dass dieser von einem Entwickler „von Hand“ geschrieben werden muss. Dabei existieren durchaus Ähnlichkeiten zu Compilern, die ebenfalls aus einem zuvor geschriebenen Quellcode maschinenles- und ausführbare Programme generieren. Codegeneratoren setzten allerdings einen Schritt früher an und generieren den Quelltext, den der Compiler entgegennehmen kann. Genau wie gesonderte Programmiersprachen zur Formulierung von Quelltext existieren, gibt es auch Sprachen zur Formulierung von Modellen, aus denen Quelltext erzeugt werden kann.

Die Technologie sollte es auch ermöglichen, zahlreiche Varianten derselben Fabrik automatisch zu planen, zu erzeugen und zu visualisieren. Eine solche Technologie kann nicht nur den Planungsprozess beschleunigen und dem Fabrikplaner eine valide Entscheidungsgrundlage bei Neu- und Umplanungsprojekten liefern, sondern kann darüber hinaus auch die Einbindung anderer Fachdisziplinen in den Planungsprozess vereinfachen. Denkbar sind zum Beispiel Studien zur Arbeitsplatzgestaltung oder zur Verwendung von Assistenzsystemen für Shop-Floor-Mitarbeiter in der Soziologie, die mittels VR-Technologie direkt in der virtuellen Fabrik durchgeführt werden könnten.

Außerdem bietet sich die Erweiterung des automatischen Kompositionsansatzes auf Aspekte der Fabrikplanung an, die über die Konzeption von Planungsworkflows hinausgehen. Denkbar sind Aufgabenfelder wie die Maschinenbelegungsplanung oder die Layout Planung. Im Bereich der Maschinenbelegungsplanung ergeben sich beispielsweise Anknüpfungspunkte an die Arbeiten von *Christin Schumacher* aus der ersten Kohorte des Graduiertenkollegs 2193, in denen sich Varianten von verschiedenen Belegungsplänen erzeugen ließen. Diese können anschließend durch Optimierungsalgorithmen von Frau Schumacher optimiert und evaluiert werden. Das gleiche gilt für die automatische Generierung von Fabriklayouts.

Anhang A

JSON Vorlage

```
{
  "$schema": "http://json-schema.org/draft-07/schema",
  "type": "object",
  "required": [ "planningModules","informationTypes", "modeling"],
  "properties": {
    "planningModules": {
      "type": "object",
      "additionalProperties": false,
      "patternProperties": {
        "^[0-9a-f]{8}-[0-9a-f]{4}-[1-5][0-9a-f]{3}-[89ab][0-9a-f
            ]{3}-[0-9a-f]{12}$": {
          "type": "object",
          "required": [ "name","abbreviation", "inputInformation", "
              outputInformation"],
          "properties": {
            "name": {
              "type": "string"
            },
            "abbreviation": {
              "type": ["string", "null"]
            },
            "inputInformation": {
              "type": "array",
              "items": {
                "type": "string",
                "pattern": "^[0-9a-f]{8}-[0-9a-f]{4}-[1-5][0-9a-f
                    ]{3}-[89ab][0-9a-f]{3}-[0-9a-f]{12}$"
              }
            },
            "outputInformation": {
              "type": "array",
```

Abbildung A.1: JSON Formatvorlage (Seite 1/4)

```
29              "items": {
30                "type": "string",
31                "pattern": "^[0-9a-f]{8}-[0-9a-f]{4}-[1-5][0-9a-f
                     ]{3}-[89ab][0-9a-f]{3}-[0-9a-f]{12}$"
32              }
33            },
34          }
35        }
36      }
37    },
38    "informationTypes": {
39      "type": "object",
40      "additionalProperties": false,
41      "patternProperties": {
42        "^[0-9a-f]{8}-[0-9a-f]{4}-[1-5][0-9a-f]{3}-[89ab][0-9a-f
            ]{3}-[0-9a-f]{12}$": {
43          "type": "object",
44          "required": [ "name"],
45          "properties": {
46            "name": {
47              "type": "string"
48            }
49          }
50        }
51      }
52    },
53    "modeling": {
54      "type": "object",
55      "required": [ "modules","links", "selected"],
56      "properties": {
57        "modules": {
58          "type": "object",
59          "additionalProperties": false,
60          "patternProperties": {
61            "^[0-9a-f]{8}-[0-9a-f]{4}-[1-5][0-9a-f]{3}-[89ab][0-9a-f
                ]{3}-[0-9a-f]{12}$": {
62              "type": "object",
63              "required": [ "moduleId","attributes", "position"],
64              "properties": {
65                "moduleId": {
66                  "type": "string",
67                  "pattern": "^[0-9a-f]{8}-[0-9a-f]{4}-[1-5][0-9a-f
                       ]{3}-[89ab][0-9a-f]{3}-[0-9a-f]{12}$"
68                },
69                "attributes": {
70                  "type": "object",
```

Abbildung A.2: JSON Formatvorlage (Seite 2/4)

```
71                "required": [ "numEmployees","cost", "duration", "
                     custom"],
72                "properties": {
73                  "numEmployees": {
74                    "type": "integer"
75                  },
76                  "cost": {
77                    "type": "integer"
78                  },
79                  "duration": {
80                    "type": "integer"
81                  },
82                  "custom": {
83                    "type": "object",
84                    "additionalProperties": false,
85                    "patternProperties": {
86                      "^[0-9]+$": {
87                        "type": "object",
88                        "required": ["key", "value"],
89                        "properties": {
90                          "key": {
91                            "type": "string"
92                          },
93                          "value": {
94                            "type": "string"
95                          }
96                        }
97                      }
98                    }
99                  }
100               }
101             },
102             "position": {
103               "type": "object",
104               "required": [ "x","y"],
105               "properties": {
106                 "x": {
107                   "type": "integer"
108                 },
109                 "y": {
110                   "type": "integer"
111                 }
112               }
113             }
114           }
115         }
```

Abbildung A.3: JSON Formatvorlage (Seite 3/4)

```
116           }
117         },
118         "links": {
119           "type": "object",
120           "additionalProperties": false,
121           "patternProperties": {
122             "^[0-9a-f]{8}-[0-9a-f]{4}-[1-5][0-9a-f]{3}-[89ab][0-9a-f
                  ]{3}-[0-9a-f]{12}$": {
123               "type": "object",
124               "required": [ "fromModule", "toModule", "informationId"],
125               "properties": {
126                 "fromModule": {
127                   "type": "string",
128                   "pattern": "^[0-9a-f]{8}-[0-9a-f]{4}-[1-5][0-9a-f
                      ]{3}-[89ab][0-9a-f]{3}-[0-9a-f]{12}$"
129                 },
130                 "toModule": {
131                   "type": "string",
132                   "pattern": "^[0-9a-f]{8}-[0-9a-f]{4}-[1-5][0-9a-f
                      ]{3}-[89ab][0-9a-f]{3}-[0-9a-f]{12}$"
133                 },
134                 "informationId": {
135                   "type": "string",
136                   "pattern": "^[0-9a-f]{8}-[0-9a-f]{4}-[1-5][0-9a-f
                      ]{3}-[89ab][0-9a-f]{3}-[0-9a-f]{12}$"
137                 }
138               }
139             }
140           }
141         },
142         "selected": {
143           "type": ["string", "null"]
144         }
145       }
146     }
147   }
148 }
```

Abbildung A.4: JSON Formatvorlage (Seite 4/4)

Anhang B

Überischt Repository

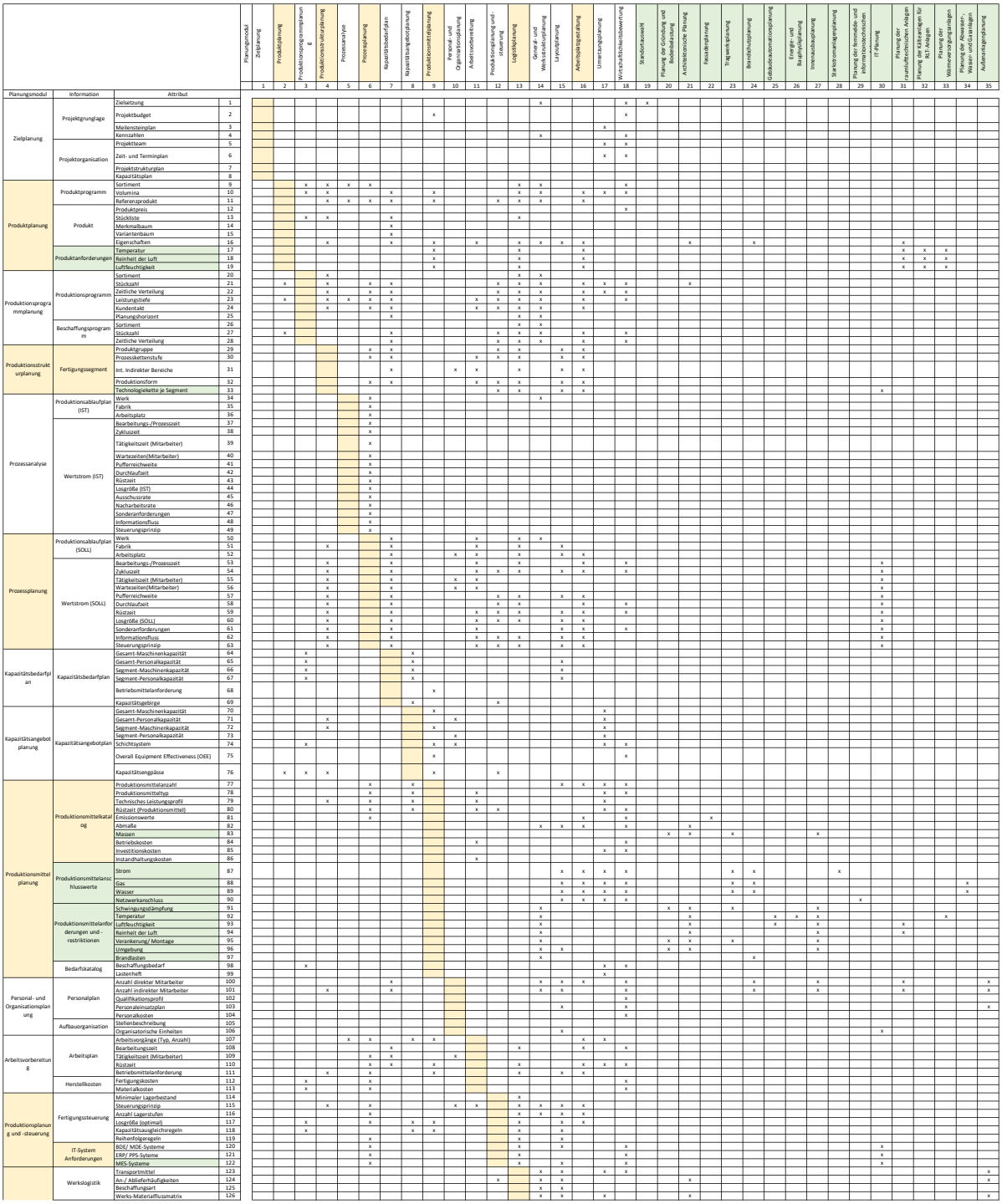


Abbildung B.1: Übersicht Planungsmodule im Basis-Repository (Seite 1)

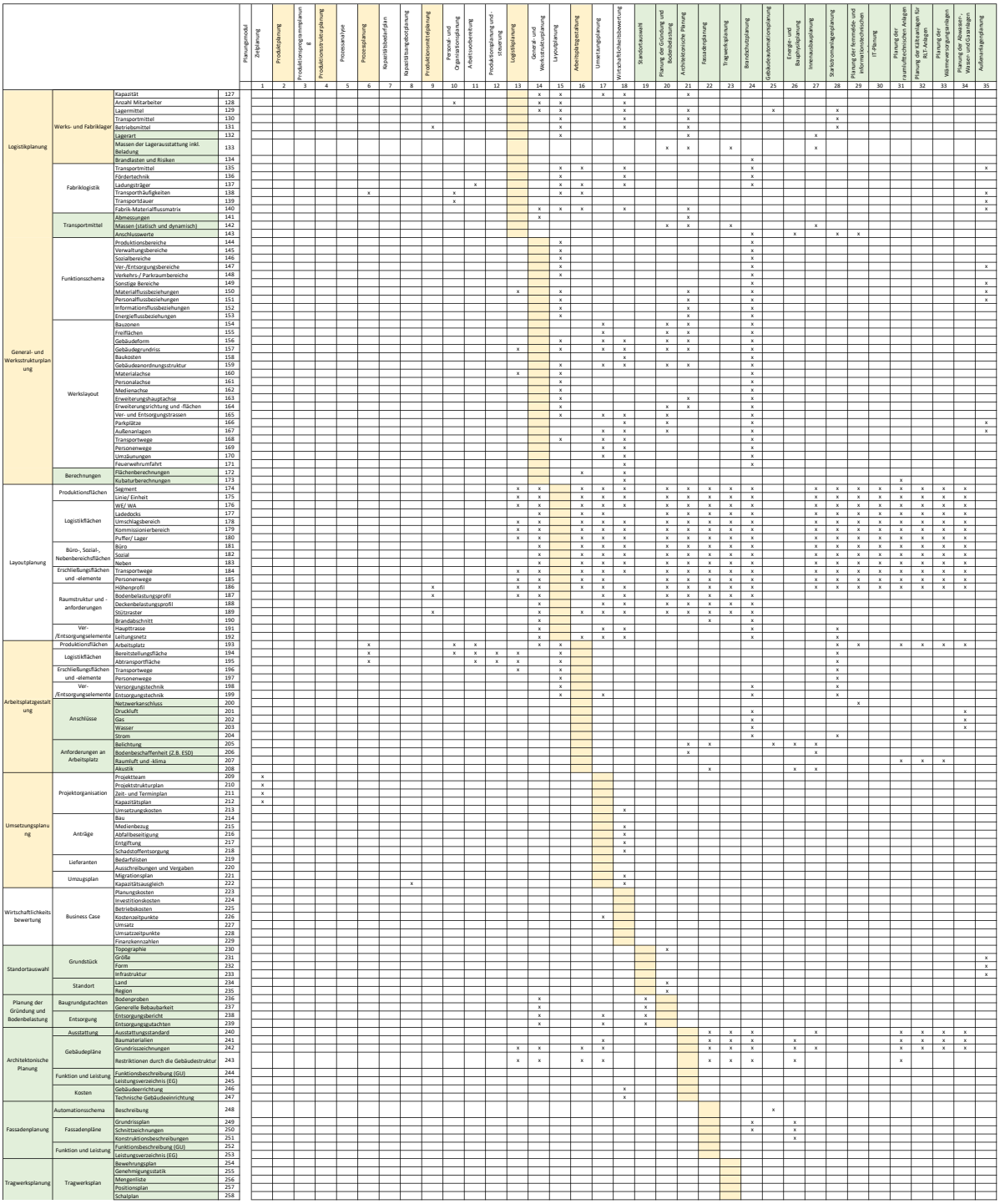
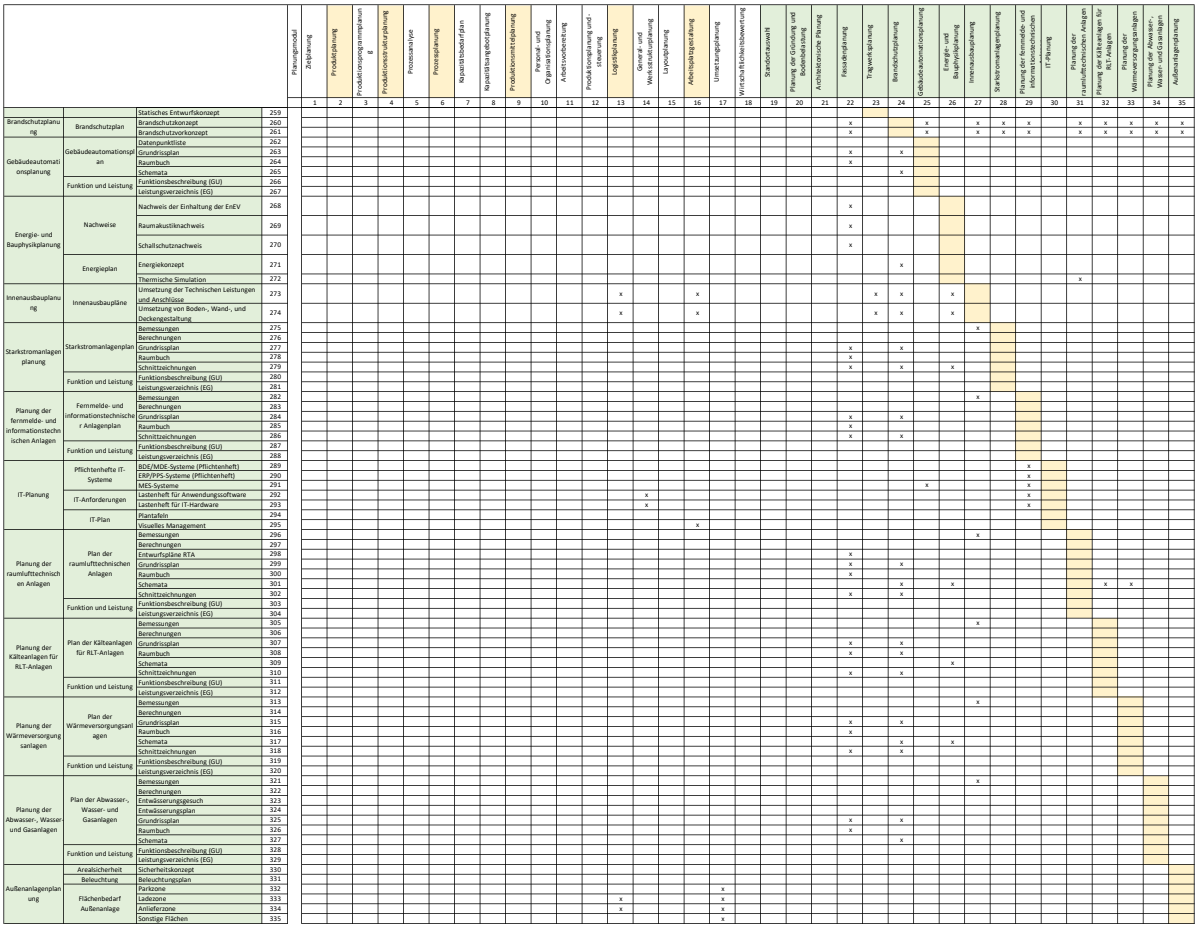


Abbildung B.2: Übersicht Planungsmodule im Basis-Repository (Seite 2)



Planungsmodul	Zeitplanung	Produktplanung	Produktionsprogrammplanung	Produktionsstrukturplanung	Prozessanalyse	Prozessplanung	Kapazitätsbedarfsplan	Kapazitätsangebotsplanung	Produktionsmittelplanung	Personal- und Organisationsplanung	Arbeitsvorbereitung	Produktionsplanung und -steuerung	Logistikplanung	General- und Werkstattplanung	Layoutplanung	Arbeitsplatzgestaltung	Umweltplanung	Wirtschaftlichkeitsbewertung	Standortauswahl	Planung der Gründung und Bodenbeladung	Architektonische Planung	Fassadenplanung	Tragwerksplanung	Brandschutzplanung	Gebäudeautomationsplanung	Energie- und Bauphysikplanung	Innenausbauplanung	Starkstromanlagenplanung	Planung der fernmelde- und informationstechnischen Anlagen	IT-Planung	Planung der raumlufttechnischen Anlagen	Planung der Kälteanlagen für RLT-Anlagen	Planung der Wärmeversorgungsanlagen	Planung der Abwasser-, Wasser- und Gasanlagen	Außenanlagenplanung
		Statisches Entwurfskonzept	259																																	
Brandschutzplanung	Brandschutzplan	Brandschutzkonzept	260																				x		x		x		x		x	x	x	x	x	x
		Brandschutzvorkonzept	261																				x		x		x		x		x	x	x	x	x	x
Gebäudeautomationsplanung	Gebäudeautomationsplan	Datenpunktliste	262																																	
		Grundrissplan	263																				x		x											
		Raumbuch	264																				x													
		Schemata	265																						x											
	Funktion und Leistung	Funktionsbeschreibung (GU)	266																																	
		Leistungsverzeichnis (EG)	267																																	
Energie- und Bauphysikplanung	Nachweise	Nachweis der Einhaltung der EnEV	268																				x													
		Raumakustiknachweis	269																				x													
		Schallschutznachweis	270																				x													
	Energieplan	Energiekonzept	271																						x											
		Thermische Simulation	272																													x				
Innenausbauplanung	Innenausbaupläne	Umsetzung der Technischen Leistungen und Anschlüsse	273											x			x							x	x		x									
		Umsetzung von Boden-, Wand-, und Deckengestaltung	274											x			x							x	x		x									
Starkstromanlagenplanung	Starkstromanlagenplan	Bemessungen	275																									x								
		Berechnungen	276																																	
		Grundrissplan	277																				x		x											
		Raumbuch	278																				x													
		Schnittzeichnungen	279																				x		x		x									
	Funktion und Leistung	Funktionsbeschreibung (GU)	280																																	
		Leistungsverzeichnis (EG)	281																																	
Planung der fernmelde- und informationstechnischen Anlagen	Fernmelde- und informationstechnische Anlagenplan	Bemessungen	282																									x								
		Berechnungen	283																																	
		Grundrissplan	284																				x		x											
		Raumbuch	285																				x													
		Schnittzeichnungen	286																				x		x											
	Funktion und Leistung	Funktionsbeschreibung (GU)	287																																	
		Leistungsverzeichnis (EG)	288																																	
IT-Planung	Pflichtenhefte IT-Systeme	BDE/MDE-Systeme (Pflichtenheft)	289																												x					
		ERP/PPS-Systeme (Pflichtenheft)	290																												x					
		MES-Systeme	291																							x					x					
	IT-Anforderungen	Lastenheft für Anwendungssoftware	292												x																x					
		Lastenheft für IT-Hardware	293												x																x					
	IT-Plan	Plantafeln	294																																	
		Visuelles Management	295														x																			
Planung der raumlufttechnischen Anlagen	Plan der raumlufttechnischen Anlagen	Bemessungen	296																									x								
		Berechnungen	297																																	
		Entwurfspläne RTA	298																				x													
		Grundrissplan	299																				x		x											
		Raumbuch	300																				x													
		Schemata	301																						x		x							x	x	
		Schnittzeichnungen	302																				x		x											
	Funktion und Leistung	Funktionsbeschreibung (GU)	303																																	
		Leistungsverzeichnis (EG)	304																																	
Planung der Kälteanlagen für RLT-Anlagen	Plan der Kälteanlagen für RLT-Anlagen	Bemessungen	305																									x								
		Berechnungen	306																																	
		Grundrissplan	307																				x		x											
		Raumbuch	308																				x		x											
		Schemata	309																								x									
		Schnittzeichnungen	310																				x		x											
	Funktion und Leistung	Funktionsbeschreibung (GU)	311																																	
		Leistungsverzeichnis (EG)	312																																	
Planung der Wärmeversorgungsanlagen	Plan der Wärmeversorgungsanlagen	Bemessungen	313																									x								
		Berechnungen	314																																	
		Grundrissplan	315																				x		x											
		Raumbuch	316																				x													
		Schemata	317																						x		x									
		Schnittzeichnungen	318																				x		x											
	Funktion und Leistung	Funktionsbeschreibung (GU)	319																																	
		Leistungsverzeichnis (EG)	320																																	
Planung der Abwasser-, Wasser- und Gasanlagen	Plan der Abwasser- Wasser- und Gasanlagen	Bemessungen	321																									x								
		Berechnungen	322																																	
		Entwässerungsgesuch	323																																	
		Entwässerungsplan	324																																	
		Grundrissplan	325																				x		x											
		Raumbuch	326																				x													
		Schemata	327																						x											
	Funktion und Leistung	Funktionsbeschreibung (GU)	328																																	
		Leistungsverzeichnis (EG)	329																																	
Außenanlagenplanung	Arealsicherheit	Sicherheitskonzept	330																																	
	Beleuchtung	Beleuchtungsplan	331																																	
	Flächenbedarf Außenanlage	Parkzone	332															x																		
		Ladezone	333											x				x																		
		Anlieferzone	334											x				x																		
		Sonstige Flächen	335															x																		

Abbildung B.3: Übersicht Planungsmodule im Basis-Repository (Seite 3)

Anhang C

Generierte Netzpläne aus Experiment 5.3

Nr.	Vorgangsname	Nr.	Vorgangsname
1	Einreichung Bauantrag	46	Deckenleuchten, TGA-Feininstallation
2	Teilbaugenehmigung Erdarbeiten u. Entwässerung	47	Tore innenliegend
3	Freigabe FT-Produktion durch BH	48	Schlosserarbeiten
4	Genehmigungsstatik	49	Ausbau Halle
5	Prüfung Genehmigungsstatik	50	Ausbau Mezzanine
6	Vorlage 1.Prüfbericht Prüfstatik	51	Gebäudereinigung
7	Vorlage der Baugenehmigung	52	Medienbereitstellung durch Versorger
8	**Ausführungsplanung**	53	Übergabe Halle 1
9	Ausführungsplanung Halle	54	**Bauaufführung**
10	Durchbruchsplanung TGA Halle	55	Fundamentaushub Erdarbeiten
11	Ausführungsplanung Büroflächen	56	Fundamente
12	Architekturfreigaben durch Bauherr	57	FT-Montage mit zweiter Kolonne
13	Durchbruchsplanung TGA Büroflächen	58	Dachabdichtung
14	Ausführungsstatik	59	Ausbau
15	Prüfung Ausführungsstatik	60	Übergabe zum Mieterausbau Küche
16	Übersichtsplanung	61	Gebäudereinigung
17	Elementplanung	62	**Planung der Abwasser, Wasser und Gasanlagen**
18	Elementplanprüfung Prüfingenieur	63	Erdarbeiten, Grundleitunge
19	Freigabe Produktionsbeginn durch Prüfingenieur	64	Ortbetonarbeiten
20	Ausführungsplanung TGA	65	FT-Montage
21	Freigabe durch Bauherr	66	Dacharbeiten
22	Außenanlage	67	Bodenplatte
23	Freigabe durch Bauherr	68	TGA-Arbeiten
24	Fertigteilproduktion	69	Bodenplatte Sprinkler
25	Geländeprofilierung Cut & Fill	70	Sprinklerbehälter
26	**Budgetplanung**	71	**Planung der Sicherheitstechnik**
27	Grundlagenermittlung Budgetleistungen	72	Erdarbeiten, Grundleitunge
28	Planung Budgetleistungen	73	Ortbetonarbeiten
29	Prüfung und Freigabe durch AG	74	FT-Montage
30	Erstellung Anfrage und Ausschreibung Budgetleistungen	75	Dacharbeiten
31	Auswertung	76	Bodenplatte
32	Prüfung und Auswertung AG	77	Ausbau
33	Vergabe Budgetleistungen	78	**Planung der Außenanlagen**
34	**Bauausführungsplanung**	79	Geländeprofilierung Erdarbeiten
35	Fundamentaushub Erdarbeiten	80	Entwässerungsleitungen
36	Fundamente	81	Inbetriebnahme Vorflut
37	FT-Montage	82	Befestigte Flächen
38	Dach	83	Betonflächen Verladehöfe
39	Fassade	84	Torabdichtung
40	TGA-Deckeninstallation	85	Schwarzdecken
41	Montage Deckenlaufkran (AG)	86	Zaun, Toranlagen
42	Überladebrücken	87	Landschaftsbau

Abbildung C.1: Verwendete Module und Submodule in Experiment 5.3

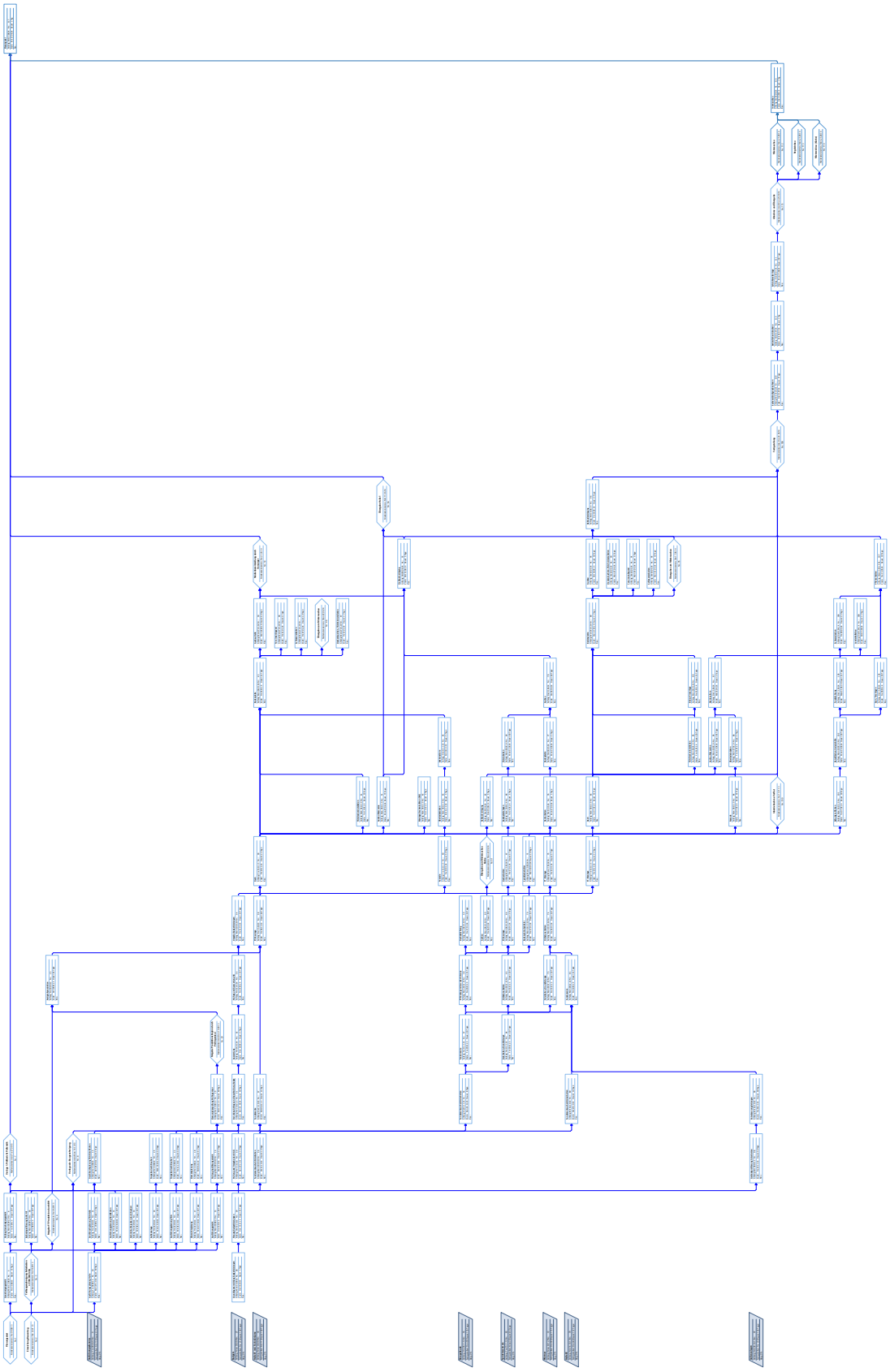


Abbildung C.2: Generierter Netzplan des Experiments 5.3 - Variante 1

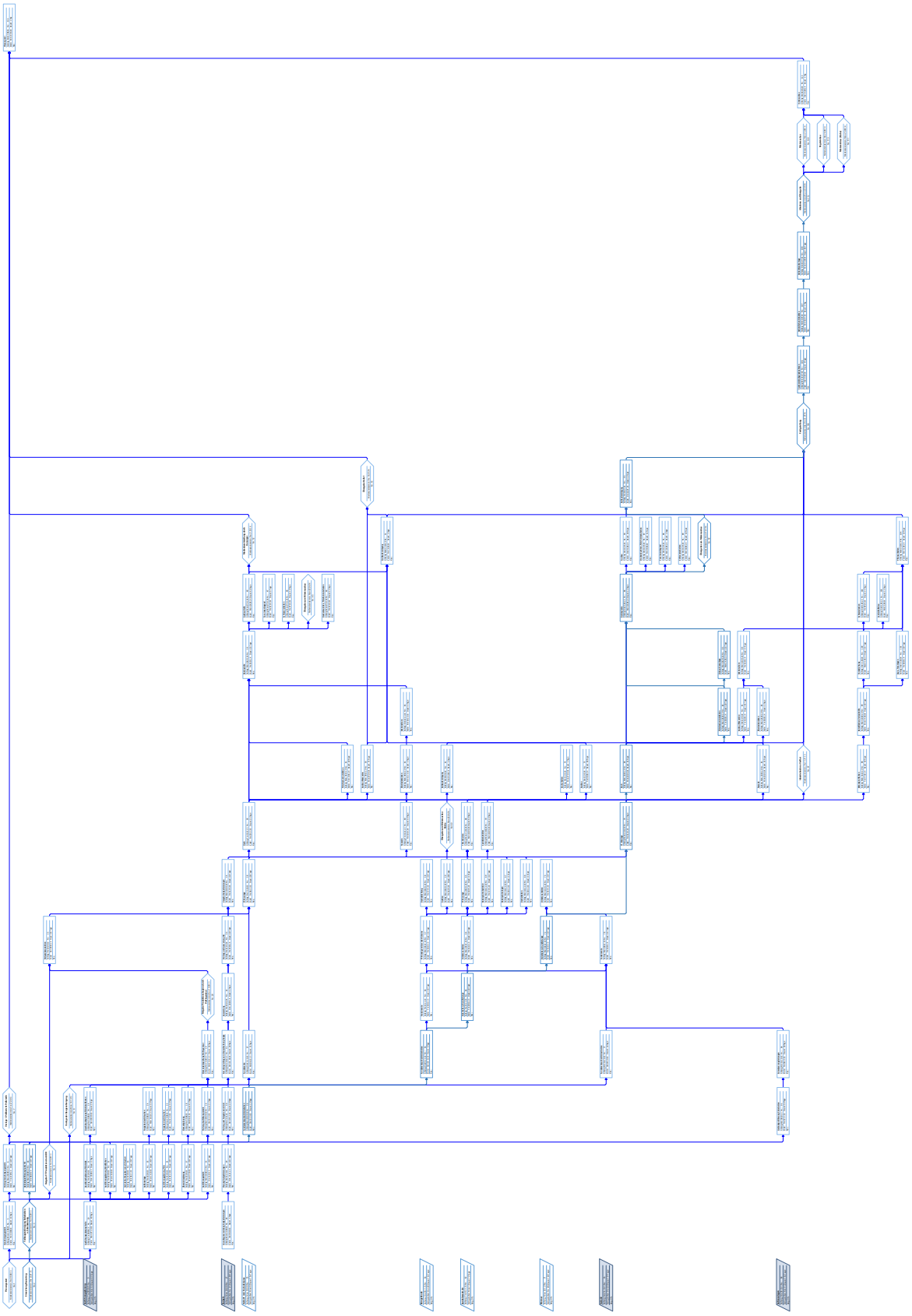


Abbildung C.3: Generierter Netzplan des Experiments 5.3 - Variante 2

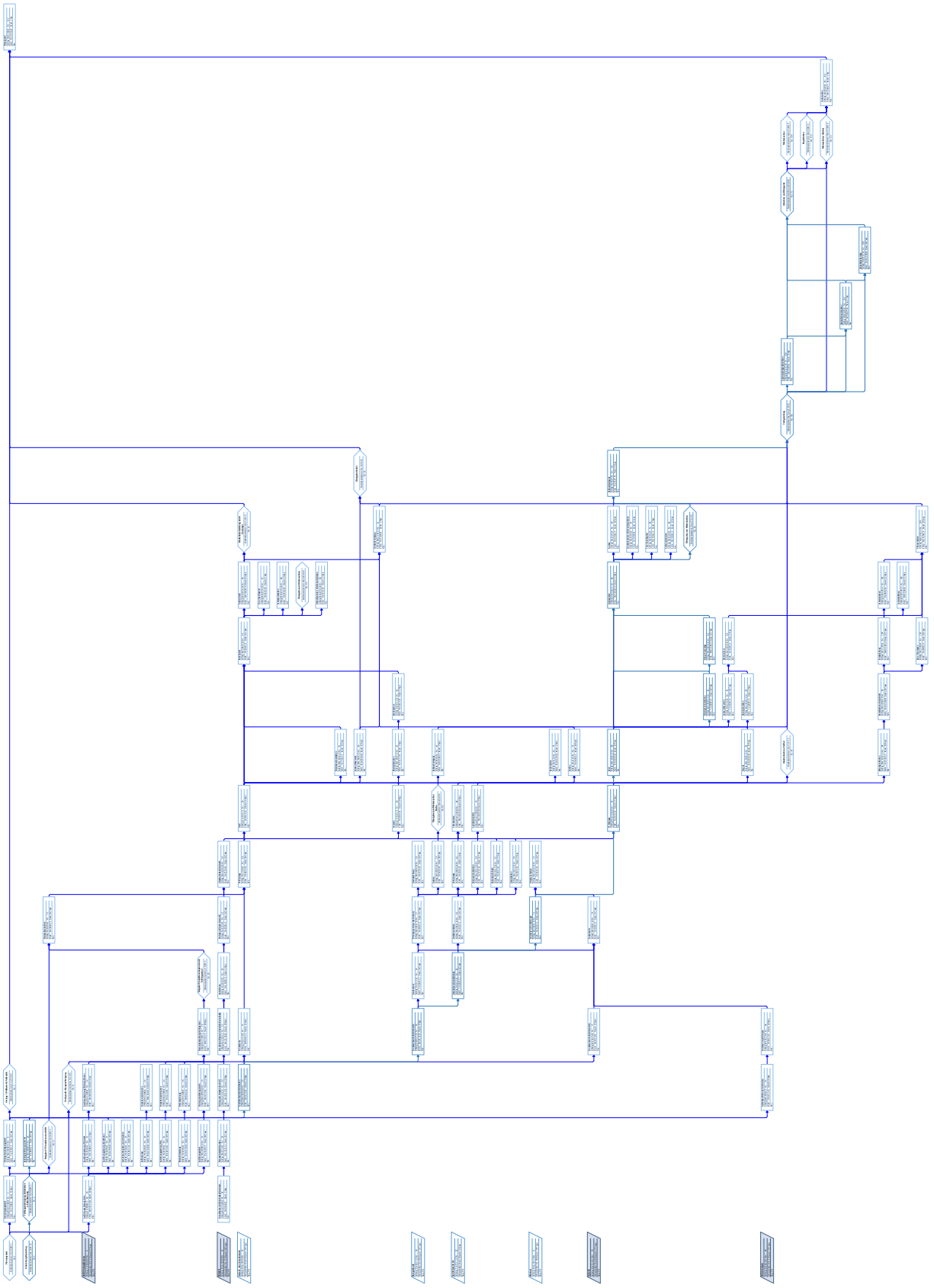


Abbildung C.4: Generierter Netzplan des Experiments 5.3 - Variante 3

Abbildungsverzeichnis

Literaturverzeichnis

[1] Seiten 343–358. Springer, 2019.

[2] Aggteleky, Béla: *Fabrikplanung: Werksentwicklung und Betriebsrationalisierung. 1. Grundlagen-Zielplanung-Vorarbeiten, unternehmerische und systemtechnische Aspekte, Marketing und Fabrikplanung.* Hanser, 1987.

[3] Ahlemann, Frederik: *Eine neue DIN-Norm zum Projektmanagement: Eine kritische Analyse aus Sicht der Wirtschaftsinformatik.* HMD Praxis der Wirtschaftsinformatik, 44(3):104–114, 2007.

[4] Altrogge, Günter: *Netzplantechnik.* Walter de Gruyter GmbH & Co KG, 2018.

[5] Alur, Rajeev, Rastislav Bodik, Garvit Juniwal, Milo MK Martin, Mukund Raghothaman, Sanjit A Seshia, Rishabh Singh, Armando Solar-Lezama, Emina Torlak und Abhishek Udupa: *Syntax-guided synthesis.* In: *Formal Methods in Computer-Aided Design (FMCAD), 2013*, Seiten 1–8. IEEE, 2013.

[6] Apel, Sven, Don Batory, Christian Kästner und Gunter Saake: *Feature-oriented software product lines.* Springer, 2016.

[7] Austen, A. D. und R. H. Neale (Herausgeber): *Managing construction projects: A guide to processes and procedures.* International Labour Office, Geneva, [Fourth impression] Auflage, 1995.

[8] Autili, Marco, Davide Di Ruscio, Amleto Di Salle, Paola Inverardi und Massimo Tivoli: *A model-based synthesis process for choreography realizability enforcement.* In: *International Conference on Fundamental Approaches to Software Engineering*, Seiten 37–52. Springer, 2013.

[9] Awad, Ahmed, Rajeev Goré, James Thomson und Matthias Weidlich: *An iterative approach for business process template synthesis from compliance rules.* In: *International Conference on Advanced Information Systems Engineering*, Seiten 406–421. Springer, 2011.

[10] Barrett, Clark, Pascal Fontaine und Cesare Tinelli: *The SMT-LIB Standard: Version 2.5, 2015.*, 2015.

[11] Barrett, Clark und Cesare Tinelli: *Satisfiability modulo theories.* In: *Handbook of Model Checking*, Seiten 305–343. Springer, 2018.

[12] Basic, Robert: *Fertigungsinseln statt Fließband.* Audi Blog, 23, 2016.

[13] Bea, Franz Xaver und S. Hesselmann St Scheurer: *Projektmanagement; Lucius u*, 2008.

[14] Becker, Marcel, Limei Gilham, Douglas R Smith et al.: *Planware II: Synthesis of schedulers for complex resource systems.* 2003.

[15] Berg, Kathrin, Judith Bishop und Dirk Muthig: *Tracing software product line variability: from problem to solution space.* In: *Proceedings of the 2005 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries*, Seiten 182–191, 2005.

[16] Bergholz, Markus: *Objektorientierte Fabrikplanung*, Band 2006,31 der Reihe *Berichte aus der Produktionstechnik.* Shaker, Aachen, 2006.

[17] Bernecker, Michael und Klaus Eckrich: *Handbuch Projektmanagement.* Walter de Gruyter, 2010.

[18] Berner, Fritz, Bernd Kochendörfer und Rainer Schach: *Netzplantechnik.* In: *Grundlagen der Baubetriebslehre 2*, Seiten 111–140. Springer, 2013.

[19] Bessai, Jan: *A Typetheoretic Framework For Component Software Synthesis.* Doktorarbeit, Technische Universität Dortmund, 2019.

[20] Bessai, Jan, Andrej Dudenhefner, Boris Düdder, Moritz Martens und Jakob Rehof: *Combinatory Logic Synthesizer.* In: Margaria-Steffen, Tiziana und Bernhard Steffen (Herausgeber): *Leveraging applications of formal methods, verification and validation*, Band 8802 der Reihe *LNCS sublibrary. SL 1, Theoretical computer science and general issues*, Seiten 26–40. Springer, Heidelberg, 2014.

[21] Bessai, Jan, Andrej Dudenhefner, Boris Düdder, Moritz Martens und Jakob Rehof: *Combinatory Process Synthesis.* 9952:266–281, 2016.

[22] Bimbó, Katalin: *Combinatory Logic.* Chapman and Hall/CRC, 2011.

[23] Bin Yang, A. Bundy, A. Smaill und L. Dixon: *Deductive synthesis of workflows for e-Science.* In: *CCGrid 2005. IEEE International Symposium on Cluster Computing and the Grid, 2005.*, Band 1, Seiten 168–175 Vol. 1, May 2005.

[24] Blaine, Lee, Limei Gilham, Junbo Liu, Douglas R Smith und Stephen Westfold: *Planware-domain-specific synthesis of high-performance schedulers.* In: *Automated Software Engineering, 1998. Proceedings. 13th IEEE International Conference on*, Seiten 270–279. IEEE, 1998.

[25] Bleicher, Knut und Christian Abegglen: *Das Konzept Integriertes Management: Visionen-Missionen-Programme, plus E-Book inside (ePub, mobi oder pdf)*, Band 1. Campus Verlag, 2017.

[26] Bodik, Rastislav und Barbara Jobstmann: *Algorithmic program synthesis: introduction.* International Journal on Software Tools for Technology Transfer, 15(5-6):397–411, 2013.

[27] Bosselmann, Steve, Markus Frohme, Dawid Kopetzki, Michael Lybecait, Stefan Naujokat, Johannes Neubauer, Dominic Wirkner, Philip Zweihoff und Bernhard Steffen: *DIME: a programming-less modeling environment for web applications.* In: *International Symposium on Leveraging Applications of Formal Methods*, Seiten 809–832. Springer, 2016.

[28] Bouyer, Patricia: *Weighted timed automata: Model-checking and games.* Electronic Notes in Theoretical Computer Science, 158:3–17, 2006.

[29] Buchi, J Richard und Lawrence H Landweber: *Solving sequential conditions by finite state strategies.* 1967.

[30] Burghardt, Manfred: *Projektmanagement: Leitfaden für die Planung, Überwachung und Steuerung von Projekten.* Publicis, Erlangen, 9., wesentlich überarb. und erw. Aufl. Auflage, 2012.

[31] Bussemer, Felix: *Methode zur systematischen Strukturierung von Fabrikplanungsprojekten*, Band 1/2019 der Reihe *Berichte aus dem IFA*. TEWISS, Garbsen, 1. Erstausgabe Auflage, 2019.

[32] Cardelli, Luca und Peter Wegner: *On Understanding Types, Data Abstraction, and Polymorphism.* ACM Comput. Surv., 17(4):471–523, Dezember 1985.

[33] Chasins, Sarah und Julie L Newcomb: *Using SyGuS to Synthesize Reactive Motion Plans.* arXiv preprint arXiv:1611.07620, 2016.

[34] Chun, Soon Ae, Vijayalakshmi Atluri und Nabil R. Adam: *Domain Knowledge-Based Automatic Workflow Generation.* In: Hameurlain, Abdelkader, Rosine Cicchetti und Roland Traunmüller (Herausgeber): *Database and Expert Systems Applications*, Seiten 81–93, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[35] CIOB, The Chartered Institute of Building: *Guide to good practice in the management of time in complex projects.* Wiley-Blackwell, Chichester, West Sussex, U.K and Ames, Iowa, 2011.

[36] CLARKE, DAVE, MICHIEL HELVENSTEIJN und INA SCHAEFER: *Abstract delta modelling.* Mathematical Structures in Computer Science, 25(3):482–527, 2015.

[37] Cooke, Brian und Peter Williams (Herausgeber): *Construction planning, programming and control.* Blackwell Publishing, Oxford, 2nd ed. Auflage, 2004.

[38] Corsten, Hans und Hilde Corsten: *Projektmanagement: Einführung.* Oldenbourg Verlag, 2000.

[39] CZARNECKI, K ANDEISENECKER: *U. 2000. Generative Programming: Methods, Tools, and Applications. ACM.*

[40] Dadam, Peter und Manfred Reichert: *The ADEPT project: a decade of research and development for robust and flexible process support.* Computer Science-Research and Development, 23(2):81–97, 2009.

[41] De Alfaro, Luca, Thomas A Henzinger und Rupak Majumdar: *Discounting the future in systems theory.* In: *International Colloquium on Automata, Languages, and Programming*, Seiten 1022–1037. Springer, 2003.

[42] Deutsches Institut für Normung e.V. (Herausgeber): *Projektmanagement – Projektmanagementsysteme – Teil 5: Begriffe.* Beuth Verlag GmbH, Berlin, Wien, Zürich, 9 Auflage, 2009.

[43] Düdder, Boris, Moritz Martens und Jakob Rehof: *Staged composition synthesis.* In: *European Symposium on Programming Languages and Systems*, Seiten 67–86, 2014.

[44] Eich, Rainer,: *HOAI 2013: Honorarordnung für Architekten und Ingenieure ; Textausgabe mit Interpolationstabellen ; Textausgabe mit Erläuterung der Neuerungen, Musterrechnung und Interpolationstabellen.* Müller, Köln, 5., aktualis. Aufl. Auflage, 2013.

[45] Eichberg, Michael, Karl Klose, Ralf Mitschke und Mira Mezini: *Component composition using feature models.* In: *International Symposium on Component-Based Software Engineering*, Seiten 200–215. Springer, 2010.

[46] Fernandes, Abilio, Angelo EM Ciarlini, Antonio L Furtado, Michael G Hinchey, Marco A Casanova und Karin K Breitman: *Adding flexibility to*

workflows through incremental planning. Innovations in Systems and Software Engineering, 3(4):291–302, 2007.

[47] Feser, John K, Swarat Chaudhuri und Isil Dillig: *Synthesizing data structure transformations from input-output examples.* In: *ACM SIGPLAN Notices*, Band 50, Seiten 229–239. ACM, 2015.

[48] Flener, Pierre und Ute Schmid: *An introduction to inductive programming.* Artificial Intelligence Review, 29(1):45–62, 2008.

[49] Frankle, Jonathan, Peter-Michael Osera, David Walker und Steve Zdancewic: *Example-directed synthesis: a type-theoretic interpretation.* ACM SIGPLAN Notices, 51(1):802–815, 2016.

[50] Franzle, Martin, Christian Herde, Tino Teige, Stefan Ratschan und Tobias Schubert: *Efficient Solving of Large Non-linear Arithmetic Constraint Systems with Complex Boolean Structure.* Journal on Satisfiability, Boolean Modeling and Computation, 1(0):209–236, 2007.

[51] Freitag, Burkhard, Tiziana Margaria und Bernhard Steffen: *A pragmatic approach to software synthesis*, Band 29. ACM, 1994.

[52] Froschauer, Uwe: *Projektmanagement mal anders–humorvoll und leicht verständlich: Handlungsfelder des Managements.* Diplomica Verlag, 2015.

[53] Golenko, Dmitrij Isaakovič: *Statistische Methoden der Netzplantechnik.* Springer-Verlag, 2013.

[54] Graefenstein, Julian, David Scholz, Michael Henke, Jan Winkels und Jakob Rehof: *Intelligente Orchestrierung von Planungsprozessen.* ZWF Zeitschrift für wirtschaftlichen Fabrikbetrieb, 112(4):209–214, 2017.

[55] Graefenstein, Julian, David Scholz, Oliver Seifert, Jan Winkels, Michael Henke und Jakob Rehof: *Automated Processing of Planning Modules in Factory Planning by Means of Constraint Solving Using the Example of Production Segmentation.* In: Hankammer, Stephan, Kjeld Nielsen, Frank T. Piller, Günther Schuh und Ning Wang (Herausgeber): *Customization 4.0*, Band 48 der Reihe *Springer proceedings in business and economics*, Seiten 157–172. Springer International Publishing, Cham, 2018.

[56] Grambow, Gregor, Roy Oberhauser und Manfred Reichert: *Semantically-driven workflow generation using declarative modeling for processes in software engineering.* In: *Enterprise Distributed Object Computing Conference Workshops (EDOCW), 2011 15th IEEE International*, Seiten 164–173. IEEE, 2011.

[57] Grundig, Claus-Gerold und Dieter Hartrampf: *Fabrikplanung.* Service-Agentur des HDL, 2006.

[58] Gulwani, Sumit, William R Harris und Rishabh Singh: *Spreadsheet data manipulation using examples.* Communications of the ACM, 55(8):97–105, 2012.

[59] Gulwani, Sumit, Susmit Jha, Ashish Tiwari und Ramarathnam Venkatesan: *Synthesis of loop-free programs.* In: *PLDI*, Band 11, Seiten 62–73, 2011.

[60] Hahn, Eugene David: *Mixture densities for project management activity times: A robust approach to PERT.* European Journal of Operational Research, 188(2):450–459, 2008.

[61] Hamilton, Albert (Herausgeber): *Managing projects for success: a trilogy.* Telford, London, 2001.

[62] Hawkins, Peter, Alex Aiken, Kathleen Fisher, Martin Rinard und Mooly Sagiv: *Concurrent data representation synthesis.* In: *ACM SIGPLAN Notices*, Band 47, Seiten 417–428. ACM, 2012.

[63] Hennicke, Ludwig H.: *Wissensbasierte Erweiterung der Netzplantechnik*, Band 47. Springer-Verlag, 2013.

[64] Hildebrandt, Thomas T und Raghava Rao Mukkamala: *Declarative event-based workflow as distributed dynamic condition response graphs.* arXiv preprint arXiv:1110.4161, 2011.

[65] Hindley, J. Roger und Jonathan P. Seldin: *Lambda-calculus and Combinators, an Introduction*, Band 13. Cambridge University Press Cambridge, 2008.

[66] Hindley, J. Roger und Jonathan P. Seldin: *Lambda-calculus and combinators: An introduction.* Cambridge Univ. Press, Cambridge, Repr Auflage, 2010.

[67] Hofstedt, Petra und Armin Wolf: *Einführung in die Constraint-Programmierung.* Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[68] Howar, Falk, Fadi Jabbour und Malte Mues: *JConstraints: A Library for Working with Logic Expressions in Java. (B60, to appear).*

[69] (Hrsg.), BitKom: *Industrie 4.0 - Die neue Rolle der IT.*

[70] Iacocca, L. und Novak, W.: *Iacocca: An autobiography.* Bantam Books, 1986.

[71] Ilghami, Okhtay und Dana S Nau: *A general approach to synthesize problem-specific planners.* Technischer Bericht, MARYLAND UNIV COLLEGE PARK DEPT OF COMPUTER SCIENCE, 2003.

[72] Jain, Ramesh, Holger Lausen, Tiziana Margaria, Charles Petrie, Amit Sheth und Michal Zaremba: *Semantic Web Services Challenge: Results from the First Year*, Band 8 der Reihe *Semantic Web And Beyond, Computing for Human Experience*. Springer US, Boston, MA, 2009.

[73] Jha, Susmit, Sumit Gulwani, Sanjit A. Seshia und Ashish Tiwari: *Oracle-guided component-based program synthesis*. In: Kramer, Jeff, Judith Bishop, Prem Devanbu und Sebastian Uchitel (Herausgeber): *ACM/IEEE 32nd International Conference on Software Engineering, 2010*, Seite 215, Piscataway, NJ, 2010. IEEE.

[74] Johannes Neubauer, Markus Frohme, Bernhard Steffen Tiziana Margaria: *Prototype-driven development of web applications with dywa*. In: *Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change*, Seiten 56–72, Berlin, Heidelberg, Januar 2014. Springer-Verlag.

[75] Jörges, Sven, Anna-Lena Lamprecht, Tiziana Margaria, Ina Schaefer und Bernhard Steffen: *A constraint-based variability modeling framework*. International Journal on Software Tools for Technology Transfer, 14(5):511–530, 2012.

[76] Julian Graefenstein: *Aufgabenorientierte Fabrikplanung- Standardisierung und teilautomatisierung des Planungsprozesses zur Fabrikanpassung*. Doktorarbeit, Technische Universität Dortmund, Dortmund, 10 / 2019.

[77] Kang, K.C., S.G. Cohen, J.A. Hess, W.E. Novak und A.S. Peterson: *Feature-oriented domain analysis (FODA) feasibility study, Technical Report CMU/SEI-90-TR-021*. SEI, Carnegie Mellon University, November 1990.

[78] Kern, Nikolaus: *Netzplantechnik*. Springer, 1969.

[79] Kettner, Hans, Jürgen Schmidt, Hans-Robert Greim et al.: *Leitfaden der systematischen Fabrikplanung*. Hanser München, 1984.

[80] Kiczales, Gregor, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier und John Irwin: *Aspect-oriented programming*. In: *European conference on object-oriented programming*, Seiten 220–242, 1997.

[81] Kitzelmann, Emanuel: *Inductive programming: A survey of program synthesis techniques*. In: *International workshop on approaches and applications of inductive programming*, Seiten 50–73. Springer, 2009.

[82] Kroening, Daniel und Strichmann Ofer: *Decision Procedures: An algorithmic point of view*. Springer, [Place of publication not identified], 2019.

[83] Krunke, M.: *Reifegradmanagement in der Fabrikplanung.* Dissertation, RWTH Aachen, Aachen, 2017.

[84] Küpper, X. Y., X. Y. Lüder und X. Y. Streitferdt: *Netzplantechnik.* Springer-Verlag, 2013.

[85] Kuster, Jürg, Eugen Huber, Robert Lippmann, Alphons Schmid, Emil Schneider, Urs Witschi und Roger Wüst: *Handbuch Projektmanagement*, Band 3. Springer, 2011.

[86] Lager, Hendrik, Philipp Regelmann, Julian Graefenstein und Daniel Silveira Pereira: *Rollen- und Kompetenzentwicklungen im Zuge der Industrie 4.0.* ZWF Zeitschrift für wirtschaftlichen Fabrikbetrieb, 113(6):415–419, 2018.

[87] Lamprecht, Anna-Lena: *User-Level Workflow Design: A Bioinformatics Perspective*, Band v.8311 der Reihe *Lecture Notes in Computer Science / Programming and Software Engineering.* Springer Berlin Heidelberg, Berlin/Heidelberg, 2013.

[88] Lamprecht, Anna-Lena, Stefan Naujokat, Tiziana Margaria und Bernhard Steffen: *Synthesis-based loose programming.* In: *Quality of Information and Communications Technology (QUATIC), 2010 Seventh International Conference on the*, Seiten 262–267. IEEE, 2010.

[89] Lau, Tessa, Pedro Domingos und Daniel S Weld: *Learning programs from traces using version space algebra.* In: *Proceedings of the 2nd international conference on Knowledge capture*, Seiten 36–43. ACM, 2003.

[90] Le, Vu und Sumit Gulwani: *FlashExtract: a framework for data extraction by examples.* In: *ACM SIGPLAN Notices*, Band 49, Seiten 542–553. ACM, 2014.

[91] Leofante, Francesco, Erika Ábrahám, Tim Niemueller, Gerhard Lakemeyer und Armando Tacchella: *On the synthesis of guaranteed-quality plans for robot fleets in logistics scenarios via optimization modulo theories.* In: *2017 IEEE International Conference on Information Reuse and Integration (IRI)*, Seiten 403–410, 2017.

[92] Lisa Lenz, Jan Winkels, Philipp Regelmann und Mike Gralla: *Automatisierungspotenziale für Controlling-Kennzahlen aus den Daten eines BIM Projektes.* Bauwirtschaft, 4(2):90–98, Juni 2019.

[93] Lisa Theresa Lenz, Julian Graefenstein, Jan Winkels und Mike Gralla: *Smart Factory Adaptation Planning by means of BIM in Combination of Constraint Solving Techniques.* Proceedings of the International Council for Research

and Innovation in Building and Construction (CIB), World Building Congress 2019 – Constructing Smart Cities, 2019.

[94] Lygeros, John, Claire Tomlin und Shankar Sastry: *Controllers for reachability specifications for hybrid systems.* Automatica, 35(3):349–370, 1999.

[95] Maler, Oded, Amir Pnueli und Joseph Sifakis: *On the synthesis of discrete controllers for timed systems.* In: *Annual Symposium on Theoretical Aspects of Computer Science*, Seiten 229–242. Springer, 1995.

[96] Margaria, Tiziana, Daniel Meyer, Christian Kubczak, Malte Isberner und Bernhard Steffen: *Synthesizing Semantic Web Service Compositions with jMosel and Golog.* In: Bernstein, Abraham, David R. Karger, Tom Heath, Lee Feigenbaum, Diana Maynard, Enrico Motta und Krishnaprasad Thirunarayan (Herausgeber): *The Semantic Web - ISWC 2009*, Seiten 392–407, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[97] Meckelnborg, Alexander: *Integrative Fabrikplanung durch effiziente Koordinationsmodelle.* Apprimus-Verlag, 2015.

[98] Medvidovic, Nenad und Richard N Taylor: *Software architecture: foundations, theory, and practice.* In: *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2*, Seiten 471–472. ACM, 2010.

[99] Microsoft: *Getting started with Z3 - A Guide*, 2019.

[100] Microsoft Office: *Von Project unterstützte Dateiformate: Project Professional 2019 Project Professional 2016 Project 2010 Project 2007 Project Online-Desktopclient Project Professional 2013 Project Server 2007 Project Standard 2007 Project Standard 2010 Project Standard 2013 Project Standard 2016 Project Standard 2019*, 2019.

[101] Morales, Roberto Hernández: *Systematik der Wandlungsfähigkeit in der Fabrikplanung.* VDI Verlag, 2003.

[102] Moura, Leonardo de und Nikolaj Bjørner: *Z3: An Efficient SMT Solver.* In: Ramakrishnan, C. R. und Jakob Rehof (Herausgeber): *Tools and Algorithms for the Construction and Analysis of Systems*, Band 4963 der Reihe *Lecture Notes in Computer Science*, Seiten 337–340. Springer-Verlag Berlin Heidelberg, Berlin, Heidelberg, 2008.

[103] Naujokat, Stefan, Anna-Lena Lamprecht und Bernhard Steffen: *Loose Programming with PROPHETS.* In: Lara, Juan de und Andrea Zisman (Herausgeber): *Fundamental Approaches to Software Engineering*, Seiten 94–98, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[104] Naujokat, Stefan, Michael Lybecait, Dawid Kopetzki und Bernhard Steffen: *CINCO: a simplicity-driven approach to full generation of domain-specific graphical modeling tools.* International Journal on Software Tools for Technology Transfer, 20(3):327–354, 2018.

[105] Naujokat, Stefan, Johannes Neubauer, Tiziana Margaria und Bernhard Steffen: *Meta-level reuse for mastering domain specialization.* In: *International Symposium on Leveraging Applications of Formal Methods*, Seiten 218–237. Springer, 2016.

[106] Neale, R. H. und David E. Neale (Herausgeber): *Construction planning.* Engineering management. T. Telford, London, 1989.

[107] Neumann, Klaus: *Netzplantechnik.* In: *Grundlagen des Operations Research*, Seiten 165–260. Springer, 1989.

[108] Nickel, Wolfram: *Von Fahrzeug fürs Volk zum Auto für alle - Tradition: 75 Jahre Volkswagen Käfer.* Welt-Online, 23, 2013.

[109] Nöcker, Jan Christoph (Herausgeber): *Zustandsbasierte Fabrikplanung*, Band Bd. 2012,6 der Reihe *Produktionssystematik.* Apprimus-Verl., Aachen, 1. Aufl. Auflage, 2012.

[110] Oliver Seifert: *Automatische Planung von Produktionssystemen mithilfe des Constraint-Solvings: Masterarbeit.*

[111] Parsotto, Emilio, Abdel-rahman Mohamed, Rishabh Singh, Lihong Li, Dengyong Zhou und Pushmeet Kohli: *Neuro-symbolic program synthesis.* arXiv preprint arXiv:1611.01855.

[112] Pawellek, Günther: *Ganzheitliche Fabrikplanung: Grundlagen, Vorgehensweise, EDV-Unterstützung.* Springer-Verlag, 2014.

[113] Pohl, Klaus, Günter Böckle und Frank J van Der Linden: *Software product line engineering: foundations, principles and techniques.* Springer Science & Business Media, 2005.

[114] Pontow, Stephan: *Theorie und Anwendung constraintbasierter Planungsverfahren.* Berichte aus der Informatik. Shaker, Aachen, Als Ms. gedr Auflage, 2000.

[115] Rabin, Michael Oser: *Automata on infinite objects and Church's problem*, Band 13. American Mathematical Soc., 1972.

[116] REFA-VERBAND, F. ARBEITSSTUDIENÜ.R. und E. V. BETRIEBSORGANISATION: *Methodenlehre der Planung und Steuerung.* Aufl. München: Hanser, 1985.

[117] Rehof, Jakob: *Minimal typings in atomic subtyping.* In: *Proceedings of the 24th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, Seiten 278–291. ACM, 1997.

[118] Rehof, Jakob: *Towards Combinatory Logic Synthesis.* BEAT'13, 1st International Workshop on Behavioural Types, 2013.

[119] Reichert, Manfred, Peter Dadam, Stefanie Rinderle-Ma, Andreas Lanz, Rüdiger Pryss, Michael Predeschly, Jens Kolb, Linh Thao Ly, Martin Jurisch, Ulrich Kreher et al.: *Enabling Poka-Yoke workflows with the AristaFlow BPM Suite.* 2009.

[120] Reynolds, John C: *Using category theory to design implicit conversions and generic operators.* In: *International Workshop on Semantics-Directed Compiler Generation*, Seiten 211–258. Springer, 1980.

[121] Reynolds, John C.: *The coherence of languages with intersection types.* In: Ito, Takayasu und Albert R. Meyer (Herausgeber): *Theoretical Aspects of Computer Software*, Seiten 675–700, Berlin, Heidelberg, 1991. Springer Berlin Heidelberg.

[122] Rintanen, Jussi: *Computational Complexity of Automated Planning and Scheduling.* In: *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling (ICAPS'16)*, Seiten 12–17.

[123] Rockstroh, Wolfgang: *Grundlagen und Methoden der Projektierung.* Verlag Technik, 1980.

[124] Roser, Stephan, Florian Lautenbacher und Bernhard Bauer: *Generation of workflow code from DSMs.* In: *Proceedings of the 7th OOPSLA Workshop on Domain-Specific Modeling*, Band 4, 2007.

[125] Runzheimer, Bodo: *Operations research: Lineare Planungsrechnung, Netzplantechnik, Simulation und Warteschlangentheorie*, Band 7. Springer-Verlag, 2013.

[126] Schaefer, Ina, Rick Rabiser, Dave Clarke, Lorenzo Bettini, David Benavides, Goetz Botterweck, Animesh Pathak, Salvador Trujillo und Karina Villela: *Software diversity: state of the art and perspectives*, 2012.

[127] Schenk, Michael, Siegfried Wirth und Egon Müller: *Wandlungsfähige Fabrikmodelle.* In: *Fabrikplanung und Fabrikbetrieb*, Seiten 649–678. Springer, 2014.

[128] Schmigalla, Hans: *Fabrikplanung: Begriffe und Zusammenhänge.* Hanser Verlag, 1995.

[129] Schönfinkel, M.: *Über die Bausteine der mathematischen Logik.* Mathematische Annalen, 92(3-4):305–316, 1924.

[130] Schot, Bram, Alexander Seitz und Peter Kössler: *AUDI AG Jahrespressekonferenz 2019.*

[131] Schreckeneder, Berta C.: *Projektcontrolling-mit Arbeitshilfen online: Projekte überwachen, steuern, präsentieren.* Haufe Lexware, 2013.

[132] Schuh, G., A. Gulden, S. Gottschalk und A. Kampker: *Komplexitätswissenschaft in der Fabrikplanung.* wt Werkstattstechnik online, 4(2006):167–170, 2006.

[133] Schulte, Helmut: *Marktanforderungen verandern Fabrikstrukturen.* Zeitschrift fur Wirtschaftlichen Fabrikbetrieb, 92(1):12–13, 1997.

[134] Schumacher, Christin, Hendrik Lager, Philipp Regelmann, Jan Winkels und Julian Graefenstein: *Einfluss der Industrie 4.0 auf Kompetenz- und Rollenprofile. Disruption von Berufsbildern durch den erhöhten Bedarf von IT-Kompetenzen im produzierenden Gewerbe.* Industrie Management, (35 (2)):42–57, 2019.

[135] Sigrid Wenzel, Jakob Rehof, Jana Stolipin, Jan Winkels: *Trends In Automatic Composition Of Structures For Simulation Models In Production And Logistics.* Proceedings of the 2019 Winter Simulation Conference, 2019.

[136] Slaats, Tijs, Raghava Rao Mukkamala, Thomas Hildebrandt und Morten Marquard: *Exformatics Declarative Case Management Workflows as DCR Graphs.* In: Daniel, Florian, Jianmin Wang und Barbara Weber (Herausgeber): *Business Process Management*, Seiten 339–354, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[137] Smaragdakis, Yannis und Don S Batory: *DiSTiL: A Transformation Library for Data Structures.* In: *DSL*, Band 97, Seite 28, 1997.

[138] Solar-Lezama, Armando, Liviu Tancau, Rastislav Bodik, Sanjit Seshia und Vijay Saraswat: *Combinatorial sketching for finite programs.* ACM Sigplan Notices, 41(11):404–415, 2006.

[139] Srinivas, Yellamraju V und Richard Jüllig: *Specware: Formal support for composing software.* In: *International Conference on Mathematics of Program Construction*, Seiten 399–422. Springer, 1995.

[140] Steffen, Bernhard, Anna-Lena Lamprecht und Tiziana Margaria: *User-level synthesis: treating product lines as systems of constraints.* In: *Proceedings of*

the 19th International Conference on Software Product Line, Seiten 427–431. ACM, 2015.

[141] Steffen, Bernhard, Tiziana Margaria und Michael von der Beeck: *Automatic synthesis of linear process models from temporal constraints: An incremental approach.* In: *ACM/SIGPLAN Int. Workshop on Automated Analysis of Software (AAS'97)*. Citeseer, 1997.

[142] Steffen, Bernhard, Tiziana Margaria und Burkhard Freitag: *Module configuration by minimal model construction.* 1993.

[143] Steffen, Bernhard und Stefan Naujokat: *Archimedean points: the essence for mastering change.* In: *Transactions on Foundations for Mastering Change I*, Seiten 22–46. Springer, 2016.

[144] Süss, Gerda und Dieter Eschlbeck: *Der Projektmanagement-Kompass.* Springer, 2002.

[145] Sven Hausen, Stefan Haffner: *Arbeiten mit Microsoft Project 2010: Microsoft setzt bei "Project 2010" auf SharePoint als Basis und eine Excel-ähnliche Bedienung.* Computerwoche - Voice of Digital, 2013(01), 11.01.2013.

[146] Thüm, Thomas, Christian Kästner, Fabian Benduhn, Jens Meinicke, Gunter Saake und Thomas Leich: *FeatureIDE: An extensible framework for feature-oriented software development.* Science of Computer Programming, 79:70–85, 2014.

[147] Tim Delbrügger, Frederik Döbbeler, Julian Graefenstein Hendrik Lager Lisa T. Lenz Matthias Meissner Daniel Müller Philipp Regelmann David Scholz Christin Schumacher Jan Winkels Andreas Wirtz und Felix Zeidler: *Anpassungsintelligenz von Fabriken im dynamischen und komplexen Umfeld.*

[148] Trexler, Robert C. und Paul E. Loustaunau: *A Plan for ULMS Weapon System Maintenance and Its Personnel Implications.*

[149] Van Gurp, Jilles, Jan Bosch und Mikael Svahnberg: *On the notion of variability in software product lines.* In: *Software Architecture, 2001. Proceedings. Working IEEE/IFIP Conference on*, Seiten 45–54. IEEE, 2001.

[150] Vanbrabant, Robbie: *Google Guice: agile lightweight dependency injection framework.* APress, 2008.

[151] VDI: *Fabrikplanung.*

[152] Vechev, Martin T, Eran Yahav und David F Bacon: *Correctness-preserving derivation of concurrent garbage collection algorithms.* In: *ACM SIGPLAN Notices*, Band 41, Seiten 341–353. ACM, 2006.

[153] Vechev, Martin T, Eran Yahav, David F Bacon und Noam Rinetzky: *CGCExplorer: a semi-automated search procedure for provably correct concurrent collectors.* In: *ACM SIGPLAN Notices*, Band 42, Seiten 456–467. ACM, 2007.

[154] Wegener, Dieter: *Industrie 4.0–Chancen und Herausforderungen für einen Global Player.* In: *Industrie 4.0 in Produktion, Automatisierung und Logistik*, Seiten 343–358. Springer, 2014.

[155] Weig, Sebastian: *Konzept eines integrierten Risikomanagements für die Ablauf- und Strukturgestaltung in Fabrikplanungsprojekten*, Band Bd. 220 der Reihe *Forschungsberichte / IWB*. Utz, München, 2008.

[156] Weiser, Mark: *The Computer for the 21st Century.* SIGMOBILE Mob. Comput. Commun. Rev., 3(3):3–11, Juli 1999.

[157] Wieczorrek, Hans W. und Peter Mertens: *Management von IT-Projekten: von der Planung zur Realisierung.* Springer-Verlag, 2010.

[158] Wiendahl, H. P., V. Ahrens, M. Burmeister et al.: *Fabrikplanung–Grundlagen der Fabrikplanung.* Produktion und Management, 3, 1999.

[159] Wiendahl, Hans-Peter, Dirk Nofen, Jan Hinrich Klussmann und Frank Breitenbach: *Planung modularer Fabriken.* Hanser, München, 2005.

[160] Wiendahl, Hans-Peter, Jürgen Reichardt und Peter Nyhuis: *Handbuch Fabrikplanung: Konzept, Gestaltung und Umsetzung wandlungsfähiger Produktionsstätten.* Carl Hanser Verlag GmbH Co KG, 2014.

[161] Wilson, James M.: *Gantt charts: A centenary appreciation*, 2003.

[162] Winkels, Jan: *Unterstützung von Serious Games mit Echtzeitsimulation in der DyWA*, 2016.

[163] Winkels, Jan, Julian Graefenstein, Tristan Schäfer, David Scholz, Jakob Rehof und Michael Henke: *Automatic Composition of Rough Solution Possibilities in the Target Planning of Factory Planning Projects by Means of Combinatory Logic.* In: Margaria, Tiziana und Bernhard Steffen (Herausgeber): *Leveraging Applications of Formal Methods, Verification and Validation. Industrial Practice*, Band 11247 der Reihe *Theoretical Computer Science and General Issues*, Seiten 487–503. Springer International Publishing, Limassol, Cyprus, 2018.

[164] Wirth, Siegfried, Michael Schenk und Egon Müller: *Wandlungsfähige und ressourceneffiziente Fabriken: Konsequenzen für Fabrikplanung und-betrieb sowie Unternehmen.* ZWF Zeitschrift für wirtschaftlichen Fabrikbetrieb, 107(6):391–397, 2012.

[165] Yu, Jian, Yan-Bo Han, Jun Han, Yan Jin, Paolo Falcarin und Maurizio Morisio: *Synthesizing service composition models on the basis of temporal business rules.* Journal of computer science and technology, 23(6):885–894, 2008.

[166] Zell, Helmut: *Projektmanagement:-zehn Module zu ausgewählten Themen.* BoD–Books on Demand, 2017.

[167] Zhang, Haoqi, Eric Horvitz und David C. Parkes: *Automated Workflow Synthesis.* In: *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*, AAAI'13, Seiten 1020–1026. AAAI Press, 2013.