Integrationskonzepte in der modellbasierten Produktentwicklung

Andreas Günzler

Technische Universität München
Institut für Informatik

Integrationskonzepte in der modellbasierten Produktentwicklung

Andreas Robert Günzler

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender: Univ.-Prof. Dr. Hans-Joachim Bungartz

Prüfer der Dissertation:

1. Univ.-Prof. Dr. Dr. h.c. Manfred Broy
2. Univ.-Prof. Dr. Klaus Bender

Die Dissertation wurde am 24. März 2005 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 11. Juli 2005 angenommen.

Kurzfassung

Die Entwicklung komplexer technischer Produkte ist geprägt durch steigende Anforderungen an Funktionsweise und Qualität der Ergebnisse sowie Effizienz und Effektivität ihrer Entstehung. In wachsendem Maße werden digitale Modelle von Produkten genutzt, welche sich mithilfe moderner Software-Lösungen im Vergleich zu realen Modellen wesentlich schneller und kostengünstiger bearbeiten lassen. Inkompatible Teilmodelle, isolierte Software-Werkzeuge und eine unzureichende Verzahnung mit zugrunde liegenden Prozessen sind jedoch erhebliche Hemmnisse, deren Bewältigung zunehmend eine strategische Herausforderung darstellt.

Ausgehend von einer Analyse der Charakteristika und Defizite heutiger Produktentwicklungsprozesse präsentiert die vorliegende Arbeit einen übergreifenden, ganzheitlichen und methodischen Integrationsansatz, welcher Teilmodelle eines technischen Produkts, zugehörige Schritte des Entwicklungsprozesses und eingesetzte Software-Werkzeuge gleichermaßen berücksichtigt. Im Mittelpunkt steht die Konzeption und Fundierung eines integrierten Produktmodells, das Produktinformationen über verschiedene Entwicklungsphasen hinweg an zentraler Stelle erfasst, zueinander in Beziehung setzt, spezifische Produktsichten anbietet und somit ein solides Fundament für die Koordination von Entwicklungsaufgaben sowie die Anbindung von Software-Werkzeugen bildet. Dabei werden Produktsichten als eigenständige Partialmodelle gemäß einer einheitlichen Modellierungstechnik aufgefasst, deren Querbezüge explizit festzulegen sind.

Der gewählte Ansatz der Produktmodellierung eröffnet einen flexiblen und differenzierten Umgang mit Konsistenzeigenschaften. Die Nutzung verschiedenartiger Techniken der Wissensrepräsentation zur Spezifikation von Querbezügen zwischen Produktsichten wird ebenso ermöglicht, wie eine Einbeziehung des Entwicklungskontexts in Konsistenzprüfungen und der Einsatz variabler Strategien der Konsistenzsicherung. Die praktische Anwendbarkeit der erarbeiteten Konzeption wird anhand der Fallstudie eines Turbinenrotors durchgängig illustriert. Mit Betrachtungen zur Automatisierung der Konsistenzsicherung schließt die Arbeit.

Danksagung

Die tief gehende Beschäftigung mit einem umfangreichen Thema über einen langen Zeitraum hinweg ist nur durch Unterstützung und Ermutigung von vielen Seiten möglich. Mein Dank gebührt allen, die auf ihre Weise zum Gelingen dieser Arbeit beigetragen haben.

Für die Möglichkeit, am Lehrstuhl für Software & Systems Engineering der Technischen Universität München in großer wissenschaftlicher Freiheit tätig zu sein, danke ich Herrn Prof. Dr. Dr. h.c. Manfred Broy. In gleichem Maße sage ich Dank für die wohlwollende und geduldige Betreuung dieser Arbeit. Herrn Prof. Dr. Klaus Bender danke ich für die aufgewendete Mühe und Zeit bei der Erstellung des zweiten Gutachtens. Ebenso gilt mein Dank Herrn Prof. Dr. Frank Schiller für sein Interesse an der Arbeit und vielfältige Anmerkungen.

Sehr besonderer Dank gebührt meinem aufrechten Kollegen Dr. Alexander Vilbig. Seine Ideen, Anregungen und nicht zuletzt offene Kritik zu den fachlichen Inhalten der Arbeit haben mich in ungezählten Diskussionen inspiriert und bereichert. Den Kolleginnen und Kollegen des gesamten Lehrstuhls danke ich für die angenehme Atmosphäre, die unkomplizierte Zusammenarbeit und so manche kurzweilige Diskussion in den letzten Jahren. Ganz gewiss nicht zuletzt gilt mein Dank meinen Freunden für ihre stetige Ermutigung, ihr Vertrauen und ihre Geduld.

Inhalt

Einführung

1 Einleitung

1.1 Motivation

Seit Anbeginn befasst sich die Menschheit mit der Erforschung, Entwicklung und stetigen Verbesserung technischer Erzeugnisse jeder Art. Allen voran haben die Fortschritte der Neuzeit die Konstruktion mannigfaltiger Werkzeuge, Maschinen, Apparate, Geräte und Instrumente in zuvor nie gekannter Vielfalt ausgelöst. Gestützt auf industrielle Fertigungsprozesse, welche eine effiziente Fabrikation in großen Stückzahlen ermöglichen, stellen technische Produkte wie Telekommunikationsgeräte, Automobile oder Software-Systeme heute unentbehrliche Hilfsmittel moderner Gesellschaften dar, die in nahezu jeden Bereich menschlichen Handelns hineinwirken.

Stetig steigende Anforderungen an Funktionsweise und Leistungsmerkmale, kontinuierlich verbesserte Produktionsverfahren sowie gleich bleibend hohe Qualitätserwartungen führen über alle Wirtschaftsbereiche hinweg zu einer signifikant anwachsenden Komplexität technischer Produkte, die sich in umfangreichen, vielschichtigen und eng vernetzten Entwicklungsabläufen widerspiegelt. Um am Markt erfolgreich bestehen zu können, sind Unternehmen aufgrund eines zunehmenden und oftmals globalen Wettbewerbsdrucks zugleich gezwungen, Entwicklungskosten zu reduzieren und Entwicklungszyklen zu verkürzen.

Unter den genannten Gesichtspunkten kommt dem Einsatz moderner Kommunikations- und Informationstechnologien eine herausragende Rolle zu. Durch ihre Verwendung haben Ingenieursdisziplinen einen grundlegenden Wandel erfahren. Ist zuvor eine Konstruktion realer Prototypen eines Produkts erforderlich gewesen, um konstruktive Annahmen über angestrebte Produkteigenschaften in aufwändigen Versuchen zu überprüfen, so erlauben es Software-Lösungen, digitale Modelle von Produkten zu bilden, welche im Vergleich eine bedeutend kostengünstigere und schnellere Erstellung, Bearbeitung und Simulation ermöglichen.

In der Vergangenheit hat sich im Ingenieursumfeld ein umfangreicher

Markt für zahlreiche kommerzielle Anbieter vielfältiger und ausgereifter Software-Werkzeuge etabliert. Jedoch sind diese in aller Regel auf die optimale Unterstützung fachspezifischer Einzelaktivitäten fokussiert, ohne die Einbettung in übergreifende Produktentwicklungsprozesse ausreichend zu berücksichtigen. In der Folge entstehen auf diese Weise komplexe und heterogene IT-Landschaften aus isolierten und inkompatiblen Software-Lösungen mit proprietären und oftmals inkonsistenten Modellen eines zu entwickelnden Produkts.

Die genannten Defizite führen in ihrer Gesamtheit zu erheblichen Einschränkungen, Hemmnissen und Reibungsverlusten, deren Bewältigung und Überwindung zunehmend an Bedeutung gewinnen und für zahlreiche Unternehmen eine strategische Herausforderung darstellen. Eine signifikante Steigerung der Effizienz und Effektivität heutiger Entwicklungsprozesse lässt sich in erster Linie nicht länger durch die Weiterentwicklung bestehender Software-Lösungen zur Unterstützung fachspezifischer Einzelaktivitäten erzielen, sondern erfordert vielmehr einen ganzheitlichen, übergreifenden und methodischen Ansatz zu ihrer Integration.

Diese vielversprechende Zielsetzung wirft eine Reihe grundlegender, weit reichender und anspruchsvoller Fragestellungen auf, die sich auf verschiedenste Gebiete des Software und Systems Engineering erstrecken. So sind fachliche wie technische Lösungsansätze zu erarbeiten, um die in aller Regel zahlreichen, vielfältigen und sehr unterschiedlichen Modelle eines technischen Produkts, zugrunde liegende Aktivitäten und Subprozesse über verschiedene Fachdisziplinen hinweg sowie eingesetzte Software-Werkzeuge unterschiedlicher Technologien, Systemplattformen und Entwicklungsumgebungen sowohl untereinander als auch miteinander zu verknüpfen.

Insbesondere eine übergreifende und durchgängige Integration verwendeter Teilmodelle aus allen Phasen eines Entwicklungsprozesses stellt aufgrund der meist hohen Modellkomplexität, signifikanten Abweichungen in den je individuellen Auffassungen beteiligter Produktentwickler und Fachdisziplinen sowie Divergenzen zwischen eingesetzten Modellierungsarten eine überaus anspruchsvolle Aufgabe dar. Eine besonders wichtige Rolle spielt die Erarbeitung flexibler Mechanismen für einen differenzierten und kontrollierten Umgang mit Querbezügen, Abhängigkeiten und Überschneidungen zwischen Modellen.

In den folgenden Kapiteln entwickelt die vorliegende Arbeit Integrationskonzepte zur Überwindung der genannten Problemstellungen. Die dabei gewählte Vorgehensweise ist an drei Grundprinzipien orientiert: Im Vordergrund steht eine wissenschaftlich fundierte Behandlung, die auf der Grundlage einer eingehenden Analyse mögliche Lösungsansätze gegeneinander abwägt und schließlich eine Auswahl trifft. Darüber hinaus werden stets die An-

forderungen, Möglichkeiten und Beschränkungen einer praxisgerechten Umsetzung berücksichtigt. Schließlich verfolgt die Arbeit durchgängig die Einbindung, Anpassung oder Erweiterung bewährter Ansätze.

1.2 Zielsetzung und Beitrag

Die industrielle Entwicklung komplexer technischer Produkte ist geprägt durch umfangreiche und anspruchsvolle Arbeitsabläufe aus aufwändigen und spezialisierten Aktivitäten in einer hochgradig interdisziplinären Umgebung. Sie erfordert ein koordiniertes Zusammenspiel vieler Bearbeiter, die ein je eigenes Verständnis des Produkts aufweisen, mit unterschiedlichsten Teilaufgaben einer Produktentwicklung befasst sind und durch verschiedenste Software-Werkzeuge unterstützt werden. Gemeinsames Ziel ist der Entwurf und die Fertigung qualitativ hochwertiger Produkte aus vielfältigen und zahlreichen Bestandteilen.

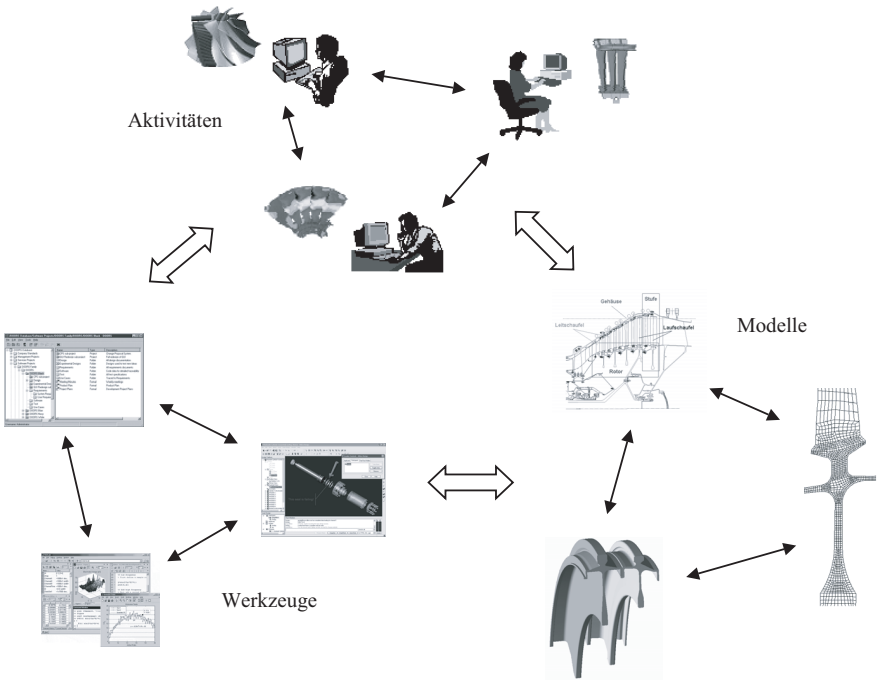

Abbildung 1.1: Ausgangssituation

In Abbildung 1.1 ist diese Ausgangssituation schematisch dargestellt. So repräsentieren zum einen Aktivitäten die verschiedenen Aufgaben einer Produktentwicklung, welche je individuell durch Software-Werkzeuge unterstützt werden. Jedes Werkzeug benutzt dazu üblicherweise ein spezifisches Modell, um ausgesuchte und im aktuellen Kontext relevante Aspekte eines technischen Produkts optimal abzubilden. Aus übergreifender Sicht sind Aktivitäten, Werkzeuge und Modelle jedoch nicht isoliert, sondern weisen vielmehr zahlreiche Wechselwirkungen auf, sowohl untereinander, als auch miteinander.

Ihre unzureichende Integration in heutigen Produktentwicklungsprozessen führt zu erheblichen Hemmnissen: Einerseits sind die Aufgaben einer Produktentwicklung und insbesondere ihr Zusammenspiel über Fachdisziplinen hinweg unzureichend erfasst und koordiniert. Verwendete Modelle werden vorwiegend unabhängig voneinander behandelt und auf proprietäre Software-Werkzeuge und ihre Formate ausgerichtet. Dies verhindert eine Erfassung von Querbezügen, Abhängigkeiten und Überschneidungen. Verschiedene Software-Technologien, Systemplattformen und Entwicklungsumgebungen führen schließlich zu zahlreichen technischen Schwierigkeiten.

Existierende Ansätze zur Überwindung der genannten Problemstellungen – etwa aus dem Bereich der Geschäftsprozessmodellierung oder dem als Enterprise Application Integration (EAI) bezeichneten Umfeld – sind auf eine Integration in Teilbereichen fokussiert und tragen somit nur eingeschränkt zu einer Lösung bei. Die zentrale Zielsetzung der vorliegenden Arbeit ist dagegen die Erarbeitung eines ganzheitlichen, übergreifenden und methodischen Integrationsansatzes, der verwendete Modelle eines Produkts, zugrunde liegende Entwicklungsprozesse und eingesetzte Software-Werkzeuge gleichermaßen berücksichtigt.

Gemäß dieser Zielsetzung ist es nahe liegend, in einem ersten Schritt zwischen den beiden Dimensionen der *horizontalen Integration* und der *vertikalen Integration* zu unterscheiden:

Horizontale Integration: Mit horizontaler Integration wird die Integration von Aktivitäten eines Entwicklungsprozesses, verwendeten Produktmodellen und eingesetzten Software-Werkzeugen je untereinander bezeichnet. Dies führt auf die Begriffe der *Prozessintegration*, der *Produktintegration* und der *Werkzeugintegration*.

Vertikale Integration: Die vertikale Integration hat darüber hinaus die Aufgabe, Entwicklungsprozesse, Produktmodelle und Software-Werkzeuge in übergreifender Weise miteinander zu integrieren. Sie stützt sich auf Lösungen der Prozessintegration, der Produktintegration und der Werkzeugintegration ab.

Während vorhandene Ansätze vorwiegend dem Bereich der horizontalen Integration zuzuordnen sind, zielt die vorliegende Arbeit gemäß der beschriebenen Zielsetzung auf die Entwicklung eines vertikalen Integrationsansatzes ab. In diesem Rahmen wird für die Bereiche der Prozessintegration und Werkzeugintegration auf bereits existierende Lösungen zurückgegriffen (vgl. Abschnitt 1.3). Eine hinreichend leistungsfähige, modellbasierte Produktintegration erfordert dagegen die Erarbeitung neuartiger Lösungsansätze, die den Kern dieser Arbeit bilden. Insgesamt lassen sich die Ergebnisse wie folgt zusammenfassen:

Integrationsansatz: Die Arbeit präsentiert einen vertikalen Integrationsansatz für digitale Modelle eines komplexen technischen Produkts, zugrunde liegende Entwicklungsprozesse und eingesetzte Software-Werkzeuge.

- Als Ausgangspunkt der Überlegungen werden die Charakteristika moderner Produktentwicklungsprozesse und ihre Defizite aus der Sicht einer integrativen Unterstützung durch Software-Lösungen herausgearbeitet.
- Eine konkrete Fallstudie aus dem Bereich der Turbinenentwicklung verdeutlicht den Umfang und die Komplexität einer Produktentwicklung und dient durchgängig zur Validierung der erarbeiteten Ergebnisse.
- Die Arbeit stellt die drei horizontalen Integrationsansätze der Prozessintegration, der Produktintegration und der Werkzeugintegration schematisch vor und diskutiert individuelle Vorteile, Nachteile und Beschränkungen.
- Schließlich werden die Anforderungen an einen vertikalen Integrationsansatz präzisiert, ein geeigneter Architekturentwurf für eine übergreifende Integrationsplattform abgeleitet und das Zusammenspiel ihrer Komponenten aufgezeigt.

Produktmodellierung: Im Mittelpunkt der Arbeit steht die Konzeption und Fundierung eines integrierten Produktmodells, das Produktinformationen durchgängig erfasst, zueinander in Beziehung setzt und spezifische Produktsichten ermöglicht.

- Es werden die wesentlichen Anforderungen formuliert, die vor dem Hintergrund des übergreifenden Integrationsansatzes an ein Gesamtmodell für Produktinformationen aus allen Phasen eines Entwicklungsprozesses zu stellen sind.

- Die Arbeit propagiert den Lösungsansatz eines segmentierten Produktmodells als Kombination eigenständiger und gleich berechtigter Partialmodelle gemäß einer gemeinsamen Modellierungstechnik sowie explizit definierter Querbezüge.
- Als zentrales Ergebnis wird ein Metamodell für komplexe technische Produkte vorgestellt, das neben Produktsichten und ihren Querbezügen die Erfassung von Produktstrukturen und elementaren Produkteigenschaften ermöglicht.
- Ein formales, mathematisch fundiertes Modell präzisiert die Semantik der im Metamodell enthaltenen Modellabstraktionen mit Mitteln der Graphentheorie und Techniken der algebraischen Spezifikation.

Konsistenzsicherung: Ein wesentlicher Beitrag der Arbeit ist die Entwicklung von Konzepten, Techniken und Strategien für einen differenzierten und flexiblen Umgang mit Konsistenzeigenschaften eines segmentierten Produktmodells.

- Die Arbeit definiert einen praxisgerechten Konsistenzbegriff, der vollständig den Entwicklungskontext einbezieht und leitet daraus eine dreigeteilte Methodik zur Festlegung, Prüfung und Sicherung von Konsistenzeigenschaften ab.
- Existierende Techniken der Wissensrepräsentation und Wissensverarbeitung werden erläutert, miteinander verglichen und Einsatzmöglichkeiten zur Sicherung der Konsistenz eines segmentierten Produktmodells diskutiert.
- Es wird gezeigt, wie sich die Schritte einer Konsistenzsicherung flexibel und kontrolliert anhand verschiedener Strategien als spezifische Aktivitäten in übergreifende Entwicklungsprozesse integrieren lassen.
- Schließlich untersucht die Arbeit Wechselwirkungen und potenzielle Konflikte zwischen Konsistenzsicherungsprozessen, ermittelt prinzipielle Grenzen ihrer Automatisierbarkeit und leitet ein algorithmisches Verfahren ab.

1.3 Einordnung

Angesichts der Breite und Reichhaltigkeit der Themenstellung weist die vorliegende Arbeit zahlreiche Querbezüge zu verschiedensten Teilgebieten der Informatik und darüber hinaus auf. In der folgenden Übersicht sind die wichtigsten verwandten Ansätze – ohne Anspruch auf Vollständigkeit – nach thematischen Schwerpunkten gruppiert dargestellt:

Integrationsansätze

Um Produktdaten in komplexen Entwicklungsprozessen zu organisieren, werden in der Praxis seit geraumer Zeit Software-Lösungen des *Product Data Management (PDM)* bzw. des *Engineering Data Management (EDM)* eingesetzt (s. beispielsweise [AMD96], [Ben01], [Sch99], [BKS+96, BKZ+97]). Diese ermöglichen u.a. die verteilte Bearbeitung, persistente Speicherung, Versionierung und Archivierung von Entwicklungsdokumenten. PDM-Systeme und EDM-Systeme dienen jedoch in erster Linie als zentrale Datenspeicher und bieten weder eine vertiefte Integration von Produktmodellen auf der Ebene von Modellelementen, noch unterstützen sie die Integration von Entwicklungsaktivitäten oder Software-Werkzeugen.

Als Weiterentwicklung des Product Data Management wird unter dem Oberbegriff *Product Lifecycle Management (PLM)* eine Reihe verschiedener Ansätze zusammengefasst, die – wie die vorliegende Arbeit – auf eine übergreifende Integration von Entwicklungsprozessen, Produktdaten und Software-Werkzeugen abzielen [SI03], [Tea05, IBM05]. Ergänzend zu einer einheitlichen Verwaltung von Produktdaten propagieren diese die Etablierung durchgängiger Prozessketten über alle Phasen eines Produktlebenszyklus hinweg. Allerdings bieten auch PLM-Lösungen keine hinreichend granulare Verknüpfung verwendeter Modelle und sind zudem vorwiegend auf die Software-Werkzeuge je eines Herstellers zugeschnitten.

Auf den Bereich der Prozessintegration sind auch die Ansätze der im Jahre 1993 gegründeten *Workflow Management Coalition (WfMC)* [WfM04] ausgerichtet. Diese bietet verschiedene Standards, um beliebige Entwicklungsprozesse zu modellieren und heterogene Workflow-Anwendungen sowohl untereinander, als auch mit verschiedenen Software-Systemen zu integrieren [WPD98b, XPD02, WfM98]. Aufgrund der Fokussierung auf Prozessaspekte werden Datenmodelle in den Arbeiten der Workflow Management Coalition jedoch nur am Rande berücksichtigt. Folglich ist lediglich eine lose und allgemeine Kopplung von Aktivitäten eines Entwicklungsprozesses mit zugehörigen Modellen möglich.

Lösungen des als *Enterprise Application Integration (EAI)* benannten Be-

reichs nutzen Middleware-Technologien wie J2EE [J2E05], [FCF02, MH02] oder CORBA [COR00] für die technische Integration heterogener, meist betriebswirtschaftlicher Software-Anwendungen innerhalb eines Unternehmens. Sie unterstützen zudem die Erfassung und Steuerung von Geschäftsprozessen [Kai02, Kel02], [Biz05, Net05]. Basierend auf Schnittstellenstandards und Adaptoren ermöglichen EAI-Lösungen zwar eine feingranulare Kopplung und Transformation von Datenmodellen durch die Einbindung freien Programmcodes. Jedoch werden Querbezüge zwischen Modellelementen nicht explizit spezifiziert und sind somit intransparent sowie schwer zu modifizieren.

Segmentierte Produktentwicklung

Die Grundidee, Sichten auf komplexe Systeme oder Abläufe zu bilden, um ausgesuchte Aspekte besser zu verstehen und einfacher zu beherrschen, ist ein bekanntes Prinzip der Informatik. So finden sich unter dem Oberbegriff *(multi-dimensional) separation of concerns* [TOHS99] zahlreiche Forschungsansätze, um verschiedene Aspekte großer Software-Systeme voneinander getrennt zu entwerfen und das Gesamtsystem gemäß spezifizierter Regeln weit gehend automatisiert zu generieren. Als Weiterentwicklung der Objektorientierung seien insbesondere das *Subjekt-Orientierte Design (SOD)* bzw. die *Subjekt-Orientierte Programmierung (SOP)* [HO93, CHOT99, Vli98] sowie *Hyperspaces* und *Hyperslices* [OT00, NCA99], [Hyp05] genannt.

Eine Übertragung auf komponentenbasierte Software-Systeme und ihre Verhaltensschnittstellen unter Nutzung markierter Transitionssysteme wird ausführlich in [GV03a] aufgezeigt. In vereinfachter Form wird die Vorstellung der Aufteilung eines Systems in Sichten auch von der an Bedeutung gewinnenden *Aspekt-Orientierten Programmierung (AOP)* [KLM+97], [Asp05] aufgegriffen, die sich jedoch darauf beschränkt, nur vollständig orthogonale Aspekte eines Systems voneinander zu separieren. Zahlreiche Arbeiten, die das Konzept der Perspektive bzw. Sicht als primäre Einheit der Systementwicklung auffassen, finden sich im Umfeld des *ViewPoint Framework* [FKG90, FKN+92, EFKN94].

Die Vorstellung, komplexe Systeme vollständig durch Sichten beschreiben zu können, wird schließlich auch in modernen Vorgehensmodellen, Modellierungstechniken und Entwicklungswerkzeugen genutzt. So propagiert beispielsweise der weit verbreitete *Rational Unified Process (RUP)* [Kru99] ein *4+1-Sichtenmodell* zur Darstellung von Software-Architekturen. Die ebenfalls weithin bekannte *Unified Modeling Language (UML)* [UML03] definiert eine Reihe von Diagrammtypen als Sichten auf ein System. Entsprechend unterstützen auch Entwicklungswerkzeuge [Tog05, Tel05, Aut05] eine Aufteilung in Sichten, um z.B. die statische Dekomposition eines Systems und

seine dynamische Ablaufstruktur getrennt zu modellieren.

Die vorliegende Arbeit überträgt die Vorstellung einer segmentierten Systementwicklung auf die Modellierung komplexer technischer Produkte. In diesem Zusammenhang sind Sichten durch die in Produktentwicklungsprozessen verwendeten Teilmodelle gegeben, welche je ausgesuchte Aspekte eines Produkts beschreiben. Im Gegensatz zur Subjekt-Orientierten Programmierung und Aspekt-Orientierten Programmierung besteht die Zielsetzung jedoch nicht darin, Teilmodelle durch Komposition schließlich in ein Gesamtmodell zu überführen. Vielmehr werden Querbezüge zwischen Sichten explizit als eigenständige Modellelemente spezifiziert, um eine hinreichend flexible Behandlung der Modellkonsistenz zu ermöglichen.

Modellbildung

Die Erstellung von Modellen in Anwendungsbereichen aller Art ist eine zentrale Aufgabe der Informatik. Folgerichtig stehen zur Modellbildung ebenso zahlreiche wie vielfältige Modellierungstechniken mit unterschiedlichen Abstraktionen und Konzepten zur Verfügung. Einen allgemein verwendbaren und leicht verständlichen Ansatz bietet das bereits im Jahre 1976 vorgeschlagene *Entity-Relationship-Modell* [Che76]. Es basiert auf den Modellelementen der Entität (engl.: *entity*), um wesentliche, klar voneinander abgrenzbare Gegenstände eines Anwendungsbereichs zu repräsentieren, sowie der Beziehung (engl.: *relationship*) zwischen Entitäten. Charakteristische Eigenschaften von Entitäten lassen sich zudem durch Attribute erfassen.

Trotz der weiten Verbreitung des Entity-Relationship-Modells in Wissenschaft und Praxis werden seit geraumer Zeit vorwiegend objektorientierte Modellierungstechniken – allen voran die *Unified Modeling Language (UML)* [UML03], [Oes04, Fow03] – zur Modellbildung eingesetzt. Diese definiert verschiedene Diagrammtypen, um sowohl den Aufbau als auch das Verhalten von Software-Systemen auf graphische Weise zu beschreiben und geht weit über die Möglichkeiten des Entity-Relationship-Modells hinaus. Aufgrund ihres Umfangs und einer unzureichend fundierten Semantik weist die Unified Modeling Language jedoch eine Reihe an Unklarheiten und Mehrdeutigkeiten auf.

Ein umfassender, integrativer und mathematisch präzise fundierter Ansatz der Modellbildung wird in [Bom99] vorgestellt. Er verbindet zahlreiche bewährte Modellabstraktionen zur Beschreibung von Strukturen und Abläufen zu einer einheitlichen und leistungsfähigen Modellierungstechnik auf der Grundlage von Graphen mit Knoten- und Kantenmarkierungen. Insbesondere wird eine potenziell unbegrenzte Anzahl an Abstraktionsebenen explizit unterstützt, um eine gleichartige Erfassung von Modellen und Meta-

modellen zu ermöglichen. Schließlich seien noch *Semantische Netze* [CQ69, Sow83] und *Ontologien* [CJB99], [UG96] als sehr allgemeine und variantenreiche Möglichkeiten der Modellbildung erwähnt.

Speziell auf die Modellierung technischer Produktdaten ist die im Jahre 1994 veröffentlichte Normenreihe *ISO 10303 – Product Data Representation and Exchange* [STE94] ausgerichtet. Bekannt unter der inoffiziellen Bezeichnung *STEP*, wird darin die Modellierungssprache EXPRESS zusammen mit einer graphischen Beschreibungstechnik definiert [EXP94]. Im Kern entspricht STEP dem Entity-Relationship-Modell, angereichert durch ein Generalisierungskonzept und Möglichkeiten zur Festlegung von Produktregeln. Jedoch sind so erstellte Modelle nicht ohne Weiteres mit etablierten Modellierungstechniken des Software Engineering vereinbar (vgl. z.B. [Ler00]), so dass STEP in der Praxis bisher nur zögerlich eingesetzt wird.

Die in der vorliegenden Arbeit entwickelte Modellierungstechnik für komplexe technische Produkte bzw. ihre Produktsichten baut auf das Entity-Relationship-Modell auf, um Produktbestandteile und ihre Beziehungen zu beschreiben. Aus objektorientierten Modellierungstechniken wird das Konzept der Generalisierung bzw. Spezialisierung sowie die Trennung zwischen Typ- und Instanzebene übernommen. Auf verhaltensbezogene Modellierungskonstrukte kann jedoch verzichtet werden. Die formale Fundierung des gewählten Ansatzes setzt – angelehnt an [Bom99], jedoch beschränkt auf die angestrebte Ausdruckskraft – ebenfalls Graphen ein. Problematische Modellabstraktionen des STEP-Ansatzes werden dagegen vermieden.

Wissensrepräsentation und Konsistenzsicherung

Ergänzend zu der Erstellung von Modellen bietet die Forschungsrichtung der *Künstlichen Intelligenz (KI)* vielfältige Ansätze zur Repräsentation und Verarbeitung komplexer Wissenszusammenhänge eines Anwendungsgebiets. Diese werden insbesondere im Rahmen wissensbasierter Systeme bzw. Expertensysteme [Lus90], [Pup88] genutzt und unter dem Oberbegriff *Knowledge-Based Engineering (KBE)* auch in Software-Lösungen des Ingenieurwesens eingesetzt [San03]. In erster Linie handelt es sich dabei um an spezifische Fachgebiete angepasste logische Kalküle insbesondere der Prädikatenlogik oder nicht-klassischer Logiken, um regelbasierte Systeme oder um geeignete Constraint-Techniken.

Aus dem Bereich XML-basierter Technologien stammt der Ansatz *xlinkit* [NCEF02], welcher ein prädikatenlogisches Kalkül erster Ordnung einführt, um Querbezüge zwischen XML-Modellen [W3C04] zu spezifizieren und eine automatisierte Prüfung der Konsistenz zu ermöglichen. Auf dem Gebiet der Graphentheorie stehen Tripelgraphgrammatiken [Sch94], [Roz97] zur Festle-

gung von Konsistenzregeln zwischen Graphen zur Verfügung. Ein umfassender und durchgängig formal fundierter Ansatz, um Beziehungen und Transformationsregeln zwischen objektorientierten Modellen bzw. Metamodellen festzulegen, ist durch die *Bidirectional Object-Oriented Transformation Language (BOTL)* [BM02, BM03, Bra03b] gegeben.

Entsprechend der verfolgten Zielsetzung einer übergreifenden und durchgängigen Modellintegration entwickelt die vorliegende Arbeit einen allgemeinen Rahmen, um verschiedenartige, formale wie informelle Techniken der Wissensrepräsentation und Wissensverarbeitung zur Festlegung und Auswertung von Überschneidungen, Querbezügen und Abhängigkeiten zwischen Modellelementen und Produktsichten eines integrierten Produktmodells einzusetzen. Auf dieser Grundlage lassen sich die genannte Ansätze – abhängig von ihrer individuellen Ausdruckskraft und Leistungsfähigkeit – als Bausteine in die vorgeschlagene Methodik einer flexiblen und differenzierten Behandlung der Modellkonsistenz modular integrieren.

1.4 Inhalt und Aufbau

Die Arbeit gliedert sich in vier Teile mit insgesamt acht Kapiteln. Der erste Teil – „Einführung“ – stellt die Anwendungsdomäne der Entwicklung komplexer technischer Produkte vor, präsentiert eine Fallstudie aus dem Bereich des Turbinenbaus, erläutert bestehende Ansätze zur Integration von Prozessen, Modellen und Werkzeugen und leitet schließlich einen übergreifenden Integrationsansatz gemäß Abschnitt 1.2 ab, welcher durchgängig den Rahmen der Arbeit bildet.

Darauf aufbauend erfolgt im zweiten Teil – „Produktmodellierung“ – der detaillierte Entwurf und die mathematisch präzise Fundierung eines integrierten Produktmodells, welches den wesentlichen und entscheidenden Bestandteil des vorgeschlagenen Integrationsansatzes darstellt. Dieses verbindet die grundlegende Vorstellung einer segmentierten Produktentwicklung gemäß gleich berechtigter, je individuell strukturierter Produktsichten mit bewährten Abstraktionen der Modellbildung.

Der dritte Teil der vorliegenden Arbeit – „Konsistenzsicherung“ – wendet sich der Erfassung und Behandlung von Überschneidungen, Querbezügen und Abhängigkeiten zwischen Modellelementen unterschiedlicher Produktsichten zu. Er entwickelt eine übergreifende Methodik für einen flexiblen, differenzierten und kontrollierten Umgang mit Konsistenzeigenschaften eines segmentierten Produktmodells unter Einbindung verschiedenartiger Techniken der Wissensrepräsentation und Wissensverarbeitung.

Im vierten Teil – „Epilog“ – werden die erarbeiteten Ergebnisse schließ-

lich zusammengefasst. Sie sind im Einzelnen wie folgt auf die verschiedenen Kapitel verteilt:

Kapitel 1 – Einleitung: In Kapitel 1 wird die der Arbeit zugrunde liegende Aufgabenstellung motiviert, die zentrale Zielsetzung erläutert, eine Einordnung in verwandte Ansätze vorgenommen und ein Überblick über den Inhalt gegeben.

Kapitel 2 – Produktentwicklung: Kapitel 2 betrachtet die Charakteristika modellbasierter Entwicklungsprozesse komplexer technischer Produkte aus der Sicht einer durchgängigen Unterstützung durch Software-Werkzeuge und arbeitet die wesentlichen Integrationsdefizite in heutiger Zeit heraus. Zudem erfolgt die Einführung des zur Illustration und Validierung gewählten Fallbeispiels einer Niederdruckturbine als wichtiges und komplexes Bauteil moderner Flugzeugtriebwerke.

Kapitel 3 – Integrationsansätze: Kapitel 3 diskutiert die drei Grundprinzipien der Prozessintegration, der Produktintegration sowie der Werkzeugintegration hinsichtlich ihrer Vorteile und Beschränkungen. Als geeignete Kombination wird daraus ein übergreifender und hinreichend leistungsfähiger Integrationsansatz abgeleitet, der Architekturentwurf einer Software-technischen Integrationsplattform vorgestellt und das Zusammenspiel ihrer Komponenten aufgezeigt.

Kapitel 4 – Entwurf des Produktmodells: In Kapitel 4 werden die Anforderungen an ein integriertes Produktmodell präzisiert, die grundlegenden Alternativen zur Unterstützung spezifischer Produktsichten eingehend erläutert und diskutiert, erforderliche Modellabstraktionen zur Erfassung von Produktstrukturen und Produkteigenschaften ausgewählt und schließlich in einem Metamodell zur Beschreibung technischer Produkte vereinigt.

Kapitel 5 – Fundierung: Kapitel 5 nutzt Mittel der Graphentheorie und Techniken der algebraischen Spezifikation zur formalen Fundierung der Semantik des in Kapitel 4 entwickelten Metamodells für technische Produkte. Auf dieser Grundlage werden die durch Entwicklungsprozesse induzierten Bearbeitungsschritte eines Produktmodells als mathematische Abbildungen zwischen Graphstrukturen interpretiert und klassifiziert.

Kapitel 6 – Techniken und Strategien: Ausgehend von der in Kapitel 4 erarbeiteten Konzeption eines segmentierten Produktmodells erläutert

Kapitel 6 Techniken und Strategien für eine flexible Prüfung und Sicherung der Modellkonsistenz. Es wird eine praxisgerechte und leistungsfähige Methodik eingeführt, die verschiedenartige Techniken zur Spezifikation von Konsistenzbedingungen vorsieht und den Entwicklungskontext in Konsistenzprüfungen vollständig berücksichtigt.

Kapitel 7 – Automatisierung: Aufbauend auf die in Kapitel 5 gewählte Formalisierung eines Produktmodells und die in Kapitel 6 erarbeitete Methodik untersucht Kapitel 7 die Automatisierung von Konsistenzsicherungsprozessen sowie potenzielle Konfliktsituationen. Es werden prinzipielle Grenzen der Automatisierbarkeit aufgezeigt, Kriterien eines konfliktfreien Ablaufs formuliert und ein algorithmisches Verfahren abgeleitet.

Kapitel 8 – Zusammenfassung: Schließlich präsentiert Kapitel 8 einen Überblick über die wesentlichen Ergebnisse der vorliegenden Arbeit und rekapituliert ihr Zusammenspiel zur Erfüllung der gewählten Zielsetzung.

Kapitel 4 und Kapitel 6, welche die Grundkonzepte eines integrierten Produktmodells festlegen und flexible Mechanismen der Konsistenzsicherung einführen, bilden den Kern der vorliegenden Arbeit. Die in Kapitel 5 entwickelte formale Fundierung ist für das Verständnis von Kapitel 6 nicht zwingend erforderlich. Die dort vorgestellten Modelltransformationen dienen jedoch als Grundlage für die in Kapitel 7 untersuchte Automatisierung von Konsistenzsicherungsprozessen.

2 Produktentwicklung

Die Entwicklung und Herstellung technischer Produkte in großen Stückzahlen bildet einen wesentlichen Bestandteil des Wirtschaftslebens moderner Gesellschaften. Wie sich in verschiedensten Industriezweigen beobachten lässt, steigt die Komplexität gefertigter Produkte und zugehöriger Entwicklungsprozesse kontinuierlich an. Die Ursache liegt in einem zunehmenden und oftmals globalen Wettbewerbsdruck, der unter gleich bleibend hohen Qualitätserwartungen wachsende Anforderungen an die Funktionsweise industrieller Produkte stellt und eine stetige Verbesserung eingesetzter Verfahren und Techniken erfordert.

Vor diesem Hintergrund kommt der Nutzung zeitgemäßer Informations- und Kommunikationstechnologien in der Entwicklung und Fertigung komplexer technischer Produkte eine bedeutende Rolle zu. Diese bieten zahlreiche Lösungen zur Unterstützung von Entwicklungsprozessen und erlauben es, digitale Modelle von Produkten zu bilden, welche im Vergleich zu realen Modellen wesentlich kostengünstiger und schneller bearbeitet werden können. Folgerichtig existiert im Ingenieursumfeld seit geraumer Zeit ein umfangreicher Markt für zahlreiche, kommerzielle Anbieter vielfältiger und ausgereifter Software-Werkzeuge.

Die unzureichende Integration dieser Software-Lösungen führt jedoch unter vielerlei Gesichtspunkten zu erheblichen Hemmnissen und Einschränkungen in der weiteren Verbesserung heutiger Produktentwicklungsprozesse. In den folgenden Abschnitten werden gemäß der Zielsetzung der vorliegenden Arbeit – der Konzeption eines integrativen Ansatzes zur durchgängigen und modellbasierten Entwicklung technischer Produkte – die wesentlichen Ursachen auftretender Defizite zusammen mit den spezifischen Charakteristika und Anforderungen der Software-gestützten Produktentwicklung in ihrer heutigen Form herausgearbeitet.

Abschnitt 2.1.1 geht zunächst auf die verschiedenen Phasen des Lebenszyklus komplexer technischer Produkte ein. Darauf aufbauend werden in Abschnitt 2.1.2 wesentliche Eigenschaften ihrer Entwicklungsprozesse vorgestellt, während in Abschnitt 2.1.3 die bedeutende Rolle digitaler Modell

erläutert wird. Einen Überblick über den Einsatz von Software-Werkzeugen gibt Abschnitt 2.1.4. Abschließend präsentiert Abschnitt 2.2 eine Fallstudie aus dem Bereich der Turbinenentwicklung, welche für den weiteren Fortgang der Arbeit eine durchgängige Grundlage zur Illustration sowie Validierung entwickelter Konzepte zur Verfügung stellt.

2.1 Prozesse, Modelle und Werkzeuge

2.1.1 Phasen eines Produktlebenszyklus

Als Ausgangspunkt der Überlegungen dient der Begriff des Produktlebenszyklus, welcher in seiner allgemeinen Form den Kreislauf der regelmäßig wiederkehrenden Abschnitte bezeichnet, die in der Entwicklung und Verwertung eines technischen Produkts auftreten. Trotz der vielfältigen Unterschiede, die beispielsweise in Hinblick auf die Dauer oder die spezifischen Verfahren der Konstruktions- und Herstellungsprozesse oder verschiedenen Einsatzzwecken unterschiedlicher Produkte anzutreffen sind, lassen sich Produktlebenszyklen in übergreifender Weise in abgrenzbare und stets auftretende Abschnitte unterteilen. In Anlehnung an [AMD96] werden die folgenden sechs Phasen unterschieden:

Planung: Im Zuge der Produktplanung werden zu Beginn der Produktentwicklung grundsätzliche Rahmenbedingungen festgelegt. Hierzu zählen elementare Funktionsmerkmale des zu fertigenden Produkts ebenso, wie eine Festlegung zeitlicher und finanzieller Richtlinien sowie eine Aufstellung benötigter Ressourcen.

Konstruktion: Die Konstruktionsphase umfasst den kreativen Prozess des Entwurfs eines Produkts. Ausgehend von seinem Einsatzzweck werden auf der Grundlage der Kenntnisse und Fähigkeiten der beteiligten Entwickler alle erforderlichen Überlegungen zur Gestaltung des Produkts durchgeführt und detaillierte Pläne zu seiner Fertigung ausgearbeitet.

Herstellung: In dieser Phase erfolgt schließlich die Fertigung eines Produkts gemäß den in der Konstruktionsphase entstandenen Plänen. Die Voraussetzung hierfür ist, dass alle erforderlichen Produktionsmittel, wie Fertigungsmaschinen, Rohstoffe oder Zwischenprodukte zur Verfügung stehen und ein Produktionsplan vorliegt.

Vertrieb: Als Schnittstelle zwischen Produktion und Konsumtion ist es die Aufgabe des Vertriebs, das entstandene Produkt in die Nutzung

überzuleiten. Dies kann beispielsweise die Durchführung eines geeigneten Marketings oder den Transport des Produkts zu seinem Bestimmungsort umfassen.

Nutzung: Während der Produktnutzung erfüllt des gefertigte Produkt seine spezifische Funktion gemäß seines Einsatzzwecks. Ebenfalls in dieser Phase enthalten sind ggf. erforderliche Wartungs- und Reparaturmaßnahmen, um die Funktionsweise des Produkts zu erhalten, wieder herzustellen oder abzuändern.

Entsorgung: Nach dem Ende der Produktnutzung, beispielsweise aufgrund eines endgültigen Funktionsversagens des Produkts, tritt die Phase der Entsorgung als letzter Abschnitt des Produktlebenszyklus in Erscheinung. Darin enthalten ist insbesondere das Recycling wieder verwertbarer Produktbestandteile.

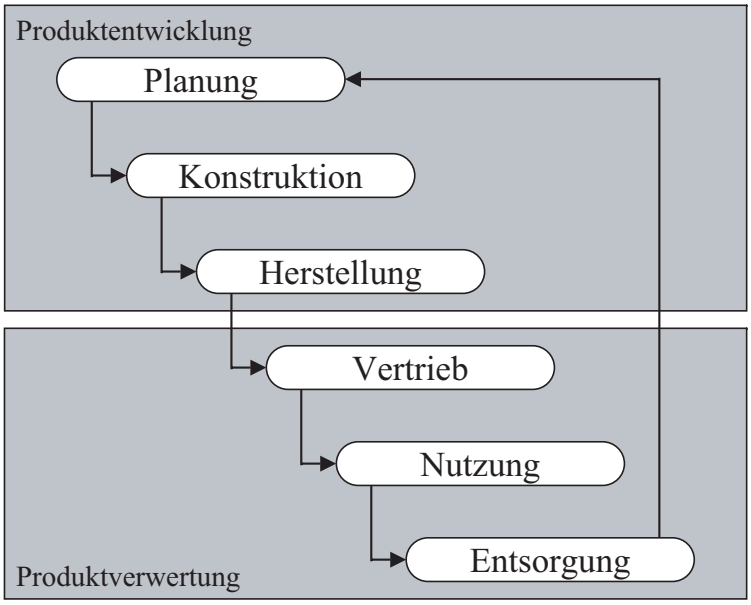

Abbildung 2.1: Phasen eines Produktlebenszyklus

Abbildung 2.1 zeigt die einzelnen Phasen eines Produktlebenszyklus im Überblick. Während Planung, Konstruktion und Herstellung der Produktentwicklung dienen, sind der Vertrieb, die Nutzung und die Entsorgung der Produktverwertung zuzuordnen. Unter den verschiedenen Phasen eines Produktlebenszyklus verursacht die Konstruktionsphase üblicherweise den größten

Aufwand, während der Abschnitt der Produktnutzung die größte zeitliche Ausdehnung besitzt.

Bereits aus dieser abstrakten und allgemeinen Darstellung eines Produktlebenszyklus lassen sich zwei wichtige Schlussfolgerungen ziehen: Zum einen wird deutlich, dass in seinem Verlauf sehr verschiedene Sichtweisen auf ein Produkt auftreten können. Während beispielsweise in der Konstruktionsphase die vielfältigen technischen Details eines komplexen Produkts zu betrachten sind, stehen in der Vertriebsphase übergreifende Qualitäts- und Kostenaspekte im Vordergrund.

Zum anderen ist erkennbar, dass die einzelnen Phasen eines Produktlebenszyklus nicht losgelöst voneinander sind, sondern im Gegenteil antizipiert werden müssen oder sich iterativ beeinflussen können. So sind beispielsweise beschränkte technische Möglichkeiten der zur Verfügung stehenden Fertigungsmaschinen bereits in der Konstruktionsphase zu berücksichtigen. Umgekehrt können aus der Vertriebs- und Nutzungsphase gewonnene Erkenntnisse über Schwächen eines Produkts in die Planung der nächsten Produktgeneration einfließen.

Im weiteren Verlauf der Arbeit werden vor allem die Phasen der Produktentwicklung aus der Perspektive einer durchgängigen Unterstützung durch Software-Werkzeuge betrachtet. Insbesondere in diesen Phasen deuten die genannten Punkte auf die Notwendigkeit eines integrativen Ansatzes hin, welcher verschiedene Sichtweisen auf ein Produkt zulässt und zugleich gegenseitige Abhängigkeiten und Wechselwirkungen berücksichtigt.

2.1.2 Produktentwicklungsprozesse

Ein Produktlebenszyklus erfasst die grundlegenden Abschnitte, die im Zuge der Entwicklung und Verwertung technischer Produkte durchlaufen werden. Die Produktentwicklung besteht aus den Phasen der Produktplanung, Produktkonstruktion und Produktherstellung. Diese können ihrerseits zahlreiche spezifische Tätigkeiten enthalten, deren Durchführung zur Erfüllung des übergreifenden Ziels einer Phase erforderlich ist. So sind beispielsweise während der Planungsphase grundlegende Funktionsmerkmale des zu entwickelnden Produkts zu spezifizieren und eine Ressourcenplanung auszuarbeiten.

Die Betrachtung einzelner Tätigkeiten der Produktentwicklung sowie ihrer Abhängigkeiten und Wechselwirkungen führt auf den Begriff des *Prozesses*, welcher in der Literatur (s. beispielsweise [SS97] oder [BK02]) uneinheitlich definiert ist. Im Rahmen dieser Arbeit wird unter dem Begriff *Produktentwicklungsprozess*, kurz *Prozess*, die Gesamtheit aller zur Entwicklung eines Produkts notwendigen Aktivitäten und ihrer gegenseitigen Abhängigkeiten

verstanden. Solche Abhängigkeiten können z.B. Einschränkungen bezüglich der zeitlichen Reihenfolge der durchzuführenden Aktivitäten eines Prozesses repräsentieren.

Insbesondere im Bereich komplexer technischer Produkte weisen Entwicklungsprozesse eine Reihe spezifischer Charakteristika auf, welche im Folgenden erläutert werden. Aufbauend auf die Arbeiten von [PR91], [HJ01], [Göt97], [Bec96] und [Vog96] wird in [Küh02] ein Klassifikationsschema vorgeschlagen, welches es allgemein erlaubt, Prozesse anhand ausgesuchter Kriterien einzuordnen. Diese lassen sich demnach unter den Gesichtspunkten der *Strukturiertheit*, der *Ausführungshäufigkeit*, der *Komplexität*, der *Dauer*, der *Veränderlichkeit*, der *Arbeitsteiligkeit* und der *Automatisierbarkeit* klassifizieren.

Die Strukturiertheit eines Prozesses bezeichnet das Ausmaß, in dem ein Prozess a priori in einzelne, exakt abgrenzbare Aktivitäten unterteilt ist. Während sich unstrukturierte Prozesse dadurch auszeichnen, dass zahlreiche spontane Entscheidungen den nächsten Bearbeitungsschritt bestimmen, ist der Ablauf eines strukturierten Prozesses bereits zu Beginn festgelegt. In Anlehnung an [Göt97] zählen Produktentwicklungsprozesse zu den semistrukturierten Prozessen. Sie enthalten einerseits also einen strukturierten Anteil, der bereits a priori eine Ablaufstruktur vorgibt, umfassen andererseits jedoch ebenso Abschnitte, deren genauer Verlauf nicht im Einzelnen planbar ist, wie beispielsweise kreative Tätigkeiten während der Konstruktionsphase (vgl. Abschnitt 2.1.1).

Die Ausführungshäufigkeit eines Prozesses gibt an, wie oft dieser in derselben Art und Weise durchgeführt wird. Im Rahmen von Entwicklungsprozessen ist hierbei zu unterscheiden zwischen Planungs- und Konstruktionsprozessen einerseits und Herstellungsprozessen andererseits (vgl. Abschnitt 2.1.1). Die Ausführungshäufigkeit der zuerst genannten Prozesse ist davon abhängig, wie abstrakt oder granular diese betrachtet werden. So weisen sicherlich die Konstruktionsprozesse verschiedener Varianten eines Produkts aus abstrakter Sicht große Ähnlichkeiten auf, während sie sich in ihrer konkreten Ausprägung in vielfältiger Weise unterscheiden können. Dagegen zeichnen sich Herstellungsprozesse durch die Fertigung eines Produkts in großen Stückzahlen auf immer gleiche Weise aus.

Ein weiteres Kriterium der Klassifikation stellt die Komplexität eines Prozesses dar. Diese lässt sich ermitteln anhand der Anzahl, des Umfangs und des Grades der Verzahnung der durchzuführenden Subprozesse und Aktivitäten, der Zahl der daran beteiligten Personen sowie ihrer erforderlichen Kenntnisse und Fähigkeiten, wie auch mittels des Ausmaßes des Werkzeugeinsatzes. Vor diesem Hintergrund sind Entwicklungsprozesse technischer Produkte durch ein hohes Maß an Komplexität besonders gekennzeichnet. In ihnen wirkt ei-

ne große Anzahl an Bearbeitern verschiedener Disziplinen zusammen, um in zahlreichen, spezialisierten und teils sehr aufwändigen Aktivitäten und unter Nutzung verschiedenster (Software-)Werkzeuge ein hoch komplexes Produkt zu entwickeln.

Anhand ihrer Dauer können Prozesse auf einfache Weise kategorisiert werden. Entwicklungsprozesse technischer Produkte erstrecken sich aufgrund ihrer hohen Komplexität oftmals über mehrere Jahre, besitzen also eine vergleichsweise sehr lange zeitliche Ausdehnung. Eine der wesentlichen Zielsetzungen in der weiteren Verbesserung heutiger Produktentwicklungsprozesse ist es, ihre Dauer unter Wahrung der Produktqualität kontinuierlich zu verkürzen. Dies lässt sich beispielsweise durch ein erhöhtes Maß an Parallelität der durchzuführenden Entwicklungsaktivitäten erreichen. Voraussetzung hierfür ist jedoch eine weit gehende Strukturiertheit eines Prozesses als Grundlage zur Koordination beteiligter Bearbeiter und Synchronisation der Entwicklungsergebnisse.

Wie statisch oder dynamisch ein Prozess ist, lässt sich anhand seiner Veränderlichkeit erkennen. Während statische Prozesse auf immer gleiche Weise ablaufen und damit gut strukturierbar sind, verfügen dynamische Prozesse über einen nicht vorhersehbaren Anteil, dessen konkrete Ausprägung erst im Prozessverlauf festgelegt wird. Analog zur Strukturiertheit und Ausführungshäufigkeit, hängt die Veränderlichkeit eines Prozesses von der gewählten Abstraktionsebene ab. Wie bereits aus der Darstellung des Produktlebenszyklus in Abschnitt 2.1.1 erkennbar ist, besitzen Produktentwicklungsprozesse abstrakt gesehen eine stets gleich bleibende Ablaufstruktur. Innerhalb einzelner Aktivitäten spezifischer Produktentwicklungsprozesse können sie jedoch eine hohe Veränderlichkeit aufweisen.

Die Arbeitsteiligkeit eines Prozesses gibt an, in welchem Maße und auf welche Weise sich durchzuführende Aktivitäten vollständig oder anteilig auf verschiedene Personen bzw. Rollen aufgliedern. Besonders charakteristisch für Produktentwicklungsprozesse ist ein sehr hoher Grad der Arbeitsteiligkeit. So vollzieht sich die Entwicklung komplexer technischer Produkte in einem stark interdisziplinär geprägten Umfeld. Dabei sind einerseits zahlreiche sehr spezialisierte Aktivitäten in paralleler und verteilter Gruppenarbeit von Experten verschiedener Fachdisziplinen, welche über spezifische Kenntnisse und Fähigkeiten verfügen, durchzuführen. Andererseits führt nur ein koordiniertes und synchronisiertes Zusammenwirken der zahlreichen Bearbeiter zu einer erfolgreichen Produktentwicklung.

Schließlich lässt sich als letztes Kriterium der Prozessklassifikation auch ihre Automatisierbarkeit heranziehen. Diese gibt an, inwieweit Aktivitäten rechnergestützt ohne Benutzerinteraktion durchführbar sind. Gemäß den bereits betrachteten Kriterien können strukturierte, statische und kurze Pro-

zesse mit hoher Ausführungshäufigkeit und geringer Komplexität sowie Arbeitsteiligkeit auf besonders einfache und vorteilhafte Weise automatisiert werden. Betrachtet man die Entwicklungsprozesse komplexer Produkte, so lässt sich eine eingeschränkte Automatisierbarkeit feststellen. Durch den zunehmenden Einsatz von Software-Werkzeugen sind einzelne Aktivitäten sehr gut automatisierbar, während sich kreative Tätigkeiten beispielsweise in der Konstruktionsphase nicht automatisieren lassen.

Die Analyse und Einordnung von Produktentwicklungsprozessen in den vorhergehenden Abschnitten zeigen ihre Charakteristika deutlich auf. Besonders kennzeichnend ist ein sehr hoher Grad inhärenter Komplexität, der eine umfassende und durchgängige Unterstützung durch Software-Werkzeuge erfordert, beispielsweise zur Steuerung des Entwicklungsprozesses oder zur Durchführung fachspezifischer Aktivitäten. In der Betrachtung heutiger Produktentwicklungsprozesse lassen sich diesbezüglich folgende Defizite feststellen:

- Als Grundlage für eine durchgängige Software-technische Unterstützung von Entwicklungsprozessen – beispielsweise durch Workflow-Management-Systeme – ist eine Erfassung der zur Verfügung stehenden Ressourcen im Rahmen eines *Organisationsmodells* erforderlich. Hierzu zählen die an einem Entwicklungsprozess beteiligten Personen und ihre Fähigkeiten, innerhalb durchzuführender Aktivitäten bestimmte Rollen zu übernehmen, ebenso, wie vorhandene Maschinen, z.B. Rechenanlagen oder Fertigungsmaschinen, als auch verfügbare Software-Werkzeuge. Nur ein solches Modell ermöglicht es, die für einen Entwicklungsprozess benötigten Ressourcen rechnergestützt zu verwalten und zu koordinieren.

  Explizite und übergreifende Organisationsmodelle sind in der Praxis jedoch nicht oder nur vereinzelt anzutreffen. Obwohl die betriebswirtschaftliche Organisationstheorie (s. z.B. [Wei02] oder [Sch98]) wie auch die Informatik im Bereich des Workflow-Management (s. beispielsweise [WPD98a]) bereits entsprechende Konzepte und Modelle anbieten, sind in realen Produktentwicklungsprozessen vorhandene bzw. erforderliche Ressourcen unvollständig erfasst, unzureichend dokumentiert, nicht auf einem aktuellen Stand oder nur isoliert vorhanden und somit nicht übergreifend nutzbar. In [Poh03] werden existierende Organisationsmodelle evaluiert und ihre Bedeutung anhand eines realen Anwendungsszenarios aufgezeigt.

- Analog zur Erfassung der zugrunde liegenden Organisationsstrukturen ist für eine durchgängige Unterstützung der Produktentwicklung durch

Software-Systeme auch eine Modellierung des Prozesses notwendig. Ein solches *Prozessmodell* beinhaltet sowohl die durchzuführenden Aktivitäten, als auch ihre gegenseitigen Abhängigkeiten, wie beispielsweise Einschränkungen hinsichtlich ihrer chronologischen Reihenfolge, und ordnet diesen verfügbare Ressourcen auf der Grundlage des Organisationsmodells zu. Dieser, in der Literatur auch als Workflow-Management bezeichnete Ansatz, erlaubt die rechnergestützte Verwaltung und Steuerung von Produktentwicklungsprozessen und der daran beteiligten Ressourcen.

Wie Organisationsstrukturen sind auch Entwicklungsprozesse im Bereich komplexer Produkte sehr umfangreich, vielschichtig und unübersichtlich und daher in der Praxis nur auf einer hohen Abstraktionsebene vollständig erfasst. Einzelne Teilprozesse besitzen dagegen einen großen unstrukturierten Anteil, welcher von impliziten Abhängigkeiten sowie spontanen Entscheidungen und unzureichend koordinierten Bearbeitungsschritten geprägt ist, und verhindern so die Ermittlung eines klaren Prozesszustands. Folgerichtig ist auch ihre Unterstützbarkeit durch Software-Lösungen entsprechend eingeschränkt. Ein Ansatz zur durchgängigen Workflow-Unterstützung in Entwicklungsprozessen wird dagegen in [Küh02] exemplarisch entworfen.

- Ein weiteres Defizit in der Software-technischen Unterstützung heutiger Produktentwicklungsprozesse ist die unzureichende technische wie fachliche Integration verwendeter Software-Systeme. Aufgrund des Einsatzes verschiedenster Software-Werkzeuge in den unterschiedlichsten Bereichen der Produktentwicklung (vgl. Abschnitt 2.1.4) gewinnt diese Herausforderung zunehmend an Bedeutung (s. Kapitel 3). Insbesondere fehlt bisher eine tragfähige und leistungsstarke Konzeption für ein flexibles und wohldefiniertes Zusammenspiel von Prozess und zu entwickelndem Produkt bzw. zugehörigen Modellen. Ein solcher Ansatz wird in Teil II und Teil III der vorliegenden Arbeit entworfen und detailliert.

Die genannten Punkte verdeutlichen Charakteristika und Defizite moderner Produktentwicklungsprozesse aus Sicht einer durchgängigen Unterstützung durch Software-Systeme. Neben einer Modellierung relevanter Organisationsstrukturen und Prozesse spielen Modelle des zu entwickelnden Produkts eine entscheidende Rolle. Ihre Bedeutung wird im folgenden Abschnitt erläutert.

2.1.3 Digitale Produktmodelle

In der Entwicklung komplexer technischer Produkte, wie sie sich z.B. im Avionikbereich oder im Automobilbau finden, kommen in zunehmendem Maße Software-Werkzeuge zum Einsatz, um Bearbeiter in ihren spezifischen Tätigkeiten zu unterstützen. Beispielsweise hat sich seit vielen Jahren die Methode des *Computer Aided Design (CAD)* in der Praxis bewährt, welche es erlaubt, die geometrische Gestalt eines Produkts in Form zwei- oder dreidimensionaler Konstruktionszeichnungen vollständig am Computer zu entwerfen. Voraussetzung für die Verwendung von Software-Werkzeugen ist allgemein die Erstellung eines Modells, welches durch Rechenanlagen verarbeitbar ist. Ein solches *Produktmodell* stellt ein digitales Abbild des tatsächlichen Produkts dar und erfasst sämtliche Gegenstände und ihre Eigenschaften, die für eine gegebene Zielsetzung relevant sind (vgl. hierzu auch Abschnitt 4.1). Auf dieser Grundlage lassen sich Software-Werkzeuge nutzen, um ein Modell sukzessive zu bearbeiten, bis es gewünschte Eigenschaften aufweist. Diese Vorgehensweise wird auch als *virtuelle Produktentwicklung* bezeichnet, da das Produkt weit gehend mithilfe von Rechenanlagen entworfen und ein stofflich vorhandener Prototyp erst am Ende des Entwicklungsprozesses gefertigt wird.

Abbildung 2.2 zeigt das Prinzip der virtuellen Produktentwicklung. Ausgehend von Anforderungen, die übergreifende Funktionsmerkmale eines Produkts festlegen, wird ein initiales Modell erstellt, welches sich daraufhin mithilfe von Software-Werkzeugen modifizieren lässt bis alle gewünschten Eigenschaften erreicht sind. Das so resultierende, finale Modell dient zur Vorlage für die Produktfertigung. Auf diese Weise lassen sich Produkte wesentlich schneller und kostengünstiger entwickeln, da eine aufwändige Fertigung stofflicher Prototypen im Entwicklungsprozess entfällt. Zudem erlaubt die virtuelle Produktentwicklung eine verhältnismäßig einfache und schnelle Simulation und Überprüfung angestrebter Produkteigenschaften, beispielsweise unter Nutzung numerischer Simulationsverfahren.

Gerade für komplexe technische Produkte spielt die virtuelle Produktentwicklung eine entscheidende Rolle. Hierbei treten jedoch einige Besonderheiten auf, welche im Folgenden näher betrachtet werden. Entsprechend der inhärent hohen Komplexität der Produkte dieses Bereichs repräsentieren auch ihre Modelle sehr reichhaltige Produktstrukturen mit vielfältigen Bestandteilen und zahlreichen Querbezügen. Je mehr Aspekte eines Produkts in einem Modell vertreten sind, d.h. je aussagekräftiger dieses Modell ist, desto schwieriger ist seine Übersichtlichkeit, Verständlichkeit und Handhabbarkeit zu gewährleisten.

Besonders kennzeichnend für die Entwicklungsprozesse komplexer Pro-

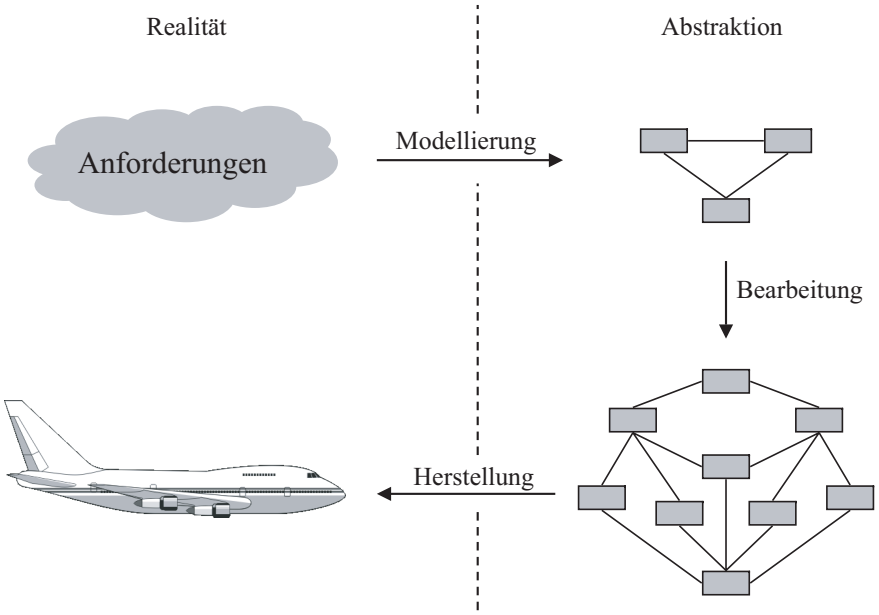

Abbildung 2.2: Prinzip der virtuellen Produktentwicklung

dukte ist ein stark interdisziplinär geprägtes Umfeld (vgl. Abschnitt 2.1.2). Jeder Bearbeiter hat ein für seinen Aufgabenbereich je eigenes Verständnis des Produkts, das sich auf ausgesuchte, für den aktuellen Kontext relevante Aspekte konzentriert. Diese individuellen *Produktsichten* resultieren in zahlreichen verschiedenen *Partialmodellen*, welche jeweils nur die für den aktuellen Entwicklungsschritt relevanten Produkteigenschaften repräsentieren und auf die Modellierungsmöglichkeiten dabei verwendeter Software-Werkzeuge ausgerichtet sind.

Trotz der übergreifenden Zielsetzung – der Erstellung eines gemeinsamen Produkts – treten also im Laufe eines Entwicklungsprozesses je nach dem Verständnis des aktuellen Bearbeiters, dem Zweck der Bearbeitung oder den Möglichkeiten verwendeter Software-Werkzeuge sehr verschiedene Produktmodelle auf. Diese Modelle können jedoch nicht unabhängig voneinander betrachtet und bearbeitet werden, sondern sind über zahlreiche Überschneidungen, Querbezüge und Abhängigkeiten miteinander verbunden. So muss beispielsweise die geometrische Form eines Produkts grundlegende, z.B. aerodynamische Zielsetzungen berücksichtigen.

Eine durchgängig modellbasierte Produktentwicklung erfordert deshalb

einen kontrollierten und koordinierten Umgang mit verschiedenen, umfangreichen und komplexen Modellen eines gemeinsamen Produkts sowie eine transparente Verwaltung ihrer zahlreichen Beziehungen. Vor diesem Hintergrund weisen heutige Entwicklungsprozesse folgende Defizite auf:

- Aufgrund komplexer Entwicklungsprozesse mit zahlreichen Einzelaktivitäten sowie vieler verschiedener Bearbeiter unterschiedlicher Fachdisziplinen existiert kein übergreifendes, allen gemeinsames Produktmodell, welches jeden relevanten Produktaspekt gleichermaßen berücksichtigt. Gegebenenfalls wäre ein solches Modell gemäß der Komplexität des Produkts sehr umfangreich und unübersichtlich zu Lasten seiner Verständlichkeit und Handhabbarkeit. Dagegen werden je nach Zielsetzung der Entwicklungsaktivitäten, dem Verständnis der Bearbeiter und den Möglichkeiten verwendeter Software-Werkzeuge unterschiedliche Partialmodelle eines Produkts als individuelle Produktsichten gebildet.

  Als wesentliches und zentrales Hindernis in der weiteren Verbesserung moderner Entwicklungsprozesse erweist sich jedoch eine unzureichende Integration dieser zahlreichen Partialmodelle. So werden relevante Produktaspekte nur isoliert und unvollständig gemäß der im aktuellen Kontext gegebenen Aufgabenstellung erfasst. Zudem orientiert sich die Modellbildung überwiegend und in zu starkem Maße an den Modellierungsmöglichkeiten verwendeter Software-Werkzeuge. Diese Vorgehensweise führt zu zahlreichen verbindungslosen, inkompatiblen und proprietären Partialmodellen, welche eine durchgängige, modellbasierte Produktentwicklung auf der Grundlage einer einheitlichen Modellierungstechnik erheblich erschweren.

- Die verschiedenen, im Laufe der Entwicklung auftretenden Modelle repräsentieren unterschiedliche Produktaspekte gemäß den Erfordernissen des aktuellen Entwicklungsschrittes. Da diese sich auf ein gemeinsames Produkt beziehen, existieren folgerichtig vielfältige Querbezüge, welche jedoch nicht explizit erfasst und verwaltet werden. Vielfach redundant vorliegende Produktinformationen in verschiedenen Partialmodellen stellen somit eine Quelle der *Inkonsistenz* dar, welche nicht oder erst in späten Phasen des Entwicklungsprozesses entdeckt wird. So führt beispielsweise die Änderung eines Partialmodells nicht unmittelbar zu einem kontrollierten Abgleich redundant vorliegender Produktinformationen in anderen Partialmodellen.

- Die verwendeten Partialmodelle eines Produkts werden in verschiedenen Entwicklungsaktivitäten durch unterschiedliche Bearbeiter weit ge-

hend getrennt voneinander modifiziert. Vorhandene Beziehungen werden dagegen nicht explizit erfasst und verwaltet, so dass ein Abgleich zwischen Partialmodellen nur spontan und auf unstrukturierte Weise vorgenommen werden kann. Analog zur unzureichenden Erfassung des Entwicklungsprozesses durch ein Prozessmodell (vgl. Abschnitt 2.1.2) verhindert dies die Einführung eines transparenten Zustandsbegriffs für das Produktmodell. So lässt sich beispielsweise nicht unmittelbar feststellen, welche Partialmodelle dem gleichen Entwicklungsstand zuzuordnen sind.

Das bisher beschriebene Maß an Komplexität in der Entwicklung technischer Produkte, resultierend aus anspruchsvollen Prozessen und umfangreichen Produkten, wird durch den zunehmenden Einsatz zahlreicher, fachspezifischer Software-Werkzeuge weiter gesteigert. Im folgenden Abschnitt werden daher Charakteristika und Defizite der Werkzeugunterstützung betrachtet.

2.1.4 Werkzeugunterstützung

Modelle des Entwicklungsprozesses sowie des zu entwickelnden Produkts dienen letztlich als Grundlage zur Verwendung von Software-Werkzeugen, um Bearbeiter in ihren individuellen Tätigkeiten zu unterstützen. Gemäß Abbildung 2.2 erlauben diese eine virtuelle Produktentwicklung mittels digitaler Modelle, welche sich im Vergleich zur Erstellung realer Modelle wesentlich kostengünstiger und schneller vollzieht. Gerade im Bereich komplexer technischer Produkte ist eine große Anzahl kommerzieller Software-Werkzeuge entstanden, welche auf die am Entwicklungsprozess beteiligten Fachdisziplinen ausgerichtet sind und oftmals als *CAx-Werkzeuge* – z.B. CAD- oder CAE-Werkzeuge – bezeichnet werden. Darüber hinaus spielen auch zahlreiche übergreifende Software-Systeme eine wichtige Rolle, welche unter die Begriffe *Workflow-Management (WFM)*, *Product Lifecycle Management (PLM)*, *Enterprise Ressource Planning (ERP)* oder *Product Data Management (PDM)* bzw. *Engineering Data Management (EDM)* zusammengefasst werden können.

Abbildung 2.3 illustriert in Anlehnung an [DWD87] die Vielfalt rechnergestützter Lösungen, welche in der Entwicklung komplexer technischer Produkte Verwendung finden, gruppiert nach verschiedenen Bereichen des Produktlebenszyklus (vgl. Abschnitt 2.1.1). Wie daraus unmittelbar erkenntlich ist, können die unterschiedlichsten Aufgabenbereiche der Produktentwicklung durch Software-Werkzeuge unterstützt werden, welche ihrerseits oftmals sehr spezialisiert und auf eine ausgesuchte Fachdisziplin zugeschnitten sind.

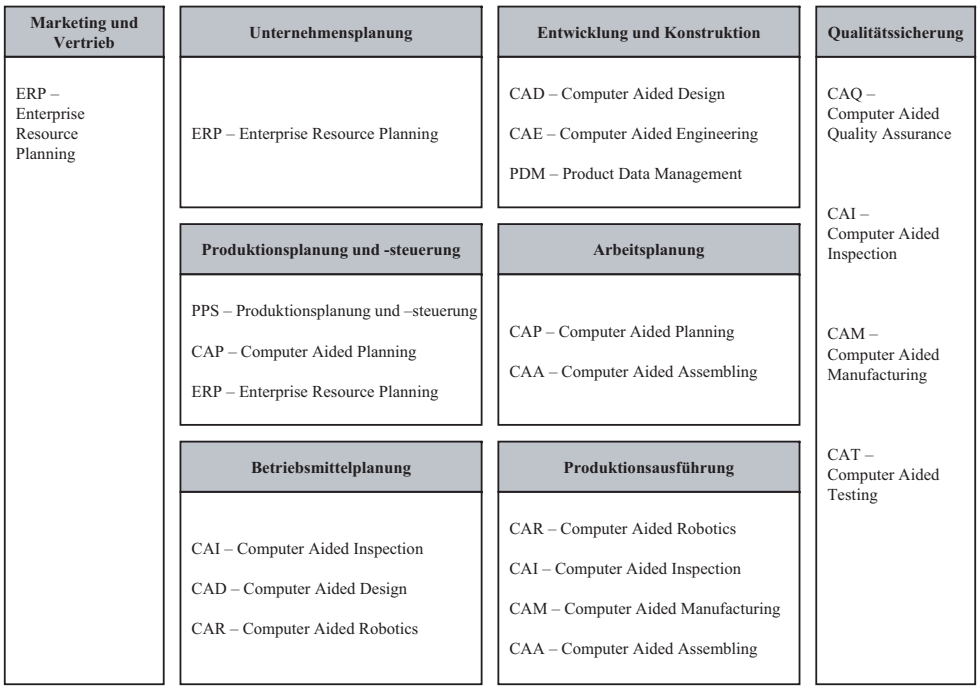

Abbildung 2.3: Werkzeuge der Produktentwicklung

Folgerichtig sind auch die in diesen Werkzeugen verwendeten Modelle des zu entwickelnden Produkts entsprechend verschieden und schwer aufeinander abzubilden (vgl. Abschnitt 2.1.3).

Neben kommerziellen Werkzeugen, welche auch als sog. *COTS-Produkte (commercial-off-the-shelf)* bezeichnet werden, kommen in der Entwicklung komplexer technischer Produkte zahlreiche Eigenentwicklungen zum Einsatz, beispielsweise um spezifische Aufgaben eines Unternehmens zu unterstützen. Diese historisch gewachsene Vielfalt an Software-Lösungen führt zu einem Nebeneinander zahlreicher Werkzeuge mit unterschiedlichen, fachspezifischen Funktionsumfängen, welche sich auf verschiedene Aspekte des zu entwickelnden Produkts bzw. des zugehörigen Prozesses konzentrieren, auf der Grundlage unterschiedlicher Technologien implementiert sind und verschiedene Systemplattformen voraussetzen.

In der Betrachtung heutiger Produktentwicklungsprozesse lässt sich aufgrund der beschriebenen Komplexität der auftretenden Werkzeuglandschaft, resultierend aus zahlreichen Software-Lösungen mit unterschiedlichen fachlichen und technischen Charakteristika eine Reihe an Defiziten feststellen, welche im Folgenden beschrieben sind:

- Die Nutzung sehr zahlreicher Software-Lösungen für die verschiedensten Aufgaben der Produktentwicklung führt zu einer unzureichenden Transparenz des Werkzeugeinsatzes im Entwicklungsprozess. So existiert keine übergreifende und strukturierte Werkzeugdokumentation, um vorhandene Software-Werkzeuge und ihre Funktionsumfänge zu erfassen. Vielmehr bleibt die Auswahl eines geeigneten Software-Werkzeugs zur Unterstützung einer Entwicklungsaktivität dem aktuellen Bearbeiter überlassen und ist somit von vielerlei zufälligen Faktoren abhängig, wie beispielsweise dem Kenntnisstand bzw. den Präferenzen des Bearbeiters oder der aktuell verfügbaren Version eines Software-Werkzeugs.

- Die historisch gewachsene Vielfalt verwendeter Software-Lösungen basiert auf einem Nebeneinander zahlreicher Software-Technologien und Systemplattformen. Ein erfolgreicher Produktentwicklungsprozess erfordert es jedoch, Werkzeuge nicht isoliert voneinander zu betrachten, sondern im Sinne einer durchgängigen Werkzeugkette miteinander zu verzahnen. Dementsprechend fehlt eine übergreifende technische Integration, welche einen Rahmen zur einheitlichen Anbindung und Verwendung benutzter Software-Werkzeuge anbietet. Diese, auch als Enterprise Application Integration (EAI) bezeichnete Problemstellung stellt ein wesentliches Hemmnis in der weiteren Verbesserung moderner Produktentwicklungsprozesse dar.

- Neben einer unzureichenden Transparenz und technischen Integration stellt die unzureichende fachliche Integration eingesetzter Software-Lösungen das wichtigste Defizit dar. Die weit gehend isolierte Verwendung von Software-Werkzeugen verhindert ein fundiertes Verständnis des Zusammenspiels zwischen Entwicklungsaktivitäten des Prozesses einerseits und dabei relevanten Modellen des Produktes andererseits. Der aktuelle Entwicklungsstand eines Produktes lässt sich somit weder klar ermitteln noch verfolgen und es treten in der Folge zahlreiche Inkompatibilitäten und Inkonsistenzen zwischen bearbeiteten Modellen auf, welche nicht oder erst in späten Phasen der Produktentwicklung bemerkt werden.

Insgesamt betrachtet stellen Software-Werkzeuge das entscheidende Mittel der virtuellen Produktentwicklung dar, welche es erlaubt, Produkte vollständig rechnergestützt zu entwerfen. Die Vorteile, die sich aus einem stetig zunehmenden Einsatz von Software-Lösungen ergeben, werden jedoch durch die unzureichende technische wie fachliche Integration erheblich beeinträchtigt.

2.2 Die Fallstudie

Die Charakteristika des Anwendungsbereichs, seine Herausforderungen und Schwierigkeiten werden in den folgenden Abschnitten anhand einer realitätsnahen Fallstudie verdeutlicht. Diese bietet darüber hinaus eine solide Grundlage, Lösungsansätze für die beschriebenen Defizite moderner, Software-gestützter Produktentwicklungsprozesse zu erarbeiten, an Beispielen zu illustrieren und auf ihre durchgängige Verwendbarkeit validieren.

Die Fallstudie ist vor dem Hintergrund der Forschungskooperation ARTWORK [ART05] gewählt, einem mehrjährigen Forschungs- und Entwicklungsprojekt in Zusammenarbeit mit der MTU Aero Engines GmbH, dem führenden deutschen Hersteller von Flugzeug- und Industriegasturbinen. Zentrale Zielsetzung des Projekts ist die Entwicklung von Konzepten und Techniken zur weiteren Verbesserung rechnergestützter Turbinenentwicklungsprozesse (s. [ART03b] und [ART03a]). Als konkretes Fallbeispiel wird demgemäß eine *Niederdruckturbine* betrachtet, ein wichtiges und komplexes Bauteil moderner Flugzeugtriebwerke.

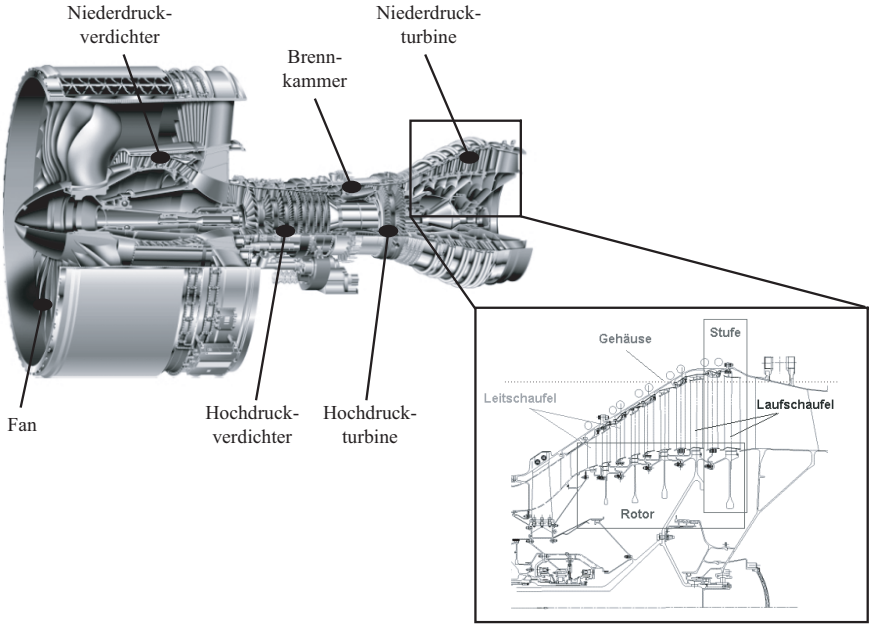

Abbildung 2.4: Aufbau eines Triebwerks

Abbildung 2.4 zeigt den Aufbau eines Zwei-Wellentriebwerks für die zivile Luftfahrt, bestehend aus einem sog. *Fan*, einem *Niederdruckverdichter*, einem

Hochdruckverdichter, einer *Brennkammer*, einer *Hochdruckturbine* und einer *Niederdruckturbine*. Gegenwärtig wird dieses Triebwerk u.a. für den Airbus A380 entwickelt, könnte aber ebenso im Airbus A330, der Boing 747 oder der Boing 767 eingesetzt werden. Ebenfalls in Abbildung 2.4 dargestellt ist der schematische Aufbau einer Niederdruckturbine, welche als abschließendes Bauteil eines Flugzeugtriebwerks schließlich für die Erzeugung der Antriebskraft zuständig ist.

Die Niederdruckturbine besteht im Kern aus einem *Rotor*, der alle rotierenden Bauteile umfasst, welche mit einer Triebwerkswelle verbunden sind. Die Aufgabe dieser Baugruppe besteht darin, Kräfte, die durch eine Expansion des Gasgemisches innerhalb der Niederdruckturbine entstehen, an den angeschlossenen Verdichter zu übertragen. Daneben beinhaltet die Niederdruckturbine als weitere Baugruppen *Laufschaufeln*, welche mit dem Rotor entsprechend verbunden sind, *Leitschaufeln*, die am Gehäuse fixiert sind und die Gasströmung leiten, das Gehäuse, welches den Rahmen bildet, sowie zahlreiche und vielfältige Einzelteile.

Der Rotor ist eine unter hoher Belastung arbeitende und daher sehr funktionskritische Baugruppe eines Flugzeugtriebwerks. Da er einen entscheidenden Einfluss auf die Funktionstüchtigkeit und somit die Lebensdauer eines Triebwerks ausübt, bestehen hinsichtlich einer Zulassung durch Luftfahrtbehörden hohe Anforderungen an diese Baugruppe. Im Wesentlichen besteht ein Rotor aus mehreren *Turbinenscheiben*, welche über Flanschverbindungen miteinander verschraubt sind. Die Anzahl dieser Turbinenscheiben ist von den Leistungsanforderungen und somit von dem Typ des betrachteten Triebwerks abhängig.

Als durchgängiges Fallbeispiel wird im weiteren Verlauf der Arbeit ein vereinfachter Rotor bestehend aus zwei Turbinenscheiben, die mit einer Flanschverbindung verschraubt sind, betrachtet. Andere Einzelbauteile des Rotors werden dagegen vernachlässigt, da sie keine prinzipiell neuen Anforderungen einbringen. Eine Hinzunahme weiterer Turbinenscheiben ist ohne Schwierigkeit möglich, würde die Aussagekraft des Fallbeispiels jedoch nicht erhöhen. Darüber hinaus wird von einer deutlich vereinfachten, jedoch repräsentativen geometrischen Gestalt der Turbinenscheiben ausgegangen, um das gewählte Fallbeispiel im gegeben Rahmen darstellen zu können.

Abbildung 2.5 zeigt die im Wesentlichen rotationssymmetrische Gestalt eines solchen Rotors im Aufriss. Bezug nehmend auf Abbildung 2.4 bilden die dargestellten Turbinenscheiben die rotierenden Teile einer Niederdruckturbine und dienen der Kraftübertragung zur Triebwerkswelle. Jede Turbinenscheibe (engl.: *disk*) besteht aus einem Scheibenhauptkörper (engl.: *body*), sowie aus einem vorderen und einem hinteren Arm (engl.: *wing* oder *drivearm*). Die beiden Turbinenscheiben sind über je einen Arm mittels ei-

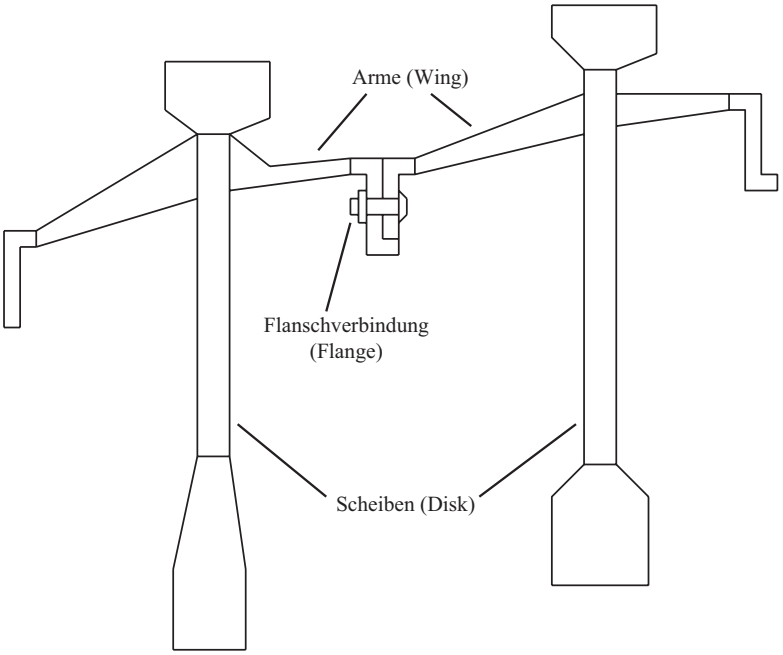

Abbildung 2.5: Turbinenscheiben

ner Flanschverbindung (engl.: *flange*) miteinander form- und kraftschlüssig verschraubt vermöge eines Schraubbolzens.

Flugzeugturbinenen des zivilen oder militärischen Bereichs, wie sie in der Realität entwickelt und eingesetzt werden, bestehen aus mehreren tausend Einzelbauteilen. Im Zuge einer rechnergestützten Produktentwicklung ist diese sehr große Anzahl an Bauteilen durch Modelle zu erfassen. Jedes Bauteil muss dazu üblicherweise aus einfachen, zur Verfügung stehenden Grundelementen aufgebaut werden. Daneben sind verschiedene Sichten auf Bauteile zu berücksichtigen (s. Abschnitt 2.2.2). Bereits anhand des stark vereinfachten Fallbeispiels wird die hieraus resultierende Komplexität einer virtuellen Produktentwicklung ersichtlich.

2.2.1 Der Entwicklungsprozess

Nach der Vorstellung der im Rahmen der Fallstudie gewählten Anwendungsdomäne der Turbinenentwicklung und der Auswahl der Niederdruckturbine als komplexes technisches Produkt bzw. der betrachteten Baugruppe eines vereinfachten Rotors mit zwei Turbinenscheiben, wird der zugehörige Ent-

wicklungsprozess beleuchtet. Einerseits zählt hierzu die Erläuterung relevanter Entwicklungsaktivitäten und ihrer (zeitlichen) Abhängigkeiten voneinander. Andererseits sind ebenso beteiligte Fachdisziplinen sowie Bearbeiter bzw. Rollen mit ihren spezifischen Aufgaben innerhalb des Produktentwicklungsprozesses zu erwähnen.

Wie bereits in Abschnitt 2.1.2 dargestellt, führt die außerordentlich hohe Komplexität des betrachteten Produkts einer Flugzeugturbine zu vergleichsweise sehr langen Entwicklungszeiträumen, die mehrere Jahre andauern können. So besitzt etwa der vollständige Entwicklungsprozess einer Niederdruckturbine oben angegebener Bauart eine zeitliche Ausdehnung zwischen drei und sechs Jahren, umfasst ca. dreihundert verschiedene Teilprozesse, die ihrerseits wiederum Dutzende Aktivitäten beinhalten, involviert in etwa fünfzehn unterschiedliche Fachdisziplinen und erfordert im Kern eine Zusammenarbeit von ca. zweihundert Bearbeitern.

Aufgrund des Umfangs und der daraus resultierenden Komplexität des geschilderten Entwicklungsprozesses können die Abläufe zur Erstellung einer Niederdruckturbine bzw. eines Rotors im gegebenen Rahmen nur vereinfacht auf sehr abstrakter Ebene und ohne Anspruch auf Vollständigkeit dargestellt werden. Wiederum besteht die Zielsetzung, einen charakteristischen und repräsentativen Teilausschnitt des vollständigen Entwicklungsprozesses zu wählen, der – wenn auch vereinfacht – die prinzipiellen Abläufe und Herausforderungen in der Entwicklung komplexer technischer Produkte verdeutlicht und als Fallbeispiel für weitere Betrachtungen dient.

Abbildung 2.6 zeigt schematisch die grundlegenden Abschnitte eines vereinfachten Entwicklungsprozesses einer Niederdruckturbine bzw. eines Rotors. Jeder dargestellte Entwicklungsschritt umfasst in der Realität eine Vielzahl einzelner, z.T. sehr spezialisierter und aufwändiger Aktivitäten und involviert zahlreiche Bearbeiter, Modelle und Software-Werkzeuge. Nicht dargestellt sind Aktionen des Projektmanagements, das übergreifend und parallel zu fachlichen Tätigkeiten der Produktentwicklung jeden Schritt des Entwicklungsprozesses kontinuierlich begleitet. Im Einzelnen finden sich die folgenden Abschnitte:

Konfiguration: Zu Prozessbeginn werden auf der Grundlage von Marktstudien angestrebte Produktmerkmale, etwa das maximal mögliche Gewicht oder die Schubkraft der Turbine, festgelegt. Hierbei fließen neue Technologien ebenso wie Überlegungen zur strategischen Ausrichtung des Produktportfolios ein. Soweit entspricht die Konfiguration der Planungsphase eines Produktlebenszyklus (vgl. Abschnitt 2.1.1). Darüber hinaus wird im Laufe der Konfiguration bereits ein Grobkonzept der zu entwickelnden Turbine erstellt, welches als Grundlage und Rahmen

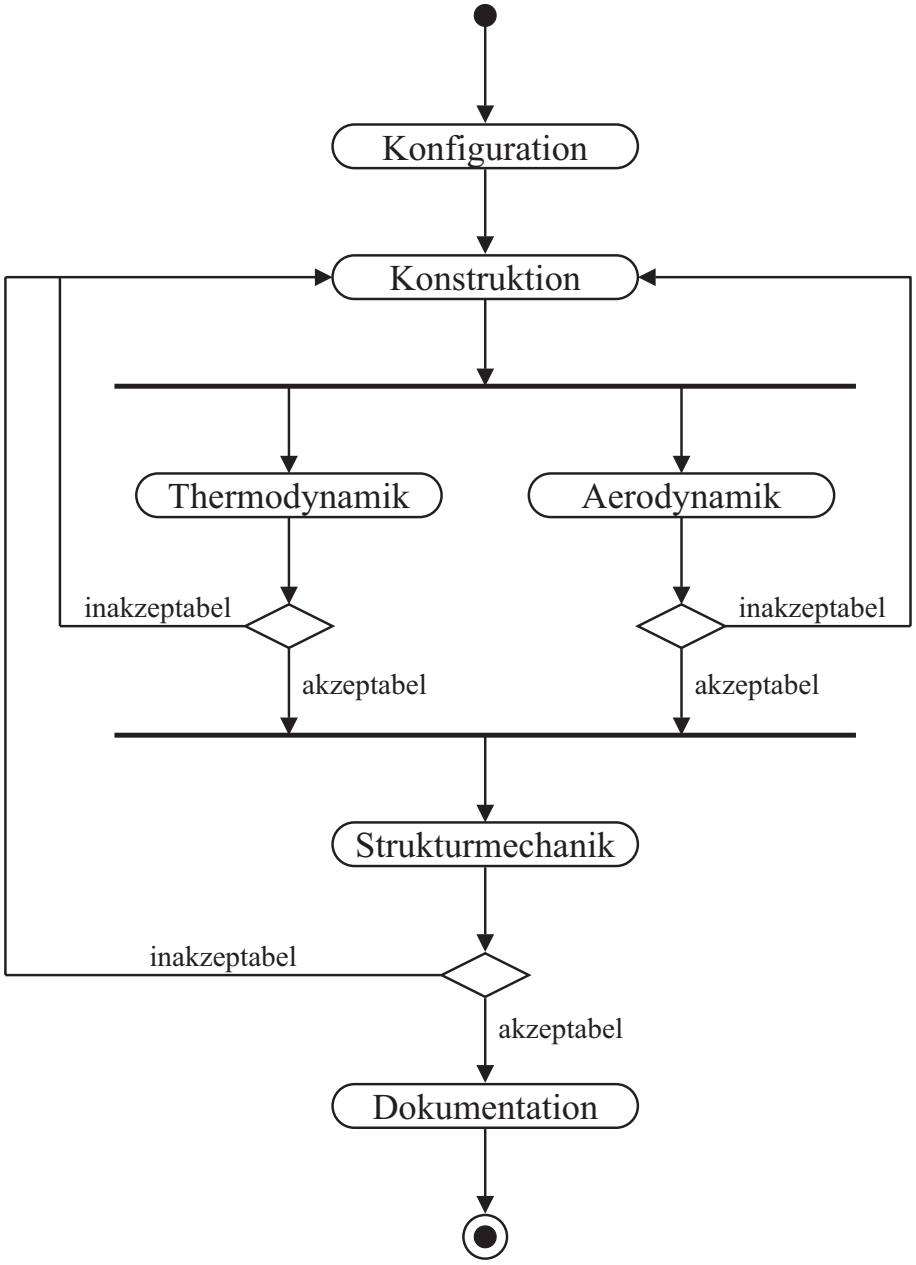

Abbildung 2.6: Abschnitte der Turbinenentwicklung

für weitere Entwicklungsschritte dient.

Konstruktion: Im Gegensatz zu Abschnitt 2.1.1 wird unter Konstruktion in diesem Zusammenhang ausschließlich die Erstellung bzw. Präzisierung der geometrischen Gestalt der Turbine im Sinne des Computer Aided Design (CAD) verstanden. Zu unterscheiden ist hierbei zwischen der zweidimensionalen Geometrieerstellung mittels einer Aufrisszeichnung (engl.: *sketch*) und einem dreidimensionalen geometrisch-topologischen Geometrieentwurf. In beiden Fällen wird die äußere Gestalt modellierter Bauteile mittels elementarer geometrischer Grundelemente hierarchisch nachgebildet.

Thermodynamik: Die Fachdisziplin der Thermodynamik ist dafür verantwortlich, die Temperaturverteilung der Bestandteile einer Turbine während verschiedener Betriebsphasen zu ermitteln. Dies geschieht auf der Grundlage der geometrischen Gestalt unter Benutzung numerischer Simulationsverfahren. Vorbereitend wird eine diskrete Approximation der geometrischen Form mittels eines sog. Finite-Elemente-Netzes [Bra03a] vorgenommen. Von besonderem Interesse für die Thermodynamik sind die Maximalwerte der Temperaturverteilung, da diese die Lebensdauer eines Bauteils bestimmen.

Aerodynamik: Die Aufgabe der Aerodynamik ist es, wiederum mittels numerischer Methoden das Strömungsverhalten einer Turbine bzw. des zugehörigen Triebwerks zu untersuchen und zu bewerten. Erst daraus lassen sich die Leistungsmerkmale eines Triebwerks, beispielsweise erreichbare Geschwindigkeiten, und ebenso seine Wirtschaftlichkeit auf der Grundlage des daraus zu folgernden Treibstoffverbrauchs ableiten. Als wesentliche Voraussetzung für Berechnungen der Aerodynamik dient eine detaillierte und vollständig erstellte dreidimensionale Geometriebeschreibung.

Strukturmechanik: Der Strukturmechanik kommt die Aufgabe zu, die Gesamtkonstruktion einer Turbine hinsichtlich ihrer Betriebssicherheit zu bewerten und die Wahrscheinlichkeit und Schwere potenzieller Schadensfälle abzuschätzen. Besondere Berücksichtigung finden die für die Herstellung einer Turbine verwendeten Materialien. Insgesamt hat die Strukturmechanik eine Abwägung zu treffen zwischen den gegenläufigen Zielsetzungen einer hohen Leistungsfähigkeit sowie leichten und kostengünstigen Materialien einerseits und einer stabilen und langen Funktionsfähigkeit andererseits.

Dokumentation: Den Abschluss des in Abbildung 2.6 dargestellten Entwicklungsprozesses bildet die Dokumentation des ermittelten Turbinenkonzepts. Zum einen fasst diese die in Modellen gehaltenen Produktinformationen und Simulationsergebnisse zum Zwecke der Archivierung zusammen, welche als Ausgangspunkt neuer Entwicklungsprojekte zur Verfügung steht. Darüber hinaus werden durch diesen letzten Entwicklungsschritt gesetzliche Vorgaben erfüllt, da eine detaillierte Dokumentation eine wesentliche Voraussetzung für die Zulassung eines neuen Turbinentyps durch Luftfahrtbehörden darstellt.

Der betrachtete Entwicklungsprozess zeigt in stark vereinfachter Weise die Abschnitte einer rechnergestützten Turbinenentwicklung. Jeder erläuterte Prozessschritt setzt sich in der Realität aus Dutzenden anspruchsvoller Einzelaktivitäten zusammen. Darüber hinaus existieren zahlreiche Querbezüge und Abhängigkeiten zwischen den verschiedenen Abschnitten, wie durch die angedeuteten Verzweigungsmöglichkeiten zum Ausdruck kommt.

2.2.2 Produktsichten

Wie bereits in Abschnitt 2.1.3 allgemein diskutiert, ist es angesichts der gewählten Fallstudie einer Niederdruckturbine bzw. eines vereinfachten Rotors vor dem Hintergrund des geschilderten Entwicklungsprozesses mit zahlreichen, verschiedenartigen Einzelaktivitäten offensichtlich, dass nicht ein einzelnes, zentrales und übergreifendes Produktmodell in der Lage ist, alle relevanten Aspekte der unterschiedlichen Fachdisziplinen angemessen und gleich berechtigt zu repräsentieren. Vielmehr werden in jedem charakteristischen Prozessabschnitt eigene, spezialisierte Partialmodelle eingesetzt, um einen spezifischen Produktaspekt optimal darzustellen.

Diese *Produktsichten* entspringen in erster Linie dem spezifischen Verständnis eines Bearbeiters, in das die Art des gewählten Bauteils, die Zielsetzung seiner Bearbeitung sowie Möglichkeiten verwendeter Software-Werkzeuge entscheidend einfließen. So kommen während der Entwicklung einer Niederdruckturbine mehrere hundert Software-Werkzeuge zum Einsatz, welche in der Regel jeweils ein eigenes, angepasstes Modell einer betrachteten Baugruppe erfordern. Allen Produktsichten gemeinsam ist die Fokussierung auf einen eingeschränkten, auf den Kontext der aktuellen Entwicklungsaktivität angepassten Ausschnitt des gemeinsamen Produkts.

Korrespondierend zu dem in Abbildung 2.6 dargestellten Entwicklungsprozess werden im Folgenden charakteristische Produktsichten vorgestellt, die während der Entwicklung einer Niederdruckturbine bzw. eines Rotors eine besonders wesentliche Rolle spielen. Wiederum kann auch an dieser Stelle

nur eine Auswahl getroffen werden, welche jedoch geeignet ist, im weiteren Verlauf der Arbeit als repräsentatives Fallbeispiel und somit als Grundlage der Konzeption einer durchgängigen und integrativen Entwicklung komplexer technischer Produkt zu dienen. Im Einzelnen werden die folgenden Produktsichten betrachtet:

Konfigurationssicht: Die Konfigurationssicht wird zu Prozessbeginn erstellt und gibt ein Grobkonzept der angestrebten Turbine wieder. Hierbei wird eine hierarchische Zerlegung des Produkts in unterscheidbare Bestandteile vorgenommen. Folgerichtig werden ausschließlich Aggregationsbeziehungen zwischen Bauteilen und Baugruppen betrachtet. Abbildung 2.7 zeigt eine sehr vereinfachte Konfigurationssicht einer Niederdruckturbine, bestehend aus einem Stator, der ihre unbeweglichen Bestandteile zusammenfasst, einem Rotor mit zwei Turbinenscheiben (engl.: *disk*), sowie einem Gehäuse (engl.: *casing*).

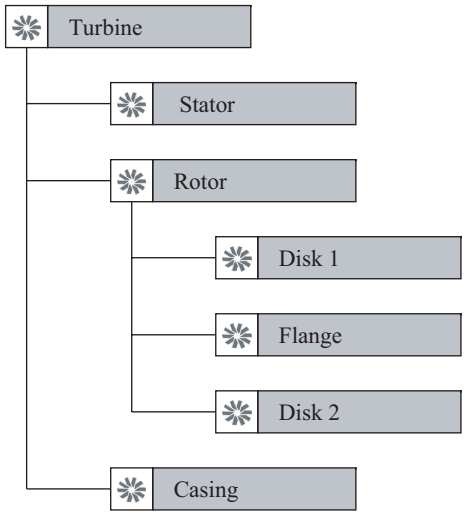

Abbildung 2.7: Vereinfachte Konfigurationssicht

Die Konfigurationssicht orientiert sich an klar trennbaren, physischen Einheiten – also Bauteilen und Baugruppen – zur Strukturierung eines Produkts. Sie repräsentiert damit eine Bauteilstückliste, welche im weiteren Prozessverlauf kontinuierlich verfeinert wird. In diesem Zusammenhang ist es ebenso möglich, einzelne Eigenschaften eines Bauteils in die Konfigurationssicht aufzunehmen. Sie umfasst i.A. jedoch keine fachbezogenen, z.B. geometrischen oder strukturmechanischen Bauteil-

informationen, sondern soll im Wesentlichen die Einordnung der Bauteile in den Gesamtentwurf darstellen.

Konstruktive Sicht: Die konstruktive oder geometrische Sicht erfasst die äußere Gestalt der zu entwickelnden Bauteile. Dies wird durch Methoden des Computer Aided Design (CAD) erreicht, welche es erlauben, mithilfe geeigneter Werkzeuge wie beispielsweise CATIA [CAT04] oder NX [NX04], zwei- oder dreidimensionale Modelle der betreffenden Bauteile konstruktiv aus einfachen geometrischen Grundelementen zu erstellen. Abbildung 2.8 zeigt das gewählte Fallbeispiel eines Rotors mit zwei Turbinenscheiben im Aufriss und in einer daraus durch Rotation entstehenden räumlichen Gestalt.

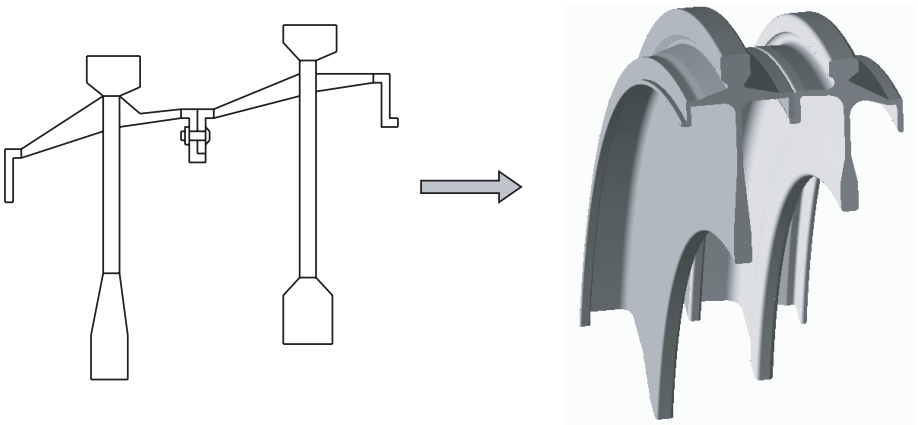

Abbildung 2.8: Aufriss und räumliche Gestalt eines Rotors

Wie bereits an dem betrachteten Beispiel deutlich wird, können verschiedene geometrische Produktsichten während der Konstruktion auftreten. Im Rahmen der vorliegenden Arbeit ist es ausreichend, sich auf zweidimensionale Aufrissmodelle (engl.: *sketch*) zu beschränken. Diese sind oftmals hierarchisch, baumartig organisiert gemäß ihres Aufbaus aus geometrischen Grundelementen, wie Linien und Kurven, welche gleichzeitig die feinste Gliederungsebene der konstruktiven Sicht darstellen. Entsprechend können geometrische Partialmodelle als Verfeinerung der Konfigurationssicht betrachtet werden.

Simulationssichten: Die Prozessanteile der Thermodynamik, Aerodynamik und Strukturmechanik beruhen im Kern darauf, ausgesuchte Ei-

genschaften einer Turbinenkonstruktion – ihre Temperaturverteilung, das Strömungsverhalten und die Betriebssicherheit – unter Nutzung numerischer Simulationsverfahren zu ermitteln. Dazu erstellen diese Fachdisziplinen ausgehend von einer detaillierten geometrischen Beschreibung eine diskretisierte Darstellung der Bauteile, sog. Finite-Elemente-Modelle, welche die Bauteilform durch ein Netz aus Dreiecken oder Polygonen nachahmen.

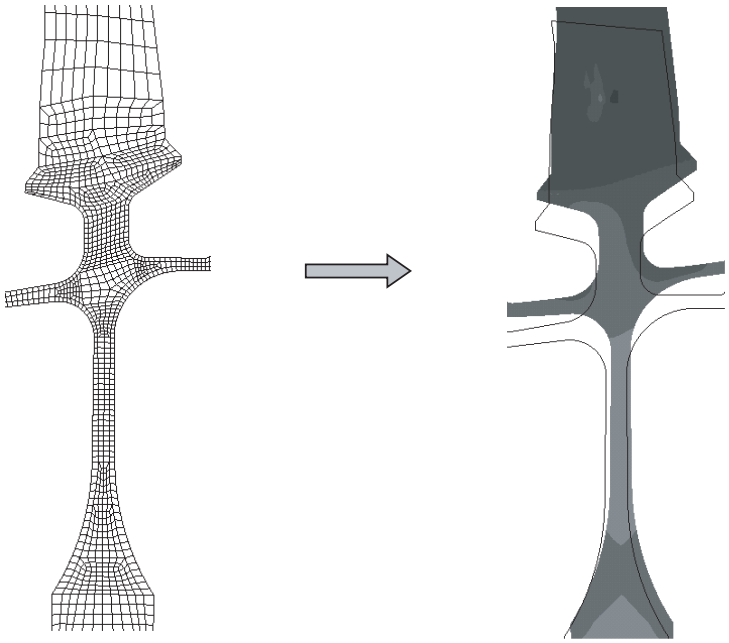

Abbildung 2.9: Vernetzung und Verformung einer Turbinenscheibe

Finite-Elemente-Modelle stellen die Voraussetzung zur Nutzung numerischer Simulationsverfahren im Bereich des Computer Aided Engineering dar. Mit ihrer Hilfe lassen sich für die genannten Fachdisziplinen relevante Bauteileigenschaften, wie z.B. die Temperaturverteilung an den Knotenstellen der netzartigen Darstellung und somit durch Interpolation für jede Stelle des Bauteils, errechnen. Abbildung 2.9 zeigt ein solches Modell einer Turbinenscheibe sowie eine daraus abgeleitete, farblich markierte Darstellung auftretender Kräfte zusammen mit Verformungen des Bauteils unter Belastung.

Offensichtlich existieren signifikante Unterschiede zwischen dem Produktverständnis der Konstruktion und der in den Fachdisziplinen des

Computer Aided Engineering relevanten Auffassung, obwohl allen Prozessabschnitten das gleiche, gemeinsame Produkt bzw. Bauteil zugrunde liegt. Während im Geometrieentwurf ein Bauteil als klar gegliedertes, hierarchisches Modell aus geometrischen Grundelementen dargestellt wird, erfordern numerische Simulationsverfahren eine vollständig andere, netzartige Abbildung. Entsprechend schwer gestaltet sich die Gewährleistung der Konsistenz zwischen diesen Produktsichten.

Dokumentationssicht: Zum Abschluss des exemplarischen Entwicklungsprozesses (vgl. Abbildung 2.6) wird auch im Rahmen der Dokumentation eine eigene Produktsicht erstellt. Diese fasst über verschiedene Prozessabschnitte hinweg alle wesentlichen Aspekte des gefundenen Turbinenentwurfs zusammen. Sie dient einerseits zur Dokumentation und Archivierung der Entwicklungsergebnisse und somit als Ausgangspunkt für neue Entwicklungsprozesse und stellt andererseits die Voraussetzung zur Zulassung des entwickelten Produkts durch Luftfahrtbehörden dar.

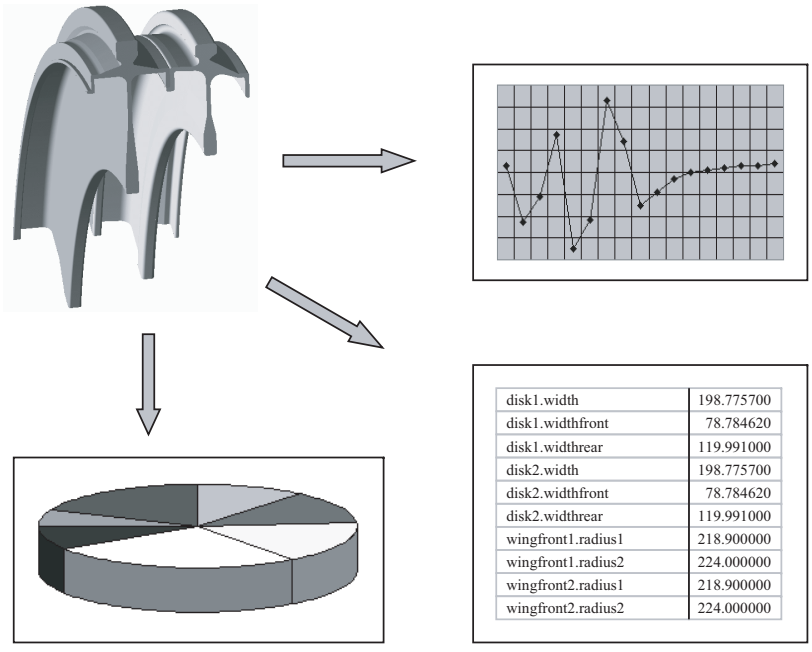

Abbildung 2.10: Dokumentation eines Rotorentwurfs

Im Gegensatz zum heutigen Stand der Produktentwicklung bietet eine digital aufbereitete Dokumentation erstrebenswerte Vorteile: So lassen

sich Entwicklungsergebnisse rasch auffinden und ggf. wieder verwenden. Abbildung 2.10 zeigt beispielhaft die Dokumentation eines Rotorentwurfs. In diese können gemäß der dokumentierten Entwicklungsergebnisse sehr unterschiedliche Darstellungsformen eingehen. So kann die Dokumentationssicht Konstruktionszeichnungen des entwickelten Produkts beinhalten, sowie tabellarisch oder graphisch erfasste Produkteigenschaften.

Anhand des gewählten Fallbeispiels wird deutlich, dass in der virtuellen, rechnergestützten Entwicklung eines komplexen Produkts verschiedenartige Partialmodelle verwendet werden gemäß dem Verständnis des jeweiligen Bearbeiters, der Art des Produkts, dem Zweck seiner Bearbeitung oder auch den Möglichkeiten und Anforderungen verwendeter Software-Werkzeuge. Diese Produktsichten stellen das betrachtete Produkt bzw. Bauteil auf je eigene und ggf. sehr verschiedene Arten dar. Während beispielsweise die konstruktive, geometrische Sicht ein Bauteil streng hierarchisch gliedert, werden im Rahmen numerischer Simulationen flache, graphartige Strukturen benutzt.

In heutigen Produktentwicklungsprozessen werden Produktsichten als eigene Teilmodelle erstellt, welche jedoch oftmals isoliert und zueinander inkompatibel sind (vgl. Abschnitt 2.1.3). In der Folge müssen sie im Laufe der Produktentwicklung mehrmals auf aufwändige und unstrukturierte Weise manuell aufeinander abgestimmt werden. Dagegen bietet ein konzeptioneller Rahmen für eine integrative, modellbasierte Produktentwicklung, welche verschiedene Produktsichten und ihre Querbezüge explizit berücksichtigt sowie einen methodischen Umgang mit diesen vorsieht erstrebenswerte Möglichkeiten der Verbesserung.

2.3 Zusammenfassung

In zunehmender Weise finden Informations- und Kommunikationstechnologien Eingang in die Entwicklungs- und Herstellungsprozesse komplexer technischer Produkte. Mithilfe digitaler Modelle ermöglichen sie es, die Effektivität und Effizienz zu steigern und Entwicklungskosten und Entwicklungszeiten bei gleich bleibender oder gar erhöhter Produktqualität zu reduzieren. Entsprechend haben sich bisher über verschiedene Wirtschaftszweige hinweg zahlreiche und vielfältige Software-Lösungen etabliert zur Unterstützung unterschiedlicher Teilaufgaben der Entwicklungsprozesse verschiedenster technischer Produkte.

Durch die beständig wachsende Anzahl und Vielfalt eingesetzter Software-Werkzeuge werden jedoch neue Fragestellungen aufgeworfen. Während zur

Unterstützung fachspezifischer Einzelaktivitäten auf vielen Gebieten inzwischen zahlreiche ausgereifte Lösungen zur Verfügung stehen, stellt die Schaffung einer durchgängigen und integrierten Kette reibungslos ineinander greifender Werkzeuge eine bisher unzureichend beantwortete Problemstellung in Wissenschaft und Praxis dar. Vor dem Hintergrund dieser Zielsetzung werden zunächst Charakteristika und Defizite gegenwärtiger Produktentwicklungsprozesse in dreierlei Bereichen betrachtet.

So verfügen komplexe technische Produkte über lange Produktlebenszyklen, welche umfangreiche Entwicklungsprozesse beinhalten. Diese sind jedoch, wie auch zugrunde liegende Organisationsstrukturen, in unzureichender Weise strukturiert und mit verwendeten Modellen und zugehörigen Werkzeugen verknüpft. Produkte werden ihrerseits durch isolierte und oft inkompatible Teilmodelle erfasst ohne explizite Berücksichtigung vorhandener Querbezüge. Schließlich stellt eine unzureichende technische wie fachliche Integration verwendeter Software-Werkzeuge ein weiteres Hindernis für die Verbesserung heutiger Entwicklungsprozesse dar.

Die Einführung einer konkreten Fallstudie im zweiten Teil dieses Kapitels verdeutlicht die genannten Punkte und stellt für den weiteren Verlauf der Arbeit eine durchgängige Grundlage zur Illustration und Validierung der zu entwickelnden Konzepte zur Verfügung. Als Fallbeispiel eines komplexen technischen Produkts wird basierend auf dem Forschungs- und Entwicklungsprojekt ARTWORK [ART05, ART03b, ART03a] in vereinfachter aber hinreichend repräsentativer Weise der Entwurf einer Niederdruckturbine bzw. eines Rotors samt zugehörigem Entwicklungsprozess, auftretenden Produktsichten und eingesetzten Software-Werkzeugen erläutert.

3 Integrationsansätze

Die Betrachtung des Anwendungsbereichs im vorangehenden Kapitel verdeutlicht die Charakteristika und Defizite von Produktentwicklungsprozessen in ihrer heutigen Form aus der Sicht einer durchgängigen Unterstützung durch Software-Werkzeuge. Als ein wesentliches Hemmnis der weiteren Verbesserung wird eine unzureichende Integration in dreierlei Bereichen identifiziert: Zunächst sind Entwicklungsprozesse wie auch benötigte Ressourcen nicht ausreichend erfasst und mit den relevanten Modellen des Produkts sowie den eingesetzten Software-Werkzeugen integriert. Ferner fehlt weit gehend eine fachliche Integration auftretender Produktmodelle untereinander. Gleiches gilt für die zur Erstellung und Bearbeitung der Modelle verwendeten Software-Lösungen.

In verschiedenen Studien über die Entwicklung komplexer Produkte (vgl. [CBC99], [Har93] oder [HY99]) wird die Aufteilung der Arbeitszeit während der Konstruktionsphase untersucht. Abbildung 3.1 zeigt die ermittelten Ergebnisse in Anlehnung an [Har93]. Demnach wird der größte Teil der Arbeitszeit in Höhe von 34% für administrative Aufgaben wie Planung oder Controlling verwendet. Ein Anteil in Höhe von 31% entfällt auf Kommunikationstätigkeiten, z.B. die Durchführung von Arbeitstreffen oder die Bearbeitung von E-Mails. Dagegen werden 24% der verfügbaren Zeit unproduktiv mit dem Warten auf Informationen und Entscheidungen verbracht. Nur der weitaus kleinste Teil der Arbeitszeit in Höhe von 11% wird für kreative Entwicklungsaufgaben genutzt.

Wie daraus unmittelbar erkenntlich ist, existiert ein hohes Potenzial zur weiteren Verbesserung der Effizienz und Effektivität heutiger Produktentwicklungsprozesse durch den Einsatz integrierter Informations- und Kommunikations-Technologien. Gemäß Kapitel 2 ergeben sich hierzu drei verschiedene Ansatzpunkte, welche in den folgenden Abschnitten beschrieben sind. Zunächst wird in Abschnitt 3.1 die Integration auf der Grundlage eines durchgängigen Prozessmodells erläutert. Abschnitt 3.2 betrachtet die fachliche Integration verwendeter Produktmodelle, während Abschnitt 3.3 auf die Integration eingesetzter Software-Werkzeuge eingeht. Eine übergreifende

Konzeption, welche die vorgestellten Ansätze verbindet, wird schließlich in Abschnitt 3.4 präsentiert.

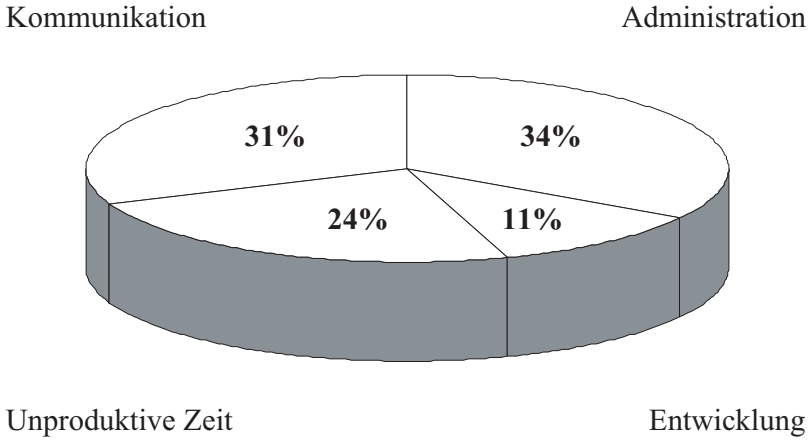

Abbildung 3.1: Arbeitszeitverteilung

3.1 Prozessintegration

Als *Prozessintegration* oder auch *Verfahrenstechnische Integration* wird ein in der Literatur häufig anzutreffender Integrationsansatz bezeichnet. Hierbei werden die Abläufe der Produktentwicklung durch eine übergreifende Prozesskomponente, wie beispielsweise ein Workflow-Management-System, geplant, gesteuert und überwacht. Vor allem seitens der Workflow Management Coalition (WfMC) [WfM04], einer 1993 entstandenen internationalen Organisation von Herstellern, Anwendern und Forschergruppen im Workflow-Bereich, wird dieser Ansatz vorangetrieben. Seitdem ist eine Reihe an Standards entstanden, wie insbesondere das Workflow-Referenzmodell [Hol95] und die XML-basierte Prozessdefinitionssprache XPDL [XPD02], welche in zahlreichen Software-Werkzeugen dieses Bereichs Verwendung finden.

Abbildung 3.2 stellt die Prozessintegration im Überblick dar. Im Mittelpunkt steht ein übergreifendes *Prozessmodell*, welches alle Entwicklungstätigkeiten und ihre Beziehungen erfasst. Hierbei kann Bezug genommen werden auf ein zugrunde liegendes *Organisationsmodell*, in dem die zur Verfügung stehenden und für die Durchführung einer Tätigkeit erforderlichen Ressourcen hinterlegt sind. Durch eine aktive Prozesskomponente, wie z.B. ein

Workflow-Management-System, werden hieraus die gegenwärtig durchzuführenden Aktivitäten bestimmt und entsprechenden Entwicklern zugeordnet. Diese können daraufhin mithilfe geeigneter Software-Werkzeuge das aktuell relevante Partialmodell des Produkts gemäß der Zielsetzung der ihnen zugewiesenen Aktivität bearbeiten.

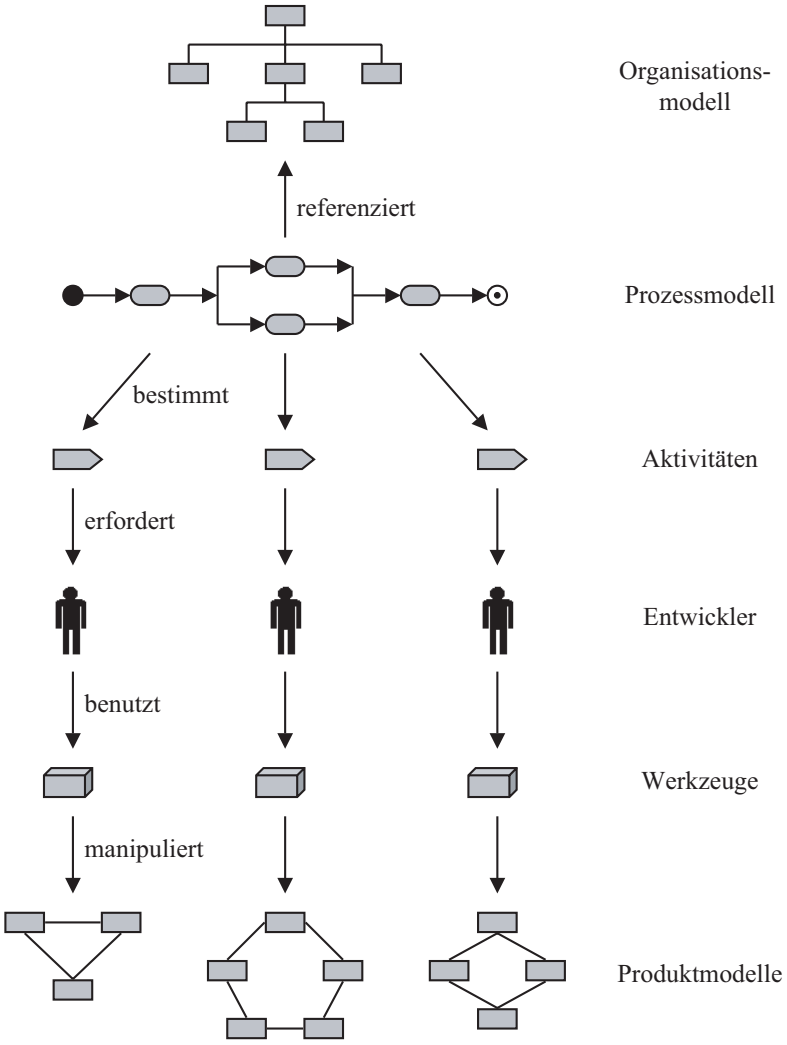

Abbildung 3.2: Prinzip der Prozessintegration

Auf diese Weise lässt sich ein strukturiertes und koordiniertes Vorgehen in der Produktentwicklung erreichen. Die Integration unterschiedlicher Fachdisziplinen und Bearbeiter erfolgt über eine gemeinsame Prozesskomponente,

welche auf der Grundlage eines durchgängigen Prozess- und Organisationsmodells den Entwicklungsprozess steuert und überwacht. Zudem erlaubt es ein solcher Ansatz, Techniken des *Computer Supported Cooperative Work (CSCW)* [BS98, SSU01, Zhu01] zur nebenläufigen und verteilten Produktentwicklung zu nutzen. Nicht zuletzt bietet die klare Dokumentation des Prozesses die Möglichkeit, Optimierungspotenziale, wie beispielsweise die Parallelisierung unabhängig voneinander durchführbarer Aktivitäten, zu identifizieren und umzusetzen.

Die technische wie fachliche Integration eingesetzter Software-Werkzeuge und korrespondierender Partialmodelle des Produkts wird durch den Ansatz der Prozessintegration jedoch unzureichend adressiert. So bietet beispielsweise das Workflow-Referenzmodell der WfMC [Hol95] lediglich eine sehr schmale und generische Schnittstelle zur Kommunikation mit externen Applikationen an. Auf diese Weise können weder Art noch Ausmaß der mittels eines Software-Werkzeugs vorgenommenen Manipulationen an Partialmodellen erfasst werden. Ebenso fehlt eine explizite Möglichkeit zur Spezifikation, Verwaltung und Überprüfung fachlicher Querbezüge zwischen verschiedenen Partialmodellen, welche nicht zuletzt auch Auswirkungen auf den Entwicklungsprozess haben können.

3.2 Produktintegration

Entscheidende Defizite der Prozessintegration gemäß Abschnitt 3.1 werden durch den Ansatz der *Produktintegration* – auch *Modelltechnische Integration* genannt – überwunden. Diese bezeichnet das Bestreben, die zahlreichen und vielfältigen Produktmodelle, welche im Laufe eines Entwicklungsprozesses erstellt und modifiziert werden (vgl. Abschnitt 2.1.3), zu einem *integrierten Produktmodell* zusammenzuführen. Hierbei sind zwei entgegengesetzte Vorgehensweisen denkbar:

Enge Integration: Dieser, auch als *enge Kopplung* bezeichnete Ansatz, hat die Zielsetzung, verschiedene Produktmodelle durch ein einziges, allen Bearbeitern gemeinsames Modell zu ersetzen, welches sämtliche Produktinformationen zentral erfasst (vgl. z.B. [Kei99]). Produktsichten sind somit nicht eigenständig und unabhängig, sondern stellen lediglich Projektionen dar, welche sich in umkehrbar eindeutiger Weise aus dem zentralen Modell ermitteln lassen.

Lose Integration: Die lose Integration oder auch *lose Kopplung* begreift das integrierte Produktmodell dagegen als eine Reihe eigenständiger

und ursprünglich unabhängiger Produktmodelle, welche auf der Grundlage gemeinsamer Modellierungskonzepte verschiedene Aspekte des zu entwickelnden Produkts repräsentieren (vgl. z.B. [GV03b]). Die Integration erfolgt durch eine explizite Erfassung und Verwaltung fachlicher Querbezüge zwischen diesen Modellen.

Eine ausführliche Gegenüberstellung, Diskussion und Bewertung der beiden Ansätze erfolgt in Abschnitt 4.3.1 als Grundlage für die im Rahmen der vorliegenden Arbeit verfolgten Konzeption. Beiden Alternativen gemeinsam ist – im Gegensatz zur Prozessintegration (s. Abschnitt 3.1) und Werkzeugintegration (s. Abschnitt 3.3) – die Verwendung eines übergreifenden Produktmodells, welches einen Rahmen zur gleichartigen Erfassung unterschiedlicher Produktinformationen zur Verfügung stellt. Abbildung 3.3 illustriert dieses Prinzip. Wiederum erfolgt die Durchführung einzelner Aktivitäten des Entwicklungsprozesses durch geeignete Entwickler, welche ihrerseits entsprechende Werkzeuge verwenden. Hierbei generierte bzw. modifizierte Produktinformationen werden jedoch nicht individuell in proprietären und isolierten Teilmodellen, sondern auf gleichartige Weise im Rahmen eines übergreifenden Produktmodells erfasst. Die spezifische Gestalt dieses Produktmodells ist dabei von der gewählten Integrationsvariante abhängig.

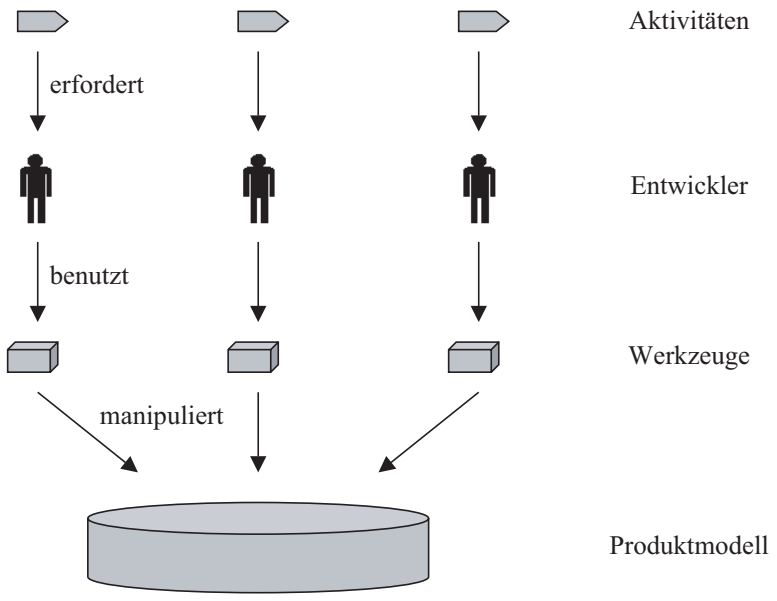

Abbildung 3.3: Prinzip der Produktintegration

Dieses integrierte Produktmodell bietet ein solides Fundament, auf dem sich im Entwicklungsprozess generierte Produktinformationen, daran beteiligte Bearbeiter sowie eingesetzte Software-Lösungen zuverlässig integrieren lassen. So werden Produktdaten nicht mehr in proprietären und verbindungslosen Teilmodellen unabhängig voneinander gehalten und bearbeitet, sondern stehen an zentraler Stelle zur Verfügung. Auf diese Weise sind Modifikationen, die an einer Stelle des Produktmodells im Rahmen einer spezifischen Entwicklungsaktivität vorgenommen werden, nicht lokal isoliert, sondern lassen sich zur Vermeidung eines inkonsistenten Gesamtzustands an andere Bestandteile des Produktmodells propagieren. Da das Produktmodell so stets den aktuellen Entwicklungsstand repräsentiert, kann es folgerichtig auch als Ausgangspunkt zur Koordination des Prozesses genutzt werden. Schließlich bietet die Verwendung eines übergreifenden Modells eine natürliche Grundlage zur Festlegung spezifischer Schnittstellen, über welche sich unterschiedliche Software-Werkzeuge anbinden lassen.

Aufgrund ihrer konzeptionellen Klarheit erscheint die Produktintegration als besonders vielversprechend. Jedoch wirft sie eine Reihe sehr grundlegender Fragestellungen auf, welche in existierenden Ansätzen nicht ausreichend beantwortet sind:

- Voraussetzung für die Erstellung eines Produktmodells ist eine Modellierungstechnik, welche grundlegende Modellierungskonstrukte zur Verfügung stellt. Hierbei ist zu klären, welche Modellierungskonzepte im Bereich komplexer technischer Produkte zur Erfassung ihrer Struktur und ihrer Eigenschaften erforderlich sind.

- Wie in Abschnitt 2.1.3 beschrieben, existieren unterschiedliche Sichtweisen, welche verschiedene Aspekte komplexer technischer Produkte beleuchten. Folgerichtig muss ein integriertes Produktmodell geeignete Mechanismen anbieten, um unterschiedliche Produktsichten zu ermöglichen und diese zueinander in Beziehung zu setzen.

- Komplexe technische Produkte – und somit auch ihre Modelle – sind nicht monolithisch, sondern bestehen aus zahlreichen und vielfältigen Bestandteilen mit Querbezügen und Abhängigkeiten. Im Rahmen eines integrierten Produktmodells sind diese geeignet zu erfassen und flexibel zu verwalten.

In Teil II und Teil III der vorliegenden Arbeit werden die genannten Punkte aufgegriffen und ein Lösungsansatz zur integrativen Entwicklung komplexer technischer Produkte entworfen.

3.3 Werkzeugintegration

Neben den fachlich motivierten Ansätzen der Prozessintegration und Produktintegration versucht die *Werkzeugintegration*, auch *Software-technische Integration* genannt, die im Rahmen eines Produktentwicklungsprozesses verwendeten Software-Werkzeuge (vgl. Abschnitt 2.1.4) auf technischer Ebene zu integrieren. Das wesentliche Ziel ist die Schaffung technischer Schnittstellen, um eine Kommunikation zwischen verschiedenen Werkzeugen zu ermöglichen. Dabei sollen die von Werkzeugen angebotenen fachlichen Dienste von der zugrunde liegenden technischen Infrastruktur abstrahiert werden, so dass beispielsweise eine Kommunikation zwischen Werkzeugen unterschiedlicher Systemplattformen möglich ist.

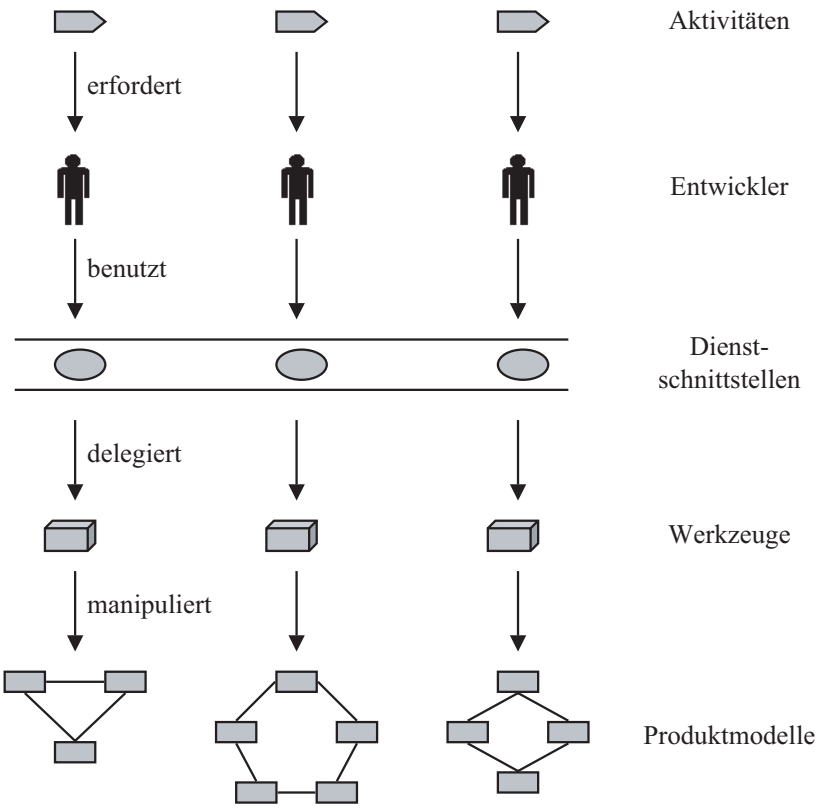

Abbildung 3.4: Prinzip der Werkzeugintegration

Unter der Bezeichnung *Enterprise Application Integration (EAI)* [Kai02, Kel02, Lin99, Cum02] ist die Werkzeugintegration ein in den verschieden-

sten Wirtschaftszweigen zu beobachtender Ansatz, der durch die wachsende Verbreitung und Weiterentwicklung von Middleware-Technologien, wie z.B. CORBA [COR00, OH98], und Internet-Technologien wie XML [W3C04] oder SOAP [SOA03] zunehmend an Bedeutung gewinnt. Die Werkzeugintegration bildet einerseits die technische Grundlage zur durchgängigen Unterstützung von Produktentwicklungsprozessen durch Software-Werkzeuge gemäß Abschnitt 3.1, ersetzt andererseits jedoch nicht eine fachliche Integration der dabei verwendeten Modelle, wie in Abschnitt 3.2 beschrieben.

Abbildung 3.4 zeigt das Prinzip der Werkzeugintegration im Überblick. Wiederum wird von einzelnen Aktivitäten eines Entwicklungsprozesses ausgegangen, welche durch qualifizierte Bearbeiter durchgeführt werden. Diese nutzen jedoch nicht mehr unmittelbar entsprechende Software-Werkzeuge, sondern wenden sich an eine übergreifende Abstraktionsschicht, die fachliche und technische Dienste zur Bearbeitung eines Produktmodells in Form abstrakter Schnittstellen anbietet. Der Aufruf eines Dienstes wird mittels einer zugehörigen Verwaltungs- und Steuerungskomponente an ein geeignetes Werkzeug delegiert, das es einem Bearbeiter schließlich gestattet, die gewünschten Modifikationen eines Produktmodells vorzunehmen.

Durch das Prinzip der Werkzeugintegration können verschiedene Zielsetzungen realisiert werden: Zum einen erlaubt eine explizite Abstraktionsschicht einen einheitlichen Zugriff auf unterschiedliche Systemplattformen, der Entwickler von Besonderheiten der technischen Infrastruktur entlastet, so dass sich diese auf fachliche Aktivitäten konzentrieren können. Ferner lässt sich auf diese Weise eine explizite Verwaltung und Steuerung eingesetzter Software-Werkzeuge erreichen. So ist es beispielsweise möglich, mithilfe eines Werkzeugs vorgenommene Bearbeitungsschritte auf einheitliche Weise zu protokollieren. Schließlich erlaubt die Separation von (fachlichen) Diensten, repräsentiert durch Schnittstellen, und ihrer Realisierung durch Software-Werkzeuge einen deutlich vereinfachten Austausch einzelner Werkzeuge.

3.4 Übergreifende Integration

Die vorangehenden Abschnitte erläutern verschiedene Integrationsansätze, wie sie im Bereich der modellbasierten Entwicklung komplexer Produkte verfolgt werden. Jeder Ansatz weist individuelle Vorteile und Beschränkungen auf. So erlaubt beispielsweise die Prozessintegration gemäß Abschnitt 3.1 eine koordinierte Steuerung der an einem Entwicklungsprozess beteiligten Personen und Ressourcen, bietet jedoch keine unmittelbare Möglichkeit, die zahlreichen Querbezüge und Abhängigkeiten zwischen verwendeten Produktmodellen zu spezifizieren.

Dagegen illustrieren die folgenden Abschnitte, wie sich die vorgestellten Integrationsansätze zu einer übergreifenden Integrationsplattform verknüpfen lassen, um ihre individuellen Vorteile auszuschöpfen und Schwächen auszugleichen. Eine solche Entwicklungsumgebung unterstützt die Koordination eines Entwicklungsprozesses und daran beteiligter Ressourcen, erfasst Produktinformationen im Rahmen eines übergreifenden Produktmodells und bietet einen einheitlichen und transparenten Zugriff auf fachliche und technische Dienste bzw. Werkzeuge der Produktentwicklung.

Abschnitt 3.4.1 fasst die Anforderungen zusammen, die vor dem Hintergrund der in Kapitel 2 beschriebenen Software-technischen Defizite heutiger Produktentwicklungsverfahren sowie den Potenzialen der vorgestellten Integrationsansätze an eine übergreifende Integrationsplattform zu stellen sind. Anschließend stellt Abschnitt 3.4.2 die Architektur einer Integrationsplattform im Überblick dar und erläutert ihre wesentlichen Bestandteile, bevor Abschnitt 3.4.3 aufzeigt, wie diese gemäß der Zielsetzung einer integrierten und durchgängigen Produktentwicklung zusammenwirken.

3.4.1 Anforderungen und Ziele

Die wesentliche Zielsetzung einer Integrationsplattform ist die Bereitstellung einer integrierten Entwicklungsumgebung für komplexe technische Produkte. Folgerichtig sind sowohl die dabei auftretenden Modelle des Entwicklungsprozesses und des zu entwickelnden Produkts, als auch daran beteiligte Software-Werkzeuge zusammenzuführen, zu verwalten und zu steuern. Darüber hinaus muss eine Reihe nicht-funktionaler Anforderungen berücksichtigt werden, um insbesondere die evolutionäre Weiterentwicklung einer solchen Integrationsplattform zu ermöglichen. Im Einzelnen können folgende Anforderungen bzw. Zielsetzungen identifiziert werden:

Modellerfassung: Eine besonders wesentliche Anforderung an eine übergreifende Integrationsplattform ist die Erfassung sämtlicher für die Produktentwicklung relevanten Informationen an zentraler Stelle durch geeignete Modelle. Wie Abbildung 3.5 illustriert, entstammen diese dreierlei Bereichen: Informationen über die zugrunde liegende Organisationsstruktur sowie verfügbare Ressourcen sind im Rahmen eines Organisationsmodells zu hinterlegen. Darauf aufbauend muss ein Prozessmodell erforderliche Entwicklungsaktivitäten samt ihrer Abhängigkeiten festlegen. Nur so lassen sich Produktentwicklungsprozesse und ihre Bearbeiter rechnergestützt koordinieren gemäß dem Prinzip der Prozessintegration (vgl. Abschnitt 3.1). Die virtuelle Produktentwicklung basiert schließlich – wie in Abschnitt 2.1.3 beschrieben – auf Modellen

des zu fertigenden Produkts, welche sukzessive mithilfe von Software-Werkzeugen bearbeitet werden, bis alle gewünschten Eigenschaften erreicht sind. Daher sind drittens unterschiedliche Teilmodelle eines Produkts sowie Querbezüge im Rahmen eines übergreifenden Produktmodells gemäß dem Prinzip der Produktintegration zusammenzuführen (vgl. Abschnitt 3.2).

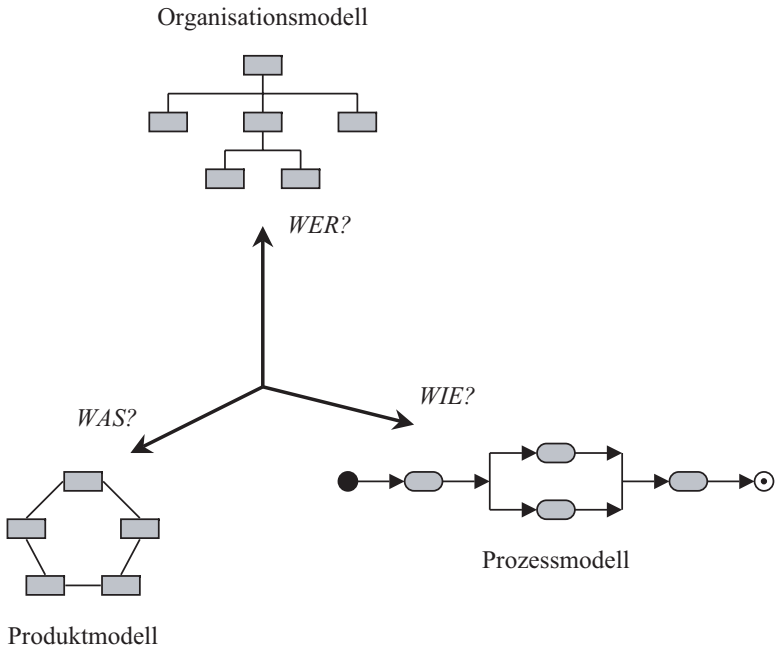

Abbildung 3.5: Modelldimensionen

Steuerung und Administration: Neben der passiven Erfassung der für die Produktentwicklung relevanten Informationen durch Modelle muss eine übergreifende Integrationsplattform ebenso aktive Komponenten für die Bearbeitung der Modelle anbieten. Hierzu ist zunächst eine Komponente zur Verwaltung der hinterlegten Modelle erforderlich, welche den Modellzugriff und Modelländerungen ermöglichen muss. Darauf aufbauend ist eine übergreifende Prozesskomponente im Sinne eines Workflow-Management-Systems erforderlich, die den Entwicklungsprozess und daran beteiligte Ressourcen und Bearbeiter steuert und administriert. Ebenso sind die von der Integrationsplattform gemäß dem Prinzip der Werkzeugintegration (s. Abschnitt 3.3) angebotenen Dienste zur Manipulation der Modelle zu verwalten. Als *Dienst* wird hierbei

eine fachliche oder technische Funktionalität eines für die Produktentwicklung erforderlichen Software-Werkzeugs bezeichnet.

Datenzugriff: Für die Produktentwicklung erforderliche Daten entstammen üblicherweise vielfältigen Datenquellen, beispielsweise relationalen oder objektorientierten Datenbanksystemen [HS00]. Eine übergreifende Integrationsplattform muss einen einheitlichen Zugriff auf unterschiedliche Datenquellen ermöglichen und von konkret verwendeten Persistenzmechanismen abstrahieren, um Änderungen an Datenquellen auf einfache Weise zu erlauben. So sollte es beispielsweise möglich sein, einzelne Datenbanksysteme gleicher Art ohne Änderung der Anwendungslogik gegeneinander auszutauschen. Gemäß dem Prinzip der Werkzeugintegration besteht darüber hinaus die Zielsetzung, von Aspekten der technischen Infrastruktur einzelner Datenquellen zu abstrahieren, wie beispielsweise ihre Verteilung auf unterschiedliche Rechnersysteme.

Werkzeugzugriff: Analog zum Datenzugriff ist auch ein einheitlicher Werkzeugzugriff von entscheidender Bedeutung für eine übergreifende Integrationsplattform. Dies betrifft sowohl die Anbindung und Verwaltung verwendeter Software-Werkzeuge, wie auch den Aufruf und die Steuerung. Die von Werkzeugen angebotene und im Laufe der Produktentwicklung erforderliche Funktionalität ist hierbei in Form abstrakter Schnittstellen gemäß dem Prinzip der Werkzeugintegration zu kapseln, um den Austausch einzelner Werkzeuge zu vereinfachen. Ebenfalls zu berücksichtigen ist eine Abstraktion von der zugrunde liegenden technischen Infrastruktur. So sollen sich Bearbeiter auf die Nutzung benötigter Dienste konzentrieren können, ohne technische Besonderheiten einzelner Werkzeuge, beispielsweise ihren Ausführungsort innerhalb eines Rechnernetzes, beachten zu müssen.

Benutzerschnittstelle: Neben der modellhaften Erfassung von Produkt-, Prozess- und Organisationsinformationen, ihrer Verwaltung und Steuerung sowie den Zugriff auf Datenquellen und Werkzeuge, ist im Rahmen einer übergreifenden Integrationsplattform auch eine interaktive Benutzerschnittstelle vorzusehen. Diese soll den zentralen Anlaufpunkt für sämtliche Beteiligte eines Produktentwicklungsprozesses darstellen, Bearbeiter über durchzuführende Aktivitäten informieren, den Zugang zu Modellen erlauben und die Benutzung von Werkzeugen bzw. Diensten ermöglichen. Individuelle Benutzerschnittstellen einzelner Werkzeuge sind – soweit technisch möglich – zu integrieren. Für die übergreifende Benutzerschnittstelle sind ferner die Grundprinzipien der *Benutzer-*

freundlichkeit (s. z.B. [Cha92]), wie geringer Lernaufwand, Fehlertoleranz oder Verlässlichkeit, zu berücksichtigen.

Nicht-funktionale Anforderungen: Außer den beschriebenen funktionalen Anforderungen an eine übergreifende Integrationsplattform sind ebenso stets zu beachtende nicht-funktionale Qualitätseigenschaften zu fordern, wie sie beispielsweise der ISO Qualitätsstandard 9126 [ISO01] festlegt. Insbesondere ist ein hohes Maß an Erweiterbarkeit, Stabilität und Effizienz notwendig. Angesichts einer kontinuierlichen Verbesserung kommerzieller Software-Systeme ist für eine dauerhaft tragfähige Integration eine flexible Basis notwendig, welche auf eine Anbindung weiter entwickelter oder gar neuer Software-Werkzeuge ausgerichtet ist. Als zentraler Integrations- und Anlaufpunkt für sämtliche Aktivitäten der Produktentwicklung muss diese besonders verlässlich sein und effiziente Abläufe gewährleisten.

Die aufgeführten Punkte benennen die grundsätzlichen Anforderungen an eine Integrationsplattform auf einer hohen Abstraktionsebene. Entsprechend sind zahlreiche Freiheitsgrade enthalten, die im Zuge eines detaillierten Feinentwurfs festzulegen sind. So führt beispielsweise die Forderung nach einem übergreifenden Produktmodell zu zahlreichen Fragestellungen, welche in Teil II der vorliegenden Arbeit untersucht werden.

3.4.2 Architekturentwurf

Aus den formulierten Anforderungen an eine übergreifende Integrationsplattform wird der Entwurf einer (logischen) Architektur abgeleitet. Gemäß der Zielsetzung einer Integrationsplattform als zentrales Informationssystem wird im Kern – wie in Abbildung 3.6 dargestellt – eine klassische Drei-Schichten-Architektur gewählt (vgl. [Bal99]). Diese besteht aus

- einer *Präsentationsschicht* für den interaktiven Dialog mit Benutzern und der Darstellung von Modellinformationen,
- einer *Anwendungsschicht*, welche Modellinformationen erfasst, administriert und steuert sowie benötigte Dienste zur Verfügung stellt,
- einer *Ressourcenschicht*, die aus Datenquellen sowie Werkzeugen zur Produktentwicklung besteht.

Jede Schicht kann auf weiter unten liegende Schichten zugreifen, jedoch nicht auf darüber liegende Schichten. Der wichtigste Vorteil dieses Ansat-

zes ist die Austauschbarkeit einzelner Schichten, beispielsweise der Benutzerschnittstelle, ohne Änderungen an darunter liegenden Schichten vornehmen zu müssen.

An oberster Stelle des in Abbildung 3.6 dargestellten Architekturentwurfs findet sich die Präsentationsschicht, welche die in Abschnitt 3.4.1 geforderte interaktive Benutzerschnittstelle realisiert. Folgerichtig ist die Präsentationsschicht dafür verantwortlich, Benutzer über durchzuführende Aktivitäten zu informieren, den Zugang zu Modellen zu erlauben sowie die Benutzung von Diensten bzw. Werkzeugen zu ermöglichen. Hierbei kann sich die Präsentationsschicht auf die Komponenten der darunter liegenden Anwendungsschicht abstützen, die in drei Bereiche aufgeteilt ist: Im Bereich *Modelle* werden Produkt-, Prozess- und Organisationsinformationen an zentraler Stelle erfasst. Daneben findet sich der Bereich *Steuerung*, der den Kern der Integrationsplattform darstellt. Dieser ist dafür verantwortlich, das übergreifende Zusammenspiel von Entwicklungsprozess, dabei verwendeten Modellen des zu fertigenden Produkts und eingesetzten Werkzeugen zu gewährleisten. Drittens tritt der Bereich *Dienste* auf, der als Abstraktionsschicht für die in der Produktentwicklung verwendeten Werkzeuge verstanden werden kann. Hierbei lassen sich *fachliche* und *technische* Dienste unterscheiden: Während fachliche Dienste zu nutzen sind, um fachspezifische Aktivitäten des Entwicklungsprozesses durchzuführen, z.B. die Erstellung der Produktgeometrie oder die rechnergestützte Simulation von Produkteigenschaften, bieten technische Dienste anwendungsunabhängige Funktionalität, beispielsweise zur Versionierung eines Entwicklungsstandes oder zur Konvertierung von Modelldaten. Aufgrund der Vielzahl und Vielfältigkeit der für die Entwicklung komplexer technischer Produkte verwendeten Software-Werkzeuge, ist eine hohe Anzahl an Diensten zu erwarten, welche zudem auf fachlicher Seite auf die Art des zu entwickelnden Produkts abzustimmen sind. Dagegen sind die Bereiche *Modelle* und *Steuerung* produktunabhängig und lassen sich in folgende Komponenten unterteilen:

Produktmodell: Ein übergreifendes Produktmodell stellt eine besonders wesentliche Komponente einer Integrationsplattform dar. An dieser Stelle sind gemäß den Prinzipien der virtuellen Produktentwicklung (vgl. Abschnitt 2.1.3) und der Produktintegration (vgl. Abschnitt 3.2) sämtliche relevanten Produktinformationen im Rahmen eines übergreifenden Modells zentral zu erfassen.

Prozessmodell: Das Prozessmodell dient dazu, den Entwicklungsprozess zu formalisieren, um eine rechnergestützte Abarbeitung zu ermöglichen. Somit sind an dieser Stelle die Aktivitäten der Produktentwicklung zusammen mit ihren Abhängigkeiten und unter Verweis auf erforderliche

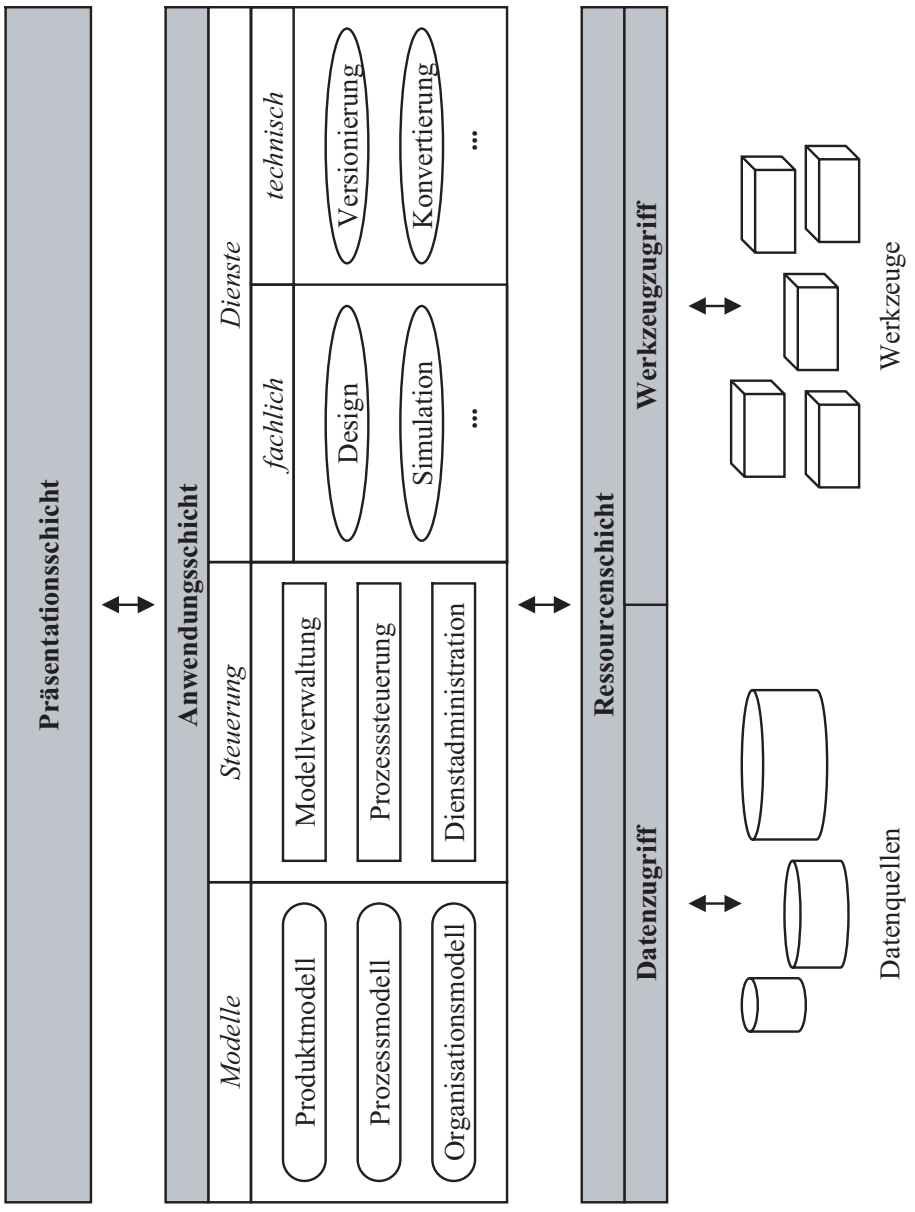

Abbildung 3.6: Architekturentwurf

Ressourcen bzw. Bearbeiter gemäß dem Prinzip der Prozessintegration (vgl. Abschnitt 3.1) zu hinterlegen.

Organisationsmodell: Als dritter Bestandteil des Modellbereichs erfasst das Organisationsmodell die zugrunde liegenden Organisationsstrukturen und verfügbaren Ressourcen bzw. Bearbeiter. Es stellt somit die Grundlage für das Prozessmodell und die rechnergestützte Abarbeitung mithilfe einer übergreifenden Komponente der Prozesssteuerung (s.u.) dar.

Modellverwaltung: Die Modellverwaltung ist dafür verantwortlich, den Zugriff auf die in den genannten Modellen erfassten Informationen zu gestatten und Änderungen an ihnen vornehmen zu können. Dementsprechend lässt sich diese Komponente in eine Produktmodellverwaltung, eine Prozessmodellverwaltung und eine Organisationsmodellverwaltung unterteilen.

Prozesssteuerung: Die Aufgabe der Prozesssteuerung ist es, auf Grundlage des Prozess- und Organisationsmodells aktuell durchzuführende Aktivitäten im Sinne eines koordinierten Produktentwicklungsprozesses geeigneten Bearbeitern zuzuweisen und ihre Abarbeitung zu überwachen. Sie stellt somit die zentrale Steuerungskomponente der Integrationsplattform dar.

Dienstadministration: Die Dienstadministration erlaubt die Verwaltung, Benutzung und Steuerung vorhandener Dienste der Integrationsplattform. Nicht zuletzt ist diese Komponente dafür verantwortlich, im Zusammenspiel mit der Modellverwaltung (s.o.) die Verbindung zwischen genutzten Diensten und dafür erforderlichen Modellen des Modellbereichs herzustellen.

Unterhalb der Anwendungsschicht mit den erläuterten Komponenten ist die Ressourcenschicht angesiedelt. Diese erlaubt einerseits einen einheitlichen Zugriff auf unterschiedliche Datenquellen und abstrahiert von konkret verwendeten Persistenzmechanismen wie Dateisystemen oder Datenbanken. Einzelne Datenquellen lassen sich ohne Änderungen der Anwendungs- oder Präsentationsschicht austauschen. Daneben besteht die Ressourcenschicht aus den erforderlichen Software-Werkzeugen der Produktentwicklung, welche zur Realisierung der in der Anwendungsschicht definierten Dienste dienen. Auch hier soll ein einheitlicher – wenn auch anwendungsabhängiger – Zugriff möglich sein, so dass technische Besonderheiten eingesetzter Werkzeuge, beispielsweise eine spezielle Ausführungsumgebung, auf Seiten der Anwendungsschicht vernachlässigt werden können.

3.4.3 Zusammenspiel

Nach der Vorstellung der wesentlichen Architekturkomponenten einer Integrationsplattform wird in diesem Abschnitt das Zusammenwirken für eine integrative und modellbasierte Entwicklung komplexer technischer Produkte aufgezeigt. Im Mittelpunkt steht eine Integration der für die Produktentwicklung erforderlichen und in Modellen erfassten Produkt-, Prozess- und Organisationsinformationen einerseits sowie der zur Bearbeitung erforderlichen Software-Werkzeuge andererseits. Abbildung 3.7 zeigt das Grundprinzip der Produktentwicklung im Rahmen einer Integrationsplattform (vgl. hierzu Abbildung 3.2, Abbildung 3.3 und Abbildung 3.4).

Gemäß dem Prinzip der Prozessintegration werden sämtliche Entwicklungsaktivitäten zusammen mit ihren gegenseitigen Beziehungen in einem übergreifenden *Prozessmodell* erfasst. Dieses nimmt Bezug auf ein zugrunde liegendes *Organisationsmodell*, in dem die zur Verfügung stehenden und für die Durchführung einer Tätigkeit erforderlichen Ressourcen hinterlegt sind. Unter Benutzung der *Modellverwaltung* bestimmt die *Prozesssteuerung* hieraus die aktuell durchzuführenden Aktivitäten und ordnet diese geeigneten Bearbeitern zu. Auf diese Weise lässt sich ein koordinierter Ablauf eines Produktentwicklungsprozesses erreichen.

Ausgewählte Bearbeiter können sich ihrerseits zur Ausführung der gegenwärtig relevanten Entwicklungsaktivitäten an die *Dienstadministration* wenden, um benötigte *Dienste* aufzufinden und zu verwenden. Die Dienstadministration regelt deren Steuerung und versorgt sie, wiederum unter Benutzung der Modellverwaltung, mit erforderlichen Modellinformationen. Die Realisierung eines Dienstes erfolgt auf Grundlage angebundener Software-Werkzeuge der Ressourcenschicht, welche einen einheitlichen *Werkzeugzugriff* gestattet, so dass rein technische Aspekte der eingesetzten Werkzeuge der Dienstadministration verborgen bleiben.

Die verwendeten Dienste bzw. Software-Werkzeuge werden genutzt, um Produktinformationen zu erzeugen oder zu bearbeiten. Gemäß dem Prinzip der Produktintegration werden diese jedoch nicht individuell in proprietären oder isolierten Teilmodellen, sondern auf gleichartige Weise im Rahmen eines übergreifenden *Produktmodells* erfasst. Der Zugriff auf das Produktmodell wird wiederum mithilfe der Modellverwaltung ermöglicht. Modellinformationen können hierbei unterschiedlichen Datenquellen der Ressourcenschicht zugeordnet sein, welche jedoch einen auf technischer Seite einheitlichen *Datenzugriff* gestatten.

Somit vereint die Integrationsplattform die Ansätze der Prozessintegration (s. Abschnitt 3.1), der Produktintegration (s. Abschnitt 3.2) und der Werkzeugintegration (s. Abschnitt 3.3), versucht ihre Vorteile jeweils zu nut-

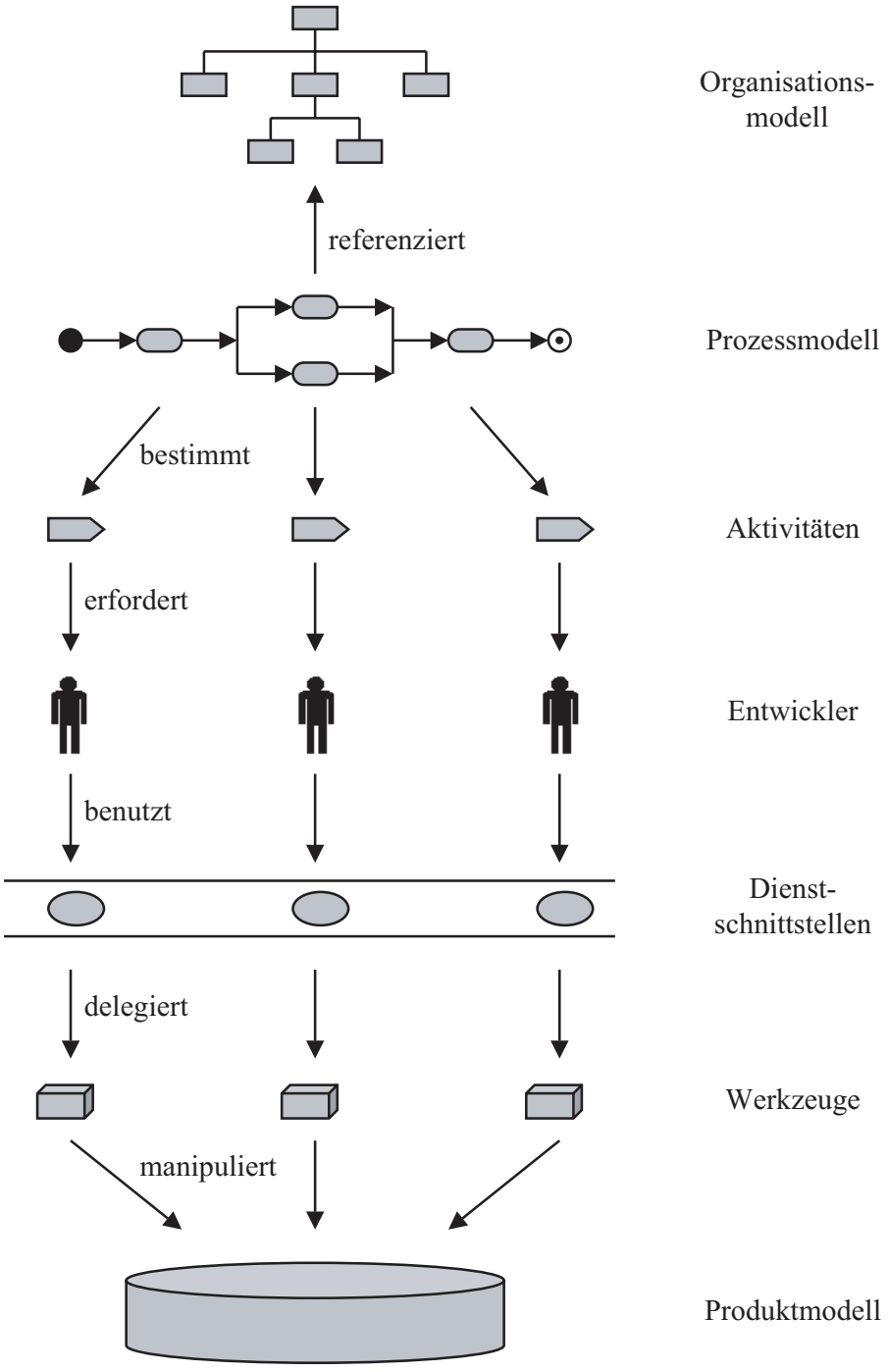

Abbildung 3.7: Zusammenspiel

zen und gegebene Schwächen gegenseitig auszugleichen. So werden an der Produktentwicklung beteiligte Bearbeiter und erforderliche Ressourcen mittels eines gemeinsamen Prozess- und Organisationsmodells koordiniert. Produktinformationen liegen nicht in proprietären oder isolierten Teilmodellen vor, sondern werden an zentraler Stelle mittels eines übergreifenden Produktmodells erfasst, welches verschiedene Sichten unterstützt. Die Unterscheidung zwischen vorhandenen Software-Werkzeugen und dadurch realisierbaren Diensten erlaubt schließlich eine Werkzeugintegration auf technischer Ebene.

3.5 Zusammenfassung

Wie verschiedene Studien über die Entwicklung komplexer technischer Produkte [CBC99, Har93, HY99] aufzeigen, dient nur etwa ein Zehntel der aufgewendeten Arbeitszeit tatsächlichen Entwicklungstätigkeiten. Als wesentliche Ursache ist ein hoher Aufwand für Administrations- und Kommunikationsaufgaben zu nennen, welcher sich auf eine unzureichende Integration des Entwicklungsprozesses mit verwendeten Modellen sowie Software-Werkzeugen zurückführen lässt. Folgerichtig existiert in diesem Bereich ein umfangreiches Potenzial zur weiteren Verbesserung der Effizienz und Effektivität heutiger Produktentwicklungsprozesse.

Vor diesem Hintergrund werden drei verschiedene Integrationsansätze vorgestellt: Zielsetzung der *Prozessintegration* ist es, Entwicklungsaktivitäten und Bearbeiter bzw. Ressourcen auf der Grundlage eines übergreifenden Prozess- und Organisationsmodells zu koordinieren. Dagegen sollen durch die *Produktintegration* die unterschiedlichen, im Zuge der Produktentwicklung verwendeten Teilmodelle eines Produkts im Rahmen eines integrierten Produktmodells zusammengeführt werden. Schließlich versucht der Ansatz der *Werkzeugintegration* eine technisch einheitliche Anbindung und Verwendung eingesetzter Software-Werkzeuge zu erreichen.

Jeder Integrationsansatz weist individuelle Vorteile und Beschränkungen auf, so dass jeweils nur ein Teil der auftretenden Integrationsdefizite überwunden werden kann. Abschnitt 3.4 kombiniert daher die vorgestellten Ansätze zu einer übergreifenden Integrationsplattform. Eine solche Entwicklungsumgebung adressiert sowohl die Koordination der an einem Entwicklungsprozess beteiligten Bearbeiter, als auch die Integration verwendeter Modelle des zu entwickelnden Produkts. Darüber hinaus bildet sie die technische Grundlage für einen einheitlichen Zugriff auf verschiedene Datenquellen und Software-Werkzeuge.

Produktmodellierung

4 Entwurf des Produktmodells

Wie die vorangehenden Kapitel aufzeigen, kann die Effektivität und Effizienz moderner Produktentwicklungsprozesse durch eine verbesserte Verzahnung genutzter Informations- und Kommunikationstechnologien entscheidend erhöht werden. Dies betrifft die Integration einzelner Prozessschritte und Organisationsstrukturen, verwendeter Produktmodelle und eingesetzter Software-Werkzeuge sowohl untereinander als auch miteinander. Auf jedem dieser Teilbereiche sind in den vergangenen Jahren Integrationsansätze entstanden, die sich im Rahmen einer übergreifenden Integrationsplattform Gewinn bringend kombinieren lassen.

Als bedeutendes Kernstück einer solchen Integrationsplattform erscheint ein leistungsfähiges und ausdrucksstarkes *Produktmodell*. Zum einen muss dieses in der Lage sein, die zahlreichen verschiedenartigen Produktinformationen und Teilmodelle, die im Laufe eines Produktentwicklungsprozesses erstellt und bearbeitet werden, an zentraler Stelle zu erfassen, zueinander in Beziehung zu setzen und geeignete Produktsichten zu unterstützen. Im Sinne einer übergreifenden Integration soll es darüber hinaus ein solides Fundament bieten für die Koordination von Entwicklungsaufgaben und die Anbindung relevanter Software-Werkzeuge.

Gegenstand des vorliegenden Kapitels ist daher die eingehende Untersuchung, Diskussion und Auswahl erforderlicher Modellierungskonzepte für eine durchgängige und integrierte Produktmodellierung sowie die Ableitung eines zugehörigen *Metamodells*, welches Begriffe, Abstraktionen und Gesetzmäßigkeiten für Modelle technischer Produkte vorgibt und ihre Gestalt, ihren Umfang und ihre Ausdruckskraft festlegt. Besonderer Wert wird zudem auf eine Veranschaulichung der gewählten Modellierungskonstrukte anhand exemplarischer Ausschnitte der in Kapitel 2.2 eingeführten Fallstudie eines Turbinenrotors gelegt.

Zu Beginn geht Abschnitt 4.1 auf die Grundbegriffe ein, die für das Verständnis der angestrebten Modellierungstechnik erforderlich sind, und präzisiert den bereits verwendeten Begriff des Produktmodells. Wesentliche Anforderungen, die an ein integriertes Produktmodell zu stellen sind, wer-

den anschließend in Abschnitt 4.2 erläutert und bilden den Ausgangspunkt der darauf folgenden Überlegungen. Abschnitt 4.3 diskutiert ausführlich die vier grundlegenden Konzepte für den Entwurf integrierter Produktmodelle, welche in ihrer Gesamtheit das angestrebte Metamodell formen. Eine Reihe möglicher Erweiterungen wird schließlich in Abschnitt 4.4 vorgestellt.

4.1 Grundbegriffe

Die Entstehungsgeschichte des Begriffs *Modell* reicht auf das lateinische Wort „modulus“ zurück, das wie die verwandten Wörter „modus“, „moderor“, „modificor“ und „modulatio“ im Sinne von „Art“, „Weise“, „Form“, „Maß“, „Größe“ oder „Menge“ gebraucht wird [Sta83]. Bereits in der Antike ist „modulus“ ebenso ein architektonischer Fachbegriff, der ein Grundmaß „für die Anlegung der Säulen und des Verhältnisses der einzelnen Teile derselben zueinander“ bezeichnet [GG84]. Daraus entsteht das frühhochdeutsche Lehnwort „Model“, welches in freierem Gebrauch im Sinne von „Muster“, „Form“ oder „Vorbild“ verwendet wird, spezifisch als gewerbliche Musterform eines Fabrikats, welche obrigkeitlich hinterlegt ist.

Im Italien der Renaissance entwickelt sich aus „modulus“ in der Fachsprache der bildenden Künste der italienische Begriff „modello“, der ein Vorbild – oftmals eine Person – zur Anfertigung eines künstlerischen Werkes bezeichnet, aber auch künstlerische Nachbildungen aus Ton oder Wachs, welche ihrerseits als Vorbilder für eine endgültige Skulptur aus Holz, Terrakotta, Marmor oder Bronze dienen. In übertragener und allgemeiner Bedeutung wird „modello“ – wie auch heute im Italienischen – im Sinne von „Vorbild“ oder „Muster“ verwendet. Zusammen mit dem frühhochdeutschen „Model“ führt „modello“ auf den deutschen Begriff *Modell* in seiner heute üblichen Bedeutung.

Ein Modell ist demnach einerseits ein der Nachahmung dienendes Vorbild, in der Kunst oftmals eine Person oder ein Gegenstand, nach deren Vorlage ein künstlerisches Werk angefertigt wird. In umgekehrter Weise wird als Modell jedoch ebenso das Ergebnis der Nachahmung eines – real existierenden oder nur imaginären – Vorbilds bezeichnet. So sind Modelle in der Architektur Nachbildungen eines erst zu schaffenden Originals in verkleinertem Maßstab. Darüber hinaus haben sich weitere, spezifische Bedeutungen des Wortes Modell entwickelt. So wird der Begriff Modell auch zur Bezeichnung der Ausführungsart eines (industriellen) Fabrikats benutzt, wie beispielsweise im Begriff „Modellreihe“ deutlich wird.

Als Nachbildungen eines realen oder imaginären Vorbilds spielen Modelle in verschiedensten Bereichen der Wissenschaft und Technik eine entschei-

dende Rolle. Gemeinsame Zielsetzung ist eine vereinfachende Erfassung der Struktur oder Funktionsweise eines Systems, die eine Untersuchung erleichtert oder erst ermöglicht. So nutzt die Physik beispielsweise Atommodelle, um den Aufbau und das Verhalten von Materie zu beschreiben. In den Finanzwissenschaften werden wahrscheinlichkeitstheoretische Modelle zur Analyse und Prognose von Kursentwicklungen eingesetzt. Ebenso werden in der Soziologie oder Medizin Modelle aufgestellt, um das Verhalten komplexer soziologischer bzw. biologischer Systeme zu erforschen.

Eine besonders zentrale Rolle spielt die Modellbildung in der Informatik, der Wissenschaft von der systematischen Darstellung, Verarbeitung und Übertragung von Informationen. Modelle werden gemäß vielfältiger Zielsetzungen in verschiedenen Teilgebieten verwendet. Sie bilden z.B. einen wesentlichen Bestandteil moderner Softwareentwicklungsprozesse („Vorgehensmodelle“) [Kru99, DW99] zur Beschreibung und Untersuchung der Struktur und des Verhaltens komplexer Software-Systeme sowie zur Definition der Entwicklungsprozesse selbst. Auch in der Softwaretechnik werden Modelle eingesetzt, beispielsweise das ISO OSI-Referenzmodell [ISO94] zur Spezifikation einer Protokollhierarchie für die Datenübertragung in Rechnernetzen.

Trotz der sehr vielfältigen Verwendungszwecke, für die Modelle genutzt werden, lassen sich Gemeinsamkeiten erkennen, die stets mit einer Modellbildung verbunden sind (vgl. [Cha92] und [Bom99]):

- Die Aufgabe eines Modells ist es, komplexe Sachverhalte zu strukturieren. Je nach der verfolgten Zielsetzung werden die relevanten Einflussfaktoren und ihre Zusammenhänge im Rahmen eines Modells erfasst, um so die Struktur oder Funktionsweise eines Systems durchschaubar zu machen. Dies erlaubt es, Situationen zu analysieren, die sich einer experimentellen Untersuchung entziehen, da sie zu komplex, zu zeitaufwändig, zu kostspielig oder schlicht unzugänglich sind.

- Das wesentliche Prinzip der Modellbildung ist die Abstraktion, also die vereinfachende Nachbildung eines realen oder imaginären Originals. Aus der Menge aller Eigenschaften eines betrachteten Systems werden nur gewisse, für die aktuelle Zielsetzung relevante Merkmale mit ihren grundlegenden Gesetzmäßigkeiten in einem Modell herauskristallisiert, während andere, dafür unmaßgebliche Eigenschaften vernachlässigt werden.

- Entsprechend ihrer vereinfachenden Annahmen besitzen Modelle stets nur eine eingeschränkte Gültigkeit gemäß der durch sie verfolgten Zielsetzung. Schließlich sollen Modelle komplexe Systeme nachbilden und ihre Untersuchung erleichtern, ohne die Komplexität vollständig zu

übernehmen. Zuweilen erfordert dies die Verwendung mehrerer, verschiedenartiger Modelle zur hinreichenden Analyse der Eigenschaften eines Systems.

Neben der geschilderten Verwendung des Modellbegriffs der Informatik als vereinfachende Nachbildung eines realen oder imaginären Originals, besitzt dieser auf dem Teilgebiet der Logik eine davon abweichende und spezifische Bedeutung. In diesem Zusammenhang wird als „Modell“ eine zu einem gegebenen System logischer Gesetzmäßigkeiten, welche als primär angesehen werden, passende „Struktur“ bezeichnet, die dieses erfüllt [Sch95, Lex01]. Im Rahmen der vorliegenden Arbeit wird der Begriff *Modell* jedoch stets in seiner erstgenannten, allgemeinen Bedeutung verwendet.

Nach der Einordnung des Modellbegriffs lässt sich der Begriff des *Produktmodells* präzisieren. Dieser ist in der Literatur in sehr unterschiedlichen Zusammenhängen anzutreffen. Auch im Sinne der vorliegenden Arbeit lassen sich verschiedene Definitionen finden. So heißt es in [And03]: „Ein Produktmodell ist die Abbildung eines Produktes in ein formales Modell. Es ist das Resultat des Produktentwicklungsprozesses, in dem alle relevanten Eigenschaften eines Produktes herausgearbeitet und im Produktmodell dokumentiert werden. Das Produktmodell entsteht durch Instanziierung des Produktdatenmodells“. In [Dyl02] „wird ein Produktmodell als Informationsmodell verstanden, welches alle im jeweiligen Kontext relevanten Informationen eines Produkts abbildet. Es ist aus formal beschriebenen Modellschemata, inklusive Definitionen der beschriebenen Sachverhalte, aufgebaut. Dadurch werden Datenstrukturen festgelegt, mit denen individuelle Produkte im Rechner abgebildet werden können“. [WD02] sieht ein Produktmodell als Vereinigung von „Merkmalen“, welche „Struktur und Gestalt des Produkts“ beschreiben und durch einen „Produktentwickler direkt festgelegt werden“ können, sowie „Eigenschaften“, die davon abgeleitet ein „Verhalten des Produkts“ ohne Einflussmöglichkeit eines Produktentwicklers definieren.

In dieser Arbeit wird unter *Produktmodell* ein formalisiertes Modell eines (technischen) Produkts verstanden, welches eine rechnergestützte Bearbeitung im Sinne von Abschnitt 2.1.3 erlaubt. Es sei zum einen darauf hingewiesen, dass diese Festlegung verschiedene Arten einer Formalisierung zulässt. Eine Möglichkeit der Formalisierung auf mathematischer Grundlage wird in Kapitel 5 aufgezeigt. Zum anderen ist festzuhalten, dass der Begriff des Produktmodells im Gegensatz zu manchen Ansätzen nicht im eingrenzenden Sinne *einer* spezifischen Fachdisziplin, beispielsweise der geometrischen Konstruktion, der Thermodynamik oder Aerodynamik, gebraucht wird. Darüber hinaus findet sich auch häufig der Begriff *integriertes Produktmodell* (s. z.B. [GAP93] oder [Ben01]). Darunter wird ein Produktmo-

dell verstanden, welches gemäß Abschnitt 3.2 sämtliche relevante Produktinformation über verschiedene Phasen des Produktlebenszyklus oder – eingeschränkt – der Produktentwicklung (vgl. Abschnitt 2.1.1) hinweg erfasst. Im Rahmen der vorliegenden Arbeit, insbesondere in den folgenden Abschnitten, wird der Begriff Produktmodell im Sinne eines integrierten Produktmodells verwendet.

Voraussetzung für die Erstellung eines Modells ist eine *Modellierungstechnik*. Diese besteht aus abstrakten *Modellierungskonzepten*, die in Modellen ausgedrückt werden sollen, aus welchen konkrete *Modellierungskonstrukte* zum konstruktiven Aufbau eines Modells abgeleitet sind. So ist beispielsweise „Strukturerfassung“ ein Modellierungskonzept, welches die Abbildung der Struktur eines betrachteten Gegenstands zur Zielsetzung hat und aus dem sich verschiedene Modellierungskonstrukte ergeben können, etwa die Konstrukte „Bestandteil“ und „Beziehung“ zur Modellierung unterscheidbarer Einheiten und Verbindungen zwischen diesen. In ihrer Gesamtheit bilden die Modellierungskonstrukte das *Metamodell*. Dieser, in jüngerer Zeit zunehmend an Verbreitung gewinnende Begriff, bezeichnet in allgemeiner Bedeutung ein Modell, das der Spezifikation von Modellen dient. Sollen Modelle technischer Produkte beispielsweise deren Bestandteile, Beziehungen zwischen diesen und elementare Produkteigenschaften darstellen, so bilden die Modellierungskonstrukte „Bestandteil“, „Beziehung“ und „Eigenschaft“ das Metamodell, während konkrete Ausprägungen („Instanzen“) davon, wie z.B. „Zahnrad“, „Schraubverbindung“ und „Gewicht“, ein Modell definieren. Eine weiter gehende Einführung in die Thematik der Metamodellierung findet sich in [Jec00] und [Str98].

Im Rahmen einer Modellierungstechnik ist für jedes Modellierungskonstrukt des Metamodells eine *Syntax* und eine *Semantik* festzulegen. Die Syntax, auch *Beschreibungstechnik* genannt, definiert eine Notation zur Darstellung von Modellierungskonstrukten und somit von Modellen. Hierbei kann zwischen textueller und graphischer Syntax unterschieden werden. Während erstere Modellierungskonstrukte durch eine Folge von Zeichen eines zugrunde liegenden Alphabets darstellt, welche nach vorgegebenen Regeln aneinander gereiht werden können, beschreiben die vor allem in jüngerer Zeit Verbreitung findenden graphischen Beschreibungstechniken Modelle durch graphische Notationen in Diagrammen. Auf der Grundlage der Syntax, welche die Möglichkeit bietet, Modellierungskonstrukte auf definierte Weise darzustellen, legt die Semantik ihre inhaltliche Bedeutung fest. Dies kann auf verschiedene Arten erfolgen: So finden sich häufig informelle, textuell gegebene Definitionen zur Festlegung der Semantik des Metamodells einer Modellierungstechnik. Eine andere Möglichkeit ist es, die Semantik mithilfe eines präzisen, mathematisch fundierten Kalküls zu definieren.

4.2 Anforderungen

Nach der Einordnung grundlegender Begriffe lassen sich die Anforderungen formulieren, die an eine adäquate Modellierungstechnik für ein (integriertes) Produktmodell zu stellen sind. Hierbei sind sowohl fachliche Anforderungen zu betrachten, die beispielsweise die Ausdruckskraft der Modellierungstechnik betreffen, als auch technische Anforderungen in Hinblick auf eine Verarbeitung des Produktmodells mithilfe moderner Rechenanlagen.

Expressivität und Angemessenheit

Die Ausdruckskraft der Modellierungstechnik ist offenbar von entscheidender Bedeutung für den Erfolg des gewählten Ansatzes. Das Metamodell muss die erforderlichen Modellierungskonstrukte zur Verfügung stellen, um alle gewünschten Aspekte technischer Produkte im Rahmen eines Modells hinreichend wiedergeben zu können. Insbesondere zählen hierzu die Erfassung der Produktstruktur, also der Unterteilung eines Produkts in unterscheidbare Bestandteile und Beziehungen zwischen diesen, sowie die Modellierung (elementarer) Produkteigenschaften, z.B. geometrische Abmessungen, verwendete Materialien oder Kostenaspekte.

Ebenso ist darauf zu achten, dass die gewählten Modellierungskonstrukte dem Anwendungsbereich angepasst sind. So muss es Anwendern möglich sein, im Rahmen ihrer Tätigkeiten erforderliche Modelle des zu entwickelnden Produkts mithilfe verständlicher und möglichst elementarer Modellierungskonstrukte auf einfache Weise zu erstellen. Das Metamodell ist daher auf eine überschaubare und handhabbare Anzahl an Elementen zu beschränken. Besonders zu berücksichtigen sind bewährte Ansätze existierender Modellierungstechniken der Informatik, beispielsweise des Entity-Relationship-Modells [Che76] oder objektorientierter Techniken [UML03].

Unterstützung von Produktsichten

Wie in Abschnitt 2.1.3 dargelegt, können sich die an einem Entwicklungsprozess beteiligten Bearbeiter gemäß ihrer Rolle in ihrem Verständnis des zu entwickelnden Produkts signifikant unterscheiden. So können je nach Art der durchzuführenden Entwicklungsaktivität sehr verschiedenartige Aspekte des gemeinsamen Produkts im Vordergrund stehen. Einzelne Anwender oder Anwendergruppen benötigen die Möglichkeit, individuelle *Sichten* auf ein Produkt zu definieren, um ausgewählte und im aktuellen Kontext relevante Eigenschaften aus ihrer spezifischen Perspektive besonders hervorzuheben und optimal zu repräsentieren.

An eine Modellierungstechnik besteht daher die explizite, weit reichende und für den gewählten Anwendungsbereich besonders charakteristische Anforderung, individuelle und gleich berechtigte Produktsichten im Rahmen eines integrierten Produktmodells zu ermöglichen. Insbesondere soll jede Produktsicht abhängig von den aktuell maßgeblichen Erfordernissen über eine eigene Strukturierung des Produkts verfügen können, ohne sich an einer übergreifenden oder in einem anderen Kontext relevanten Produktstrukturierung orientieren zu müssen. Gleiches gilt für die innerhalb einer Sicht betrachteten Produkteigenschaften.

Unterstützung von Abstraktionsebenen

Die Verwendung verschiedener Abstraktionsebenen ist ein in Modellierungstechniken häufig anzutreffendes Prinzip. Allgemein betrachtet erfolgt die Modellierung eines Gegenstandes hierbei in mehreren Schritten und somit Modellen, die zueinander in einer Verfeinerungsbeziehung stehen. Ein bekanntes Beispiel für den Einsatz von Abstraktionsebenen ist die Trennung zwischen abstrakten „Typen“ und konkreten „Instanzen“, wie sie sich in gängigen Modellierungs- oder Programmiersprachen findet. Als wesentlicher Vorteil dieser Vorgehensweise ergibt sich die Wiederverwendbarkeit abstrakt spezifizierter Modelle durch Belegung der vorgesehenen Freiheitsgrade.

Auch in einer Modellierungstechnik für integrierte Produktmodelle soll dieses bewährte Prinzip genutzt werden. So sind zwei Abstraktionsebenen vorzusehen, um zum einen die wohldurchdachte, stets gleich bleibende und somit wieder verwendbare Struktur eines Produkts zu erfassen. Auf der zweiten Abstraktionsebene sollen dagegen die eigentlichen Modelle des zu entwickelnden Produkts als konkrete Ausprägungen davon angesiedelt sein. Neben Aspekten der Wiederverwendbarkeit eröffnet dieser Ansatz nicht zuletzt einen Ausgangspunkt für die Weiterentwicklung oder Variantenbildung eines Produkts.

Durchgängigkeit und Einheitlichkeit

Entsprechend seiner Definition muss ein *integriertes* Produktmodell darauf ausgerichtet sein, Produktinformationen aus allen Phasen des Entwicklungsprozesses zu erfassen. Folgerichtig ist das Metamodell so zu gestalten, dass eine Modellierung aller im Entwicklungsprozess relevanten Produktaspekte mithilfe der angebotenen Modellierungskonstrukte möglich ist, so dass eine durchgängige Produktentwicklung auf der Grundlage eines Produktmodells gewährleistet ist. Die ausgewählten Modellierungskonstrukte sollen daher hinreichend ausdrucksstark sein, um den Erfordernissen sämtlicher Fach-

disziplinen und Interessensgruppen gerecht zu werden.

Dennoch ist darauf zu achten, dass die konzipierte Modellierungstechnik Produktinformationen über alle Phasen des Entwicklungsprozesses hinweg auf einheitliche Weise erfasst. So sind die benötigten Modellierungskonstrukte möglichst allgemein und somit vielfältig verwendbar zu wählen, um eine aus der Sicht des Metamodells gleichartige Erfassung verschiedener Produktaspekte zu erreichen. Das Metamodell muss also einerseits hinreichend ausdrucksstark sein, um eine durchgängige Modellierung zu ermöglichen, ohne andererseits spezifische, nur im Kontext einer Interessensgruppe sinnvolle Modellierungskonstrukte zu enthalten.

Unabhängigkeit und Kompatibilität

Ein wesentliches Defizit heutiger Entwicklungsprozesse ist die Konzentration auf eine proprietäre, meist Werkzeug-spezifische Modellbildung (vgl. Abschnitt 2.1.3), die zu zahlreichen, häufig inkompatiblen Teilmodellen eines zu entwickelnden Produkts führt. So ist ein Wechsel oder die Weiterentwicklung eines verwendeten Software-Werkzeugs mit aufwändigen und umfangreichen Änderungen an dadurch erstellten Modellen verbunden, ohne dass Auswirkungen auf die damit verbundene Produktinformation beabsichtigt sind. Aus diesem Grunde besteht die Anforderung einer übergreifend verwendbaren und insbesondere Werkzeug-unabhängigen Modellierungstechnik.

Jedoch ist bei der Auswahl der Modellierungskonstrukte darauf zu achten, dass die erstellten Modelle prinzipiell kompatibel und somit transformierbar sind, sowohl hinsichtlich allgemeiner, im Bereich technischer Produkte verwendeter Standards und Austauschformate wie UML [UML03] oder STEP [STE94], wie auch bezüglich spezifischer, innerhalb einzelner Fachdisziplinen verbreiteter Modellierungsarten. Zusammenfassend besteht somit die Zielsetzung, Produktmodelle unabhängig von proprietären Formaten und Technologien zu erstellen, um sie als breite Integrationsgrundlage für verschiedenste Software-Werkzeuge nutzen zu können.

Umsetzbarkeit und Handhabbarkeit

Ein integriertes Produktmodell stellt als Grundlage zur Anbindung verschiedenster Software-Werkzeuge einen wesentlichen Bestandteil einer übergreifenden Integrationsplattform dar, wie sie in Abschnitt 3.4 beschrieben ist. In diesem Rahmen besteht die Anforderung einer Software-technischen Umsetzbarkeit des Produktmodells auf der Grundlage zeitgemäßer, beispielsweise objektorientierter oder komponentenbasierter Technologien. Diese sind bereits bei der Konzeption der Modellierungstechnik zu berücksichtigen, so

dass eine einfache und nach Möglichkeit unmittelbare Umsetzung der entwickelten Konzepte gewährleistet ist.

Nicht zuletzt sollte bei der Konzeption des Produktmodells auf seine Handhabbarkeit besonders geachtet werden, um gemäß der häufigsten Anwendungsfälle einen methodischen und nach Möglichkeit einfachen Umgang mit dem Produktmodell zu erreichen. Schließlich stellt das Produktmodell nicht ein statisches Abbild eines Produkts dar, sondern ist im Gegenteil im Laufe eines Produktentwicklungsprozesses zahlreichen und vielfältigen Bearbeitungen unterworfen. So entspricht beispielsweise der vielfach zu erwartenden Tätigkeit der Modellerstellung oder -änderung die Forderung nach einem überschaubaren und verständlichen Metamodell.

4.3 Basiskonzepte

Die beschriebenen Anforderungen ergeben sich aus der Zielsetzung eines integrierten Produktmodells, welches einen wesentlichen Bestandteil einer übergreifenden Integrationsplattform zur Überwindung der Defizite heutiger Produktentwicklungsprozesse darstellt (vgl. Kapitel 2 und Abschnitt 3.4). Vor diesem Hintergrund werden in den folgenden Abschnitten die vier Basiskonzepte *Segmentierung*, *Strukturierung*, *Typisierung* und *Attributierung* als grundlegende Modellierungsprinzipien eines integrierten Produktmodells entwickelt. Aus diesen lassen sich schließlich konkrete Modellierungskonstrukte ableiten, welche in ihrer Gesamtheit das angestrebte Metamodell bilden.

Im Überblick betrachtet bietet das Prinzip der *Segmentierung* einen Lösungsansatz, um verschiedene Sichten auf ein technisches Produkt im Rahmen eines integrierten Produktmodells zu unterstützen. Besondere Beachtung findet ein enges Zusammenspiel zwischen Produktmodellen einerseits und übergreifenden Entwicklungsprozessen mit vielfältigen Aktivitäten und zahlreichen Bearbeitern und Interessensgruppen andererseits. Im Rahmen der *Strukturierung* wird untersucht, wie sich innerhalb einer gewählten Sicht die Struktur eines technischen Produkts, bestehend aus einzelnen Bauteilen und Verbindungen zwischen diesen, modellieren lässt.

Das vielfach bewährte Konzept der *Typisierung* adressiert die Unterstützung verschiedener Abstraktionsebenen innerhalb eines Produktmodells. Auf diese Weise wird eine Trennung zwischen abstrakt gehaltenen Typdefinitionen für allgemeine Bestandteile technischer Produkte und konkreten Ausprägungen davon erreicht und die Wiederverwendbarkeit einmal erstellter Modelle erzielt. Das Konzept der *Attributierung* wendet sich schließlich der Beschreibung elementarer Eigenschaften der Bestandteile technischer Produkte zu. Als Grundbausteine des Produktmodells bilden diese die Freiheits-

grade genutzter Typdefinitionen.

Das aus den Basiskonzepten entwickelte Metamodell umfasst die Gesamtheit der Modellierungskonstrukte zur Darstellung technischer Produkte und stellt somit ein Modell zur Beschreibung integrierter Produktmodelle dar. Die Elemente eines Produktmodells ergeben sich daraus durch konkrete Ausprägungen der Modellierungskonstrukte als Instanzen des Metamodells. Analog ist auch zur Beschreibung der Elemente des Metamodells eine entsprechende Modellierungstechnik erforderlich, die ein – aus Sicht des Produktmodells – Meta-Metamodell anbietet, welches seinerseits wiederum geeignet zu spezifizieren ist.

Die beschriebene Kette an Modellen und zu ihrer Spezifikation erforderlichen Metamodellen ließe sich beliebig fortsetzen. Es ist jedoch ausreichend, ein Metamodell einzuführen, das über hinreichend ausdrucksstarke Modellierungskonstrukte verfügt, um sich selbst zu beschreiben. So steht beispielsweise mit dem Standard *Meta Object Facility (MOF)* [MOF02] der *Object Management Group (OMG)* [OMG04] eine solche Modellierungstechnik zur Verfügung. Diese ist allgemein verwendbar, um im Rahmen eines vierstufigen Ansatzes Metamodelle beliebiger Anwendungsbereiche und somit Modelle und ihre Instanzen zu beschreiben.

Eine weitere Möglichkeit ist es, sich zur Definition des Metamodells auf eine Modellierungstechnik zu beziehen, die als bekannt vorausgesetzt werden kann und nicht weiter spezifiziert werden muss. Im Rahmen der vorliegenden Arbeit wird daher zur Beschreibung der Modellierungskonstrukte des angestrebten Metamodells die in Wissenschaft und Praxis weit verbreitete *Unified Modeling Language (UML)* [UML03, BRJ99, RJB99, Oes04] eingesetzt. Das Metamodell der UML stellt eine Ausprägung des Meta-Metamodells der *Meta Object Facility* dar und kann zur Erstellung von Modellen – und somit auch Metamodellen – genutzt werden.

Ursprünglich ist die Unified Modeling Language seitens der Object Management Group zur Modellierung (objektorientierter) Software-Systeme entwickelt worden. Sie wird mittlerweile jedoch erfolgreich auch in anderen Anwendungsgebieten eingesetzt. Die UML bietet verschiedene Diagrammarten, beispielsweise Klassen- oder Sequenzdiagramme, um verschiedene Aspekte eines (Software-)Systems graphisch zu beschreiben. Insbesondere Klassendiagramme sind gut geeignet, um die Modellierungskonstrukte des angestrebten Metamodells für technische Produkte zu erfassen. Eine formale Fundierung der Semantik wird schließlich in Kapitel 5 gegeben.

4.3.1 Segmentierung

Gemäß dem Prinzip der virtuellen Produktentwicklung (vgl. Abschnitt 2.1.3) stellt ein Modell des betreffenden Produkts in aller Regel die Voraussetzung für die Verwendung von Software-Werkzeugen dar. Gerade in stark interdisziplinär geprägten Entwicklungsprozessen mit zahlreichen Aktivitäten und Bearbeitern ist ein solches Produktmodell jedoch nicht universell gültig, also in jedem Kontext sinnvoll und von allen Beteiligten gleichermaßen akzeptiert. Vielmehr können je nach Art des Produkts, dem Zweck seiner Bearbeitung, dabei verwendeten Software-Werkzeugen und insbesondere dem Verständnis beteiligter Entwickler sehr verschiedenartige Produktaspekte im Vordergrund stehen. Im Rahmen der vorliegenden Arbeit wird ein in diesem Sinne festgelegtes, individuell angepasstes Modell eines betrachteten Produkts als *Produktsicht* – oder kurz *Sicht* – bezeichnet. Entsprechend der hohen Komplexität technischer Produkte sowie der Vielfältigkeit ihrer Entwicklungsprozesse muss es jeder Produktsicht gestattet sein, eine eigene Strukturierung des betreffenden Produkts vorzunehmen sowie nur ausgesuchte, im aktuellen Kontext relevante Eigenschaften aus der Menge aller Produkteigenschaften zu betrachten.

Die Unterstützung von Produktsichten stellt eine wesentliche und gerade im Bereich komplexer technischer Produkte charakteristische Anforderung an ein integriertes Produktmodell dar. Dieses muss die Erstellung, Bearbeitung und Verwaltung von Produktsichten explizit vorsehen und unterschiedliche Sichten an zentraler Stelle zusammenführen. Wie bereits in Abschnitt 3.2 angedeutet, sind hierfür zwei unterschiedliche Integrationsansätze zu beobachten: Der im Folgenden als *enge Integration* bezeichnete Ansatz, auch *enge Kopplung* genannt, hat eine Integration auf der Ebene des Produktmodells zum Ziel. Hierbei sollen alle relevanten Produktinformationen durch ein einziges, allen Bearbeitern gemeinsames Modell erfasst werden, während Produktsichten darauf beschränkt sind, dieses verschiedenartig darzustellen. Dagegen versucht der im Folgenden *lose Integration* genannte und auch als *lose Kopplung* bezeichnete Ansatz eine Integration auf der Ebene des Metamodells. Produktsichten treten dabei als eigenständige Modelle bezüglich eines gemeinsamen Metamodells auf, die über explizite fachliche Querbezüge miteinander verbunden sind.

In den folgenden Abschnitten werden die genannten Alternativen der Produktmodellintegration ausführlich vorgestellt und ihre Einbettung in einen Produktentwicklungsprozess aufgezeigt. Insbesondere werden die individuellen Vorteile und Nachteile der betrachteten Ansätze ausführlich beleuchtet. Vor dem Hintergrund des betrachteten Anwendungsbereichs erfolgt schließlich eine Entscheidung für das angestrebte Metamodell.

Enge Integration

Die enge Integration stellt einen konventionellen und nahe liegenden Integrationsansatz der virtuellen Produktentwicklung dar. Ihre wesentliche Zielsetzung ist die Erfassung sämtlicher Produktinformationen aus verschiedenen Phasen des Produktentwicklungsprozesses durch ein einziges, übergreifendes, allen Bearbeitern gemeinsames und über alle Entwicklungsschritte hinweg verwendetes Produktmodell.

Produktsichten bestehen dagegen aus angepassten Ausschnitten dieses Modells, welche ausgesuchte Elemente geeignet kombiniert darstellen. Sie sind somit nicht eigenständig, so dass jede Änderung einer Sicht unmittelbar zu Änderungen der dadurch betroffenen Elemente des zentralen Produktmodells sowie ggf. weiterer Sichten führt, die sich auf diese Elemente beziehen. Abbildung 4.1 illustriert dieses Prinzip.

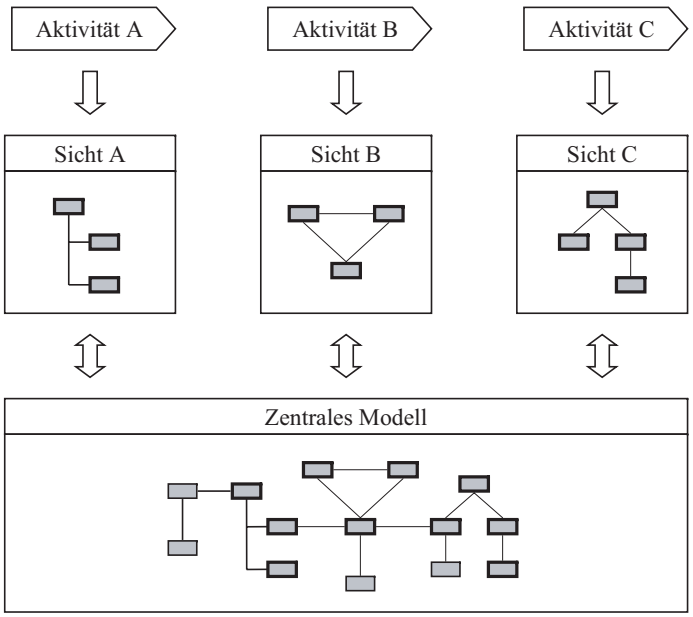

Abbildung 4.1: Prinzip der engen Integration

Im Rahmen einer übergreifenden Integrationsplattform ist die Komponente *Modellverwaltung* für den Zugriff auf das Produktmodell und Produktsichten verantwortlich (vgl. Abschnitt 3.4.2). Vor diesem Hintergrund skizzieren die folgenden Schritte die Erstellung und Bearbeitung eines Produktmodells sowie zugehöriger Produktsichten gemäß dem Paradigma der engen Integration.

1. Modellerstellung

Als erster Schritt erfolgt zu Beginn eines Produktentwicklungsprozesses der Aufbau eines zentralen Produktmodells. Dieses bietet die Integrationsgrundlage, um die im Verlauf verschiedener Entwicklungsphasen erarbeiteten Produktmerkmale zu erfassen und durch Produktsichten geeignet zu repräsentieren. Die Erstellung des zentralen Modells erfolgt mittels der Modellverwaltung, wie in Abbildung 4.2 dargestellt.

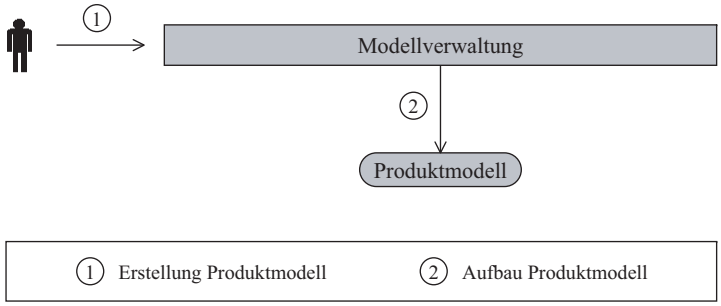

Abbildung 4.2: Modellerstellung

2. Sichtenerstellung

Auf der Grundlage des zentralen Modells lassen sich in einem zweiten Schritt verschiedene Produktsichten erstellen. Hierzu kann für eine im aktuellen Kontext relevante Auswahl der Elemente des zentralen Modells eine geeignete Repräsentation als eigene Produktsicht definiert werden. Für den Aufbau von Produktsichten ist wiederum die Modellverwaltung zuständig (s. Abbildung 4.3).

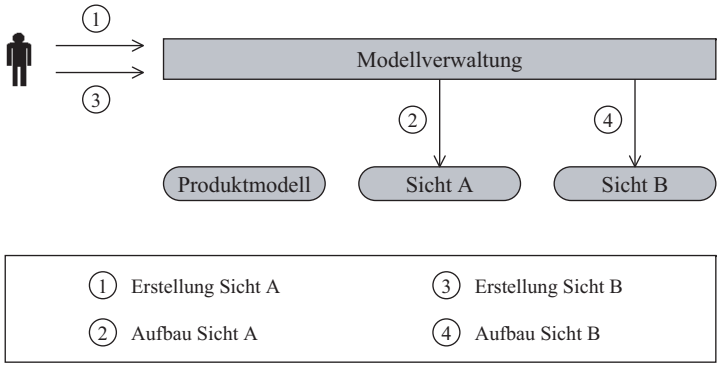

Abbildung 4.3: Sichtenerstellung

3. Modellbearbeitung

Schließlich erfolgt die sukzessive Bearbeitung des Produktmodells. Dazu hat ein Entwickler die Möglichkeit, Änderungen an der im aktuellen Kontext relevanten Produktsicht vorzunehmen. Gemäß dem Prinzip der engen Integration soll jede Modifikation einer Produktsicht unmittelbar zu einer Aktualisierung der betroffenen Elemente des Produktmodells sowie aller weiteren Produktsichten führen.

In Anlehnung an die als *Model-View-Controller* [KP88] bzw. *Observer* [GHJV95] bezeichneten Entwurfsmuster lässt sich dieses Verhalten wie in Abbildung 4.4 dargestellt erreichen: Jede gewünschte Änderung einer Produktsicht wird seitens eines Entwicklers der Modellverwaltung mitgeteilt, welche daraufhin die betroffenen Elemente des Produktmodells ermittelt und diese entsprechend modifiziert.

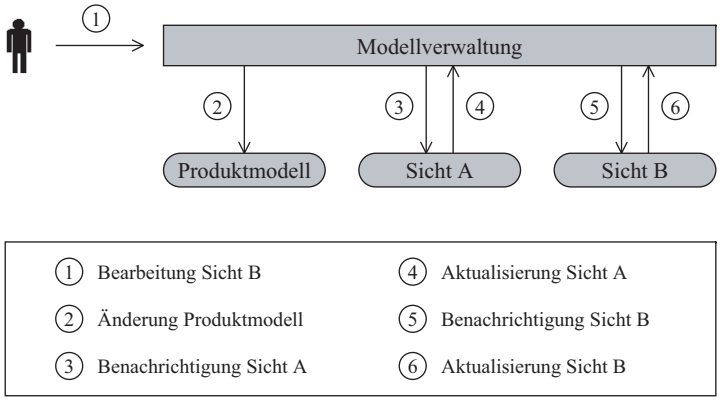

Abbildung 4.4: Modellbearbeitung

Anschließend informiert die Modellverwaltung alle vorhandenen Produktsichten darüber, dass eine Änderung des Produktmodells eingetreten ist. Diese können nun ihrerseits, wiederum unter Nutzung der Modellverwaltung, reagieren und eine entsprechende Aktualisierung veranlassen. Insgesamt lässt sich auf diese Weise ein stets aktuelles und konsistentes Gesamtmodell gewährleisten.

Im Falle der engen Integration besteht das integrierte Produktmodell aus dem zentralen Modell sowie darauf definierten Produktsichten. Darüber hinaus gehende Modelle des zu entwickelnden Produkts sind in diesem Ansatz nicht zugelassen. Folgerichtig nimmt jede Aktivität des Entwicklungsprozesses gemäß dem Prinzip der Produktintegration (vgl. Abschnitt 3.2) unmittelbar oder unter der Verwendung einer Produktsicht mittelbar Bezug auf das

zentrale Modell. Gleiches gilt für die im Entwicklungsprozess zur Manipulation des Produktmodells eingesetzten Software-Werkzeuge. Daraus ergeben sich die folgenden Vorteile der engen Integration:

- Ein übergreifendes, zentrales Modell dient allen Beteiligten eines Produktentwicklungsprozesses als gemeinsamer Bezugspunkt. Die übergreifende Festlegung einer Produktstruktur sowie der im Modell vertretenen Produkteigenschaften fördert ein gleichartiges Verständnis des Produkts und reduziert somit den Kommunikationsaufwand und Abstimmungsbedarf.

- Die Erfassung aller relevanten Produktinformationen im Rahmen *eines* zentralen Modells erlaubt eine Vermeidung von *Redundanz*, also dem mehrfachen Auftreten derselben Produktinformation an verschiedenen Stellen. Auf diese Weise ergibt sich implizit die *Konsistenz* eines zentralen Modells, da eine Produktinformation stets nur einmal hinterlegt ist.

- Ein deutlicher Vorteil der engen Integration ist die methodische Klarheit dieses Ansatzes: über die verschiedenen Phasen der Produktentwicklung hinweg werden alle relevanten Produktinformationen in einem zentralen Modell zusammengeführt, welches somit eine klar definierte Grundlage für die Prozessintegration, also die Koordination erforderlicher Aktivitäten und daran beteiligter Bearbeiter bildet.

- In gleicher Weise vorteilhaft ist die enge Kopplung hinsichtlich der Werkzeugintegration: So reduziert sich die Anbindung der im Produktentwicklungsprozess benötigten Software-Werkzeuge auf die Schaffung geeigneter Schnittstellen zu dem zentralen Modell. Dessen implizit gegebene Konsistenz gewährleistet darüber hinaus ein reibungsloses Zusammenspiel verwendeter Werkzeuge.

Dagegen lassen sich folgende Nachteile und Schwierigkeiten dieses Ansatzes nennen:

- Ein zentrales Modell erfordert die Einigung aller Beteiligten eines Produktentwicklungsprozesses auf *ein* universell gültiges Modell mit einer übergreifenden Strukturierung und einer verbindlichen Auswahl im Modell vertretener Produkteigenschaften. Eine so weit reichende Standardisierung ist jedoch gerade im Bereich komplexer technischer Produkte weit entfernt.

- Die enge Integration vereint sämtliche Produktinformationen aus verschiedenen Phasen der Produktentwicklung in einem zentralen Modell. Ein solches ist folgerichtig sehr umfangreich, unübersichtlich und durch ein hohes Maß an Komplexität gekennzeichnet. Entsprechend negativ sind die Auswirkungen dieses Ansatzes auf die Verständlichkeit, Handhabbarkeit und Wartbarkeit des integrierten Produktmodells.

- Besonders kritisch ist der Ansatz eines zentralen Modells in Hinblick auf dessen Modifizierbarkeit, insbesondere im Sinne einer evolutionären Weiterentwicklung zu sehen: In diesem Falle sind neu hinzukommende Produktaspekte und Produkteigenschaften in ein sehr komplexes und unübersichtliches Modell einzuarbeiten, ohne bereits vorhandene, übergreifende Modellelemente negativ zu beeinflussen.

- Die enge Kopplung von zentralem Modell und Produktsichten minimiert die Redundanz und gewährleistet so implizit die Konsistenz des integrierten Produktmodells. Bezüglich einer nebenläufigen oder experimentellen Produktentwicklung ist diese Form der Konsistenzsicherung jedoch sehr inflexibel, da jede Änderung des zentralen Modells unmittelbar eine Aktualisierung aller Produktsichten nach sich zieht.

Die genannten Nachteile der engen Integration lassen sich durch den Ansatz der losen Integration überwinden, der im Folgenden vorgestellt wird.

Lose Integration

Die lose Integration stellt eine Alternative zur engen Integration dar, die im höheren Maße dem heutigen Stand der rechnergestützten Produktentwicklung entspricht. Dieser Ansatz verzichtet auf ein zentrales Modell und fasst Produktsichten als eigenständige und zunächst unabhängige Partialmodelle eines Produkts auf, welche jedoch auf der Grundlage einer gemeinsamen Modellierungstechnik erstellt sind.

Vor dem Hintergrund eines gemeinsamen Produkts existieren in der Regel zahlreiche und vielfältige Querbezüge, Abhängigkeiten und Überschneidungen zwischen Elementen verschiedener Produktsichten. Daraus ergibt sich eine Reihe an Bedingungen, die über die *Konsistenz* der Partialmodelle entscheiden, also bestimmen, ob diese zueinander widerspruchsfrei sind und somit insgesamt ein potenziell reales Produkt beschreiben.

Der Ansatz der losen Integration sieht ein integriertes Produktmodell somit als eine Reihe eigenständiger Partialmodelle auf der Grundlage einer übergreifenden Modellierungstechnik in Kombination mit einer endlichen Anzahl explizit erfasster *Konsistenzbedingungen*, die Beziehungen zwischen

diesen festlegen. Abbildung 4.5 veranschaulicht dieses Prinzip anhand dreier Produktsichten mit exemplarischen Beziehungen.

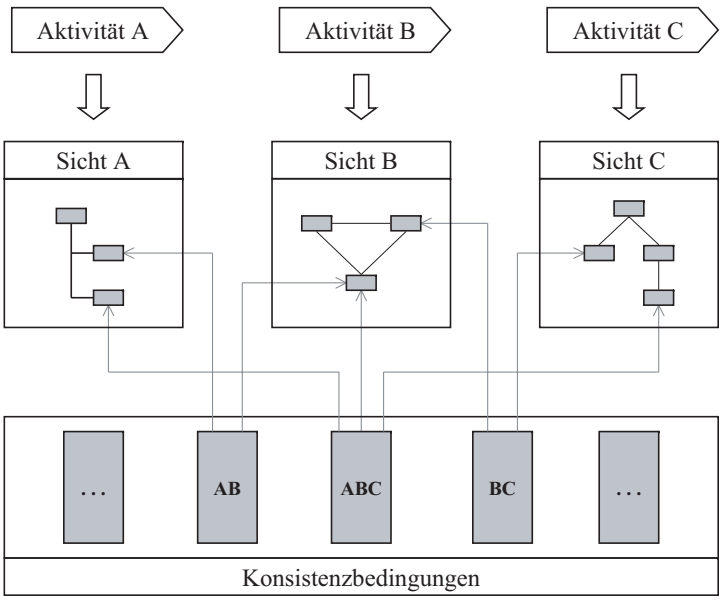

Abbildung 4.5: Prinzip der losen Integration

Im Rahmen einer übergreifenden Integrationsplattform ist wiederum die Komponente *Modellverwaltung* für die Erstellung und Bearbeitung von Produktsichten verantwortlich. Zudem hat sie die Aufgabe, die Definition und Prüfung von Konsistenzbedingungen zu ermöglichen. Entsprechend zeigen die folgenden Schritte die Bearbeitung eines Produktmodells gemäß dem Ansatz der losen Integration.

1. *Sichtenerstellung*

   Die Erstellung der gewünschten Produktsichten stellt den ersten Schritt der Produktentwicklung dar. Im Gegensatz zur engen Integration repräsentieren diese eigenständige und voneinander unabhängige Partialmodelle auf der Grundlage einer gemeinsamen Modellierungstechnik. Wie in Abbildung 4.6 dargestellt, wird die hierfür erforderliche Funktionalität Entwicklern seitens der Modellverwaltung angeboten.

2. *Definition der Konsistenzbedingungen*

   Nach der Festlegung relevanter Produktsichten lassen sich Querbezüge, Abhängigkeiten und Überschneidungen zwischen Modellelementen mittels

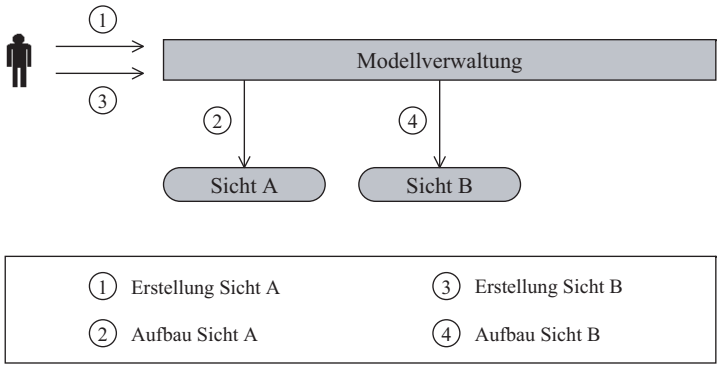

Abbildung 4.6: Sichtenerstellung

der Modellverwaltung in Form expliziter Konsistenzbedingungen in das Produktmodell einbringen (s. Abbildung 4.7). Auf diese Weise werden die Elemente definierter Produktsichten flexibel zueinander in Beziehung gesetzt.

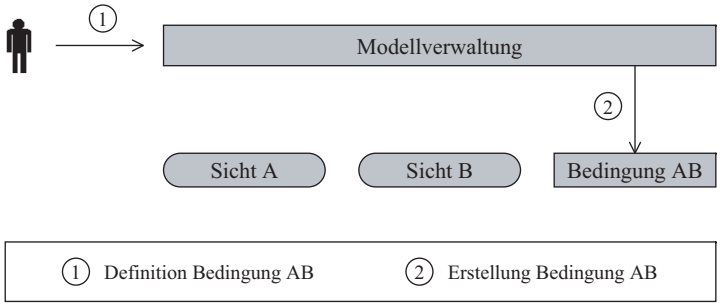

Abbildung 4.7: Definition der Konsistenzbedingungen

3. Prüfung der Konsistenz

Im Gegensatz zur engen Integration lassen sich Produktsichten im Laufe des Entwicklungsprozesses unabhängig voneinander bearbeiten. Gleichzeitig ist es zu jedem Zeitpunkt der Produktentwicklung möglich, auf der Grundlage der festgelegten Konsistenzbedingungen zu prüfen, ob Produktsichten zueinander konsistent sind, also ein potenziell reales Produkt beschreiben.

Für die Durchführung einer Konsistenzprüfung wendet sich ein Entwickler, wie in Abbildung 4.8 dargestellt, an die Modellverwaltung, welche

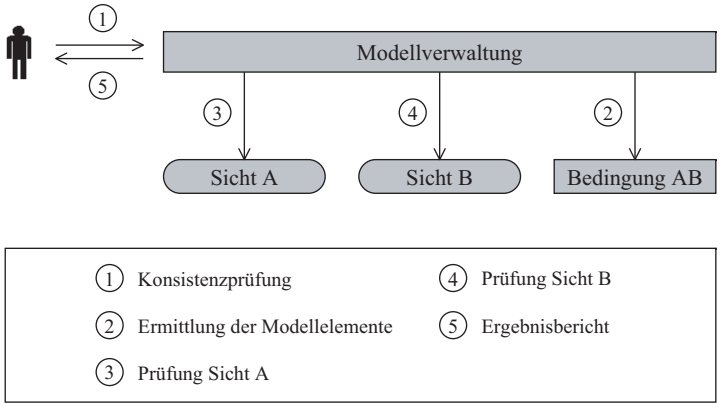

Abbildung 4.8: Konsistenzprüfung

ihrerseits aus einer festgelegten Konsistenzbedingung die betroffenen Sichten und Modellelemente ermittelt und diese inspiziert. Das Ergebnis der Konsistenzprüfung und eine Beschreibung ggf. auftretender Inkonsistenzen wird schließlich dem Bearbeiter mitgeteilt.

4. Sicherung der Konsistenz

Schließlich sind vorhandene Inkonsistenzen, wie sie durch eine Prüfung des Gesamtmodells ermittelt werden, individuell zu beheben. Je nach betrachteter Konsistenzbedingung, dem Ausmaß der Inkonsistenz und seiner Ursache können hierzu automatisierte Verfahren genutzt werden oder manuelle Änderungen an mindestens einer der betroffenen Produktsichten erforderlich sein (s. Abbildung 4.9).

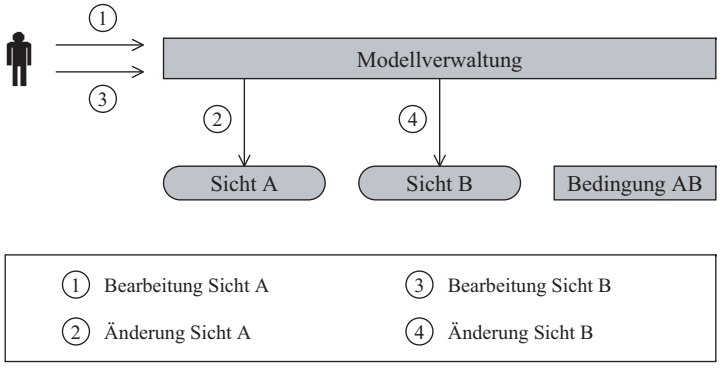

Abbildung 4.9: Konsistenzsicherung

Die lose Integration begreift das integrierte Produktmodell als eine Reihe eigenständiger, unabhängiger Partialmodelle in Verbindung mit explizit vorgegebenen Bedingungen, anhand derer die Konsistenz definierter Produktsichten geprüft werden kann. Über die Verwendung einer übergreifenden Modellierungstechnik hinaus macht dieser Ansatz keine Vorgaben, weder über die Art oder den Aufbau festgelegter Produktsichten, noch hinsichtlich einer Vorgehensweise zur Konsistenzprüfung und Konsistenzsicherung. Dementsprechend erlaubt die lose Integration ein hohes Maß an Flexibilität gemäß folgender Vorteile dieses Ansatzes:

- Im Gegensatz zur engen Integration, welche die Zielsetzung verfolgt, alle Produktaspekte in einem zentralen Modell zu vereinen, können sich voneinander unabhängige Partialmodelle auf ausgesuchte Aspekte eines technischen Produkts konzentrieren. Folgerichtig lässt sich die Komplexität auftretender Modelle entscheidend reduzieren zu Gunsten ihrer Übersichtlichkeit und Verständlichkeit.

- In einem zentralen Modell werden alle relevanten Produktinformationen an einer Stelle zusammengefasst, so dass verschiedene Produktaspekte aufeinander abzustimmen sind. Dagegen erlaubt es die lose Integration, ausgesuchte Produktaspekte in Form eigenständiger Partialmodelle optimal zu repräsentieren und auf die Erfordernisse des aktuell gültigen Kontexts zu beschränken.

- Besonders vorteilhaft erscheint der Ansatz der losen Integration im Hinblick auf die Modifizierbarkeit und Weiterentwicklung eines integrierten Produktmodells. So können einerseits Partialmodelle unabhängig voneinander geändert werden und andererseits neu hinzukommende Produktaspekte in Form zusätzlicher Partialmodelle in das integrierte Produktmodell eingearbeitet werden.

- Der Ansatz unabhängiger Produktsichten, welche über gegebene Konsistenzbedingungen zueinander in Beziehung gesetzt werden, erlaubt im Gegensatz zur engen Integration die Anwendung flexibler Strategien der Konsistenzsicherung. So wird beispielsweise im Sinne einer explorativen Produktentwicklung ein kontrollierter Umgang mit temporären und lokal begrenzten Inkonsistenzen ermöglicht (vgl. Teil III).

Jedoch ist der Ansatz der losen Integration auch mit einer Reihe an Nachteilen und Schwierigkeiten verbunden:

- Im Rahmen der losen Integration werden Produktsichten als eigenständige, voneinander unabhängige Partialmodelle aufgefasst. Diese beziehen sich jedoch auf ein gemeinsames Produkt, so dass vor dem Hintergrund zahlreicher und vielfältiger fachlicher Querbezüge mit einer im Vergleich zur engen Integration deutlich erhöhten Redundanz erfasster Modellinformationen auszugehen ist.

- Durch ein zentrales Modell mit eng gekoppelten Produktsichten ist die Konsistenz des integrierten Produktmodells bereits implizit gegeben. Die zu erwartende Redundanz eigenständiger und unabhängiger Partialmodelle erfordert dagegen explizite Mechanismen zur Prüfung und Sicherstellung der Konsistenz und erhöht somit den Aufwand einer modellbasierten Entwicklung.

- Im Gegensatz zur engen Kopplung sind Produktsichten nicht über ein zentrales Modell miteinander verbunden. Vielmehr kann jedes Partialmodell potenziell Querbezüge zu allen anderen Partialmodellen aufweisen und so die Komplexität des integrierten Produktmodells erhöhen. Eingeschränkt auf binäre Beziehungen wird dies durch Abbildung 4.10 veranschaulicht.

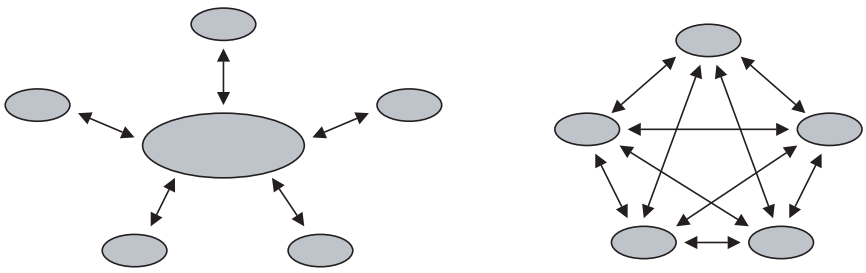

Abbildung 4.10: Querbezüge zwischen Produktsichten

- Schließlich stellt die lose Kopplung gegenüber der konventionellen Vorgehensweise einer engen Integration einen vergleichsweise neuartigen Forschungsansatz dar. Entsprechend gering sind bisherige Erfahrungen hinsichtlich der Umsetzbarkeit, Handhabbarkeit, Skalierbarkeit, Wartbarkeit und nicht zuletzt der Sicherstellung der Konsistenz eines auf diese Weise aufgebauten integrierten Produktmodells.

Diskussion

Nach der Vorstellung der beiden grundlegenden Alternativen zur Unterstützung verschiedener Sichten im Rahmen eines integrierten Produktmodells und der Darstellung ihrer spezifischen Vorteile und Nachteile werden nun Kriterien diskutiert, anhand derer eine Entscheidung zwischen dem Ansatz der engen Integration und dem Ansatz der losen Integration getroffen werden kann. Offensichtlich spielt die *Komplexität* der Produktentwicklung die entscheidende Rolle in dreierlei Hinsicht:

Prozesskomplexität: Je umfangreicher und komplexer sich ein Produktentwicklungsprozess gestaltet und je mehr Bearbeiter und Fachdisziplinen daran beteiligt sind, desto wahrscheinlicher ist es, dass große Unterschiede in dem jeweiligen Verständnis des zu entwickelnden Produkts aus Sicht einer Aktivität oder eines Bearbeiters auftreten. Entsprechend schwierig ist in diesem Fall eine Einigung auf ein zentrales und gemeinsames Produktmodell, das alle relevanten Produktaspekte gleich berechtigt repräsentiert. Während die enge Integration also vor allem für einfache, kompakte und überschaubare Entwicklungsprozesse einen nahe liegenden Integrationsansatz darstellt, gebieten umfangreiche und komplexe Entwicklungsprozesse die Verwendung der losen Integration zur Schaffung eines integrierten Produktmodells.

Produktkomplexität: Neben der Komplexität des Prozesses spielt ebenso die Komplexität des Produktes eine wichtige Rolle. Während sich einfache Produkte bereits durch ein einziges Modell hinreichend beschreiben lassen, werden zur Entwicklung komplexer Produkte zahlreiche Partialmodelle eingesetzt (vgl. Abschnitt 2.1.3). Je größer die Anzahl dieser Produktsichten ist und je mehr sich diese in Struktur und betrachteten Produkteigenschaften voneinander unterscheiden, desto schwerer lässt sich ein übergreifendes, zentrales Modell zur Integration aller Produktsichten finden. Während einfache Produkte demnach durch ein Modell mit wenigen Sichten beschrieben werden können und sich somit der Ansatz der engen Integration empfiehlt, bietet die lose Integration eine Lösung für sehr komplexe Produkte mit zahlreichen Partialmodellen.

Werkzeugkomplexität: Der dritte zu betrachtende Komplexitätsgrad resultiert aus der Verwendung von Software-Werkzeugen gemäß der virtuellen Produktentwicklung (vgl. Abschnitt 2.1.4). Jedes Software-Werkzeug setzt üblicherweise ein eigenes, darauf abgestimmtes Modell des zu entwickelnden Produkts voraus. Je mehr Werkzeuge im Laufe eines Produktentwicklungsprozesses zum Einsatz kommen, desto größer

ist auch die Anzahl so entstehender Partialmodelle und desto schwerer ist es, diese im Rahmen eines zentralen Modells zusammenzuführen. Der Ansatz der engen Integration bietet sich also bei der Verwendung weniger Software-Werkzeuge an, die auf Grundlage eines gemeinsamen Modells integriert werden können, während die lose Integration auf den Einsatz zahlreicher Software-Werkzeuge ausgerichtet ist.

Zusammenfassend lässt sich festhalten, dass die Auswahl des Integrationsansatzes von der Komplexität des Entwicklungsprozesses, des zu entwickelnden Produkts und der dazu erforderlichen Werkzeuglandschaft abhängig ist. Während sich die enge Integration vor allem für überschaubare Entwicklungsprozesse und einfache Produkte mit wenigen Produktsichten und einer geringen Anzahl eingesetzter Software-Werkzeuge eignet, bietet die lose Integration einen Lösungsansatz für umfangreiche Prozesse, in deren Mittelpunkt komplexe Produkte mit vielen Produktsichten stehen, die mithilfe zahlreicher und vielfältiger Software-Werkzeuge entwickelt werden.

Für den in der vorliegenden Arbeit betrachteten Bereich komplexer technischer Produkte wird daher der Ansatz der losen Integration zur Unterstützung verschiedener Produktsichten im Rahmen eines integrierten Produktmodells gewählt. Dieses besteht demnach aus eigenständigen und gleich berechtigten Partialmodellen des zu entwickelnden Produkts, welche unterschiedliche, im Laufe des Entwicklungsprozesses relevante Produktaspekte repräsentieren. Querbezüge, Überschneidungen und Abhängigkeiten zwischen Partialmodellen werden durch explizite Bedingungen erfasst, welche es erlauben, die Konsistenz des Gesamtmodells zu prüfen und zu sichern.

Hinsichtlich der Auswahl von Partialmodellen, also ihrer Art und ihrer inneren Struktur, legt der Ansatz der losen Integration über die Verwendung eines gemeinsamen Metamodells hinaus, keinerlei Vorgaben fest. So ist es prinzipiell möglich, jedem relevanten Aspekt der Produktentwicklung eine spezifische, optimal angepasste Produktsicht mit je eigenem Verwendungszweck zuzuordnen. Eine Aufteilung in Produktsichten kann somit anhand sehr unterschiedlicher Kriterien erfolgen und beispielsweise durch die Art des Produkts, die Struktur des Entwicklungsprozesses oder der zugrunde liegenden Organisation motiviert sein:

Produktaspekte: Ein nahe liegendes Kriterium für eine Aufteilung in Produktsichten ist die Orientierung an wesentlichen und grundlegenden Aspekten und Eigenschaften eines technischen Produkts, wie beispielsweise seine (physische) Strukturierung oder grundlegende Leistungsmerkmale.

Aktivitäten: Der Ansatz der losen Integration erlaubt es ausdrücklich, jeder individuellen Aktivität eines Produktentwicklungsprozesses eine eigene Produktsicht zuzuordnen. Dies stellt einen Ansatzpunkt zur methodischen Integration von Entwicklungsprozessen und Partialmodellen dar (s. Abbildung 4.11).

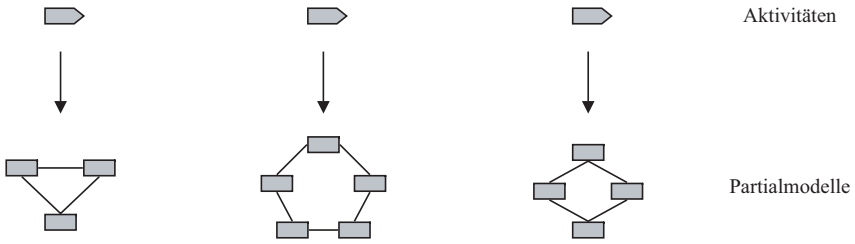

Abbildung 4.11: Aufteilung nach Aktivitäten

Teilprozesse: Neben der feingranularen Orientierung an individuellen Aktivitäten ist es ebenso vorstellbar, gemeinsame Produktsichten innerhalb verschiedener Aktivitäten eines übergreifenden Teilprozesses zu verwenden und eine Aufteilung somit gemäß charakteristischer Abschnitte der Produktentwicklung vorzunehmen (s. Abbildung 4.12).

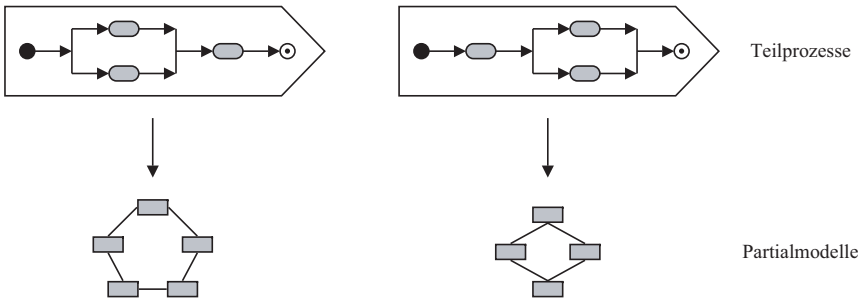

Abbildung 4.12: Aufteilung nach Teilprozessen

Bearbeiter und Rollen: Anstelle von Produkt- oder Prozessaspekten ist eine Aufteilung in Sichten ebenso anhand von Organisationskriterien möglich. So ist es vorstellbar, jedem Bearbeiter bzw. jeder Rolle eine eigene Sicht zu gestatten, die spezifisch auf seine Anforderungen und Präferenzen ausgerichtet ist (s. Abbildung 4.13).

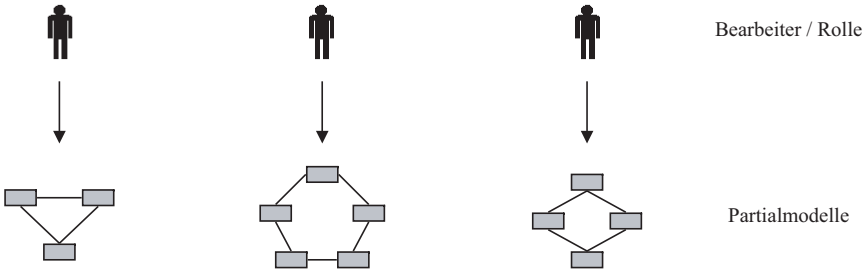

Abbildung 4.13: Aufteilung nach Bearbeiter oder Rollen

Organisationseinheiten: Analog zu Aktivitäten und Teilprozessen kann über die feingranulare Aufteilung gemäß einzelner Bearbeiter bzw. Rollen hinaus die Definition von Produktsichten ebenso anhand übergreifender Organisationseinheiten, wie Entwicklungsabteilungen oder Fachdisziplinen, erfolgen (s. Abbildung 4.14).

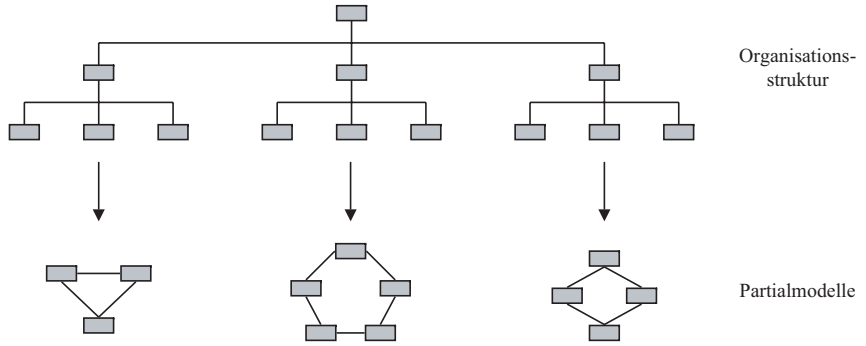

Abbildung 4.14: Aufteilung nach Organisationseinheiten

Werkzeuge: Schließlich können auch in der Produktentwicklung verwendete Software-Werkzeuge den entscheidenden Ausgangspunkt für eine Aufteilung in Partialmodelle darstellen (s. Abbildung 4.15). Ohnehin erfordern diese meist ein spezifisches und angepasstes Modell des zu entwickelnden Produkts.

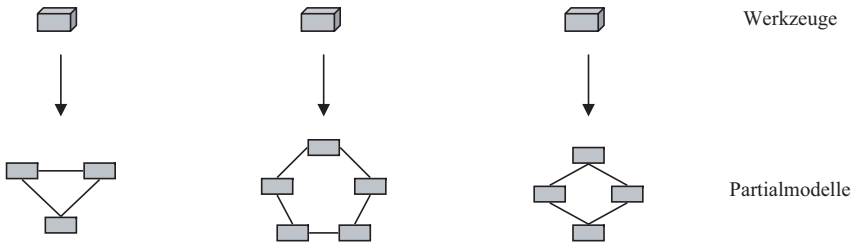

Abbildung 4.15: Aufteilung nach Werkzeugen

Weitere Möglichkeiten der Auswahl und Definition von Partialmodellen im Rahmen eines integrierten Produktmodells ergeben sich aus spezifischen Verbindungen oben genannter Kriterien. So könnten Produktsichten beispielsweise sehr individuell durch eine Kombination aus durchzuführender Entwicklungsaktivität, dem zugewiesenen Bearbeiter sowie dem dabei verwendeten Software-Werkzeug festgelegt werden. Auf einer übergreifenden Ebene könnte eine Kombination aus relevanten Fachdisziplinen und vorhandenen Teilprozessen ein wichtiges Kriterium für eine Aufteilung in Produktsichten darstellen.

Hinsichtlich des Zusammenspiels von Entwicklungsprozess und dabei verwendeten Partialmodellen des zu entwickelnden Produkts sind auf der Grundlage der losen Integration zweierlei Vorgehensweisen vorstellbar: Der konventionelle und deduktive Ansatz legt zunächst den Entwicklungsprozess fest und leitet daraus die erforderlichen Partialmodelle des betrachteten Produkts ab. Alternativ könnten induktiv auch zuerst relevante Partialmodelle definiert und darauf aufbauend die sie manipulierenden Aktivitäten festgelegt werden. Beziehungen zwischen Aktivitäten würden auf diese Weise unmittelbar mit den Beziehungen zwischen Partialmodellen korrespondieren.

Insgesamt ist auf einen methodischen Umgang mit Partialmodellen besonderer Wert zu legen. So ist zu bedenken, dass Produktsichten zwar prinzipiell beliebig erstellt werden können, jedoch geeignet verwaltet werden müssen. Eine sehr feingranulare Aufteilung in Sichten gestattet demnach zwar einerseits eine flexible Anpassung an einzelne Aktivitäten, Bearbeiter oder Werkzeuge, führt jedoch andererseits zu einer sehr großen Anzahl an Partialmodellen, welche geeignete Ordnungsmechanismen notwendig macht. Andererseits ist eine geringe Anzahl an Partialmodellen einfacher zu verwalten, setzt jedoch andererseits eine Einigung auf gemeinsame Modelle voraus.

Metamodell

Aus der Vorstellung der beiden Integrationsalternativen einer engen und losen Kopplung sowie ihrer Gegenüberstellung und der Entscheidung für den Ansatz der losen Integration, lassen sich die ersten Elemente des angestrebten Metamodells für ein integriertes Produktmodell ableiten. Abbildung 4.16 zeigt diese und ihre Beziehungen in UML-Notation. Gemäß dem Ansatz der losen Kopplung besteht ein Modell eines Produkts (engl.: *product*) demnach aus beliebig vielen, jedoch mindestens einer Sicht (engl.: *view*) sowie einer beliebigen Anzahl an Bedingungen (engl.: *condition*) zur Prüfung und Sicherstellung des Konsistenz des Gesamtmodells.

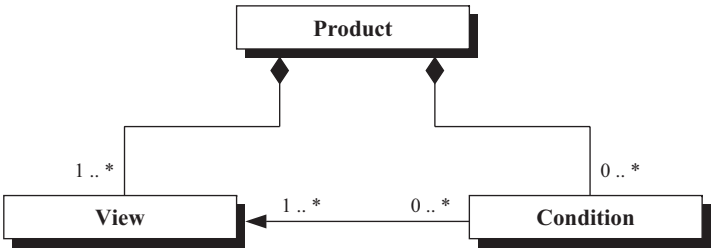

Abbildung 4.16: Initiale Version des Metamodells

Jede Konsistenzbedingung kann auf beliebig viele der festgelegten Produktsichten verweisen und sie so zueinander in Beziehung setzen. Sie muss jedoch mindestens einer Produktsicht zugeordnet sein. Umgekehrt kann jede Produktsicht in beliebig viele Konsistenzbedingungen eingehen. Auf diese Weise ergibt sich ein hinreichend allgemeiner Rahmen, um komplexe Querbezüge zwischen Produktsichten innerhalb eines integrierten Produktmodells gleichartig zu erfassen. Konkrete Ausprägungen von Konsistenzbedingungen und mögliche Techniken zu ihrer Formulierung werden in Teil III der vorliegenden Arbeit betrachtet.

4.3.2 Strukturierung

Das beschriebene Konzept der Segmentierung, d.h. die Aufteilung in mehrere Partialmodelle und eine explizite Spezifikation vorhandener Querbezüge, bildet die Grundlage zur Unterstützung verschiedener Produktsichten im Rahmen eines integrierten Produktmodells. Zur Darstellung einzelner Partialmodelle sind darüber hinaus weitere Modellierungskonzepte erforderlich. So

stellt insbesondere die Erfassung der Produktstruktur im Rahmen einer Produktsicht eine wichtige Anforderung dar, die wesentlich die Expressivität des angestrebten Metamodells beeinflusst. Auch in diesem Zusammenhang sind zwei Alternativen zu betrachten:

Baumartige Strukturierung: Ein baumartiger Ansatz, wie schematisch in Abbildung 4.17 gezeigt, ist eine häufig anzutreffende Darstellungsform für Produktstrukturen. Insbesondere im Bereich des Computer Aided Design (CAD) ist diese Strukturierungsmethode weit verbreitet. Die baumartige Strukturierung eines Produktmodells bzw. einer Produktsicht entspricht einer hierarchischen Dekomposition in logisch zusammengehörende Produktbestandteile gemäß ihres konstruktiven Aufbaus. So wird beispielsweise in der konstruktiven geometrischen Sicht (vgl. Abschnitt 2.2.2) die äußere Gestalt eines Bauteils mit Hilfe elementarer geometrischer Elemente aufgebaut, welche sukzessive miteinander kombiniert werden.

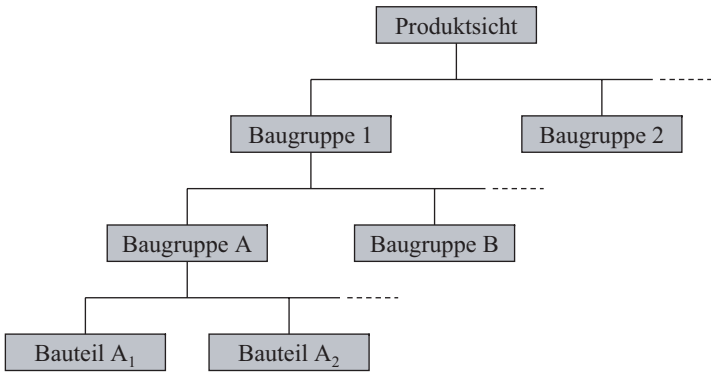

Abbildung 4.17: Baumartige Produktstrukturierung

Graphartige Strukturierung: Die Darstellung von Produktstrukturen in Form von Graphen stellt den allgemeinsten Ansatz der Strukturmodellierung dar. Dieser umfasst insbesondere die baumartige Strukturierungsmethode, da sich Bäume als gerichtete und azyklische Graphen auffassen lassen. Abbildung 4.18 zeigt schematisch eine graphartige Strukturierung eines Produkts bzw. einer Produktsicht. Hierbei können beliebige (binäre) Beziehungen zwischen Produktbestandteilen auftreten. So bestehen etwa strukturmechanische Produktsichten (vgl. Abschnitt 2.2.2) aus Finite-Elemente-Modellen, welche die geometrische

Produktgestalt durch Graphen aus Netzknoten und Netzkanten nachzubilden versuchen.

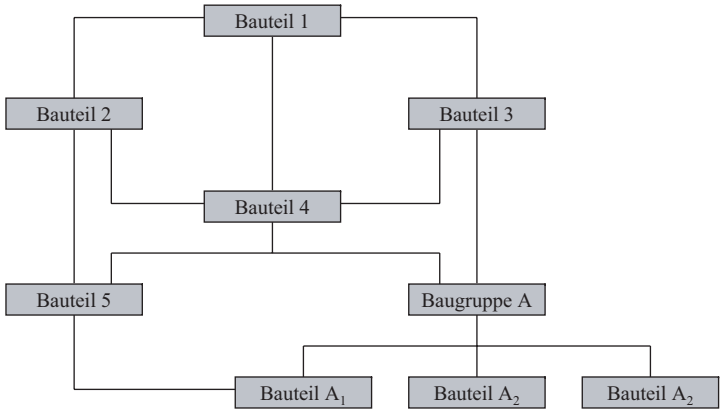

Abbildung 4.18: Graphartige Produktstrukturierung

Im Rahmen der Strukturierung ist das Metamodell für integrierte Produktmodelle um Modellierungskonstrukte zu erweitern, die es erlauben, die Struktur eines Produkts, also seine Bestandteile und ihre Beziehungen, innerhalb einer Produktsicht adäquat zu erfassen. Gemäß dem Ansatz der losen Integration können unterschiedliche Produktsichten verschiedene Strukturierungen aufweisen. Jedoch ist darauf zu achten, das Metamodell so zu konzipieren, dass die gewählten Modellierungskonstrukte hinreichend ausdrucksstark sind, um in allen Produktsichten gleichermaßen verwendet werden zu können.

Vor diesem Hintergrund ist eine Auswahl zwischen den beiden vorgestellten Strukturierungsmethoden zu treffen: So stellt die baumartige Strukturierung einen häufig verwendeten Strukturierungsansatz dar, der sich an einer hierarchischen Dekomposition eines Produkts orientiert, weit verbreitet ist und vergleichsweise einfach handhaben lässt. Eine graphartige Produktstrukturierung ist dagegen mit einer signifikant höheren Modellkomplexität verbunden. Jedoch bietet nur diese angesichts zahlreicher, verschiedenartiger Produktsichten hinreichend ausdrucksstarke Modellierungskonstrukte, um in allen Produktsichten Verwendung finden zu können.

Aus diesem Grund wird das angestrebte Metamodell um Modellierungskonstrukte erweitert, die eine graphartige Produktstrukturierung ermöglichen. Diese erlauben es, als Spezialfall insbesondere baumartige Produkt-

strukturen zu bilden. Abbildung 4.19 zeigt in UML-Notation das um Strukturierungsaspekte erweiterte Metamodell (vgl. Abbildung 4.16). Jede Produktsicht besteht demnach aus einer beliebig großen Anzahl an Produktbestandteilen (engl.: *component*), jedoch mindestens einem. Daneben können beliebig viele Assoziationen (engl.: *association*) existieren, die je zwei Produktbestandteile zueinander in Beziehung setzen.

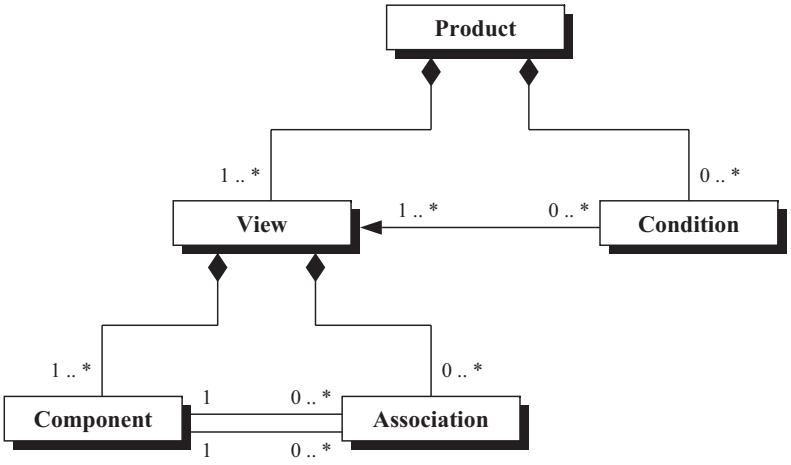

Abbildung 4.19: Erweiterung des Metamodells um Strukturaspekte

Komponenten stellen das zentrale und allgemein verwendbare Modellierungskonstrukt dar, um die innerhalb einer Produktsicht relevanten Produktbestandteile zu erfassen. Je nach Kontext und Zielsetzung einer Produktsicht repräsentieren sie die logisch zusammengehörenden Einheiten einer aktuell sinnvollen Unterteilung eines Produkts. Entsprechend können Komponenten dazu benutzt werden, Baugruppen oder Bauteile zu modellieren, aber auch als Grundbausteine jeder anderen Produktstrukturierung dienen. So repräsentieren die Komponenten der konstruktiven Sicht eines Rotors (vgl. Abschnitt 2.2.2) beispielsweise geometrische Figuren bzw. Körper.

Um innerhalb einer Produktsicht Beziehungen zwischen Komponenten ausdrücken zu können, bietet das Metamodell das Modellierungskonstrukt der *Assoziation* an. Jede Assoziation verbindet je zwei (nicht notwendig unterschiedliche) Komponenten miteinander. Umgekehrt kann jede Komponente an beliebig vielen Assoziationen beteiligt sein. Es sei darauf hingewiesen, dass das Metamodell keine einschränkenden Vorgaben hinsichtlich der Art einer Assoziation festlegt. Entsprechend können Assoziationen be-

liebige, im aktuellen Kontext sinnvolle Beziehungen zwischen Komponenten repräsentieren.

Beispiel 4.1 (*Strukturmodell der Fallstudie*)

Anhand der betrachteten Fallstudie eines Niederdruckturbinenrotors, bestehend aus zwei Turbinenscheiben, wird für die konstruktive Produktsicht die Verwendung von Komponenten und Assoziationen zur Modellierung einer Produktstruktur erläutert. Abbildung 4.20 zeigt die verschiedenen Bestandteile einer Turbinenscheibe (engl.: *disk*) im Aufriss (vgl. Abbildung 2.5). Diese besteht demnach aus einem Scheibenhauptkörper (engl.: *body*) sowie zwei Armen (engl: *wing*). Der Scheibenhauptkörper umfasst seinerseits drei Bestandteile, welche als *hub*, *rim* und *web* bezeichnet werden.

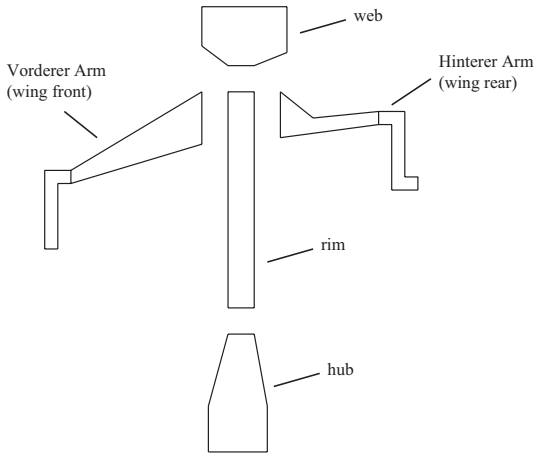

Abbildung 4.20: Bestandteile einer Turbinenscheibe

Die so beschriebene Rotorstruktur kann mithilfe von Komponenten und Assoziationen wie folgt modelliert werden: Zunächst wird für jede Turbinenscheibe je eine Komponente für die Bestandteile *web*, *rim*, *hub*, *wing front* und *wing rear* eingeführt. Die ersten drei Komponenten bilden den Scheibenhauptkörper und werden jeweils in einer Aggregationskomponente *body* zusammengefasst. In gleicher Weise wird für jede Turbinenscheibe eine Aggregationskomponente *disk* eingeführt und die Flanschverbindung zwischen diesen durch eine Komponente *flange* repräsentiert. Eine Aggregationskomponente *rotor* fasst schließlich die beschriebenen Bestandteile zusammen.

In der konstruktiven Sicht eines Rotors stehen zwei Komponenten genau dann zueinander in Beziehung, wenn sie in der Aufrisszeichnung über eine gemeinsame Begrenzungslinie verfügen. Zusätzlich besitzen Aggregationskom-

ponenten Beziehungen zu jeder in ihr enthaltenen Teilkomponente. Daraus lassen sich die Assoziationen der betrachteten Produktstruktur ermitteln. Abbildung 4.21 zeigt das so entstehende Modell in UML-artiger Notation. Diese ist jedoch nicht zu verwechseln mit dem gleichfalls in UML beschriebenen Metamodell (s. Abbildung 4.19). Vielmehr stellen die Komponenten und Assoziationen Instanzen der Metamodellelemente dar.

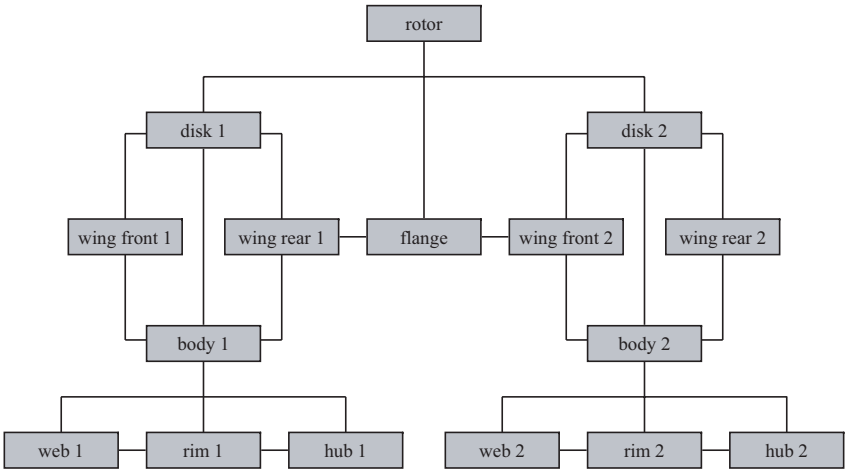

Abbildung 4.21: Strukturmodell eines Rotors

Auf die gezeigte Weise lassen sich durch Komponenten und Assoziationen beliebige Produktstrukturen nachbilden. Die genaue Art eines Modellaufbaus bleibt dabei letztlich dem Verständnis des Anwenders überlassen. So ließe sich beispielsweise die beschriebene Struktur eines Turbinenrotors auch auf andere als die in Abbildung 4.21 gezeigte Weise erfassen. Insbesondere stellt die Verwendung von Aggregationskomponenten, welche lediglich dazu dienen, vorhandene Modellelemente zusammenzufassen, einen Freiheitsgrad dar. Diese erhöhen zwar einerseits den Modellumfang, tragen jedoch andererseits durch Partionierung zur Übersichtlichkeit eines Modells bei.

4.3.3 Typisierung

Aus der Zielsetzung der Wiederverwendbarkeit einmal erstellter Modelle resultiert die Anforderung nach einer Unterstützung verschiedener Abstraktionsebenen (vgl. Abschnitt 4.2). So soll es das angestrebte Metamodell

ermöglichen, gemeinsame, abstrakte Eigenschaften gleichartiger Produktbestandteile in übergreifender Weise festzulegen und somit an zahlreichen Stellen eines Modells wieder verwenden zu können. In diesem Zusammenhang bietet sich das aus zahlreichen Modellierungssprachen (s. z.B. [Che76] oder [UML03]) bekannte und bewährte Konzept der Typisierung, also einer Unterscheidung zwischen *Typen* und *Instanzen*, an.

Abbildung 4.22 illustriert beispielhaft das Prinzip der Typisierung. So sind zunächst auf der Ebene des Metamodells geeignete Modellierungskonstrukte zur Spezifikation von Typen vorzusehen. Die Instanziierung des Metamodells ermöglicht es, verschiedene Typen zu definieren, welche ihrerseits gemeinsame Eigenschaften gleichartiger, diesem Typ zuzuordnenden Entitäten eines Anwendungsbereichs in abstrakter Weise zusammenfassen. Aus Typen ergeben sich schließlich – wiederum durch Instanziierung, also einer Belegung der vorgesehenen Freiheitsgrade – die Instanzen eines Modells, welche als konkrete Ausprägungen eines Typs die betrachteten Entitäten repräsentieren.

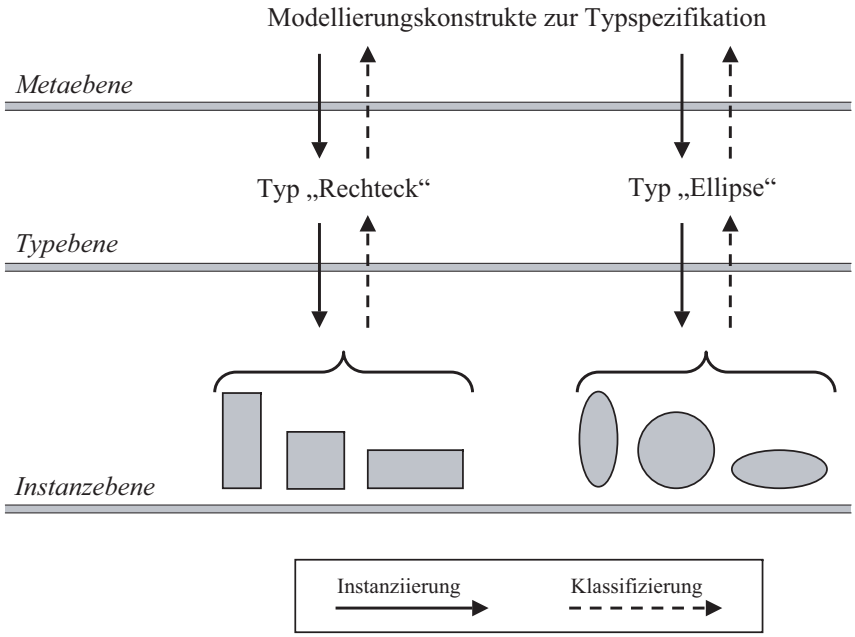

Abbildung 4.22: Prinzip der Typisierung

Produktbestandteile werden gemäß den Ergebnissen des vorhergehenden Abschnitts durch das Modellierungskonstrukt der Komponente repräsentiert. Entsprechend ist das angestrebte Metamodell für integrierte Produktmodelle

so zu erweitern, dass jede Komponente genau einem Typ zugeordnet werden kann. Hierbei erscheint es sinnvoll, zusätzlich das im Zusammenhang mit Typen häufig anzutreffende Konzept der Spezialisierung bzw. Generalisierung in das Metamodell aufzunehmen. Dieses legt eine (binäre) Verfeinerungsrelation zwischen den Typen eines Modells fest und erlaubt es so, Typen hierarchisch anzuordnen.

Das in diesem Sinne erweiterte Metamodell ist in Abbildung 4.23 in UML-Notation dargestellt (vgl. Abbildung 4.19). Eine Produktsicht beinhaltet demnach neben Komponenten und Assoziationen ebenfalls eine beliebig große Anzahl an Typen (engl.: *type*), jedoch mindestens einen Typ. Hiermit wird jeder Komponente einer Produktstruktur genau ein Typ zugeordnet, der genutzt werden kann, um gemeinsame Eigenschaften gleichartiger Komponenten in übergreifender Weise festzulegen. Entsprechend kann es zu jedem Typ beliebig viele Komponenten geben, die als Instanzen konkrete Ausprägungen dieses Typs repräsentieren.

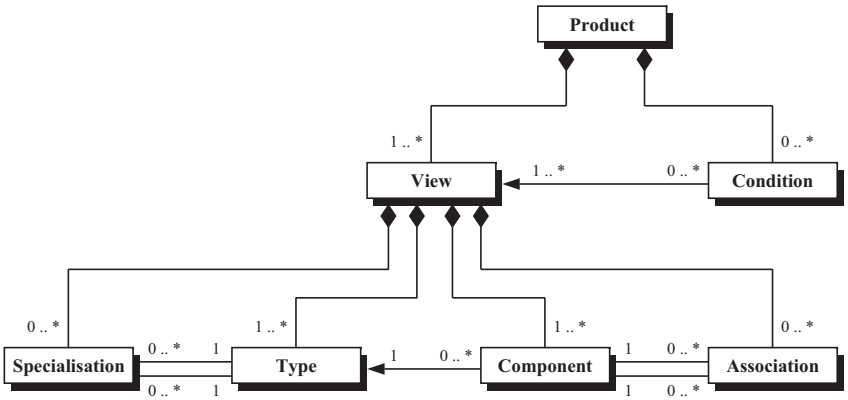

Abbildung 4.23: Erweiterung des Metamodells um Typaspekte

Zusätzlich wird zu jeder Produktsicht eine Spezialisierungsrelation (engl.: *specialisation*) eingeführt, um Typen hierarchisch anzuordnen. Ein Typ ist genau dann als Spezialisierung eines anderen Typs anzusehen, wenn er mindestens dessen Eigenschaften aufweist. In jede Spezialisierung gehen demnach genau zwei Typen ein, welche relativ zu der betrachteten Spezialisierung gemäß ihrer Anordnung in der Typhierarchie üblicherweise als *Obertyp* bzw. *Untertyp* bezeichnet werden [Kla04]. Umgekehrt kann jeder Typ an beliebig vielen Spezialisierungen beteiligt sein, wobei gefordert sei, dass es zu jedem Typ höchstens einen Obertyp gibt.

Beispiel 4.2 (*Typmodell der Fallstudie*)

Aufbauend auf das im vorangehenden Beispiel 4.1 vorgestellte Strukturmodell der konstruktiven Sicht eines Turbinenrotors, bestehend aus Komponenten und Assoziationen, wird im Folgenden die Verwendung des beschriebenen Konzepts der Typisierung illustriert. Grundlegend ist die Einführung einer Typhierarchie, welche es erlaubt, die Komponenten des betrachteten Strukturmodells anhand gleichartiger Eigenschaften zu klassifizieren. Abbildung 4.24 zeigt ein solches Typmodell sowie Spezialisierungsbeziehungen zwischen den auftretenden Typen in UML-Notation.

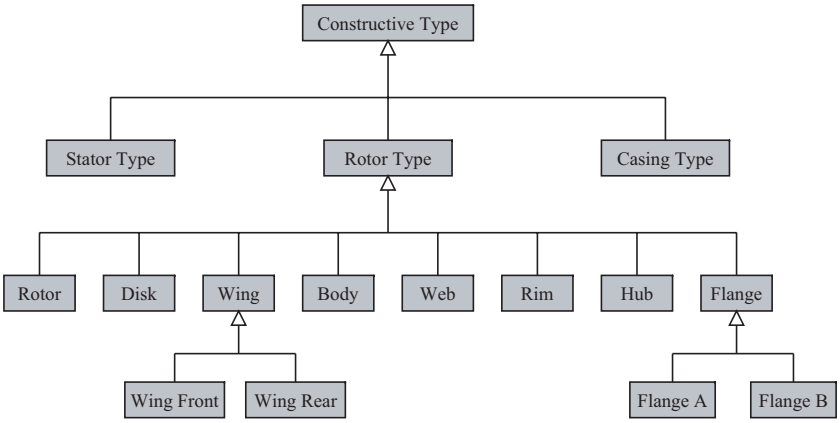

Abbildung 4.24: Typmodell der konstruktiven Sicht

Zunächst wird der Typ *Constructive Type* eingeführt, welcher nicht unmittelbar der Klassifizierung von Komponenten dient, sondern als allgemeiner Obertyp einen Ausgangspunkt bietet, um die Typen der konstruktiven Sicht anzuordnen und übergreifende, alle Typen betreffende Eigenschaften bereits an dieser Stelle festlegen zu können. Als Spezialisierungen werden davon die drei Typen *Stator Type*, *Rotor Type* und *Casing Type* abgeleitet, die gemäß der grundlegenden Aufteilung einer Niederdruckturbine in Stator, Rotor und Gehäuse (vgl. Abschnitt 2.2) ihrerseits als allgemeine Obertypen zur Typisierung der Komponenten dieser drei Bereiche dienen.

Ausgehend vom Typ *Rotor Type* wird je charakteristischem Bauelement eines Turbinenrotors ein eigener Typ eingeführt. So dient der Typ *Wing* beispielsweise dazu, die Komponenten *wing front 1*, *wing rear 1*, *wing front 2* und *wing rear 2* des betrachteten Strukturmodells (s. Abbildung 4.21) als gleichartige Komponenten zu kennzeichnen und gemeinsame Eigenschaften

in übergreifender Weise festzulegen. Für ggf. zu modellierende Unterschiede zwischen dem vorderen Arm und dem hinteren Arm einer Turbinenscheibe stehen darüber hinaus die beiden Spezialisierungen *Wing Front* und *Wing Rear* als Typen zur Verfügung

In gleicher Weise sei angenommen, dass verschiedenartige Flanschverbindungen zwischen den beiden Turbinenscheiben *disk 1* und *disk 2* des betrachteten Rotors möglich sind. Entsprechend steht ein allgemeiner Obertyp *Flange* zur Verfügung, der beispielsweise die Spezialisierungen *Flange A* und *Flange B* besitze. Somit kann seitens eines Bearbeiters flexibel eine optimale Flanschverbindung ausgewählt werden, welche einerseits in beiden Fällen die durch den Obertyp festgelegten Eigenschaften eines allgemeinen Flanschs aufweist und andererseits über die spezifischen Charakteristika der aktuell gewählten Flanschverbindung verfügt.

4.3.4 Attributierung

Gemäß den Ergebnissen des vorhergehenden Abschnitts stellt die Verwendung von Typen eine Grundlage dar, um Produktbestandteile als gleichartig zu kennzeichnen und somit zu klassifizieren. Darüber hinaus ist eine Modellierungsmöglichkeit erforderlich, um elementare Eigenschaften einzelner Komponenten zu erfassen. Diese werden im Bereich der Modellbildung üblicherweise als *Attribute*, *Variablen* oder *Parameter* bezeichnet und legen die Freiheitsgrade eines Typs fest. Instanzen eines Typs, also die Repräsentanten realer Produktbestandteile, ergeben sich daraus durch eine Belegung der vorgesehenen Attribute.

Die Zielsetzung dieser, auch als *parametrisierter Produktmodellierung* bezeichneten Vorgehensweise ist es, alle relevanten, im Laufe der Produktentwicklung erarbeiteten Produkteigenschaften soweit möglich durch Attribute zu erfassen. Hierzu zählen beispielsweise geometrische Abmessungen ebenso, wie Temperatur- und Festigkeitswerte, auftretende Kräfte, Materialeigenschaften oder Fertigungsvorgaben. Aber auch Kosten-, Lagerhaltungs- oder Wartungsaspekte können auf diese Weise in das Produktmodell einfließen. Entsprechend ist das angestrebte Metamodell um Modellierungskonstrukte zur Erfassung von Attributen zu erweitern.

Die in diesem Sinne ergänzte Version des Metamodells für integrierte Produktmodelle ist in Abbildung 4.25 dargestellt (vgl. Abbildung 4.23). Jeder Typ verfügt zur Modellierung seiner spezifischen Eigenschaften über eine beliebig große Anzahl an Attributen (engl.: *attribute*). Für eine genauere Spezifikation eines Attributs wird zusätzlich die Menge der Werte – also ein

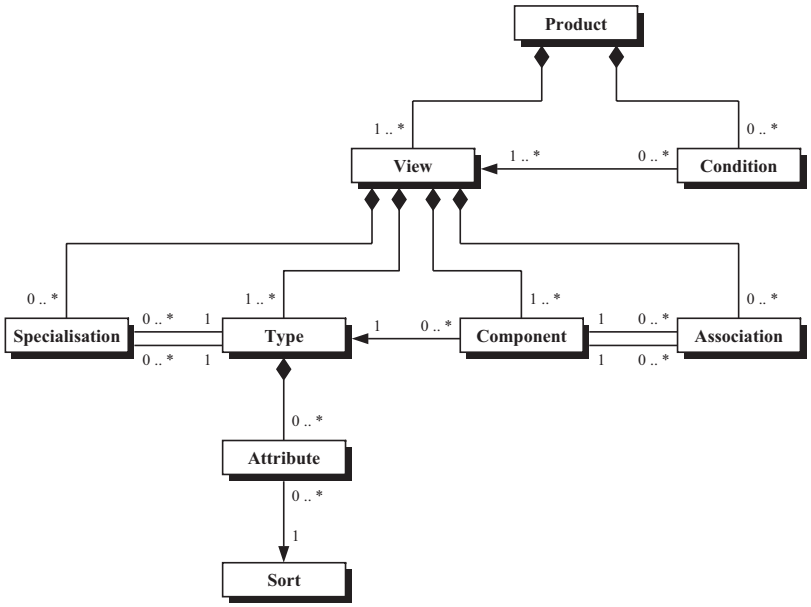

Abbildung 4.25: Erweiterung des Metamodells um Attributierungsaspekte

Datentyp – festgelegt, die auf Instanzseite auftreten können. Um eine Verwechslung mit dem in Abschnitt 4.3.3 eingeführten Typkonzept zu vermeiden, wird hierfür der ebenfalls gebräuchliche Begriff der *Sorte* (engl.: *sort*) verwendet.

Jedes Attribut verweist demnach auf genau eine Sorte, die den Bereich möglicher Ausprägungen festlegt, welche dieses Attribut auf Instanzseite annehmen kann. In umgekehrter Weise kann eine einmal definierte Sorte für beliebig viele Attribute verwendet werden. Entsprechend der Vielfältigkeit auftretender Produktinformationen können für Attribute je nach Produktsicht und dem Typ verwendeter Komponenten sehr unterschiedliche Sorten sinnvoll und erforderlich sein, beginnend bei einfachen numerischen Wertebereichen oder Zeichenketten bis hin zu Vektoren, Matrizen, Listen, Mengendarstellungen oder allgemeinen Aufzählungen.

Abbildung 4.26 ergänzt das in Abbildung 4.25 dargestellte Metamodell für komplexe Produkte um ein Sortenkonzept, welches die üblicherweise bei Attributen auftretenden Wertebereiche abdeckt und in einer Spezialisierungshierarchie anordnet. Analog zu Komponenten und ihren Typen (vgl. Abschnitt 4.3.3) wird in diesem Zusammenhang von (abstrakten) *Obersorten* und *Untersorten* gesprochen. Im Einzelnen werden folgende Wertebereiche berücksichtigt:

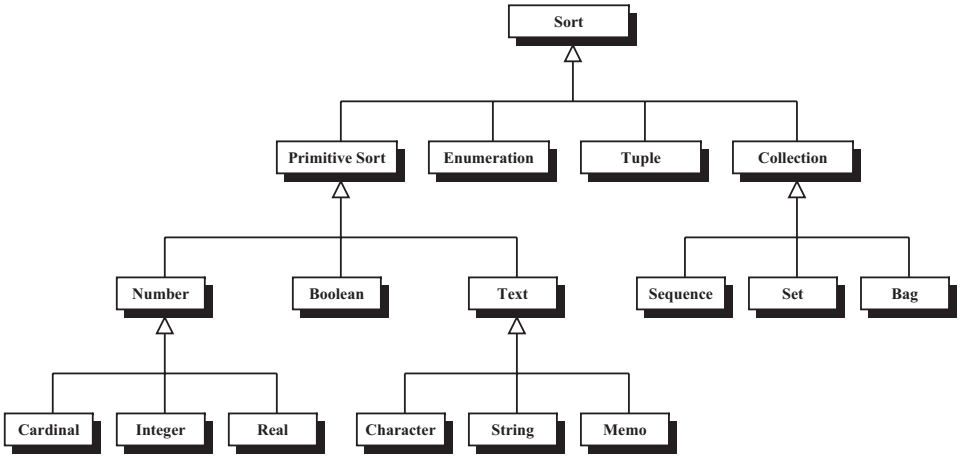

Abbildung 4.26: Sortenkonzept

Zahlenbereiche: Die Sorten *Cardinal*, *Integer* und *Real* repräsentieren die aus der Mathematik bekannten Zahlenbereiche der natürlichen Zahlen, der ganzen Zahlen sowie der reellen Zahlen.

Wahrheitswerte: Zur Darstellung elementarer boolscher Attributwerte ist die Sorte *Boolean* vorgesehen gemäß der Menge $\mathbb{B} = \{\texttt{false}, \texttt{true}\}$ der Wahrheitswerte.

Zeichen und Zeichenketten: Die Sorten *Character*, *String* und *Memo* ermöglichen die Darstellung einzelner Zeichen, endlicher Zeichenketten, sowie formatierter Texte.

Aufzählungen: Die Sorte *Enumeration* steht als Obersorte für alle die Sorten zur Verfügung, deren Wertebereich als endliche Aufzählung einzelner Werte einer gleichen Sorte vorgegeben ist. Auf diese Weise ließe sich etwa die Sorte *Colour* mit dem Wertebereich $\{\texttt{red}, \texttt{green}, \texttt{blue}\}$ als Aufzählung einzelner Farbwerte, welche durch Zeichenketten repräsentiert sind, in das Sortenkonzept einbringen.

Tupel: Analog dient die Sorte *Tuple* als allgemeine Obersorte für die Definition beliebiger Sorten, deren Wertebereich sich als kartesisches Produkt der Wertebereiche anderer Sorten darstellen lässt. Beispielsweise könnte so eine Sorte *Pair* definiert werden, die aufbauend auf die Sorte

Real als Wertebereich über die Menge aller Paare zweier reeller Zahlen verfügt.

Sammlungen: Neben primitiven Sorten, Aufzählungen und Tupeln spielen Sammlungen von Werten eine wichtige Rolle. Diese werden unter der Obersorte *Collection* zusammengefasst. Im Einzelnen finden sich die Sorten *Sequence* für endliche, geordnete Sequenzen einer Grundsorte und *Set* zur Darstellung des mathematischen Mengenkonzepts, sowie die Sorte *Bag*, die das Konzept der Multimenge repräsentiert.

Insgesamt betrachtet stellen die beschriebenen Sorten das Fundament dar, auf das die Modellierungskonstrukte des Produktmodells aufbauen. Sie geben das Werteuniversum vor, welches benutzt werden kann, um elementare Produktinformationen zu repräsentieren. Das in Abbildung 4.26 illustrierte Sortenkonzept erhebt nicht den Anspruch auf Vollständigkeit und ist nur als eine Möglichkeit zu verstehen, um Sorten festzulegen, die sich bei Bedarf ohne Änderung der grundlegenden Konzeption anpassen lässt. So ist es beispielsweise jederzeit möglich, auf einige der vorgestellten Sorten zu verzichten oder andere Sorten hinzuzufügen. Insbesondere könnten sich eigene Sorten zur Darstellung von Vektoren und Matrizen als nützlich erweisen, obwohl sich diese bereits über die erläuterte Sorte *Tuple* realisieren lassen. Aber auch allgemeine Konzepte zur Festlegung alternativer Sorten oder rekursiver Sorten könnten wertvolle Erweiterungen darstellen.

Beispiel 4.3 (*Attributiertes Modell der Fallstudie*)

Das folgende Beispiel zeigt auf, wie sich die erläuterten Konzepte der Segmentierung, Strukturierung, Typisierung und Attributierung insgesamt für die Erstellung eines Produktmodells verwenden lassen. Ausgehend von der in Abschnitt 2.2 eingeführten Fallstudie eines Turbinenrotors mit zwei Turbinenscheiben erfolgt als erster Schritt gemäß dem Prinzip der Segmentierung die Festlegung relevanter Produktsichten, orientiert an der Art des betrachteten Produkts, der Struktur des Entwicklungsprozesses, den Möglichkeiten eingesetzter Software-Werkzeuge und dem Verständnis beteiligter Entwickler.

Entsprechend wird das Produktmodell in eine Konfigurationssicht, eine konstruktive Produktsicht, eine bzw. mehrere Simulationssichten sowie eine Dokumentationssicht unterteilt, wie in Abschnitt 2.2 erläutert. In einem zweiten Schritt kann daraufhin für jede Produktsicht eine individuelle und optimal angepasste Struktur spezifiziert werden, bestehend aus Komponenten und Assoziationen gemäß den Vorgaben des Metamodells (vgl. Abbildung 4.19). Abbildung 4.21 zeigt ein mögliches Strukturmodell der kon-

struktiven Produktsicht eines Turbinenrotors. Auf gleiche Weise ließe sich die Struktur aller weiteren Produktsichten erfassen.

Um Komponenten als gleichartig zu kennzeichnen und gemeinsame Eigenschaften in übergreifender Weise festzulegen, werden drittens Typen definiert. So zeigt Abbildung 4.24, wie sich die Komponenten des betrachteten Strukturmodells eines Turbinenrotors klassifizieren und in einer Spezialisierungshierarchie anordnen lassen. Als vierter und letzter Schritt werden schließlich gemäß dem Prinzip der Attributierung die spezifischen Eigenschaften der gewählten Typen spezifiziert. Zur Verdeutlichung wird hierzu exemplarisch das vordere Armelement einer Turbinenscheibe betrachtet (vgl. Abbildung 4.20).

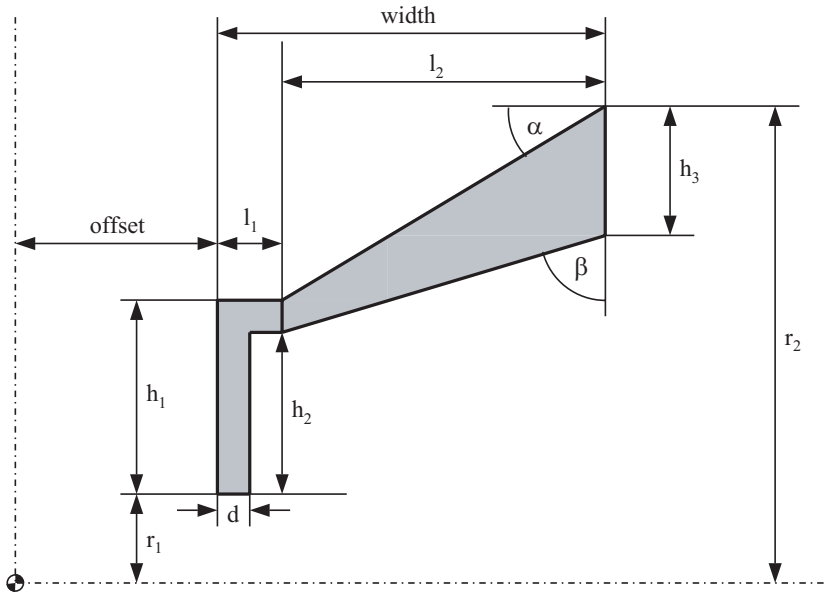

Abbildung 4.27: Vorderes Armelement einer Turbinenscheibe

In Abbildung 4.27 ist ein solches Armelement in Form einer zweidimensionalen parametrisierten Geometriebeschreibung dargestellt. Dies bedeutet, dass der prinzipielle geometrische Aufbau des betrachteten Bauteils vorgegeben ist, während vorhandene Freiheitsgrade, also geometrische Abmessungen und auftretende Winkel, durch charakteristische Attribute relativ zu einem gegebenen Koordinatensystem beschrieben werden. Es sei darauf hingewiesen, dass die verwendeten Attribute z.T. redundante Informationen beinhalten können. So ergibt sich beispielsweise das Attribut *width*, die Gesamtbreite des Bauteils, als Summe der Attribute l_1 und l_2.

Aus der parametrisierten Geometriebeschreibung lassen sich die Attribute des Komponententyps für ein vorderes Armelement (engl.: *wing front*) unmittelbar ableiten. Abbildung 4.28 zeigt in UML-Notation einen Ausschnitt des in Beispiel 4.2 eingeführten Typmodells für Komponenten einer Turbinenscheibe, erweitert um zugehörige Attribute und auf der Grundlage des beschriebenen Sortenkonzepts. Hierbei bezeichne das Attribut *origin* den relativen Koordinatenursprung, die Sorte *Pair* einen Wertebereich für Paare reeller Zahlen sowie die Sorte *Angle* einen entsprechenden Wertebereich für die Angabe geometrischer Winkelgrößen.

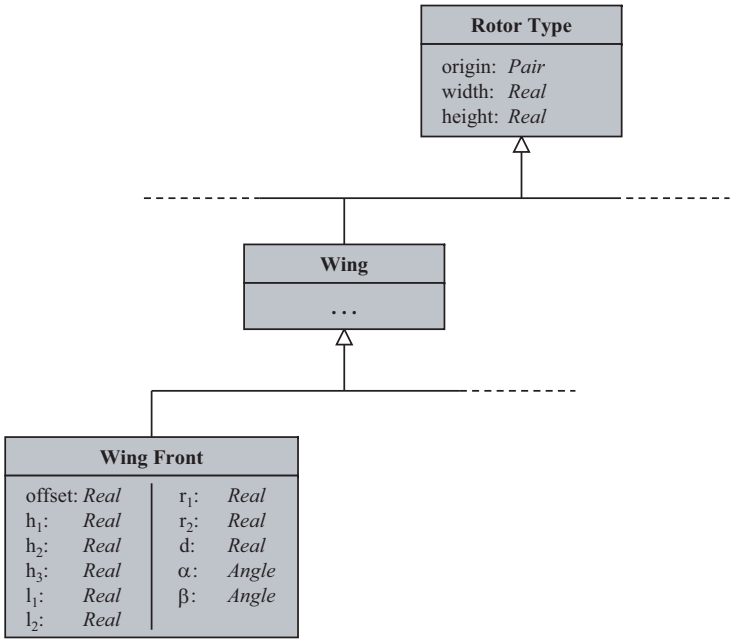

Abbildung 4.28: Attributierte Komponententypen

Vermöge der Spezialisierungsrelation zwischen Typen verfügt jeder Typ nicht nur über lokal definierte Attribute, sondern zusätzlich über alle die Attribute, die in Obertypen spezifiziert sind. Somit sind z.B. dem Typ *Wing Front* die Attribute *origin*, *width* und *height* des allgemeinen Obertyps *Rotor Type* sowie ggf. die Attribute des Typs *Wing* zugeordnet. Auf die gezeigte Weise können auch alle weiteren Typen des betrachteten Fallbeispiels mit Attributen versehen werden. Insgesamt ergibt sich daraus eine typisierte und attributierte Struktur der gewählten konstruktiven Produktsicht eines Turbinenrotors.

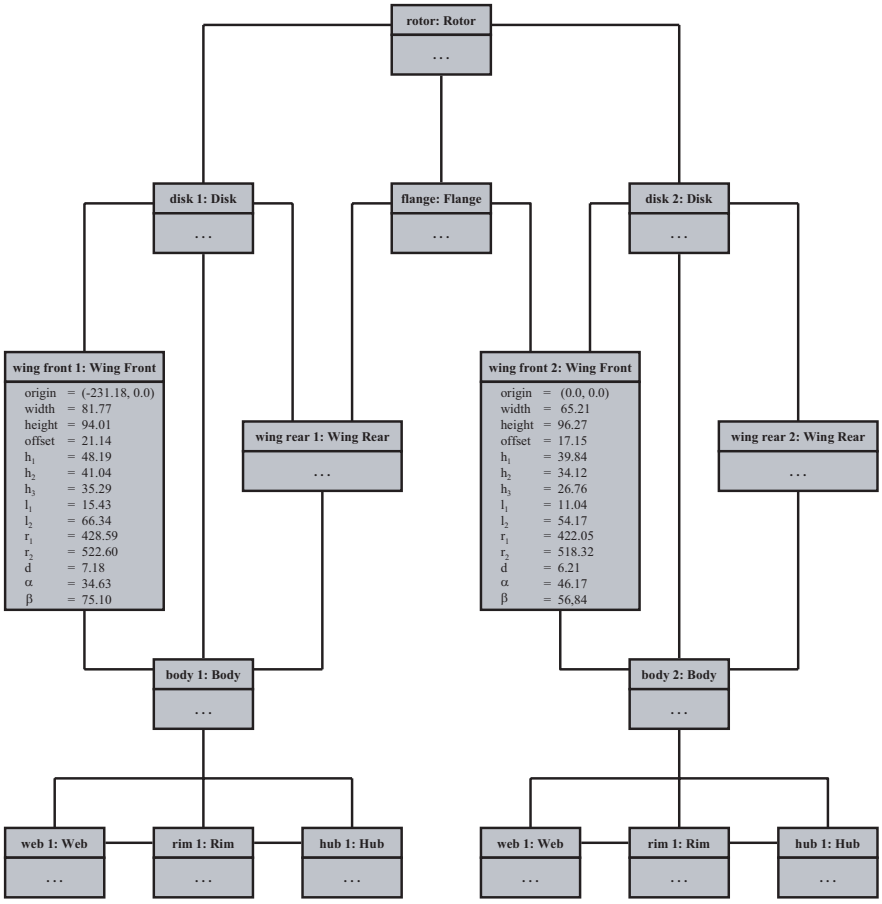

Abbildung 4.29: Attributiertes und typisiertes Partialmodell

Mittels einer Belegung der durch Attribute vorgegebenen Freiheitsgrade gemäß der Wertebereiche zugeordneter Sorten lässt sich schließlich ein Modell der Instanzebene ableiten, welches das betrachtete Fallbeispiel eines Turbinenrotors aus der Sicht der konstruktiven, parametrisierten Geometrieerstellung repräsentiert. Abbildung 4.29 zeigt auf der Grundlage der Beispiele 4.1 und 4.2 die Struktur, die zugeordneten Typen sowie exemplarische Attributwerte dieses Partialmodells. In analoger Weise können die gleichen Modellierungskonstrukte des Metamodells genutzt werden, um jede weitere der festgelegten Produktsichten zu erfassen.

4.4 Erweiterungen

In den vorangehenden Abschnitten 4.3.1 bis 4.3.4 werden die Basiskonzepte einer integrativen Produktmodellierung ausführlich motiviert, erläutert, diskutiert und daraus schließlich das in Abbildung 4.25 dargestellte Metamodell abgeleitet, welches es mit seinen Begriffen und Gesetzmäßigkeiten ermöglicht, Modelle technischer Produkte auf die in Beispiel 4.3 aufgezeigte Weise zu erstellen. So bietet das Metamodell allgemeine Modellierungskonstrukte zur Darstellung von Produktsichten, Produktkomponenten und Produktstrukturen, Komponententypen und elementarer Produkteigenschaften in Form von Attributen.

Die gewählten Konzepte erfüllen die Grundanforderungen, die an eine übergreifende und durchgängige Modellierungstechnik zur Darstellung komplexer Produkte zu stellen sind (vgl. Abschnitt 4.2). In diesem Sinne repräsentiert das entwickelte Metamodell den Kern einer solchen Modellierungstechnik und bietet ein solides Fundament für verschiedene Erweiterungen, die sich abhängig von der betrachteten Anwendungsdomäne, der Komplexität modellierter Produkte oder der angestrebten Ausdruckskraft eines Produktmodells als sinnvoll, wünschenswert oder notwendig darstellen können.

Im Folgenden wird ohne Anspruch auf Vollständigkeit eine Reihe möglicher Erweiterungen vorgestellt, welche sich auf einfache Weise und ohne wesentliche Änderungen bereits vorhandener Modellierungskonstrukte in das Metamodell integrieren lassen und dessen Ausdruckskraft entscheidend erhöhen können:

Sortenerweiterungen: Wie bereits in Abschnitt 4.3.4 kurz angedeutet, stellt die Hinzunahme neuer Sorten oder Möglichkeiten der Sortendefinition eine einfache, aber ggf. sehr wertvolle Erweiterungsmöglichkeit des Metamodells bzw. des in Abbildung 4.26 vorgestellten Sortenkonzepts dar. Neben der Einführung neuer Wertebereiche, beispielsweise für rationale oder komplexe Zahlen, könnten sich insbesondere Konzepte zur Definition sog. *varianter Sorten* – also Sorten die alternative Wertebereiche ermöglichen – sowie *rekursiver Sorten* als Gewinn bringend erweisen.

Attributerweiterungen: Das erarbeitete Metamodell sieht vor, dass jedes Attribut auf eine Sorte verweist, welche den Bereich möglicher Attributwerte festlegt. In gleicher Weise ließen sich Attribute um eine Reihe weiterer Charakteristika erweitern, um Eigenschaften erfasster Produktinformationen genauer zu spezifizieren. Beispiele hierfür sind

die Erfassung einer zugeordneten Maßeinheit für numerische Attributwerte, die Definition eines initialen Vorgabewerts, die Festlegung eines angestrebten Optimalwerts oder die Angabe zulässiger Toleranzen für Attributwerte.

Assoziationstypen: Assoziationen stellen ein allgemeines und frei verwendbares Modellierungskonstrukt des Metamodells zur Repräsentation von Beziehungen jeder Art zwischen verschiedenen Komponenten einer Produktstruktur dar. Analog zu dem in Abschnitt 4.3.3 beschriebenen Konzept der Typisierung von Produktbestandteilen können auch auftretende Assoziationen durch Assoziationstypen klassifiziert werden. Dies erlaubt es, die Menge zulässiger Assoziationen abhängig von einer betrachteten Produktsicht klar vorzugeben, ihre Semantik präzise zu definieren und eine Wiederverwendung zu ermöglichen.

Assoziationsattribute: Eine Erweiterung des Metamodells um Assoziationstypen (s.o.) bietet zugleich eine Grundlage, um Assoziationen gemäß dem Prinzip der Attributierung (s. Abschnitt 4.3.4) spezifische Eigenschaften in übergreifender Weise zuzuordnen. Dies ist beispielsweise immer dann sinnvoll, wenn eine Verbindung zweier Komponenten zusätzliche Informationen erfordert, welche sich nicht aus den Eigenschaften der betrachteten Komponenten ableiten lässt. Allerdings führt die Verwendung von Assoziationstypen und Assoziationsattributen zu einer erheblichen Komplexitätssteigerung.

N-äre Assoziationen: Den Gesetzmäßigkeiten des entwickelten Metamodells folgend, sind Assoziationen stets binär, verbinden also immer zwei verschiedene Komponenten einer Produktstruktur miteinander. Darüber hinaus kann es sinnvoll sein, n-äre Assoziationen zuzulassen, so dass durch eine Assoziation n verschiedene Komponenten zueinander in Beziehung gesetzt werden. Dies kann dazu beitragen, die Anzahl der Assoziationen eines Modells deutlich zu verringern, stellt jedoch keinen prinzipiellen Gewinn an Ausdruckskraft dar und erhöht die Komplexität des Metamodells.

Hierarchische Komponenten: Wie in Abschnitt 4.3.2 dargelegt wird, bietet eine hierarchische Produktstrukturierung im Vergleich zu der gewählten graphartigen Strukturerfassung des Metamodells zahlreiche Vorteile, ist jedoch nicht hinreichend allgemein, um stets Verwendung finden zu können. Dieser Gegensatz kann durch die zusätzliche Einführung hierarchischer Komponenten behoben werden, welche als spezielle Erweiterung des Metamodells Produktbestandteile repräsentieren,

die ausschließlich über Aggregationsbeziehungen zu untergeordneten Komponenten verfügen.

Sichtenorganisation: Der Ansatz der losen Integration (s. Abschnitt 4.3.1) erlaubt die Definition beliebiger Produktsichten als eigenständige Partialmodelle zur optimal angepassten Beschreibung ausgewählter Aspekte eines technischen Produkts. Vor diesem Hintergrund kann abhängig von der Komplexität eines Produkts bzw. der Granularität seiner Modellierung eine hohe Anzahl verschiedener Produktsichten auftreten. Entsprechend können geeignete Organisationsprinzipien wie beispielsweise die Nutzung von Hierarchien zur übersichtlichen Verwaltung von Produktsichten erforderlich werden.

Die genannten Punkte stellen eine Auswahl möglicher Erweiterungen des entwickelten Metamodells dar, die ohne Änderung der grundlegenden Konzeption integriert werden können. Aus einer detaillierten Betrachtung fachspezifischer Anforderungen sowie einer praxisgerechten Umsetzung des Ansatzes sind zahlreiche weitere Ergänzungen zu erwarten. Im Mittelpunkt einer Entscheidungsfindung über eine Aufnahme möglicher Erweiterungen steht stets die Abwägung zwischen dem Ausmaß der Komplexitätssteigerung des Metamodells – und somit seiner Instanzen – einerseits und dem Gewinn an Ausdruckskraft andererseits.

4.5 Zusammenfassung

Die Verwendung von Modellen ist ein in Wissenschaft und Technik unentbehrliches Hilfsmittel, um komplexe Sachverhalte zu strukturieren, Analysen durchzuführen und Schlussfolgerungen abzuleiten. Als wesentliches Prinzip der Modellbildung erscheint die Abstraktion. So werden aus der Menge aller Eigenschaften eines betrachteten Systems nur die Eigenschaften ausgewählt, die in Hinblick auf eine gegebene Zielsetzung relevant sind. Entsprechend stellt ein Produktmodell eine vereinfachte Nachbildung eines (technischen) Produkts dar, die im Vergleich zu realen Modellen mithilfe moderner Rechenanlagen wesentlich schneller und kostengünstiger bearbeitet werden kann.

Ein integriertes Produktmodell bildet zudem einen zentralen und entscheidenden Bestandteil einer Integrationsplattform, wie sie in Kapitel 3 beschrieben ist. Seine Aufgabe besteht darin, sämtliche über die verschiedenen Phasen eines Entwicklungsprozesses hinweg erstellten und bearbeiteten Teilmodelle und Eigenschaften eines technischen Produkts zu erfassen und sie zueinander in Beziehung zu setzen. Darüber hinaus bildet das Produktmodell

ein solides Fundament für eine durchgängige und übergreifende Koordination von Entwicklungsaktivitäten, beteiligten Bearbeitern sowie eingesetzten Software-Werkzeugen

Voraussetzung für die Erstellung von Produktmodellen ist eine Modellierungstechnik, welche die grundlegenden Modellierungskonzepte beschreibt und insbesondere ein Metamodell vorgibt. Dieses legt mit seinen Begriffen, Abstraktionen und Gesetzmäßigkeiten fest, auf welche Art und Weise Modelle aufgebaut sind und welche Produktinformationen enthalten sein können. Es bestimmt somit wesentlich die Ausdruckskraft, Angemessenheit und Umsetzbarkeit des angestrebten Modellierungsansatzes vor dem Hintergrund der betrachteten Anwendungsdomäne. Entsprechend sind sehr weit reichende Anforderungen zu beachten, wie in Abschnitt 4.2 erläutert.

Der Entwurf des Metamodells orientiert sich an vier Basiskonzepten, die in den Abschnitten 4.3.1 bis 4.3.4 ausführlich erläutert, diskutiert und illustriert werden. Zunächst gibt das Prinzip der *Segmentierung* einen übergreifenden Rahmen vor, um verschiedene Produktsichten – eine für den Bereich komplexer technischer Produkte besonders charakteristische Anforderung – innerhalb eines integrierten Produktmodells darzustellen und miteinander zu verknüpfen. Produktsichten werden hierbei als eigenständige Partialmodelle aufgefasst und über explizit zu definierende Konsistenzbedingungen zueinander in Beziehung gesetzt.

Die innere Struktur einzelner Produktsichten wird gemäß dem Konzept der *Strukturierung* durch Graphen, bestehend aus Produktbestandteilen und zugehörigen Verbindungen, modelliert. Vor dem Hintergrund der betrachteten Fallstudie erscheint nur ein solcher Ansatz als hinreichend ausdrucksstark, um durchgängig Verwendung finden zu können. Das Konzept der *Typisierung* entspricht schließlich der Forderung nach verschiedenen Abstraktionsebenen und erlaubt es, Produktbestandteile als gleichartig zu kennzeichnen, gemeinsame Eigenschaften in übergreifender Weise festzulegen und somit eine Wiederverwendung zu ermöglichen.

Das vierte und letzte Basiskonzept der entwickelten Modellierungstechnik ist die *Attributierung* strukturierter und typisierter Produktsichten, also die Erweiterung definierter Typen um Attribute zur Erfassung elementarer Produkteigenschaften. In ihrer Gesamtheit bestimmen die vier Basiskonzepte das erarbeitete Metamodell, welches als das wichtigste Ergebnis des vorliegenden Kapitels zu nennen ist. Die Verwendung der dort spezifizierten Modellierungskonstrukte wird anhand der durchgängigen Fallstudie eines Niederdruckturbinenrotors durch aufeinander aufbauende Beispiel ausführlich veranschaulicht.

Mit Ausnahme des Konzepts der Segmentierung, welches einen Lösungsansatz gemäß der für den Bereich komplexer technischer Produkte spezifi-

schen Anforderung nach sehr verschiedenartigen Produktsichten bildet, basiert das erarbeitete Metamodell auf wohlbekannten und bewährten Modellierungskonzepten der Informatik, angepasst an die betrachtete Anwendungsdomäne der Produktmodellierung auf der Grundlage einer sorgfältigen Diskussion. Zusätzlich bietet die Segmentierung einen Ansatzpunkt zur methodischen Verknüpfung des Produktmodells mit Aktivitäten des übergeordneten Entwicklungsprozesses sowie verwendeten Software-Werkzeugen.

5 Fundierung

In Kapitel 4 sind die grundlegenden Konzepte eines durchgängigen Produktmodells beschrieben, das als Kernstück einer übergreifenden Integrationsplattform in der Lage ist, Produktinformationen über verschiedene Phasen eines Entwicklungsprozesses hinweg zu erfassen und darüber hinaus ein solides Fundament für eine methodische Koordination einzelner Entwicklungsaktivitäten, beteiligter Bearbeiter und eingesetzter Werkzeuge zur Verfügung stellt. Ausgehend von einfachen und bewährten Modellabstraktionen sind allgemein verwendbare Modellierungskonstrukte abgeleitet, die in ihrer Gesamtheit das vorgestellte Metamodell bilden.

Die Intention der gewählten Modellierungskonzepte wird in Kapitel 4 ausführlich erläutert und die Art ihrer Verwendung anhand einer Reihe aufeinander aufbauender Beispiele der Fallstudie veranschaulicht. Zusätzlich wird implizit bereits eine graphische Beschreibungstechnik eingeführt, also eine *Syntax* zur Darstellung von Modellen technischer Produkte, die sich an der in Wissenschaft und Praxis weit verbreiteten Modellierungssprache *Unified Modeling Language* (UML) [UML03] orientiert. Ihre *Semantik* ist dagegen rein informell beschrieben (vgl. hierzu auch die Ausführungen zu Grundbegriffen der Modellbildung in Abschnitt 4.1).

Vor diesem Hintergrund besteht die Zielsetzung des vorliegenden Kapitels darin, eine formale und mathematisch präzise Fundierung der entwickelten Modellierungskonstrukte des Metamodells für integrierte Produktmodelle durch einen Kalkül typisierter und attributierter Graphen anzugeben. Im Einzelnen entspricht dies folgenden Zielen:

- Nur ein formales Modell ist hinreichend präzise, um die Semantik der entwickelten Modellierungskonstrukte in eindeutiger Weise zu definieren. Durch die Nutzung wohlbekannter mathematischer Grundkonzepte wie Mengen, Relationen und Abbildungen lassen sich Mehrdeutigkeiten oder Unklarheiten, die aus einer rein informellen Beschreibung des Metamodells resultieren, vermeiden und so ein falsches Verständnis der gewählten Modellabstraktionen ausschließen.

- Das Produktmodell ist ein wesentlicher und entscheidender Bestandteil einer übergreifenden Entwicklungsumgebung, die gemäß Kapitel 3 als Software-technische Integrationsplattform zur Anbindung verschiedenster Werkzeuge der Produktentwicklung dient. Entsprechend ist die semantisch präzise Festlegung des Produktmodells eine wichtige Voraussetzung für die Umsetzung der entwickelten Modellierungskonzepte im Rahmen eines Software-Systems.

- Nicht zuletzt bildet eine Fundierung des Produktmodells die Grundlage für eine Nutzung formaler Methoden zur Sicherung der Konsistenz verschiedener Produktsichten sowie ihrer Automatisierung, wie in Teil III der vorliegenden Arbeit erläutert wird. Diese setzen meist ein formales Modell der betrachteten Anwendungsdomäne voraus, welches vorhandene Modellstrukturen und auftretende Freiheitsgrade in geeigneter Weise erfasst.

Entsprechend der beschriebenen Zielsetzung ist die Gliederung der folgenden Abschnitte wie folgt gewählt: Zuerst stellt Abschnitt 5.1 allgemeine Grundlagen der Graphentheorie sowie die Technik der algebraischen Spezifikation vor, die für die angestrebte Fundierung des Produktmodells benötigt werden. Diese wird anschließend in Abschnitt 5.2 eingeführt und an Beispielen veranschaulicht. Auf der dadurch gegebenen Basis untersucht Abschnitt 5.3, wie sich Bearbeitungsschritte des Produktmodells, also Entwicklungsaktivitäten, in Form von Modelltransformationen erfassen lassen und klassifiziert diese.

5.1 Grundlagen

Für die formale Fundierung des Produktmodells werden zwei bewährte Theorien der Informatik bzw. Mathematik miteinander verknüpft: Zum einen wird das vielfach genutzte Konzept der *Graphen* eingesetzt, welche in natürlicher Weise geeignet sind, Produktstrukturen gemäß des in Abschnitt 4.3.2 gewählten Ansatzes zu beschreiben. Zur Erfassung elementarer Produkteigenschaften durch Attribute, wie in Abschnitt 4.3.4 ausgeführt, bietet sich zudem die Methode der *Algebraischen Spezifikation* an. Die folgenden beiden Abschnitte geben eine kurze Einführung in die erforderlichen Grundlagen der genannten Theorien.

5.1.1 Graphen

Graphen stellen ein einfaches und allgemein verwendbares Mittel zur formalen Modellierung beliebiger Strukturen dar. Als Ursprung der Graphentheorie gilt eine im Jahre 1736 von Leonhard Euler veröffentlichte Arbeit zur Lösung des sog. Königsberger Brückenproblems [Cha85]. Rasch sind daraufhin in vielerlei Bereichen die reichhaltigen Anwendungmöglichkeiten für Graphen erkannt worden, so dass sich die Graphentheorie als eigenes Forschungsgebiet entwickelt hat. Heute lassen sich viele Problemstellungen der Mathematik und Informatik durch Graphen erfassen und auf eine Untersuchung ihrer Eigenschaften zurückführen.

Im Bereich der Modellbildung erlauben Graphen eine einfache, intuitiv verständliche und sehr anschauliche, aber zugleich formale Beschreibung von Strukturen. Entsprechend vielfältig ist ihre Verwendung in unterschiedlichsten Anwendungsgebieten. So werden Graphen etwa genutzt zur Modellierung semantischer Netze [Sow83, CQ69], zur Beschreibung von Verbindungsstrukturen in Rechnernetzwerken oder Telekommunikationssystemen [Sch03], zur formalen Erfassung von Prozessabläufen durch Petri-Netze [Stü97] oder zur Darstellung von Verkehrswegen und Optimierung von Transportproblemen [Gal92].

Anschaulich besteht ein Graph aus einer Menge von Entitäten eines betrachteten Anwendungsbereichs, z.B. Produktbestandteilen, welche *Knoten* (engl.: *vertice*) genannt werden, sowie aus einer Menge von *Kanten* (engl.: *edge*), welche je zwei Knoten miteinander verbinden. Je nach der verfolgten Zielsetzung werden unterschiedliche Ausprägungen von Graphen betrachtet. So können beispielsweise Kanten gerichtet oder ungerichtet oder Knoten durch mehrere Kanten miteinander verbunden sein (vgl. [Cha85]). Die folgende Definition präzisiert daher den Graphbegriff, wie er im weiteren Verlauf der vorliegenden Arbeit verwendet wird:

Definition 5.1 (*Einfacher, endlicher Graph*)

Ein (einfacher und endlicher) Graph G ist ein Paar $G = (V, E)$ mit einer endlichen und nicht-leeren Knotenmenge $V \neq \emptyset$ sowie einer Kantenmenge $E \subseteq \{\{x, y\} : x, y \in V,\ x \neq y\}$.

Neben der reinen Erfassung der Struktur ist es häufig wünschenswert, weitere Eigenschaften eines betrachteten Systems in formale Graphenmodelle aufzunehmen. Hierzu können sowohl die Knoten, als auch die Kanten eines Graphen zusätzlich mit Markierungen versehen werden. Entsprechend entstehen Knoten- oder Kanten-markierte Graphen. Je nach Kontext und Art

der Markierung wird von markierten, attributierten, typisierten, gewichteten oder auch gefärbten Graphen gesprochen. Im Rahmen der vorliegenden Arbeit werden nur Knoten-markierte Graphen betrachtet gemäß folgender Definition:

Definition 5.2 (*Markierter Graph*)

Gegeben sei eine nicht-leere Menge M von Markierungen. Ein (einfacher und endlicher) markierter Graph G ist ein Tripel $G = (V, E, \mu)$, bestehend aus einem einfachen und endlichen Graphen (V, E) sowie einer (totalen) Abbildung $\mu : V \to M$.

Unter dem Begriff (markierter) Graph wird im Folgenden stets ein einfacher und endlicher (markierter) Graph verstanden. Das folgende Beispiel veranschaulicht die gegebenen Definitionen anhand eines sehr einfachen, abstrakten Graphen und stellt eine Notation für die Darstellung von Graphen vor:

Beispiel 5.1 (*Markierter Graph*)

Gegeben seien die Markierungen A und B, so dass $M := \{A, B\}$ die Menge der Markierungen bezeichne. Die Menge der Knoten bestehe aus den Elementen a_1, a_2, a_3, b_1, b_2, wobei jeder Knoten (zyklisch in dieser Reihenfolge) jeweils mit dem übernächsten Knoten durch eine Kante verbunden sei. Jeder der Knoten a_i, $1 \leq i \leq 3$, besitze die Markierung A, während die Knoten b_1 und b_2 über die Markierung B verfügen sollen. Es ergibt sich der markierte Graph $G = (V, E, \mu)$ mit

$$V := \{a_1, a_2, a_3, b_1, b_2\},$$

$$E := \{\{a_1, a_3\}, \{a_2, b_1\}, \{a_3, b_2\}, \{b_1, a_1\}, \{b_2, a_2\}\},$$

$$\mu : V \to M, \quad \mu(a_i) := A, \quad \mu(b_j) := B, \quad 1 \leq i \leq 3,\ 1 \leq j \leq 2,$$

illustriert in Abbildung 5.1.

Vergleicht man Abbildung 5.1 mit den in den Abschnitten 4.3.2 und 4.3.4 betrachteten Beispielen der Fallstudie, so ist bereits erkennbar, wie sich Graphen zur formalen Beschreibung eines Produktmodells nutzen lassen. Zuvor werden jedoch weitere Grundlagen für die in Abschnitt 5.2 angestrebte Fundierung benötigt. Schließlich sei darauf hingewiesen, dass die gegebenen Erläuterungen nur die grundlegendsten Begriffe der Graphentheorie vorstellen, wie sie im Rahmen der vorliegenden Arbeit benötigt werden. Weiter gehende Ausführungen zu diesem umfangreichen Themenfeld findet der interessierte Leser beispielsweise in [Cha85], [Die00] oder [Ste01].

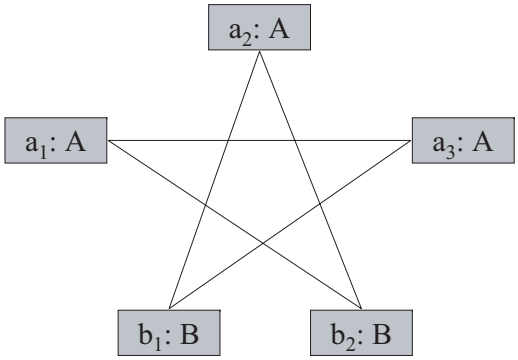

Abbildung 5.1: Darstellung eines markierten Graphen

5.1.2 Algebraische Spezifikation

Graphen bieten ein anschauliches Mittel, um Produktstrukturen, also die Unterteilung von Produkten in unterscheidbare Produktbestandteile und Beziehungen zwischen diesen, formal zu erfassen. Gemäß den in Kapitel 4 entwickelten Basiskonzepten der Produktmodellierung kann jeder Produktbestandteil darüber hinaus über elementare Produkteigenschaften in Form von Attributen verfügen (vgl. Abschnitt 4.3.4). Die Gestalt möglicher Ausprägungen eines Attributs wird durch die Verwendung unterschiedlicher Sorten eingegrenzt. Üblicherweise sind mit dem Begriff der *Sorte* die folgenden Vorstellungen verbunden:

- Eine Sorte fasst gleichartige Datenelemente zu einer *Trägermenge* zusammen und definiert auf diese Weise einen Wertebereich für mögliche Ausprägungen eines Attributs.
- Um Datenelemente der Trägermenge bearbeiten zu können, legt eine Sorte – ggf. unter Verwendung weiterer Sorten – eine Reihe zugehöriger *Operationen* fest.

Darüber hinaus hat es sich als nützlich erwiesen, zwischen der Spezifikation von Sorten und ihrer Implementierung zu unterscheiden: So lässt sich eine Zugriffssicht bilden, welche die grundlegenden Operationen, die für eine Sorte zur Verfügung stehen, deren Funktionalität sowie charakteristische Gesetzmäßigkeiten in Form einer Schnittstelle deklarativ beschreibt, ohne sich auf ihre Implementierung abzustützen. Die Benutzung einer Sorte erfordert somit nur die Kenntnis der Zugriffssicht, so dass Details der Implementierung verborgen bleiben und sich bei Bedarf unter Beibehaltung der Schnittstelle ändern lassen.

Vor dem Hintergrund der genannten Anforderungen und Zielsetzungen bietet die Technik der algebraischen Spezifikation einen idealen Rahmen, um Sorten als Grundelemente einer formalen Fundierung des Produktmodells zu erfassen. In Anlehnung an [EM85] und [Bro92, Bro95] werden im Folgenden die Grundzüge dieser Theorie vorgestellt und ihre Verwendung an Beispielen aufgezeigt. Im Mittelpunkt stehen die Begriffe *Signatur* und *Algebra*, welche zunächst definiert werden:

Definition 5.3 (*Signatur*)

Eine Signatur SIG ist ein Paar $SIG = (S, F)$, bestehend aus einer nichtleeren Menge $S = \{s_1, \ldots, s_m\}$, $m \in \mathbb{N}$, von Sortensymbolen und einer Menge $F = \{f_1, \ldots, f_n\}$, $n \in \mathbb{N}_0$, von Funktionssymbolen. Dabei ist jedem Funktionssymbol $f \in F$ eine Funktionalität $f : (s_{i_1}, \ldots, s_{i_k}) \rightarrow s_j$ mit $s_{i_1}, \ldots, s_{i_k}, s_j \in S$, $1 \leq i_1, \ldots, i_k, j \leq m$, $k \in \mathbb{N}_0$, zugeordnet. Im Falle $k = 0$, also $f : s_j$, $1 \leq j \leq m$, heißt f Konstantensymbol der Sorte $s_j \in S$.

Eine Signatur dient somit dazu, Bezeichnungen für Sorten und Operationen, sowie die Funktionalität von Operationen festzulegen. Darauf aufbauend definiert eine Algebra, auch *Rechenstruktur* genannt, zugehörige Trägermengen und Abbildungen:

Definition 5.4 (*Algebra*)

Gegeben sei eine Signatur $SIG = (S, F)$ mit $S = \{s_1, \ldots, s_m\}$, $m \in \mathbb{N}$, und $F = \{f_1, \ldots, f_n\}$, $n \in \mathbb{N}_0$. Eine Algebra ALG der Signatur SIG ist ein Paar $ALG = (S^{ALG}, F^{ALG})$ mit folgenden Eigenschaften:

(1) $S^{ALG} = \{s_1^{ALG}, \ldots, s_m^{ALG}\}$ besteht aus einer Familie nicht-leerer Trägermengen für die Sortensymbole $s_1, \ldots, s_m \in S$.

(2) $F^{ALG} = \{f_1^{ALG}, \ldots f_n^{ALG}\}$ besteht aus einer Familie von Abbildungen und Konstanten, wobei gilt:

   a) Ist $f \in F$ ein Konstantensymbol der Sorte $s \in S$, so ist f^{ALG} eine Konstante der Trägermenge $s^{ALG} \in S^{ALG}$, also $f^{ALG} \in s^{ALG}$.

   b) Verfügt $f \in F$ über die Funktionalität $f : (s_{i_1}, \ldots, s_{i_k}) \rightarrow s_j$ mit $s_{i_1}, \ldots, s_{i_k}, s_j \in S$, $1 \leq i_1, \ldots, i_k, j \leq m$, $k \in \mathbb{N}$, so ist f^{ALG} eine (partielle) Abbildung $f^{ALG} : s_{i_1}^{ALG} \times \ldots \times s_{i_k}^{ALG} \rightarrow s_j^{ALG}$.

Eine Algebra oder Rechenstruktur gibt demnach für jede der im Rahmen einer Signatur eingeführten Sorte eine Trägermenge an und ordnet jedem Konstanten- bzw. Operationssymbol eine entsprechende Konstante der Trägermenge bzw. eine Abbildung zu. Das folgende Beispiel veranschaulicht die Verwendung von Signaturen und Algebren anhand der Sorte *Boolean*.

Beispiel 5.2 (*Rechenstruktur der Wahrheitswerte*)

Die Sorte *Boolean* repräsentiert die Rechenstruktur der Wahrheitswerte. Entsprechend werden neben zwei Konstanten zur Darstellung elementarer boolscher Werte die logische Negation, die logische Konjunktion und die logische Disjunktion betrachtet. Es ergibt sich somit die Signatur $SIG = (S, F)$ mit

$$S = \{Boolean\} \quad \text{und} \quad F = \{true, false, not, and, or\},$$

wobei den angegebenen Konstanten- und Funktionssymbolen $f \in F$ die folgende Funktionalität zugeordnet sei:

$$\begin{aligned} true &: Boolean, \\ false &: Boolean, \\ not &: Boolean \to Boolean, \\ and &: (Boolean, Boolean) \to Boolean, \\ or &: (Boolean, Boolean) \to Boolean. \end{aligned}$$

Die zugehörige Algebra ist durch $ALG = (S^{ALG}, F^{ALG})$ gegeben mit

$$S^{ALG} = \mathbb{B} := \{\mathbf{O}, \mathbf{L}\},$$

sowie

$$F^{ALG} = \{true^{ALG}, false^{ALG}, not^{ALG}, and^{ALG}, or^{ALG}\},$$

wobei gilt:

$$\begin{aligned} true^{ALG} &:= \mathbf{L} \in \mathbb{B}, \\ false^{ALG} &:= \mathbf{O} \in \mathbb{B}, \\ not^{ALG} &: \mathbb{B} \to \mathbb{B}, \quad x \mapsto \begin{cases} \mathbf{L} & : \ x = \mathbf{O}, \\ \mathbf{O} & : \ x = \mathbf{L}, \end{cases} \\ and^{ALG} &: \mathbb{B} \times \mathbb{B} \to \mathbb{B}, \quad (x, y) \mapsto \begin{cases} \mathbf{L} & : \ x = y = \mathbf{L}, \\ \mathbf{O} & : \ \text{sonst}, \end{cases} \\ or^{ALG} &: \mathbb{B} \times \mathbb{B} \to \mathbb{B}, \quad (x, y) \mapsto \begin{cases} \mathbf{O} & : \ x = y = \mathbf{O}, \\ \mathbf{L} & : \ \text{sonst}. \end{cases} \end{aligned}$$

Eine Signatur legt lediglich Bezeichnungen fest und gibt die Funktionalität von Operationen an, ohne ihre Funktionsweise vorzugeben. So ließen sich zu der betrachteten Signatur für die Rechenstruktur der Wahrheitswerte auch weitere Algebren finden, deren Operationen ein anderes Verhalten aufweisen. Aus diesem Grunde ist es erforderlich, zusätzlich charakteristische Gesetzmäßigkeiten, *Axiome* genannt, zu spezifizieren, die in einer Algebra stets erfüllt sein sollen. Im betrachteten Beispiel könnte ein solches Axiom durch

$$\forall x \forall y : not(and(x, y)) = or(not(x), not(y))$$

beschrieben sein. Hierzu sind offenbar *Variablen*, *Terme* und *Termauswertungen* erforderlich, die in den folgenden Definitionen eingeführt werden:

Definition 5.5 (*Variable*)

Gegeben sei eine Signatur $SIG = (S, F)$ und für jedes $s \in S$ eine Menge X_s mit folgenden Eigenschaften:

(1) $X_{s_1} \cap X_{s_2} = \emptyset$ für alle $s_1, s_2 \in S$, $s_1 \neq s_2$,

(2) $X_s \cap F = \emptyset$ für alle $s \in S$.

Dann heißt X_s Menge von Variablen der Sorte s und $X := \bigcup_{s \in S} X_s$ Menge von Variablen der Signatur SIG.

Definition 5.6 (*Term*)

Gegeben sei eine Signatur $SIG = (S, F)$, für jedes $s \in S$ eine Menge X_s von Variablen der Sorte s, sowie $X := \bigcup_{s \in S} X_s$ als Menge von Variablen der Signatur SIG.

(1) Die Menge $T_s(X)$ der Terme der Sorte $s \in S$ ist induktiv als kleinste Menge mit folgenden Eigenschaften definiert:

   a) $X_s \subseteq T_s(X)$,

   b) Für jedes Konstantensymbol $f \in F$ der Sorte s gilt $f \in T_s(X)$,

   c) Hat $f \in F$ die Funktionalität $f : (s_1, \ldots, s_k) \to s$, $s_1, \ldots, s_k \in S$, $k \in \mathbb{N}$, und sind $t_1 \in T_{s_1}(X), \ldots, t_k \in T_{s_k}(X)$ Terme, so gilt $f(t_1, \ldots, t_k) \in T_s(X)$.

(2) $T_{SIG}(X) := \bigcup_{s \in S} T_s(X)$ heißt Menge der Terme der Signatur SIG.

(3) Für $s \in S$ heißt $T_s := T_s(\emptyset)$ Menge der Grundterme der Sorte s.

(4) $T_{SIG} := \bigcup_{s \in S} T_s$ heißt Menge der Grundterme der Signatur SIG.

Definition 5.7 (*Termauswertung*)

Gegeben sei eine Signatur $SIG = (S, F)$, eine Algebra $ALG = (S^{ALG}, F^{ALG})$ der Signatur SIG, für jedes $s \in S$ eine Menge X_s von Variablen der Sorte s, sowie $X = \bigcup_{s \in S} X_s$ als Menge von Variablen der Signatur SIG. Eine Abbildung $\alpha : T_{SIG}(X) \to \bigcup_{s \in S} s^{ALG}$ heißt Termauswertung, falls die folgenden Eigenschaften erfüllt sind:

(1) Ist $x \in X_s$ eine Variable der Sorte $s \in S$, so gilt $\alpha(x) \in s^{ALG}$.

(2) Ist $f \in F$ ein Konstantensymbol der Sorte $s \in S$, so gilt $\alpha(f) = f^{ALG}$.

(3) Ist $f \in F$ ein Funktionssymbol der Funktionalität $f : (s_1, \ldots, s_k) \to s$, $s_1, \ldots, s_k, s \in S$, $k \in \mathbb{N}$, und sind $t_1 \in T_{s_1}(X), \ldots, t_k \in T_{s_k}(X)$ Terme, so gilt (sofern definiert) $\alpha(f(t_1, \ldots, t_k)) = f^{ALG}(\alpha(t_1), \ldots, \alpha(t_k))$.

Variablen und Terme erlauben die Festlegung von Gesetzmäßigkeiten, die in einer Algebra erfüllt sein sollen. Auf diese Weise ist es möglich, das Verhalten der im Rahmen einer Signatur festgelegten Funktionssymbole durch Axiome abstrakt zu charakterisieren. Zunächst erfolgt die Definition dieses Begriffs:

Definition 5.8 (*Axiom, Gültigkeit*)

Gegeben sei eine Signatur $SIG = (S, F)$, eine Algebra $ALG = (S^{ALG}, F^{ALG})$ der Signatur SIG sowie eine Menge X von Variablen der Signatur SIG.

(1) Ein Axiom a (der Sorte $s \in S$ bzgl. der Signatur SIG) ist ein Paar $a = (l, r)$ mit $l, r \in T_s(X)$.

(2) Ein Axiom $a = (l, r)$ heißt gültig in der Algebra ALG, falls $\alpha(l) = \alpha(r)$ für jede Termauswertung $\alpha : T_{SIG}(X) \to \bigcup_{s \in S} s^{ALG}$ erfüllt ist.

Ein Axiom $a = (l, r)$ mit Variablen $x_1, \ldots, x_k$, $k \in \mathbb{N}$, entspricht der prädikatenlogischen Formel

$$\forall x_1 \ldots \forall x_k : l = r.$$

Entsprechend wird für a auch vereinfachend die Schreibweise $a : l = r$ verwendet. Betrachtet man eine Signatur zusammen mit einer Menge von Axiomen, so wird von einer *Spezifikation* gesprochen. Diese beschreibt die Zugriffssicht einer Rechenstruktur, ohne sich auf Details der Implementierung, also der konkreten Gestalt der Operationen einer Algebra, abzustützen.

Definition 5.9 (*Spezifikation*)

Gegeben sei eine Signatur $SIG = (S, F)$ und eine Algebra ALG der Signatur SIG mit $ALG = (S^{ALG}, F^{ALG})$.

(1) Eine Spezifikation $SPEC$ ist ein Tripel $SPEC = (S, F, A)$, wobei A eine Menge von Axiomen bzgl. der Signatur SIG ist.

(2) Die Algebra ALG erfüllt die Spezifikation $SPEC = (S, F, A)$, falls alle Axiome $a \in A$ in ALG gültig sind.

Ergänzend zu der in Beispiel 5.2 betrachteten Signatur für die Rechenstruktur der Wahrheitswerte definiert das folgende Beispiel eine Reihe von Axiomen, welche die dort beschriebene Algebra charakterisieren, ohne auf diese Bezug zu nehmen.

Beispiel 5.3 (*Axiome der Sorte Boolean*)

Wir betrachten die Spezifikation $SPEC = (S, F, A)$, wobei S und F wie in Beispiel 5.2 durch

$$S = \{Boolean\} \quad \text{und} \quad F = \{true, false, not, and, or\},$$

gegeben seien und die Konstanten- und Funktionssymbole $f \in F$ über die dort beschriebene Funktionalität verfügen. Zusätzlich werden die folgenden Axiome $a_1, \ldots, a_8$ gefordert:

$$\begin{aligned}
a_1 &: and(x, x) = x \\
a_2 &: and(x, y) = and(y, x) \\
a_3 &: and(and(x, y), z) = and(x, and(y, z)) \\
a_4 &: and(x, or(x, y)) = x \\
a_5 &: and(x, true) = x \\
a_6 &: and(x, or(y, z)) = or(and(x, y), and(x, z)) \\
a_7 &: and(x, not(x)) = false \\
a_8 &: or(x, y) = not(and(not(x), not(y)))
\end{aligned}$$

Wie sich leicht ermitteln lässt, erfüllt die in Beispiel 5.2 gegebene Algebra ALG mit den dort beschriebenen Operationen die gegebenen Axiome. Für das Axiom a_1 wird dies exemplarisch verifiziert:

(1) $x = false^{ALG} = \mathbf{O}$:

$$\begin{aligned} & and^{ALG}(x,x) = and^{ALG}(false^{ALG}, false^{ALG}) = \\ = \; & and^{ALG}(\mathbf{O},\mathbf{O}) = \mathbf{O} = false^{ALG} = x, \end{aligned}$$

(2) $x = true^{ALG} = \mathbf{L}$:

$$\begin{aligned} & and^{ALG}(x,x) = and^{ALG}(true^{ALG}, true^{ALG}) = \\ = \; & and^{ALG}(\mathbf{L},\mathbf{L}) = \mathbf{L} = true^{ALG} = x. \end{aligned}$$

Zusammenfassend bietet die vorgestellte Technik der algebraischen Spezifikation einen Rahmen, um Sorten als Grundelemente eines Produktmodells gemäß Abschnitt 4.3.4 durch Signaturen und Axiome zu spezifizieren sowie Wertebereiche und zugehörige Operationen mittels Algebren formal zu fundieren.

5.2 Modellierung

Nach den vorbereitenden Erläuterungen der letzten Abschnitte kann die formale Modellierung der in Kapitel 4 beschriebenen Konzepte eines integrierten Produktmodells erfolgen. Vorausgesetzt sei stets eine *Spezifikation* $SPEC$, welche eine Reihe $s_1, \ldots, s_k$, $k \in \mathbb{N}$, von *Sorten*, beispielsweise das in Abschnitt 4.3.4 vorgestellte Sortenkonzept, gemäß Definition 5.9 einführt. Entsprechend sei eine Algebra ALG gegeben, welche die Spezifikation $SPEC$ erfüllt und zu jedem Sortensymbol s_i eine Trägermenge s_i^{ALG}, $1 \leq i \leq k$, definiert. Auf dieser Grundlage bezeichnet

$$SORT := \{s_1, \ldots, s_k\}$$

die Menge der Sorten bzw. Sortensymbole, sowie

$$VAL := \bigcup_{1 \leq i \leq k} s_i^{ALG}$$

die Vereinigung aller Trägermengen, also das gegebene Werteuniversum der Datenelemente. Zusätzlich ordne die Abbildung

$$\nu : SORT \to \wp(VAL), \quad s_i \mapsto s_i^{ALG}, \quad 1 \leq i \leq k,$$

jedem Sortensymbol die zugehörige Trägermenge zu.

5.2.1 Typmodelle

Gemäß dem in Abschnitt 4.3.3 vorgestellten Basiskonzept der Typisierung bieten Typen die Möglichkeit, Produktbestandteile als gleichartig zu kennzeichnen und ihre Eigenschaften – in Form von Attributen – auf übergreifende Weise festzulegen. Einmal erstellte Komponententypen lassen sich somit für mehrere Komponenten eines Produktmodells wieder verwenden. Zusätzlich ist es vorteilhaft, Typen in einer Spezialisierungshierarchie anzuordnen, so dass Untertypen die Eigenschaften eines Obertyps übernehmen und an dessen Stelle treten können. Die folgende Definition präzisiert den Begriff des *Typmodells*:

Definition 5.10 (*Typmodell*)

Gegeben sei eine nicht-leere Menge ATT von Attributbezeichnungen. Ein Typmodell ist ein Tupel $(T, \sqsubseteq, \alpha, \sigma)$, bestehend aus folgenden Komponenten:

(1) T ist eine nicht-leere, endliche Menge, deren Elemente *Typen* genannt werden.

(2) $\sqsubseteq \subseteq T \times T$ ist eine Spezialisierungsrelation über der Menge T der Typen, d.h. eine partielle Ordnung mit folgenden Eigenschaften:

   a) $\sqsubseteq$ ist reflexiv, d.h. für jeden Typ $t \in T$ ist $(t, t) \in \sqsubseteq$.

   b) $\sqsubseteq$ ist antisymmetrisch, d.h. aus $t_1 \sqsubseteq t_2$ und $t_2 \sqsubseteq t_1$ folgt stets $t_1 = t_2$ für alle Typen $t_1, t_2 \in T$.

   c) $\sqsubseteq$ ist transitiv, d.h. mit $t_1 \sqsubseteq t_2$ und $t_2 \sqsubseteq t_3$ gilt auch $t_1 \sqsubseteq t_3$ für alle Typen $t_1, t_2, t_3 \in T$[1].

   d) Aus $t \sqsubseteq t_1$ und $t \sqsubseteq t_2$ folgt stets $t_1 \sqsubseteq t_2$ oder $t_2 \sqsubseteq t_1$ für alle Typen $t, t_1, t_2 \in T$. Dies garantiert, dass jeder Typ t über höchstens einen direkten Obertyp verfügt.

(3) Die Abbildung $\alpha : T \rightarrow \wp(ATT)$ ordnet jedem Typ eine endliche Menge von Attributbezeichnungen zu und heißt Attributzuordnung.

(4) Die Abbildung α ist mit $\sqsubseteq$ verträglich, d.h. für alle Typen $t_1, t_2 \in T$ folgt aus $t_1 \sqsubseteq t_2$ stets $\alpha(t_2) \subseteq \alpha(t_1)$[2].

[1] Die Eigenschaften a) bis c) besagen, dass $\sqsubseteq$ eine partielle Ordnung auf der Menge der Typen definiert.

[2] Dies bedeutet, dass ein Untertyp mindestens über die Attribute jedes Obertyps verfügt.

(5) Die partielle Abbildung $\sigma : T \times ATT \to SORT$ heißt Sortenzuordnung. Hierbei sei $\sigma(t, a)$ für $t \in T$ und $a \in ATT$ genau dann definiert, wenn $a \in \alpha(t)$ gilt.

(6) Die Abbildung σ ist mit $\sqsubseteq$ verträglich, d.h. für alle Typen $t_1, t_2 \in T$ folgt aus $t_1 \sqsubseteq t_2$ und $a \in \alpha(t_2)$ stets $\sigma(t_1, a) = \sigma(t_2, a)$[3].

Ein Typmodell basiert demnach auf einer nicht-leeren Menge ATT von Attributbezeichnungen, welche im Folgenden stets vorausgesetzt werde, sowie einer Menge T von Typen. Mittels der Abbildung α wird jedem Typ eine endliche Menge von Attributbezeichnungen zugeordnet. Diese sind jedoch nicht direkt mit einer Sorte verknüpft, da ansonsten Attribute gleichen Namens in verschiedenen Typen stets dieselbe Sorte aufweisen müssten. Vielmehr erfolgt die Zuordnung eines Attributs zu einer Sorte mittels der Abbildung σ unter Berücksichtigung des betrachteten Typs.

Spezialisierungsbeziehungen zwischen Typen werden durch die binäre Relation $\sqsubseteq \subseteq T \times T$ erfasst, an welche die in Abschnitt 4.3.3 beschriebenen Anforderungen gestellt werden. So besagen die Eigenschaften a), b) und c), dass jeder Typ als Unter- bzw. Obertyp seiner selbst auftritt, ein Typ nicht gleichzeitig echter Untertyp und echter Obertyp eines anderen Typs sein kann und die Spezialisierungsbeziehung zwischen Typen einen transitiven Charakter aufweist. Zusätzlich garantiert Eigenschaft d), dass ein Typ über höchstens einen direkten Obertyp verfügt.

Beispiel 5.4 (*Typmodell der Fallstudie*)

Das folgende Beispiel präsentiert eine formale Erfassung der in den Abschnitten 4.3.3 und 4.3.4 eingeführten Typen zur Beschreibung der konstruktiven Sicht eines Turbinenrotors auf der Grundlage der gegebenen Definition. Vorausgesetzt sei eine Spezifikation sowie eine zugehörige Algebra für die verwendeten Sorten `Pair`, `Real` und `Angle` (vgl. Beispiel 4.3). Darüber hinaus sei die Menge ATT der Attributbezeichnungen durch

$$\begin{aligned} ATT \;=\; \{&\texttt{origin}, \texttt{width}, \texttt{height}, \texttt{offset}, \texttt{h1}, \texttt{h2}, \texttt{h3}, \texttt{l1}, \texttt{l2},\\ &\texttt{r1}, \texttt{r2}, \texttt{d}, \texttt{alpha}, \texttt{beta}\} \end{aligned}$$

gegeben. Die Menge T der Typen ergibt sich gemäß Abbildung 4.24 zu

[3]Ein Typ kann die Sorten der von Obertypen übernommenen Attribute also nicht ändern.

$$\begin{aligned} T \;=\; \{&\texttt{Constructive Type}, \texttt{Stator Type}, \texttt{Rotor Type},\\ &\texttt{Casing Type}, \texttt{Rotor}, \texttt{Disk}, \texttt{Wing}, \texttt{Wing Front},\\ &\texttt{Wing Rear}, \texttt{Body}, \texttt{Web}, \texttt{Rim}, \texttt{Hub}, \texttt{Flange},\\ &\texttt{Flange A}, \texttt{Flange B}\}. \end{aligned}$$

Entsprechend umfangreich ist die Beschreibung der Spezialisierungsrelation $\sqsubseteq$, so dass an dieser Stelle nur der in Abbildung 4.28 dargestellte Ausschnitt angegeben wird:

$$\begin{aligned} \sqsubseteq \;=\; \{&\ldots, (\texttt{Wing Front}, \texttt{Wing Front}), (\texttt{Wing Front}, \texttt{Wing}),\\ &(\texttt{Wing Front}, \texttt{Rotor Type}), \ldots, (\texttt{Wing}, \texttt{Wing}),\\ &(\texttt{Wing}, \texttt{Rotor Type}), \ldots, (\texttt{Rotor Type}, \texttt{Rotor Type}), \ldots\}. \end{aligned}$$

Jedem Typ wird vermöge der Abbildung $\alpha : T \rightarrow ATT$ eine Menge von Attributbezeichnungen zugeordnet. So gilt für die Typen *Rotor Type* und *Wing Front*

$$\alpha(\texttt{Rotor Type}) = \{\texttt{origin}, \texttt{width}, \texttt{height}\},$$

sowie

$$\begin{aligned} \alpha(\texttt{Wing Front}) \;=\; \{&\texttt{origin}, \texttt{width}, \texttt{height}, \texttt{offset}, \texttt{h1}, \texttt{h2}, \texttt{h3},\\ &\texttt{l1}, \texttt{l2}, \texttt{r1}, \texttt{r2}, \texttt{d}, \texttt{alpha}, \texttt{beta}\}. \end{aligned}$$

Die Zuordnung von Attributen zu Sorten erfolgt schließlich mittels der Abbildung $\sigma : T \times ATT \rightarrow SORT$. So gilt beispielsweise

$$\begin{aligned} &\sigma(\texttt{Rotor Type}, \texttt{origin}) = \texttt{Pair},\\ &\sigma(\texttt{Wing Front}, \texttt{h1}) = \texttt{Real},\\ &\sigma(\texttt{Wing Front}, \texttt{alpha}) = \texttt{Angle}. \end{aligned}$$

Auf der Grundlage gegebener Sorten und Attributbezeichnungen definieren Typmodelle also die zur Beschreibung technischer Produkte verfügbaren Komponententypen und erfassen Spezialisierungsbeziehungen zwischen diesen.

5.2.2 Strukturmodelle

Dem in Kapitel 4 entwickelten Metamodell folgend, stehen zur Beschreibung von Produktstrukturen die Modellierungskonstrukte der *Komponente* und der *Assoziation* zur Verfügung. Komponenten repräsentieren Produktbestandteile und verfügen stets über einen Typ, der sie klassifiziert und verfügbare Attribute festlegt. Beziehungen zwischen verschiedenen Komponenten einer Produktstruktur werden durch (binäre) Assoziationen modelliert (vgl. Abschnitt 4.3.2). Ein Strukturmodell einer Produktsicht lässt sich demnach durch einen markierten Graphen beschreiben, gemäß der in Abschnitt 5.1.1 gegebenen Definition.

Definition 5.11 (*Strukturmodell*)

Gegeben sei ein Typmodell $(T, \sqsubseteq, \alpha, \sigma)$. Ein Strukturmodell ist ein markierter Graph (C, A, τ), wobei gilt:

(1) Die Elemente der Menge C werden Komponenten genannt.

(2) Die Elemente der Menge $A \subseteq \{\{c_1, c_2\} : c_1, c_2 \in C,\ c_1 \neq c_2\}$ werden Assoziationen genannt.

(3) Die Abbildung $\tau : C \to T$ heißt Typzuordnung.

Die Verwendung von Strukturmodellen in Verbindung mit Typmodellen wird anhand der in Abschnitt 4.3.2 vorgestellten Grundstruktur einer Turbinenscheibe aufgezeigt:

Beispiel 5.5 (*Strukturmodell der Fallstudie*)

Vorausgesetzt sei das in Beispiel 5.4 betrachtete Typmodell $(T, \sqsubseteq, \alpha, \sigma)$. Wie in Beispiel 4.1 beschrieben und in Abbildung 4.21 veranschaulicht, ergibt sich die Menge C des Strukturmodells (C, A, τ) eines Turbinenrotors zu

$$\begin{aligned} C \;=\; \{&\texttt{rotor}, \texttt{disk 1}, \texttt{disk 2}, \texttt{wing front 1},\\ &\texttt{wing rear 1}, \texttt{flange}, \texttt{wing front 2},\\ &\texttt{wing rear 2}, \texttt{body 1}, \texttt{body 2}, \texttt{web 1},\\ &\texttt{rim 1}, \texttt{hub 1}, \texttt{web 2}, \texttt{rim 2}, \texttt{hub 2}\}. \end{aligned}$$

Zwischen den gegebenen Komponenten existieren zahlreiche Assoziationsbeziehungen, die durch die Menge A beschrieben werden. Diese hat die

Gestalt

$$\begin{aligned} A \;=\; \{&\{\texttt{rotor}, \texttt{disk 1}\}, \{\texttt{rotor}, \texttt{flange}\}, \{\texttt{rotor}, \texttt{disk 2}\}, \\ &\{\texttt{disk 1}, \texttt{wing front 1}\}, \{\texttt{disk 1}, \texttt{body 1}\}, \\ &\{\texttt{disk 1}, \texttt{wing rear 1}\}, \{\texttt{disk 2}, \texttt{wing front 2}\}, \\ &\{\texttt{disk 2}, \texttt{body 2}\}, \{\texttt{disk 2}, \texttt{wing rear 2}\}, \\ &\ldots\}. \end{aligned}$$

Schließlich ordnet die Abbildung τ jeder Komponente $c \in C$ einen Typ gemäß des zugrunde liegenden Typmodells $(T, \sqsubseteq, \alpha, \sigma)$ zu. So gilt beispielsweise

$$\begin{aligned} &\tau(\texttt{rotor}) &&= \texttt{Rotor}, \\ &\tau(\texttt{disk 1}) &&= \texttt{Disk} &&= \tau(\texttt{disk 2}), \\ &\tau(\texttt{wing front 1}) &&= \texttt{Wing Front} &&= \tau(\texttt{wing front 2}), \\ &\tau(\texttt{wing rear 1}) &&= \texttt{Wing Rear} &&= \tau(\texttt{wing rear 2}), \\ &\ldots \end{aligned}$$

Durch eine Kombination der Abbildungen τ und α lässt sich für jede Komponente $c \in C$ die Menge $\alpha(\tau(c)) \subseteq ATT$ der zugeordneten Attribute ermitteln. Analog ergibt $\sigma(\tau(c), a) \in SORT$ für jedes $c \in C$ und $a \in \alpha(\tau(c))$ die zugeordnete Sorte.

Wie sich zeigt, erlauben Typ- und Strukturmodelle eine ausdrucksstarke und vergleichsweise einfache formale Erfassung von Produktstrukturen, Komponententypen und Attributen und decken so bereits weite Teile des erarbeiteten Metamodells ab.

5.2.3 Produktmodelle

Der Ansatz der losen Integration begreift ein Produktmodell als Reihe eigenständiger Produktsichten, die durch explizit festgelegte Konsistenzbedingungen zueinander in Beziehung gesetzt werden (vgl. Abschnitt 4.3.1). Jede Produktsicht kann eine individuelle Strukturierung aufweisen und eigene Komponententypen einführen. Wie in den Abschnitten 4.3.3 und 4.3.4 beschrieben, ergibt sich ein Partialmodell der Instanzebene, also die Repräsentation eines realen Produkts im Rahmen einer Produktsicht, durch eine Zuweisung von Werten der gegebenen Trägermengen zu Attributen vorhandener Komponenten gemäß festgelegter Sorten.

Definition 5.12 (*Belegung*)

Gegeben sei ein Typmodell $(T, \sqsubseteq, \alpha, \sigma)$ und ein Strukturmodell (C, A, τ). Eine Belegung ist eine partielle Abbildung

$$\beta : C \times ATT \to VAL.$$

Hierbei sei $\beta(c, a)$ für $c \in C$ und $a \in ATT$ genau dann definiert, wenn die Eigenschaft $a \in \alpha(\tau(c))$ erfüllt ist. In diesem Falle gelte $\beta(c, a) \in \nu(\sigma(\tau(c), a))$.

Eine Belegung ordnet demnach jedem Paar (c, a), bestehend aus einer Komponente $c \in C$ eines gegebenen Strukturmodells (C, A, τ) und einer Attributbezeichnung $a \in ATT$, ein Element $\beta(c, a) \in \nu(\sigma(\tau(c), a)) \subseteq VAL$ der gegebenen Trägermengen des Werteuniversums zu, sofern a in der Menge $\alpha(\tau(c)) \subseteq ATT$ enthalten ist, d.h. ein Attribut des durch $\tau(c) \in T$ festgelegten Komponententyps von c darstellt, entsprechend einem vorhandenen Typmodell $(T, \sqsubseteq, \alpha, \tau)$. Als Kombination eines Typmodells, eines Strukturmodells und einer Belegung ergibt sich in natürlicher Weise der Begriff der Produktsicht bzw. des Partialmodells:

Definition 5.13 (*Partialmodell, Produktsicht*)

Ein Partialmodell oder Produktsicht ist ein Tupel $S = (T, \sqsubseteq, \alpha, \sigma, C, A, \tau, \beta)$, wobei gilt:

a) $(T, \sqsubseteq, \alpha, \sigma)$ ist ein Typmodell.

b) (C, A, τ) ist ein Strukturmodell mit $\tau : C \to T$.

c) β ist eine Belegung mit $\beta : C \times ATT \to VAL$.

Beispiel 5.6 (*Konstruktive Produktsicht der Fallstudie*)

Im Rahmen von Beispiel 4.3 zeigt Abbildung 4.29 ein (vereinfachtes) Modell der durchgängigen Fallstudie eines Turbinenrotors aus der Sicht einer konstruktiven, parametrisierten Geometrieerstellung. Mit dem in Beispiel 5.4 eingeführten Typmodell $(T, \sqsubseteq, \alpha, \sigma)$ und dem in Beispiel 5.5 betrachteten Strukturmodell entspricht dieses der Produktsicht $S = (T, \sqsubseteq, \alpha, \sigma, C, A, \tau, \beta)$, wobei die Belegung $\beta : C \times ATT \to VAL$ den festgelegten Attributen die in

Abbildung 4.29 beschriebenen Datenelemente der Trägermengen zuweist. So gilt beispielsweise für die Komponente *wing front 1*:

$$\begin{aligned} \tau(\textit{wing front 1}) &= \textit{Wing Front}, \\ \alpha(\textit{Wing Front}) &= \{\textit{origin}, \textit{width}, \textit{height}, \ldots\}, \\ \sigma(\textit{Wing Front}, \textit{width}) &= \textit{Real}, \end{aligned}$$

sowie

$$\beta(\textit{wing front 1}, \textit{width}) = 81.77 \in \nu(\sigma(\tau(\textit{wing front 1}), \textit{width})).$$

Schließlich kann der Begriff des Produktmodells als Reihe eigenständiger Produktsichten präzisiert werden, gemäß folgender Definition:

Definition 5.14 (*Produktmodell*)

Ein Tupel $P = (S_1, \ldots, S_n)$ von Produktsichten $S_1, \ldots, S_n$, $n \in \mathbb{N}$, heißt Produktmodell[4].

Durch die insgesamt in Abschnitt 5.2 gegebenen Definitionen ist die formale Fundierung des Produktmodells auf der Grundlage von Graphen und Techniken der algebraischen Spezifikation abgeschlossen und eine formale Semantik durch eine Rückführung auf mathematische Grundkonstrukte wie Mengen, Relationen und Abbildungen eingeführt. Ein Produktmodell besteht in diesem Sinne aus einer endlichen Anzahl von Produktsichten, welche ihrerseits drei Anteile aufweisen: Ein Typmodell beschreibt zugrunde liegende Komponententypen, ihre Spezialisierungsbeziehungen und Attribute, ein Strukturmodell den inneren Aufbau einer Produktsicht, und eine Belegung schließlich die Komponenten-spezifische Zuweisung von Werten gegebener Trägermengen zu Attributen des Modells.

5.3 Transformationen

Nach einer formalen Erfassung der Begriffe zur Beschreibung eines Produktmodells und seiner Bestandteile werden in diesem Abschnitt Bearbeitungsschritte eines Produktmodells und somit Entwicklungsprozesse näher

[4]Die zusätzlich in einem Produktmodell enthaltenen Konsistenzbedingungen gemäß des in Kapitel 4 erarbeiteten Metamodells werden im Rahmen der formalen Modellierung vernachlässigt, jedoch ausführlich in Teil III der vorliegenden Arbeit behandelt.

betrachtet. Diese lassen sich formal als Transformationen der elementaren Bestandteile eines Produktmodells gemäß festgelegter Partialmodelle, Typmodelle, Strukturmodelle und Belegungen auffassen. Als Ausgangspunkt der Überlegungen sei eine Menge $\mathcal{M}$ an Produktmodellen und eine Menge $\mathcal{S}$ an Produktsichten entsprechend Definition 5.13 und Definition 5.14 vorausgesetzt. Allgemein betrachtet, entspricht eine Produktmodell-Transformation bzw. eine Sichten-Transformation somit einer Abbildung

$$\Phi : \mathcal{M} \to \mathcal{M} \qquad \text{bzw.} \qquad \Psi : \mathcal{S} \to \mathcal{S}.$$

Für ein gegebenes Produktmodell $P \in \mathcal{M}$ und Produktmodell-Transformationen $\Phi_1, \ldots, \Phi_k : \mathcal{M} \to \mathcal{M}$ bzw. eine Produktsicht $S \in \mathcal{S}$ und Sichten-Transformationen $\Psi_1, \ldots, \Psi_k : \mathcal{S} \to \mathcal{S}$, $k \in \mathbb{N}$, vereinbaren wir die Schreibweisen

$$P[\Phi_1, \ldots, \Phi_k] := (\Phi_k \circ \ldots \circ \Phi_1)(P)$$

und

$$S[\Psi_1, \ldots, \Psi_k] := (\Psi_k \circ \ldots \circ \Psi_1)(S).$$

Ausgehend von einem Produktmodell P, welches gemäß dem Prinzip der virtuellen Produktentwicklung (vgl. Abschnitt 2.1.3) zu Beginn eines Entwicklungsprozesses als initiales Modell vorliegt, kann jede Entwicklungsaktivität als Produktmodell-Transformation aufgefasst werden. In diesem Sinne entspricht ein (sequenzieller) Entwicklungsprozess einer Folge

$$P = P[Id] \mapsto P[\Phi_1] \mapsto P[\Phi_1, \Phi_2] \mapsto \ldots \mapsto P[\Phi_1, \ldots, \Phi_k],$$

wobei Id die Identitätsabbildung und $\Phi_1, \ldots, \Phi_k$ geeignete Produktmodell-Transformationen bezeichnen.

Ein Entwicklungsprozess lässt sich demnach als sukzessive Anwendung von Produktmodell-Transformationen auf ein initiales Produktmodell darstellen. Dies bietet einen wohlfundierten Ausgangspunkt für eine Integration von Produktmodellen und Entwicklungsprozessen gemäß folgender Überlegungen:

- Die Betrachtung von Bearbeitungsschritten als Produktmodell-Transformationen führt unmittelbar zu einem methodischen Verständnis des Zusammenhangs zwischen Entwicklungsaktivitäten und daran beteiligten Elementen eines Produktmodells.
- Die Darstellung eines Entwicklungsprozesses als Komposition einzelner Produktmodell-Transformationen ermöglicht es, Analysen seiner Eigenschaften auf Untersuchungen beteiligter Produktmodell-Transformationen zurückzuführen.

- Produktmodell-Transformationen bieten schließlich eine klare Grundlage zur Kategorisierung von Entwicklungsaktivitäten. Beispielsweise lassen sich Struktur-verändernde Bearbeitungsschritte auf bestimmte Rollen eines Organisationsmodells beschränken.

Betrachtet man im Folgenden nur solche Produktmodell-Transformationen $\Phi : \mathcal{M} \to \mathcal{M}$, welche die Anzahl der Partialmodelle eines Produktmodells $P = (S_1, \ldots, S_n) \in \mathcal{M}$, $n \in \mathbb{N}$, unverändert lassen, so gibt es offenbar stets Sichten-Transformationen $\Psi_1, \ldots, \Psi_n : \mathcal{S} \to \mathcal{S}$ mit

$$P[\Phi] = (S_1[\Psi_1], \ldots, S_n[\Psi_n]).$$

Aus diesem Grund ist es ausreichend, sich auf die Untersuchung von Sichten-Transformationen

$$\Psi : \mathcal{S} \to \mathcal{S}, \quad (T, \sqsubseteq, \alpha, \sigma, C, A, \tau, \beta) \mapsto (T', \sqsubseteq', \alpha', \sigma', C', A', \tau', \beta')$$

zu beschränken, die sich in nahe liegender Weise in drei Kategorien unterteilen lassen:

Instanz-Transformationen, welche lediglich eine Änderung einzelner Attributwerte bewirken. Ψ heißt Instanz-Transformation, falls

$$T' = T, \quad \sqsubseteq' = \sqsubseteq, \quad \alpha' = \alpha, \quad \sigma' = \sigma, \quad C' = C, \quad A' = A, \quad \tau' = \tau$$

erfüllt ist.

Struktur-Transformationen, welche die Struktur einer Produktsicht verändern können. In diesem Falle ist die Eigenschaft

$$T' = T, \quad \sqsubseteq' = \sqsubseteq, \quad \alpha' = \alpha, \quad \sigma' = \sigma$$

erfüllt[5].

Typ-Transformationen, die zugrunde liegende Komponententypen verändern. Ψ ist eine Typ-Transformation, falls

$$C' = C \quad \text{und} \quad A' = A$$

gilt[6].

[5] Man beachte, dass eine Struktur-Transformation aufgrund der potenziellen Hinzunahme neuer Komponenten auch eine Änderung der Belegung erfordern kann.

[6] Aufgrund der potenziellen Wegnahme eines Typs und der möglichen Modifikation von Attributen erfordert eine Typ-Transformation ggf. eine Änderung der Typzuordnung sowie der Belegung.

Die beschriebene Klassifizierung von Sichten-Transformationen lässt sich in natürlicher Weise auf Produktmodell-Transformationen übertragen. Offenbar repräsentieren Instanz-Transformationen die in einem intuitiven Sinne einfachsten Bearbeitungsschritte eines Produktmodells. Hierbei werden weder zugrunde liegende Typen oder Strukturen, sondern lediglich zugewiesene Attributwerte verändert. Struktur-Transformationen erlauben es, die Struktur von Produktsichten, also Komponenten, Assoziationen und zugewiesene Typen zu modifizieren, lassen Typmodelle jedoch unverändert. Deren Bearbeitung stellt offenbar die schwerwiegendste Änderung einer Produktsicht bzw. eine Produktmodells dar.

Auf der Grundlage der gegebenen Klassifizierung kann jede Sichten-Transformation $\Psi : \mathcal{S} \to \mathcal{S}$ (und in analoger Weise jede Produktmodell-Transformation $\Phi : \mathcal{M} \to \mathcal{M}$) in drei Anteile $\Psi_1, \Psi_2, \Psi_3 : \mathcal{S} \to \mathcal{S}$ gemäß

$$\Psi = \Psi_3 \circ \Psi_2 \circ \Psi_1$$

aufgeteilt werden, so dass Ψ_1 eine Typ-Transformation, Ψ_2 eine Struktur-Transformation und Ψ_3 eine Instanz-Transformation darstellt. Dies bietet einen Ausgangspunkt für eine methodische Verknüpfung von Produktmodell, Entwicklungsprozess und Organisationsmodell. So ließen sich Transformationen beispielsweise gemäß des Grades ihrer Auswirkung auf bestimmte Ressourcen, Rollen oder Personen eines Organisationsmodells beschränken.

Beispiel 5.7 (*Struktur-Transformation*)

Im Folgenden wird im Detail die Wirkungsweise einer exemplarischen Struktur-Transformation aufgezeigt. Wir betrachten hierzu das in Abbildung 5.2 dargestellte Beispiel zweier Strukturmodelle, die ineinander zu überführen sind. Vorausgesetzt sei also eine Produktsicht $S = (T, \sqsubseteq, \alpha, \sigma, C, A, \tau, \beta)$ entsprechend des linken Strukturmodells der gezeigten Abbildung mit

$$T = \{t_1, t_2, t_3\}, \quad \sqsubseteq = \emptyset, \quad C = \{c_1, c_2\}, \quad A = \{\{c_1, c_2\}\}$$

und

$$\begin{aligned} \alpha: &\quad t_1 \mapsto \{x\}, \quad t_2 \mapsto \{y\}, \quad t_3 \mapsto \{z\}, \\ \sigma: &\quad (t_1, x) \mapsto s, \quad (t_2, y) \mapsto s, \quad (t_3, z) \mapsto s, \\ \tau: &\quad c_1 \mapsto t_1, \quad c_2 \mapsto t_2, \end{aligned}$$

sowie einer Belegung

$$\beta : C \times ATT \to VAL.$$

Zu dieser ist eine neue Komponente c_3 hinzuzufügen, welche den Typ t_3 aufweist und mit den Komponenten c_1 und c_2 assoziiert ist. Die zugehörige

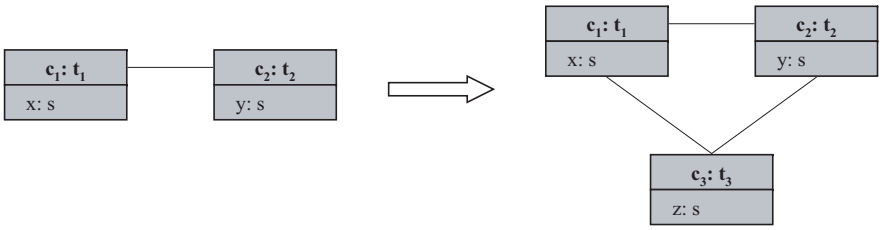

Abbildung 5.2: Struktur-Transformation

Struktur-Transformation Ψ bildet demnach die Produktsicht S auf eine Produktsicht $S' = (T, \sqsubseteq, \alpha, \sigma, C', A', \tau', \beta')$ ab, wobei gilt:

$$C' := C \cup \{c_3\} = \{c_1, c_2, c_3\},$$

$$A' := A \cup \{\{c_1, c_3\}, \{c_2, c_3\}\} = \{\{c_1, c_2\}, \{c_1, c_3\}, \{c_2, c_3\}\},$$

$$\tau' : C' \to T, \quad c \mapsto \begin{cases} \tau(c) & : \ c \in C, \\ t_3 & : \ c = c_3, \end{cases}$$

$$\beta' : C \times ATT \to VAL, \quad (c, a) \mapsto \begin{cases} \beta(c, a) & : \ c \in C, \\ \lambda & : \ (c, a) = (c_3, z) \end{cases}$$

für ein Datenelement $\lambda \in \nu(s) = \nu(\sigma(\tau'(c_3), z)) \subseteq VAL$, welches den Wert des Attributs $z \in ATT$ in Komponente $c_3 \in C'$ festlegt.

Wie in der gezeigten Weise, kann jeder Bearbeitungsschritt eines Produktmodells auf Änderungen der elementaren Bestandteile seiner formalen Repräsentation durch Mengen, Relationen und Abbildungen zurückgeführt werden. Dies erlaubt insgesamt sowohl eine präzise Klassifizierung von Bearbeitungsschritten, als auch ein genaues Verständnis ihrer Auswirkungen auf ein Produktmodell, und bietet so die Grundlage einer methodischen Produktentwicklung.

5.4 Zusammenfassung

Eine formale Fundierung der in Kapitel 4 entwickelten Modellabstraktionen verfolgt mehrere Zielsetzungen: Zum einen kann die zunächst informell beschriebene Semantik auf diesem Wege präzise erfasst werden, um Unklarheiten und Mehrdeutigkeiten auszuschließen. Dies stellt darüber hinaus

eine wichtige Voraussetzung für die Software-technische Umsetzung eines Produktmodells innerhalb einer übergreifenden Integrationsplattform dar. Schließlich bietet eine mathematisch fundierte Modellbildung die Grundlage für die Nutzung formaler Methoden zur Sicherung der Konsistenz verschiedener Produktsichten.

Die gewählte Form der Modellierung kombiniert zwei bewährte Ansätze: So eignen sich weithin bekannte Konzepte der Graphentheorie, um Produktstrukturen in natürlicher und einfacher Weise abzubilden. Für die Erfassung elementarer Produkteigenschaften durch Attribute, Sorten und zugeordnete Operationen bietet die Technik der algebraischen Spezifikation einen idealen Rahmen. Darauf aufbauend werden die Begriffe des *Typmodells*, des *Strukturmodells* und der *Belegung* eingeführt, um Produktsichten und schließlich Produktmodelle formal zu beschreiben. Ihre Verwendung wird wiederum anhand der durchgängigen Fallstudie aufgezeigt.

Auf der erarbeiteten Grundlage können schließlich Entwicklungsprozesse und einzelne Entwicklungsaktivitäten als sukzessiv durchgeführte Transformationen der elementaren Bestandteile eines formal beschriebenen Produktmodells aufgefasst werden. Eine solche Betrachtungsweise ermöglicht eine Kategorisierung von Bearbeitungsschritten, erlaubt ihre Dekomposition in elementare Modelltransformationen, vereinfacht somit zielgerichtete Analysen ihrer Wirkungsweisen und führt insgesamt zu einem methodisch klaren Verständnis des Zusammenspiels von Produktmodell, Prozessschritten und Organisationsaspekten.

Konsistenzsicherung

6 Techniken und Strategien

Ein integriertes Produktmodell bildet einen bedeutenden Bestandteil einer übergreifenden Entwicklungsplattform für komplexe technische Produkte, wie sie in Kapitel 3 beschrieben ist. Es besitzt primär die Aufgabe, Produktinformationen aus den sukzessiv durchlaufenen Phasen eines Entwicklungsprozesses in einheitlicher Weise und an zentraler Stelle zusammenzuführen, stellt zudem einen methodisch klaren Bezugspunkt für die Koordination erforderlicher Entwicklungsaktivitäten sowie daran beteiligter Bearbeiter dar, und dient schließlich als solides Fundament für eine Anbindung eingesetzter Software-Werkzeuge.

Dem in Kapitel 4 entwickelten Prinzip der Segmentierung folgend, wird ein integriertes Produktmodell als Reihe eigenständiger Partialmodelle konzipiert, welche sich unabhängig voneinander bearbeiten lassen. Jedoch resultiert aus dem gemeinsamen Bezugspunkt – dem zu entwickelnden Produkt – eine Vielzahl an Überschneidungen, Querbezügen und Abhängigkeiten zwischen verschiedenen Produktsichten. Somit ist explizit dafür Sorge zu tragen, dass das Gesamtmodell spätestens mit dem Abschluss eines Entwicklungsprozesses in sich widerspruchsfrei ist und ein potenziell reales Produkt beschreibt.

Diese gesondert zu betrachtende Gewährleistung der *Konsistenz* eines Produktmodells erscheint gegenüber dem Ansatz der engen Integration, in welchem vorhandene Produktsichten über ein zentrales Modell miteinander gekoppelt und auf diese Weise implizit konsistent sind, zunächst als Nachteil, da sie mit einem zusätzlichen Aufwand verbunden ist (vgl. Abschnitt 4.3.1). Andererseits eröffnet eine solche Vorgehensweise eine Reihe erstrebenswerter Vorteile, die sich aus einem flexiblen, aber dennoch kontrollierten und methodischen Umgang mit der Konsistenz eines Produktmodells bzw. vorhandener Inkonsistenzen ergeben.

So wird es beispielsweise möglich, in bestimmten Prozessphasen entsprechend dem kreativen und experimentellen Charakter einer Produktentwicklung vorübergehend Inkonsistenzen zwischen Produktsichten zuzulassen, sofern diese keine hinderlichen Auswirkungen auf den weiteren Fortgang der

Entwicklungsaktivitäten aufweisen und zu einem späteren Zeitpunkt aufgelöst werden können. Ebenso ließe sich durch eine sukzessive Erhöhung der Konsistenzanforderungen an Partialmodelle eine inkrementelle und iterativ ausgerichtete Produktentwicklung unterstützen, welche in frühen Entwicklungsphasen erforderliche Freiräume gewährt.

Ein segmentiertes Produktmodell bietet somit gegenüber dem Ansatz eines zentralen Modells ein deutlich erhöhtes Maß an Flexibilität hinsichtlich der Handhabung und Gewährleistung von Konsistenz. Diese erstreckt sich auf dreierlei Bereiche:

Flexible Festlegung von Querbezügen: Entsprechend der Reichhaltigkeit eines komplexen Produktmodells und der Vielfältigkeit denkbarer Querbezüge lassen sich verschiedenartige Techniken zur Festlegung gewünschter Beziehungen zwischen Produktsichten nutzen, welche von informellen Beschreibungen vorhandener Zusammenhänge bis hin zu streng formalen Ansätzen reichen.

Flexible Prüfung der Konsistenz: Eine explizite Festlegung vorhandener Querbezüge erlaubt eine flexible Überprüfung der Konsistenz, welche nicht nur den Zustand des Produktmodells selbst, sondern ebenso den Kontext des Entwicklungsprozesses berücksichtigt. So können abhängig von der aktuellen Entwicklungsphase unterschiedliche Konsistenzanforderungen eingebracht werden.

Flexible Sicherung der Konsistenz: Analog zur flexiblen Überprüfung der Konsistenz eines Produktmodells lässt sich auch im Rahmen der Auflösung ermittelter Inkonsistenzen zwischen Partialmodellen der Reifegrad verschiedener Produktsichten sowie der Kontext des Entwicklungsprozesses berücksichtigen, so dass ein optimal angepasstes Vorgehen gewählt werden kann.

Die genannten Punkte werden in den folgenden Abschnitten weiter ausgeführt. Beginnend präzisiert Abschnitt 6.1 den Begriff der Konsistenz eines integrierten Produktmodells. Darauf aufbauend geht Abschnitt 6.2 auf die drei grundlegenden Vorgehensbausteine einer flexiblen Konsistenzsicherung ein. Abschnitt 6.3 stellt anschließend eine Reihe bekannter Techniken vor, die zur Festlegung von Querbezügen zwischen Produktsichten genutzt werden können. Unterschiedliche Strategien der Konsistenzsicherung, insbesondere unter Einbeziehung zugrunde liegender Entwicklungsprozesse, werden schließlich in Abschnitt 6.4 betrachtet.

6.1 Grundbegriffe

Grundlegend für die Betrachtungen zur Konsistenz eines integrierten Produktmodells ist der Begriff der *Konsistenzbedingung*. Darunter verstehen wir allgemein eine Aussage über Produktsichten bzw. ihre Bestandteile wie beispielsweise Typen, Komponenten, Assoziationen oder Attribute, bei der zu jedem Zeitpunkt eines Entwicklungsprozesses entschieden werden kann, ob sie erfüllt oder nicht erfüllt ist. Man beachte, dass die getroffene Festlegung die konkrete Art der Formulierung einer Konsistenzbedingung offen lässt. Informelle, textuelle Beschreibungen sind somit ebenso möglich, wie formal spezifizierte Angaben (vgl. Abschnitt 6.2.1).

Wie aus dem in Kapitel 4 entwickelten Metamodell ersichtlich ist, setzt jede Konsistenzbedingung Produktsichten zueinander in Beziehung, indem sie Zusammenhänge zwischen ihren Elementen festlegt[1]. Entsprechend nennen wir zwei oder mehrere Produktsichten *adjazent*, falls eine Konsistenzbedingung existiert, in welche mindestens ein Element jeder der betrachteten Produktsichten eingeht. Zudem heißen eine Produktsicht und eine Konsistenzbedingung *inzident*, falls sich letztere auf mindestens ein Element der gegebenen Produktsicht bezieht, diese also potenziell Einfluss auf den Zustand der Konsistenzbedingung ausübt.

Auf dieser Grundlage wird ein Konsistenzbegriff für Produktsichten eingeführt. Hierzu betrachten wir zu einer gegebenen Menge von Produktsichten die Gesamtheit aller inzidenten Konsistenzbedingungen und wählen daraus jene Konsistenzbedingungen aus, die sich ausschließlich auf Elemente der betrachteten Sichten beziehen, also zu keiner der nicht betrachteten Produktsichten inzident sind. Sind alle Konsistenzbedingungen der so getroffenen Auswahl erfüllt, so nennen wir die gegebenen Produktsichten *konsistent*. Ist dagegen mindestens eine der Konsistenzbedingungen nicht erfüllt, so heißen die Produktsichten *inkonsistent*[2].

Beispiel 6.1 (*Konsistenz von Produktsichten*)

Anhand eines schematischen Beispiels werden die eingeführten Begriffe veranschaulicht. Vorausgesetzt sei das in Abbildung 6.1 dargestellte Produktmodell mit drei Produktsichten sowie zwei zugehörigen Konsistenzbedingungen, welche Querbezüge zwischen je zwei der vorhandenen Produktsichten festlegen. In der so beschriebenen Struktur sind die Sichten A und B sowie die

[1] Formal lässt sich die so entstehende Struktur aus Produktsichten und Querbezügen durch das Konzept des *Hypergraphen* [Har99] erfassen.

[2] Die gegebene Definition von Konsistenz ist nicht zu verwechseln mit dem Konsistenzbegriff der Logik, die Konsistenz nicht als Eigenschaft eines Modells, sondern als Widerspruchsfreiheit eines axiomatischen Systems begreift.

Sichten B und C adjazent, da sie über die Konsistenzbedingung AB bzw. BC zueinander in Beziehung gesetzt werden. Nicht adjazent sind dagegen die Sichten A und C, da diese nur mittelbar über Sicht B miteinander verbunden sind.

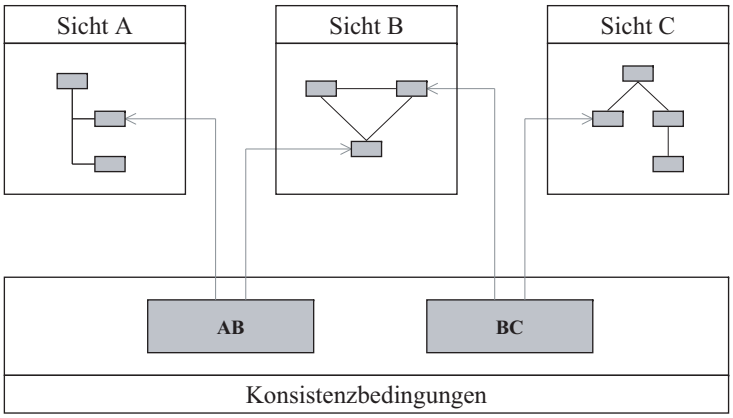

Abbildung 6.1: Exemplarisches Produktmodell

Analog ist Sicht A nur zu der Konsistenzbedingung AB und Sicht C nur zu der Konsistenzbedingung BC inzident, während Sicht B eine Inzidenz zu beiden Konsistenzbedingungen aufweist. Für die Prüfung der Konsistenz der Sichten A und B ist zunächst die Gesamtheit aller zu Sicht A oder zu Sicht B inzidenten Konsistenzbedingungen zu ermitteln, welche offenbar beide Konsistenzbedingungen umfasst. Daraufhin sind nur die Bedingungen zu betrachten, die sich ausschließlich auf die Sichten A und B beziehen. Maßgeblich für die Konsistenz dieser Sichten ist also nur die Konsistenzbedingung AB, da die Bedingung BC eine weitere, nicht betrachtete Sicht involviert.

In nahe liegender Weise kann der für Produktsichten gefundene Konsistenzbegriff auf das Produktmodell insgesamt erweitert werden. In diesem Zusammenhang ist es zweckmäßig, unterschiedliche Grade der Konsistenz zu unterscheiden:

Globale Konsistenz: Wir nennen ein Produktmodell *(global) konsistent*, falls alle adjazenten Produktsichten konsistent sind. Offenbar ist dies genau dann zutreffend, wenn alle festgelegten Konsistenzbedingungen erfüllt sind.

Lokale Konsistenz: Von *lokaler Konsistenz* eines Produktmodells sprechen wir, falls adjazente Produktsichten existieren, die konsistent sind. Die lokale Konsistenz stellt somit einen abgeschwächten Konsistenzbegriff dar.

Relative Konsistenz: Als weitere Differenzierung des Konsistenzbegriffs bezeichnen wir ein Produktmodell als *relativ konsistent* zu einer vorgegebenen, fest gewählten Konsistenzbedingung, sofern diese zu dem betrachteten Zeitpunkt erfüllt ist.

Offenbar folgt aus der (globalen) Konsistenz eines Produktmodell stets seine lokale Konsistenz, welche ihrerseits relative Konsistenz impliziert. In gleicher Weise lassen sich verschiedene Formen der Inkonsistenz eines segmentierten Produktmodells benennen:

Globale Inkonsistenz: Ein Produktmodell heißt *global inkonsistent*, falls es keine konsistenten Produktsichten gibt, also jede Auswahl adjazenter Produktsichten inkonsistent ist.

Lokale Inkonsistenz: Wir bezeichnen ein Produktmodell als *(lokal) inkonsistent*, falls es inkonsistente Produktsichten gibt, also mindestens eine der aufgestellten Konsistenzbedingungen nicht erfüllt ist[3].

Relative Inkonsistenz: Schließlich nennen wir ein Produktmodell *relativ inkonsistent* zu einer vorgegebenen Konsistenzbedingung, falls diese zu dem betrachteten Zeitpunkt nicht erfüllt ist.

Wie sich leicht erkennen lässt, stellen lokale und relative Inkonsistenz äquivalente Begriffe dar. Die globale Inkonsistenz eines Produktmodells impliziert ferner sowohl lokale, als auch relative Inkonsistenz. Intuitiv verbindet sich mit dem Begriff der Konsistenz eines Produktmodells die Vorstellung, dass alle aufgestellten Konsistenzbedingungen erfüllt sind. Umgekehrt besteht die Erwartung, dass in einem inkonsistenten Produktmodell stets mindestens eine Konsistenzbedingung nicht erfüllt ist. Durch die Begriffe der (globalen) Konsistenz bzw. der (lokalen) Inkonsistenz werden diese Auffassungen widergespiegelt.

Vor dem Hintergrund der eingeführten Begriffe lässt sich die Entwicklung eines technischen Produkts als ein zielgerichteter Prozess der Konsistenzsicherung ansehen. Zunächst führen einzelne Entwicklungsschritte meist dazu, dass Inkonsistenzen zwischen Produktsichten auftreten. Anhand explizit

[3]Man beachte, dass ein (lokal) inkonsistentes Produktmodell sowohl lokal, als auch relativ konsistent sein kann.

definierter Konsistenzbedingungen lassen sich diese bedarfsgerecht, etwa zu festgelegten Meilensteinen, auffinden und die lokale Konsistenz eines Produktmodells – also die Konsistenz einzelner Sichten – herstellen. Mit dem Abschluss einer Produktentwicklung ist schließlich die globale Konsistenz eines Produktmodells – also die Konsistenz aller Sichten – zu gewährleisten.

In ihrer Gesamtheit bilden Konsistenzbedingungen ein komplexes Netzwerk, das den Lösungsraum valider Produktmodelle sukzessive einschränkt. Dabei ist darauf zu achten, dass nicht widersprüchliche Konsistenzbedingungen auftreten, es also stets möglich ist, ein Produktmodell zu finden, welches allen aufgestellten Konsistenzbedingungen genügt. In diesem Zusammenhang bezeichnen wir eine gegebene Menge an Konsistenzbedingungen als

- *tautologisch*, falls alle Konsistenzbedingungen stets erfüllt sind, unabhängig vom Zustand des Produktmodells,
- *erfüllbar*, falls ein Produktmodell existiert, so dass alle Konsistenzbedingungen erfüllt sind,
- *unerfüllbar*, falls stets mindestens eine Konsistenzbedingung nicht erfüllt ist, also kein valides Produktmodell existiert.

Offensichtlich schränkt eine tautologische Menge an Konsistenzbedingungen den Lösungsraum valider Produktmodelle in keiner Weise ein und führt somit stets zu einem global konsistenten Produktmodell, während eine unerfüllbare Menge an Konsistenzbedingungen ein stets inkonsistentes Produktmodell impliziert. Eine erfüllbare Menge an Konsistenzbedingungen lässt dagegen alle drei der beschriebenen Konsistenzgrade zu. In den folgenden Ausführungen sei daher stets vorausgesetzt, dass die aufgestellten Konsistenzbedingungen in ihrer Gesamtheit erfüllbar sind, da nur dies ein der Realität angemessenes Szenario darstellt.

6.2 Konzeption

Im Rahmen des übergreifenden Ansatzes eines segmentierten Produktmodells spielt der erläuterte Begriff der Konsistenz offenbar eine entscheidende Rolle. Darauf aufbauend werden in den folgenden Ausführungen die Grundbausteine einer flexiblen Behandlung von Konsistenz vorgestellt. Dabei wird zwischen der *Spezifikation* einer Konsistenzbedingung, d.h. ihrer Festlegung auf der Grundlage vorhandener Elemente eines Produktmodells, ihrer *Interpretation*, also der Prüfung, ob das aktuell gültige Modell der festgelegten Bedingung genügt, und ihrer *Resolution* als ggf. erforderliche Wiederherstellung relativer Konsistenz unterschieden.

6.2.1 Spezifikation

Die explizite *Spezifikation* der zwischen den Partialmodellen eines segmentierten Produktmodells bestehenden Konsistenzbedingungen stellt einen wesentlichen Bestandteil unseres Ansatzes dar. Obwohl Produktsichten darin als eigenständig aufgefasst werden, existiert aufgrund des übergreifenden Bezugs auf ein gemeinsames Produkt in natürlicher Weise eine Vielzahl an Überschneidungen, Querbezügen und Abhängigkeiten, welche die fachlichen Zusammenhänge des Anwendungsbereichs widerspiegeln. Diese werden gemäß des in Kapitel 4 entwickelten Metamodells durch eigene Modellelemente repräsentiert (vgl. Abbildung 4.16 in Abschnitt 4.3.1).

Angesichts der Vielfältigkeit denkbarer Querbezüge zwischen Partialmodellen ist nicht zu erwarten, dass eine einzelne, fest gewählte Technik der Spezifikation ausreichend ist, um alle Konsistenzbedingungen angemessen zu beschreiben. Zwar ließe sich – abstrakt betrachtet – jede Bedingung innerhalb eines hinreichend ausdrucksstarken Kalküls der Wissensrepräsentation, beispielsweise auf Grundlage der Prädikatenlogik erster Ordnung [Sch95], spezifizieren. Jedoch sind zahlreiche Beziehungen nur schwer in einem mathematisch exakten Sinne erfassbar, so dass in der Praxis häufig informelle, umgangssprachliche Formulierungen bevorzugt werden[4].

Aus diesen Gründen lässt der in Abschnitt 6.1 bewusst allgemein gefasste Begriff der Konsistenzbedingung verschiedene Techniken der Spezifikation zu. Es wird lediglich gefordert, dass zu jedem Zeitpunkt eines Entwicklungsprozesses in endlicher Zeit und ggf. abhängig vom Entwicklungskontext (s. Abschnitt 6.2.2) festgestellt werden kann, ob eine Bedingung erfüllt oder nicht erfüllt ist[5]. So können Konsistenzbedingungen flexibel in einer optimal angepassten Weise formuliert werden, sei es als prädikatenlogische Formel, als informelle Beschreibung oder als Anweisungsfolge einer zugrunde liegenden Programmiersprache.

In umgekehrter Weise strebt der Ansatz eines segmentierten Produktmodells wie in Abschnitt 4.3.1 schematisch aufgezeigt unter ausdrücklicher Berücksichtigung verschiedener Spezifikationstechniken und Auswertungsverfahren eine integrierte, gleichartige Behandlung von Konsistenzbedingungen an. Daher ist ein einheitlicher Rahmen für Konsistenzbedingungen vorzugeben, der durch das in Kapitel 4 erarbeitete Metamodell festgelegt werden kann. Im Einzelnen gehen wir davon aus, dass jede Konsistenzbedingung

[4] Analog könnten verschiedene Konsistenzbedingungen durch logische Konjunktion zu einer Bedingung verknüpft werden. Dem entgegen steht jedoch die Zielsetzung einer möglichst differenzierten Betrachtung der Konsistenz eines Produktmodells.

[5] Im Bereich der Theoretischen Informatik entspricht dies dem Begriff der *Entscheidbarkeit* eines Prädikats bzw. einer Menge (s. z.B. [Bro95] oder [Sch97]).

einen gleichartigen Aufbau aufweist, der mindestens die folgenden Bestandteile umfasst:

Identifikator: Jeder Konsistenzbedingung wird ein *Identifikator* zugeordnet, der diese in der Menge aller Konsistenzbedingungen eindeutig kennzeichnet.

Spezifikation: Die *Spezifikation* einer Konsistenzbedingung beschreibt die an ein Produktmodell gestellte Anforderung. An dieser Stelle können wie erläutert verschiedenartige Techniken zum Einsatz kommen.

Zustand: Schließlich weist jede Konsistenzbedingung einen Zustand auf, der angibt, ob sie zum gegenwärtigen Zeitpunkt erfüllt ist, nicht erfüllt ist oder noch nicht überprüft ist.

Über die genannten Punkte hinaus sind zahlreiche weitere Attribute für Konsistenzbedingungen vorstellbar. So wäre es beispielsweise möglich, Autor und Zeitpunkt der Erstellung einer Konsistenzbedingung zu erfassen oder anzugeben, welcher Aufwand mit ihrer Überprüfung verbunden ist und welche Auswertungsmöglichkeiten zur Verfügung stehen.

Beispiel 6.2 (*Spezifikation einer Konsistenzbedingung*)

Eine wichtige Kategorie von Querbezügen zwischen Partialmodellen eines komplexen Produkts besteht aus Beziehungen, welche die innere Struktur der vorhandenen Produktsichten, also ihren Aufbau aus Komponenten und Assoziationen, betreffen (vgl. Abschnitt 4.3.2). So stellt etwa die in Abschnitt 2.2.2 beschriebene konstruktive Sicht der Fallstudie eines Turbinenrotors eine strukturelle Verfeinerung der entsprechenden Konfigurationssicht dar. Als vergleichsweise einfaches Beispiel betrachten wir Produktsichten, die eine isomorphe, d.h. unter Vernachlässigung der Komponenten- und Assoziationsbezeichnungen gleiche Struktur aufweisen.

Abbildung 6.2 zeigt ein entsprechendes Produktmodell mit zwei Produktsichten sowie eine zugehörige Konsistenzbedingung. In diesem Rahmen lässt sich die Forderung nach isomorphen Strukturen der Sichten A und B auf unterschiedliche Weise formulieren. So könnte die Spezifikation der gezeigten Konsistenzbedingung beispielsweise informell durch

„*Die Strukturen der Sichten A und B sind isomorph*“

festgelegt sein. Auf der Grundlage der in Kapitel 5 vorgenommenen Fundierung könnte ausgehend von Strukturmodellen (C_1, A_1, τ_1) für Sicht A und

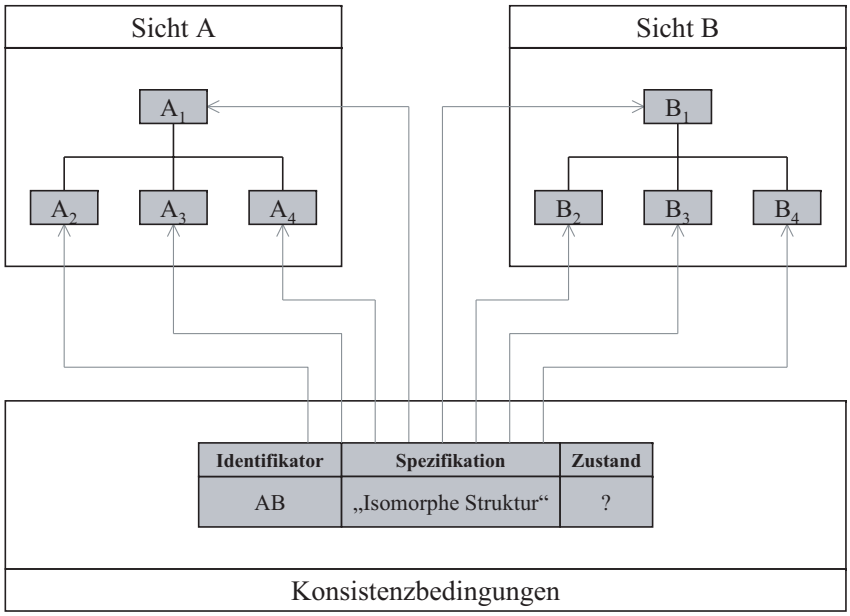

Abbildung 6.2: Spezifikation einer Konsistenzbedingung

(C_2, A_2, τ_2) für Sicht B (s. Abschnitt 5.2) die gleiche Konsistenzbedingung präzise als prädikatenlogische Formel in der Form

$$\begin{array}{lllll} \exists f \in C_2^{C_1}(& \forall c_2 \in C_2 & \exists c_1 \in C_1 & (f(c_1) = c_2) & \wedge \\ & \forall c \in C_1 & \forall c' \in C_1 & (f(c) = f(c') \rightarrow c = c') & \wedge \\ & \forall c \in C_1 & \forall c' \in C_1 & (\{c, c'\} \in A_1 \leftrightarrow \{f(c), f(c')\} \in A_2) &) \end{array}$$

angegeben sein, wobei $C_2^{C_1}$ die Menge aller Abbildungen $f : C_1 \rightarrow C_2$ bezeichne. Diese beschreibt den Isomorphiebegriff der Graphentheorie, demzufolge Sicht A und Sicht B eine isomorphe Struktur aufweisen, falls es zwischen den zugehörigen Komponenten eine bijektive Abbildung $f : C_1 \rightarrow C_2$ gibt, so dass zwei Komponenten in Sicht A genau dann miteinander durch eine Assoziation verbunden sind, wenn auch die vermöge der Abbildung f korrespondierenden Komponenten in Sicht B zueinander assoziiert sind.

Wie das gezeigte Beispiel verdeutlicht, ist für die Spezifikation einer Konsistenzbedingung eine Entscheidung zu treffen zwischen einer eher informellen und leicht verständlichen, jedoch häufig ungenauen und daher mehrdeutigen Beschreibung einerseits und einer formalen, mathematisch präzisen, aber oftmals komplexen und somit schwer verständlichen Formulierung andererseits. Diese hat zudem wesentlichen Einfluss auf den weiteren Verlauf eines

Konsistenzsicherungsprozesses. Während informell spezifizierte Konsistenzbedingungen ausschließlich manuell verifizierbar sind, erlaubt eine formale Spezifikation die Nutzung automatisierbarer Verfahren.

Zusammenfassend lässt sich festhalten, dass der vorgeschlagene Ansatz eines segmentierten Produktmodells eine explizite Behandlung der Konsistenz erfordert. Als erster Schritt sind die aus fachlichen Zusammenhängen resultierenden Querbezüge zwischen Partialmodellen in Form von Konsistenzbedingungen zu spezifizieren. Eine praxisgerechte Lösung erfordert es, hierbei einerseits variable Techniken der Spezifikation zuzulassen, welche sowohl formale als auch informelle Beschreibungsmöglichkeiten umfassen. Andererseits ist für die Zielsetzung einer gleichartigen Behandlung von Konsistenzbedingungen ein einheitlicher Rahmen vorauszusetzen.

6.2.2 Interpretation

Unter dem Begriff *Interpretation* verstehen wir die Ermittlung des Zustands einer Konsistenzbedingung, welcher angibt, ob diese durch das gegenwärtige Produktmodell erfüllt oder nicht erfüllt ist. Interpretation bezeichnet also den Vorgang der Überprüfung einer ausgesuchten Konsistenzbedingung bzw. eines Produktmodells, welcher sich definitionsgemäß stets in endlicher Zeit durchführen lässt[6]. Ist ein Produktmodell zu einer betrachteten Konsistenzbedingung relativ konsistent, so sprechen wir auch davon, dass diese *erfüllt*, *gültig* oder *zutreffend* ist. Andernfalls nennen wir sie *nicht erfüllt*, *ungültig* oder *verletzt*.

Die Betrachtung der Interpretation einer Konsistenzbedingung als expliziten und eigenständigen Bestandteil des Konsistenzsicherungsprozesses eröffnet eine flexible und praxisgerechte Handhabung des Konsistenzbegriffs. Dies bedeutet, eine Interpretation kann über die Spezifikation einer Konsistenzbedingung und den Zustand inzidenter Produktsichten hinaus den aktuellen Kontext eines Entwicklungsprozesses berücksichtigen. So wird es möglich, Konsistenzbedingungen in frühen Prozessphasen zu vernachlässigen und erst mit steigendem Reifegrad eines Produktmodells eine zunehmend restriktive Konsistenzsicherung zu betreiben.

In Abbildung 6.3 ist das Grundprinzip der Interpretation veranschaulicht. In die Interpretation einer Konsistenzbedingung gehen demnach die Konsistenzbedingung selbst sowie jede betroffene, d.h. inzidente Produktsicht ein. Mittels des zugrunde liegenden Prozessmodells wird jedoch ebenso der aktuelle Zustand des Entwicklungsprozesses berücksichtigt. Analog ist es

[6]Im Rahmen eines praxisgerechten Ansatzes spielt zudem die Betrachtung des Aufwands einer Interpretation für eine möglichst effiziente Überprüfung von Konsistenzbedingungen eine wichtige Rolle.

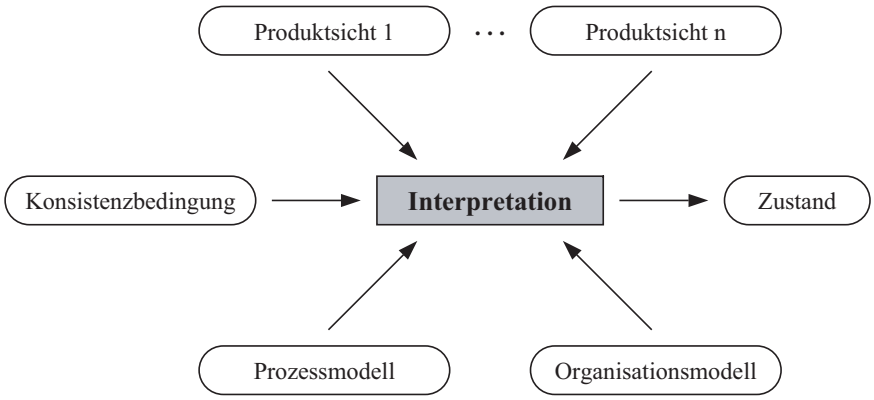

Abbildung 6.3: Prinzip der Interpretation

möglich, vermöge des Organisationsmodells gegenwärtig relevante Aspekte der Organisation einzubeziehen. Das Ergebnis der Interpretation ist eine Aktualisierung des Zustands der betrachteten Konsistenzbedingung, der über die relative Konsistenz eines Produktmodells entscheidet.

Eine Konsistenzbedingung per se legt somit keine absolute Anforderung an ein Produktmodell fest. Die Gültigkeit einer Konsistenzbedingung ist vielmehr ein relativer Begriff, der abhängig ist von

- der Konsistenzbedingung bzw. ihrer Spezifikation, welche eine Anforderung an ein Produktmodell beschreibt,
- dem Produktmodell, also dem Zustand relevanter Produktsichten und ihrer Bestandteile,
- dem Prozessmodell und somit dem Kontext der aktuell durchgeführten Entwicklungsaktivität,
- dem Organisationsmodell, welches Personen- oder Ressourcen-spezifische Aspekte einbringt.

Im Rahmen einer übergreifenden Integrationsplattform lässt sich die Interpretation auf fachliche Dienste der Anwendungsschicht abstützen (vgl. Abschnitt 3.4). Eine flexible und kontextbezogene Suche, Auswahl, Kombination und letztlich Ausführung geeigneter Dienste zur Durchführung der Interpretation einer Konsistenzbedingung kann folgerichtig durch ein Zusammenspiel aus Modellverwaltung und Dienstadministration erreicht werden. Auf diese

Weise ist es möglich, sowohl spezifische als auch generische Dienste gleichartig zu integrieren oder zu einer Konsistenzbedingung verschiedene Dienste vorzusehen, welche abhängig vom Entwicklungskontext genutzt werden.

Beispiel 6.3 (*Interpretation einer Konsistenzbedingung*)

In Beispiel 6.2 werden anhand eines exemplarischen Produktmodells Möglichkeiten der Spezifikation einer Konsistenzbedingung aufgezeigt. Vorausgesetzt sind zwei Produktsichten, deren Strukturen isomorph sein sollen. Da sich Partialmodelle jedoch voneinander unabhängig bearbeiten lassen, kann diese Strukturbedingung im Laufe eines experimentellen Produktentwicklungsprozesses temporär verletzt sein, da beispielsweise Komponenten hinzugenommen oder entfernt werden. Ein solcher Zustand der Inkonsistenz des betrachteten Produktmodells ist beispielhaft in Abbildung 6.4 dargestellt (vgl. Abbildung 6.2).

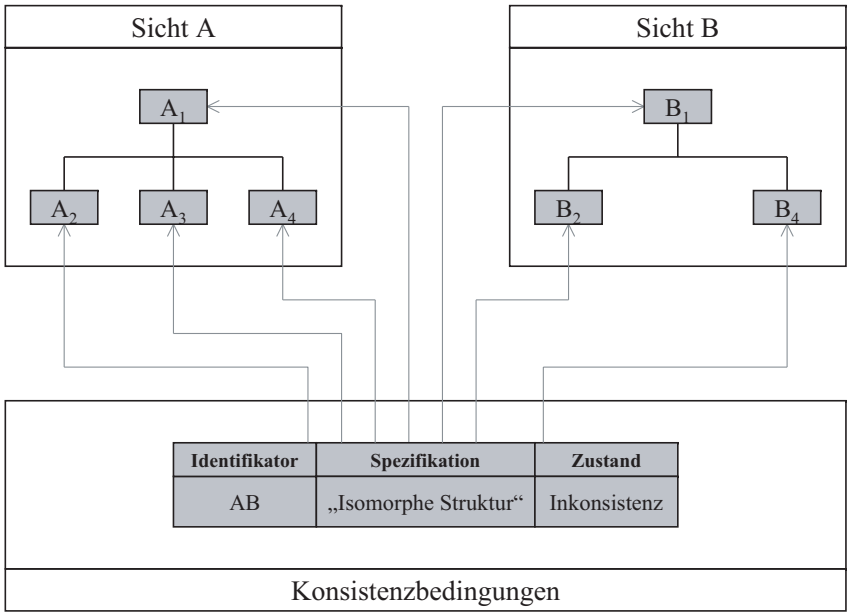

Abbildung 6.4: Interpretation einer Konsistenzbedingung

Durch die explizite Spezifikation des beschriebenen Querbezugs zwischen Sicht A und Sicht B kann zu jedem Zeitpunkt eines Entwicklungsprozesses entschieden werden, ob die gestellte Konsistenzanforderung erfüllt ist. In dem gezeigten Fall führt die Interpretation der zugehörigen Konsistenzbedingung zu einer Aktualisierung ihres Zustands, der ersichtlich auf eine Inkonsistenz des Produktmodells hinweist. Gemäß des Grundprinzips der Interpretation

wäre es jedoch durchaus möglich, das gezeigte Produktmodell zunächst als (relativ) konsistent anzusehen, da beispielsweise die Forderung nach isomorphen Strukturen in frühen Prozessphasen zu vernachlässigen ist.

Wie das gezeigte Beispiel verdeutlicht, entscheidet erst die Interpretation als eigenständiger Schritt eines Konsistenzsicherungsprozesses darüber, ob eine Konsistenzbedingung erfüllt oder verletzt ist. Insbesondere kann dabei der Entwicklungskontext flexibel berücksichtigt werden. Die Durchführung einer Interpretation lässt sich im Rahmen einer übergreifenden Integrationsplattform auf fachliche Dienste abstützen. Ihre konkrete Gestalt hängt wesentlich von der gewählten Art der Spezifikation ab. Während sich informell beschriebene Bedingungen nur manuell verifizieren lassen, können formal spezifizierte Querbezüge automatisiert geprüft werden.

6.2.3 Resolution

Mit dem Begriff *Resolution* bezeichnen wir die ggf. erforderliche Wiederherstellung relativer Konsistenz bezüglich einer ausgesuchten Konsistenzbedingung, also die Überführung eines inkonsistenten Produktmodells in einen Zustand relativer Konsistenz. Als dritter und wesentlicher Bestandteil eines Konsistenzsicherungsprozesses setzt die Resolution somit die Schritte der Spezifikation und Interpretation einer Konsistenzbedingung voraus, welche eine Konsistenzanforderung an ein Produktmodell festlegen bzw. darauf aufbauend die Konsistenz eines Produktmodells prüfen (s. Abschnitt 6.2.1 und Abschnitt 6.2.2).

Analog zur Durchführung einer Interpretation erlaubt der gewählte Ansatz eines segmentierten Produktmodells mit lose gekoppelten, eigenständigen Produktsichten auch für den Vorgang der Resolution eine praxisgerechte Berücksichtigung des aktuellen Entwicklungskontexts. So kann abhängig vom Zustand des Entwicklungsprozesses, den Erfordernissen oder Beschränkungen der gegenwärtig durchzuführenden Entwicklungsaktivität oder dem Reifegrad einzelner Partialmodelle flexibel entschieden werden, welche Maßnahmen zur Wiederherstellung relativer Konsistenz erforderlich und angemessen sind.

Abbildung 6.5 verdeutlicht das Grundprinzip der Resolution. Eine Resolution bezieht sich stets auf eine Konsistenzbedingung und den dazu inzidenten Produktsichten, hinsichtlich der das Produktmodell gemäß einer zuvor durchgeführten Interpretation zum gegenwärtigen Zeitpunkt inkonsistent ist. In gleicher Weise lässt sich wiederum der aktuelle Entwicklungskontext in Form des zugrunde liegenden Prozessmodells sowie des Organisationsmodells

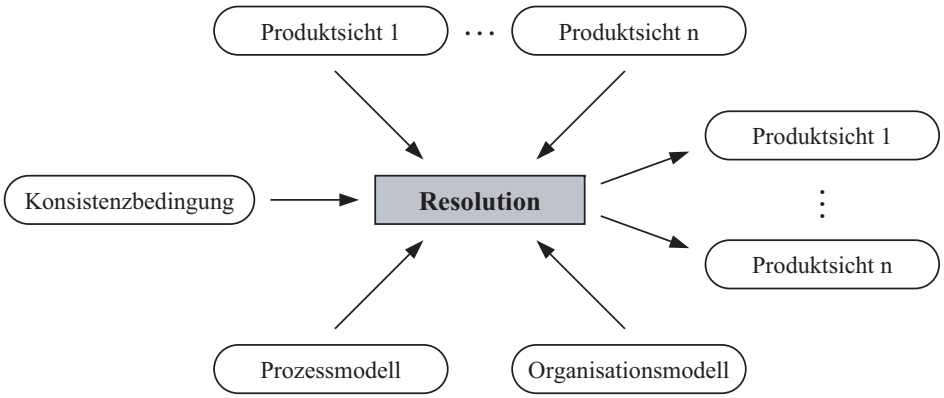

Abbildung 6.5: Prinzip der Resolution

berücksichtigen. Das Ergebnis einer Resolution ist eine Änderung der betroffenen Produktsichten, so dass die betrachtete Konsistenzbedingung nach Durchführung der Resolution erfüllt ist.

Wie der Vorgang der Interpretation lässt sich auch die Resolution vor dem Hintergrund einer übergreifenden Integrationsplattform (s. Abschnitt 3.4) auf fachliche Dienste der Anwendungsschicht abstützen. Die im Zuge einer Resolution erforderlichen Modifikationen betroffener Partialmodelle unter Berücksichtigung des Entwicklungskontexts können somit durch ein Zusammenspiel aus Modellverwaltung für den Zugriff auf das Produktmodell und Dienstadministration für die Ausführung erforderlicher Modelländerungen erreicht werden. In diesem Rahmen bestehen für die spezifische Durchführung einer Resolution die folgenden Freiheitsgrade:

Art und Ausmaß: Abhängig vom Zustand des Entwicklungsprozesses und dem Reifegrad betroffener Partialmodelle können sehr unterschiedliche Maßnahmen zur Beseitigung einer festgestellten Inkonsistenz zweckmäßig sein. Insbesondere ist zu entscheiden, welche der beteiligten Produktsichten bzw. Modellelemente für eine Wiederherstellung relativer Konsistenz zu modifizieren sind. In diesem Zusammenhang sind zwei Fälle zu unterscheiden:

- Im Zuge einer *unilateralen Resolution* werden alle erforderlichen Modifikationen zur Wiederherstellung relativer Konsistenz an ausschließlich einem Partialmodell vorgenommen. Dies ist beispiels-

weise immer dann sinnvoll, wenn eine Produktsicht in einem unmittelbaren Abhängigkeitsverhältnis zu einer weiteren Produktsicht steht, so dass Änderungen an der übergeordneten Produktsicht zu übernehmen sind.

- Dagegen sprechen wir von einer *multilateralen Resolution*, wenn zur Beseitigung einer festgestellten Inkonsistenz Elemente verschiedener Partialmodelle geändert werden. Diese Art der Resolution ist für den Fall gleich berechtigter, aus fachlicher Sicht weit gehend unabhängiger Produktsichten mit ähnlichem Reifegrad angemessen und stellt somit die üblicherweise zu erwartende Resolutionsmethode dar.

Ablauf: Gemäß der gewählten Form der Spezifikation einer Konsistenzbedingung für Partialmodelle (vgl. Abschnitt 6.2.1), können beteiligte Bearbeiter seitens einer übergreifenden Integrations- und Entwicklungsplattform (s. Abschnitt 3.4) auf unterschiedliche Weise in der Durchführung einer Resolution unterstützt werden. Insbesondere lassen sich Resolutionen hinsichtlich des Grades ihrer Automatisierung in drei Kategorien einteilen:

- Eine Resolution heißt *manuell*, wenn alle erforderlichen Modellmodifikationen zur Wiederherstellung relativer Konsistenz durch beteiligte Entwickler selbst vorgenommen werden.
- Wir nennen eine Resolution *interaktiv*, wenn sich Teilschritte automatisiert vollziehen lassen, jedoch zusätzlich Eingriffe oder Entscheidungen eines Entwicklers erforderlich sind.
- Schließlich heißt eine Resolution *automatisiert*, falls sie sich ohne Eingriff eines Entwicklers vollständig auf einen Dienst der Integrationsplattform abstützen lässt.

Anwendung: Als dritter Freiheitsgrad einer Resolution ist der Zeitpunkt ihrer Anwendung während eines Entwicklungsprozesses zu nennen. So ist es beispielsweise möglich, vorhandene Inkonsistenzen temporär zu tolerieren und sukzessive gemäß festgelegter Strategien aufzulösen. In diesem Zusammenhang lassen sich zwei Vorgehensweisen unterscheiden, auf welche in Abschnitt 6.4 dieses Kapitels ausführlich eingegangen wird:

- Wir nennen eine Resolution *Prozess-gesteuert*, falls ihre Durchführung einen eigenständigen Schritt des zugrunde liegenden Entwicklungsprozesses darstellt.

- Eine Resolution heißt *Ereignis-gesteuert*, wenn ihre Anwendung als Reaktion auf das Eintreten spezifischer Ereignisse einer Produktentwicklung erfolgt.

Angesichts der genannten Punkte beschreibt Abbildung 6.5 nur den prinzipiellen Rahmen einer Resolution als eigenständiger Bestandteil eines Konsistenzsicherungsprozesses. Darüber hinaus existieren zahlreiche Freiheitsgrade, welche die konkrete Gestalt einer Resolution und ihre Einbettung in einen Entwicklungsprozess bestimmen.

Beispiel 6.4 (*Resolution einer Inkonsistenz*)

Im Folgenden greifen wir das in Beispiel 6.2 und Beispiel 6.3 eingeführte, exemplarische Produktmodell zur Illustration der Spezifikation und Interpretation einer Konsistenzbedingung erneut auf und veranschaulichen daran die Wirkungsweise der Resolution sowie die erläuterten Freiheitsgrade ihrer Durchführung. Wie in Abbildung 6.4 dargestellt, hat der in Beispiel 6.3 beschriebene Schritt der Interpretation einen inkonsistenten Modellzustand festgestellt, da die in Beispiel 6.2 spezifizierte Konsistenzbedingung verletzt ist, d.h. die beiden beteiligten Produktsichten keine isomorphen Strukturen aufweisen.

Eine mögliche Resolution dieser Inkonsistenz besteht darin, das Sicht B zugeordnete Strukturmodell um eine Komponente (sowie eine entsprechende Assoziation) zu ergänzen, gemäß der strukturellen Vorgabe aus Sicht A. Jedoch ist es unter alleiniger Betrachtung der festgelegten Konsistenzbedingung zur Wiederherstellung relativer Konsistenz ebenso möglich, die in Abbildung 6.6 gezeigte Komponente C nicht in Sicht B einzufügen und stattdessen genau eine der Komponenten A_2, A_3 oder A_4 aus Sicht A zu entfernen. Auch eine solche Vorgehensweise führt zu isomorphen Strukturen der vorhandenen Produktsichten.

Ebenso könnten beiden Produktsichten beliebige, identische Strukturmodelle zugewiesen werden, um isomorphe Strukturen zu erzwingen. Diese weiteren Möglichkeiten der Resolution sind in Abbildung 6.7 veranschaulicht. Wie das gezeigte Beispiel bereits für den Fall einer vergleichsweise einfachen Konsistenzbedingung verdeutlicht, bestehen hinsichtlich Art und Ausmaß einer Resolution, d.h. ihrer spezifischen Ausprägung, erhebliche Freiheitsgrade. Oftmals ist somit neben der Kenntnis der Konsistenzanforderung eine Berücksichtigung des Entwicklungskontexts erforderlich, um zu entscheiden, welche Produktsichten in welcher Weise zu modifizieren sind.

Wie der Vorgang der Interpretation ist auch der Ablauf der Resolution abhängig von der gewählten Form der Spezifikation einer Konsistenzbedingung. Während formal festgelegte Konsistenzanforderungen potenziell eine

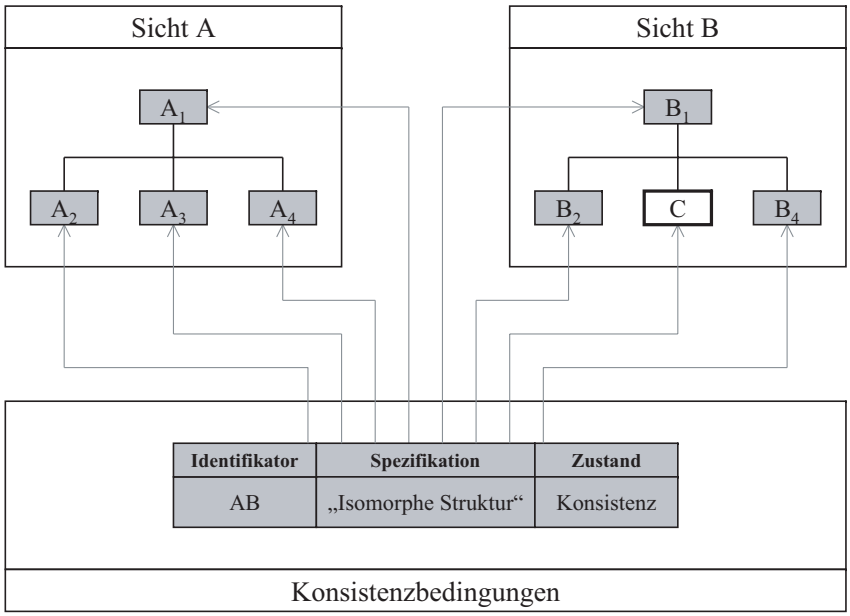

Abbildung 6.6: Mögliche Resolution einer Inkonsistenz

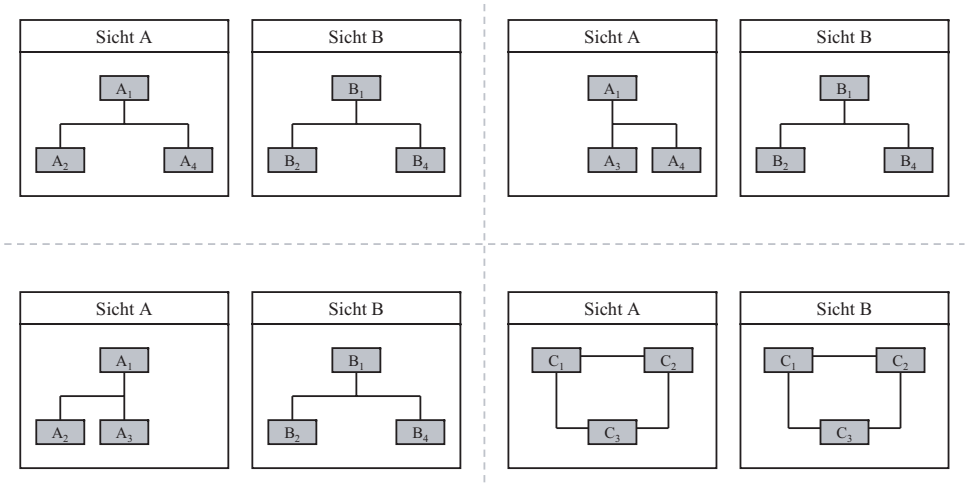

Abbildung 6.7: Weitere Möglichkeiten der Resolution

weit gehend automatisierte bzw. interaktive Resolution erlauben, ist auf der Grundlage informeller Beschreibungen lediglich eine manuelle Durchführung seitens eines geeigneten Entwicklers möglich. Je größer somit der in die präzise Spezifikation einer Konsistenzbedingung investierte Aufwand ist, desto einfacher können tendenziell die Schritte der Interpretation und Resolution automatisiert oder unterstützt werden.

Resolution ist die Wiederherstellung relativer Konsistenz bezüglich einer fest gewählten Konsistenzbedingung. Jedoch können die damit verbundenen Modellmodifikationen dazu führen, dass eine weitere, bisher erfüllte Konsistenzbedingung nach Durchführung einer Resolution verletzt ist. In diesem Fall sprechen wir von einem *Resolutionskonflikt* oder kurz *Konflikt*. Die Ursache eines Resolutionskonflikts liegt offenbar darin, dass im Zuge einer Resolution nur eine Konsistenzbedingung betrachtet wird, obwohl weitere Konsistenzbedingungen den Lösungsraum valider Produktmodelle zusätzlich einschränken.

In Abschnitt 6.1 wird vorausgesetzt, dass die Menge aller Konsistenzbedingungen erfüllbar ist, also stets ein global konsistentes Produktmodell existiert, das alle aufgestellten Konsistenzbedingungen erfüllt. Wie sich daraus unmittelbar ableiten lässt, sind Resolutionskonflikte vermeidbar durch eine geeignete Wahl der im Verlauf einer Resolution durchgeführten Modellmodifikationen. Jedoch kann es ebenso sinnvoll sein, aufgestellte Konsistenzbedingungen aufeinander abzustimmen, um häufig auftretende Konflikte zu umgehen. Insgesamt betrachtet bestehen zur Auflösung eines Resolutionskonflikts die folgenden Möglichkeiten:

- Vor dem Hintergrund einer erfüllbaren Menge an Konsistenzbedingungen können auftretende Konflikte durch die Wahl eines Produktmodells vermieden werden, das im Lösungsraum aller relevanten Bedingungen liegt. Somit ist der Ablauf einer Resolution geeignet anzupassen, um Modellmodifikationen, die zu einem Resolutionskonflikt führen, zu vermeiden. Die Spezifikation und Interpretation von Konsistenzbedingungen bleiben dagegen unverändert.

- Eine weitere Möglichkeit besteht darin, Resolutionskonflikten bereits auf der Ebene der Interpretation von Konsistenzbedingungen vorzubeugen. Da diese über die relative Konsistenz bzw. Inkonsistenz eines Produktmodells entscheiden, kann im Rahmen einer Interpretation beispielsweise eine Priorisierung von Konsistenzbedingungen vorgenommen werden. Die Spezifikation von Konsistenzbedingungen ist in diesem Fall dagegen nicht betroffen.

- Der dritte und weitest gehende Ansatzpunkt zur Vermeidung von Resolutionskonflikten besteht schließlich in der Änderung aufgestellter Konsistenzbedingungen. So lassen sich beispielsweise verwandte Bedingungen zu einer Konsistenzbedingung zusammenfassen, so dass eine erfolgreiche Resolution stets alle der dadurch adressierten Aspekte berücksichtigen muss. Entsprechend anzupassen sind die Schritte der Interpretation und Resolution.

Spezifikation, Interpretation und Resolution bilden die grundlegenden Schritte eines flexiblen Konsistenzsicherungsprozesses vor dem Hintergrund eines segmentierten Produktmodells gemäß Kapitel 4. Aus ihrer Modularität ergeben sich zahlreiche Freiheitsgrade für eine praxisgerechte Behandlung von Modellkonsistenz. Die Nutzung variabler Techniken zur Festlegung von Querbezügen zwischen Partialmodellen ist damit ebenso möglich, wie der Einsatz differenzierter und angepasster Verfahren zur Konsistenzprüfung und Konsistenzsicherung unter expliziter Berücksichtigung des Entwicklungskontexts.

6.3 Techniken

Im Rahmen der Spezifikation werden die aus fachlichen Zusammenhängen resultierenden Querbezüge zwischen den Sichten eines segmentierten Produktmodells festgelegt. Angesichts der Vielfältigkeit denkbarer Beziehungen sieht der vorgeschlagene Ansatz eine gleich berechtigte Verwendung verschiedenartiger Spezifikationstechniken vor, um vorhandenes Wissen über bestehende Zusammenhänge in optimal angepasster Weise erfassen zu können. Insbesondere werden informelle Beschreibungsformen zugelassen, da zahlreiche Querbezüge zwischen Modellelementen nur schwer in einem mathematisch exakten Sinne darstellbar sind.

Zentrale Zielsetzung der Spezifikation ist die Anreicherung des Produktmodells um Fachwissen über Abhängigkeiten, Querbezüge und Überschneidungen zwischen Produktsichten bzw. ihren Elementen, so dass die darauf folgenden Schritte der Interpretation und der Resolution die Produktentwicklung hinsichtlich einer kontrollierten Behandlung der Modellkonsistenz möglichst weit gehend unterstützen. Hierzu bieten sich aus dem Gebiet der *Künstlichen Intelligenz (KI)* verschiedene Techniken der Wissensrepräsentation an, welche beispielsweise in Expertensystemen [Pup88] seit geraumer Zeit erfolgreich eingesetzt werden.

Die folgenden Abschnitte stellen im Überblick und ohne Anspruch auf Vollständigkeit die wesentlichen Techniken dieses Bereichs sowie weitere Verfahren zur Erfassung von Produktwissen vor und ordnen sie in die erarbeite-

te Konzeption eines übergreifenden Konsistenzsicherungsprozesses ein. Dabei werden insbesondere die bevorzugten Anwendungsbereiche der einzelnen Ansätze benannt, ihre individuellen Vorteile, Nachteile und Beschränkungen erläutert, sowie ihre Eignung für die Verwendung innerhalb des vorgestellten Ansatzes der Konsistenzsicherung geprüft. Eine Zusammenfassung der ermittelten Ergebnisse findet sich in Abschnitt 6.3.6.

6.3.1 Logik

Die Disziplin der Logik ist historisch aus der Mathematik und der Philosophie entstanden und befasst sich mit der folgerichtigen Ableitung von Schlüssen aus gegebenen Aussagen. Insbesondere bietet die formale Logik als Grundlage der Mathematik verschiedene Kalküle der Wissensrepräsentation, welche sich im Rahmen des vorgeschlagenen Ansatzes zur Spezifikation von Konsistenzbedingungen nutzen lassen (vgl. Beispiel 6.2). Aufgrund ihres Umfangs können die in der Logik verwendeten Techniken zur Erfassung von Aussagen an dieser Stelle nur im Überblick vorgestellt werden. Ausführliche Darstellungen finden sich z.B. in [Sch95], [Lus90], [EFT96] oder [CM85].

Im Rahmen der Logik haben sich verschieden mächtige Techniken der Wissensrepräsentation und -deduktion entwickelt, welche üblicherweise in drei Bereiche eingeteilt werden:

- In der *Aussagenlogik* werden Verknüpfungen und Operationen für atomare Aussagen untersucht. Diese lassen sich stets auf die Konjunktion von Aussagen, ihre Disjunktion oder ihre Negation zurückführen. Die Aussagenlogik stellt somit eine sehr einfache, jedoch oftmals unzureichende Technik der Wissensrepräsentation dar.

- Die *Prädikatenlogik* erweitert die Aussagenlogik um zusätzliche Möglichkeiten zur differenzierten Erfassung von Aussagen durch die Hinzunahme von Funktionen, Prädikaten und Quantoren. Somit können nicht nur Verknüpfungen von Aussagen formal erfasst werden, sondern ebenso ihr innerer Aufbau.

- Erweiterungen der Prädikatenlogik werden im Bereich der *Nicht-klassischen Logik* untersucht. Hierzu zählen insbesondere die Berücksichtigung unsicheren und unvollständigen Wissens wie beispielsweise durch probabilistische Ansätze [Pup88] oder Fuzzy Logic [MF96], nicht-monotones Schließen sowie temporale Logiken.

Die formale Logik bietet somit eine Vielzahl von Ansätzen zur Repräsentation von Produktwissen, welche von der elementaren Verknüpfung atomarer Aussagen bis hin zur differenzierten Erfassung unsicheren und unvollständigen Wissens reichen. Jeder Zuwachs an Ausdruckskraft eines Kalküls zieht jedoch gleichermaßen eine Steigerung der Komplexität und des Aufwands nach sich, der mit einer Überprüfung einer damit formulierten Aussage verbunden ist. Folgerichtig ist eine Abwägung zu treffen zwischen dem Gewinn an Ausdruckskraft einerseits und dem erforderlichen Aufwand einer Auswertung andererseits.

Der in Abschnitt 6.2 erarbeitete Ansatz einer modularen Konsistenzsicherung sieht in übergreifender Weise eine gleichartige Behandlung der genannten Techniken vor. Im Falle der Erfassung von Querbezügen zwischen Produktsichten durch logische Formalismen gestaltet sich der Konsistenzsicherungsprozess wie folgt:

Spezifikation: Die Spezifikation einer Konsistenzbedingung besteht aus einer Formel eines geeigneten logischen Kalküls, welche ihrerseits auf ggf. zuvor definierte Aussagen, Funktionen oder Prädikate aufbauen kann. Angesichts der Vielfältigkeit vorhandener Ansätze ließe sich auf diese Weise nahezu jede Aussage über ein Produktmodell formal erfassen.

Interpretation: Die Aufgabe der Interpretation ist es, aus der im Rahmen der Spezifikation festgelegten Formel einen Wahrheitswert zu ermitteln, welcher angibt, ob die zugehörige Konsistenzbedingung erfüllt oder verletzt ist. Abhängig von der gewählten Spezifikationstechnik kann dies auf der Grundlage des Produktmodellzustands nur partiell automatisiert erfolgen. So ist beispielsweise im Falle der Aussagenlogik eine manuelle Bewertung der verwendeten atomaren Aussagen erforderlich. Zusätzlich ist für die Interpretation der Entwicklungskontext zu berücksichtigen.

Resolution: Sofern eine Inkonsistenz vorliegt, die betrachtete Konsistenzbedingung also unter Einbeziehung der Randbedingungen des Entwicklungsprozesses verletzt ist, sind im Zuge der Resolution geeignete Modellmodifikationen vorzunehmen. Aufgrund des deklarativen Charakters logischer Formeln können diese im Allgemeinen nicht konstruktiv abgeleitet werden, so dass eine manuelle Resolution notwendig ist (s. Abschnitt 6.2.3). Jedoch ist es möglich, die Ursache einer auftretenden Inkonsistenz präzise zu ermitteln und Entwickler auf diese Weise in der Resolution zu unterstützen.

Auf dem Gebiet der modellbasierten Software-Entwicklung ist die im Jahre 1995 entstandene *Object Constraint Language (OCL)* [WK98, CW02, MC99] als Bestandteil der in Wissenschaft und Praxis weithin genutzten Modellierungstechnik *Unified Modeling Language (UML)* [UML03] als ein bekanntes Beispiel zu nennen, um Konsistenzanforderungen an Modelle mithilfe eines logischen Kalküls zu spezifizieren. Diese gibt eine Reihe von Funktionen, Operationen und Prädikaten vor, welche es ermöglichen, Modellelemente in Klassendiagrammen zueinander in Beziehung zu setzen und die gewünschten Invarianten eines Modells zu definieren.

Aus der zunehmenden Verbreitung XML-basierter Technologien [W3C04] ist der Ansatz *xlinkit* [NCEF02, NEFE03] hervorgegangen, welcher ebenfalls ein prädikatenlogisches Kalkül erster Ordnung nutzt, um – aufbauend auf die Standards *XPath* [W3C99] und *XLink* [W3C01] – Konsistenzbeziehungen zwischen XML-Dokumenten festzulegen. Schließlich sei die Transformationsbeschreibungssprache *Bidirectional Object Oriented Transformation Language (BOTL)* [BM03, BM02, Bra03b] erwähnt, die über die Festlegung von Querbezügen hinaus auf automatisierte Transformationen zwischen (Meta-)Modellen mittels logisch fundierter Regelwerke ausgerichtet ist.

Insgesamt betrachtet bietet der Bereich der Logik sehr vielfältige und ausdrucksstarke Techniken der Wissensrepräsentation, eignet sich jedoch aufgrund der weit gehenden Einschränkungen hinsichtlich einer automatisierten Interpretation und Resolution nur begrenzt für eine praxisgerechte Konsistenzsicherung eines Produktmodells. Aus den genannten Ansätzen ist insbesondere die Prädikatenlogik erster Ordnung [Sch95] hervorzuheben, welche einerseits hinreichend allgemeine Konstrukte zur Spezifikation von Produktwissen anbietet und andererseits zumindest automatisierte Auswertungsverfahren zulässt.

6.3.2 Regeln

Regeln sind eine weit verbreitete und allgemein anwendbare Form der Wissensrepräsentation, welche insbesondere in Expertensystemen bevorzugt genutzt wird. Abstrakt betrachtet besteht eine Regel stets aus einer Menge von *Prämissen*, welche den Vorbedingungen der Regelanwendung entsprechen und einer *Implikation*, die eine aus den Prämissen ableitbare Aussage bezeichnet. Um Prämissen und Implikationen ihrerseits festzulegen, ist eine hinreichend ausdrucksstarke Spezifikationstechnik erforderlich, wie sie beispielsweise die in Abschnitt 6.3.1 vorgestellten Kalküle der Aussagen- oder Prädikatenlogik anbieten.

Regeln lassen sich somit ebenfalls als Formeln eines logischen Kalküls gemäß den Ausführungen des vorangehenden Abschnitts auffassen. Eine Re-

gel besitzt jedoch stets die spezifische Gestalt

$$P_1 \wedge \ldots \wedge P_n \rightarrow I,$$

wobei $P_1, \ldots, P_n$ und I, $n \in \mathbb{N}$, die Prämissen einer Regel bzw. ihre Implikation bezeichnen[7]. Aufgrund ihres vorgegebenen Aufbaus sind Regeln im Vergleich zu allgemeinen Formeln eines logischen Kalküls meist einfacher zu handhaben. Entsprechend können die Schritte einer regelbasierten Konsistenzsicherung gemäß des übergreifenden Ansatzes wie folgt präzisiert werden:

Spezifikation: Die Spezifikation einer Konsistenzbedingung eines Produktmodells in Form einer Regel geschieht durch die Festlegung aller Prämissen sowie der Implikation auf der Grundlage eines geeigneten logischen Formalismus.

Interpretation: Für die Interpretation einer Regel ist (unter Berücksichtigung des Entwicklungskontexts) zu prüfen, ob durch den aktuellen Zustand des Produktmodells jede der angegebenen Prämissen erfüllt ist. In diesem Fall muss für die Gültigkeit der Konsistenzbedingung insgesamt auch die Implikation erfüllt sein. Ist dagegen mindestens eine Prämisse verletzt, so gilt die zugehörige Konsistenzbedingung als erfüllt, unabhängig vom Wahrheitswert der Implikation.

Resolution: Liegt eine Inkonsistenz vor, so erfüllt das Produktmodell alle Prämissen der betrachteten Regel, jedoch nicht ihre Implikation. Für eine Wiederherstellung der Konsistenz sind somit geeignete Modellmodifikationen vorzunehmen, so dass mindestens eine Prämisse nicht erfüllt oder aber die Implikation erfüllt ist. Wie in Abschnitt 6.3.1 beschrieben, lässt sich dies nur eingeschränkt automatisieren und erfordert daher meist manuelle Eingriffe.

Zusammenfassend kann Produktwissen in natürlicher Weise oftmals in Form von Regeln dargestellt werden. Die in Abschnitt 6.3.1 erläuterten Kalküle der Wissensrepräsentation lassen sich hierbei einsetzen, um Prämissen und Implikation einer Regel angemessen zu erfassen. Abhängig von der Ausdruckskraft der gewählten Spezifikationstechnik ist eine weit gehend automatisierte, generische Interpretation von Regeln erreichbar. Sofern zulässige Implikationen einer Regel jedoch nicht stark eingeschränkt werden, erfordert der Ablauf einer Resolution im allgemeinen Fall manuelle Modellmodifikationen seitens eines geeigneten Entwicklers.

[7] Im Rahmen der Aussagenlogik sind Regeln unter den Begriffen *Hornklausel* bzw. *Hornformel* bekannt, benannt nach dem Logiker A. Horn.

6.3.3 Constraints

Ebenfalls im Bereich der Künstlichen Intelligenz haben sich Constraint-basierte Ansätze als eigenes Forschungsgebiet entwickelt. Constraints repräsentieren Relationen zwischen den meist numerischen Variablen eines zugrunde liegenden Modells und legen auf diese Weise Randbedingungen fest. Im Rahmen eines Produktmodells sind Constraints daher besonders geeignet, um Querbezüge zwischen den elementaren Attributen vorhandener Komponenten zu spezifizieren (vgl. Abschnitt 4.3.4). Bezeichnet man diese sukzessive mit $x_1 \in D_1, \ldots, x_n \in D_n$ gemäß der Wertebereiche (engl.: *domain*) $D_1, \ldots, D_n$, $n \in \mathbb{N}$, so entspricht ein Constraint C einer Relation

$$C \subseteq D_1 \times \ldots \times D_n.$$

Die so spezifizierte Randbedingung eines Modells ist genau dann erfüllt, falls $(x_1, \ldots, x_n) \in C$ gilt. Hierbei kann die Festlegung eines Constraints auf unterschiedliche Arten erfolgen, welche in [VV87] klassifiziert werden. Im Fall endlicher Wertebereiche bietet es sich beispielsweise an, Relationen in Form von Tabellen anzugeben, während sich Constraints für sehr große oder potenziell unendliche Wertemengen auch durch Muster bzw. Gleichungen zwischen Variablen beschreiben lassen. Constraints können daher analog zu Regeln als Kalkül einer quantorenfreien Prädikatenlogik erster Ordnung mit spezifischen Funktionen und Relationen aufgefasst werden.

Im Allgemeinen lassen sich die vielfältigen Randbedingungen eines Modells erst durch die simultane Betrachtung mehrerer Constraints adäquat erfassen. Man spricht daher von Constraint-Netzen, bestehend aus einer Reihe einzelner Constraints, die über gemeinsame Variablen miteinander verbunden sind. Die Aufgabe eines Constraint-Systems besteht darin, aus einer so beschriebenen Vorgabe eine Belegung der Modellvariablen zu ermitteln, so dass alle festgelegten Constraints erfüllt sind. Für diese, als Constraint-Propagierung bezeichnete Problemstellung existieren verschieden mächtige Lösungsansätze, die in [Pup88] im Überblick genannt werden.

Vorhandene Constraint-Systeme – insbesondere die in [Güs89] und [SS80] beschriebenen Ansätze – können modular in die vorgeschlagene Konzeption der Konsistenzsicherung integriert werden zur Festlegung, Überprüfung und Sicherung der Randbedingungen eines Produktmodells. In diesem Zusammenhang besitzen die einzelnen Schritte folgende Gestalt:

Spezifikation: Die Spezifikation einer Konsistenzbedingung mittels Constraints geschieht durch die Angabe eines Constraint-Netzes auf der Grundlage einer geeigneten Sprache zu ihrer Beschreibung. Auf eine der genannten Arten werden so die Attribute eines Produktmodells zueinander in Beziehung gesetzt.

Interpretation: Die Interpretation eines Constraint-Netzes hat die Aufgabe, alle auftretenden Attribute durch die im Produktmodell enthaltenen Werte zu ersetzen und festzustellen, ob jeder der angegebenen Constraints erfüllt ist. Dies lässt sich auf einfache Weise mittels generischer Verfahren automatisieren.

Resolution: Im Falle einer Inkonsistenz stehen schließlich Constraint-Propagierungs-Methoden zur Verfügung, welche sich im Zuge einer ggf. erforderlichen Resolution nutzen lassen. Somit ist auch dieser Schritt eines Konsistenzsicherungsprozesses weit gehend automatisierbar durch Einbindung existierender Constraint-Systeme.

Aufgrund der weit reichenden Automatisierbarkeit von Interpretation und Resolution eignen sich Constraints in besonderem Maße als Mittel der Konsistenzsicherung. Eine exemplarische Umsetzung anhand der Fallstudie eines Turbinenrotors wird in [Gün01] vorgestellt. Existierende Constraint-Systeme können unmittelbar in eine übergreifende Integrationsplattform eingebunden werden. Eine Erweiterung um Computeralgebrasysteme [KG02, Wes03] erlaubt ferner die Nutzung leistungsfähiger symbolischer Propagierungsmethoden. Weitere Möglichkeiten ergeben sich aus einer Kombination mit logischen Beweissystemen [Hom96, Bal94].

Constraints bieten jedoch nur die Möglichkeit, vergleichsweise einfache Beziehungen zwischen den elementaren Variablen eines Modells zu spezifizieren, so dass sich z.B. strukturelle Bedingungen für Elemente verschiedener Produktsichten (vgl. Beispiel 6.2) auf diese Weise nicht unmittelbar formulieren lassen. Abhängig von der Ausdruckskraft einer Constraint-Sprache, der Komplexität eines Constraint-Netzes und der Mächtigkeit eines zugehörigen Constraint-Systems können Propagierungsverfahren zudem mit einem hohen Aufwand verbunden oder nur für eingeschränkte Problemstellungen erfolgreich anwendbar sein.

6.3.4 Applikative Techniken

Eine nahe liegende und sehr leistungsfähige Methode der Konsistenzsicherung besteht darin, die Schritte der Interpretation und der Resolution auf die Ausführung von Programmcode einer Programmiersprache abzustützen. Als Voraussetzung ist eine geeignete Schnittstelle für den Zugriff auf Produktmodelle anzubieten, welche die Inspektion und Modifikation eines Modellzustands erlaubt. In Verbindung mit Plattform-übergreifenden und Sprachunabhängigen Middleware-Technologien, wie sie aus dem als *Enterprise Application Integration (EAI)* bezeichneten Bereich bekannt sind, können zudem verschiedenste Programmiertechniken eingesetzt werden.

Auch diese, sehr freie Form der Konsistenzsicherung lässt sich unmittelbar in die in Abschnitt 6.2 entwickelte Konzeption einordnen. Die einzelnen Schritte eines Konsistenzsicherungsprozesses gestalten sich dabei wie folgt:

Spezifikation: Die Spezifikation einer Konsistenzbedingung beruht im Fall applikativer Techniken auf der Angabe von Programmcode einer zugelassenen Programmiersprache in zweierlei Weise: Zum einen ist festzulegen, wie – unter Nutzung einer geeigneten Schnittstelle – Zugriff auf ein Produktmodell genommen und aus seinem Zustand ein Wahrheitswert ermittelt wird, der über die Konsistenz entscheidet. Ferner ist Programmcode anzugeben, welcher im Falle einer Inkonsistenz erforderliche Modellmodifikationen zur Wiederherstellung eines konsistenten Zustands durchführt.

Interpretation: Die Interpretation einer so beschriebenen Konsistenzanforderung entspricht der Ausführung des zuerst spezifizierten Programmcodes. Als Ergebnis wird ein Wahrheitswert ermittelt, welcher angibt, ob das aktuell gültige Produktmodell sich im Zustand relativer Konsistenz befindet.

Resolution: Sofern mittels der Interpretation ein inkonsistenter Modellzustand festgestellt wird, ist im Zuge der Resolution der zweite Teil des angegebenen Programmcodes auszuführen, welcher Zugriff auf das Produktmodell nimmt und geeignete Änderungen zur Sicherung der Konsistenz durchführt.

Offenbar stellt die Verwendung freien Programmcodes zur Formulierung von Konsistenzanforderungen eine sehr ausdrucksstarke und leistungsfähige Methode der Konsistenzsicherung dar, die unmittelbar innerhalb des vorgeschlagenen Ansatzes realisiert werden kann. Die in einer übergreifenden Integrationsplattform ohnehin erforderliche Infrastruktur zur Anbindung heterogener Anwendungen eröffnet die Möglichkeit, verschiedenste Programmiersprachen und -techniken zu nutzen. Im Vergleich sind auf diese Weise formulierte Konsistenzbedingungen und Resolutionsverfahren jedoch schwerer nachzuvollziehen und zu modifizieren.

6.3.5 Manuelle Verfahren

Wissen über Querbezüge, Abhängigkeiten und Einschränkungen hinsichtlich der Bestandteile eines komplexen Produkts liegt in heutigen Produktentwicklungsprozessen üblicherweise in Form nicht-formalisierter, verbaler Beschreibungen vor, zum Beispiel in der Gestalt von Entwicklungshandbüchern oder

Produktdokumentationen. Diese Art der Formulierung ist einerseits für jeden qualifizierten Bearbeiter einer Fachdisziplin unmittelbar verständlich, stellt jedoch andererseits eine Quelle von Mehrdeutigkeiten sowie Fehlinterpretationen dar und verhindert eine automatisierte Behandlung von Konsistenzeigenschaften.

Dennoch repräsentiert die Erfassung von Konsistenzanforderungen auf informelle, verbale Weise eine wichtige Alternative, zumal dies der bevorzugten Technik in Entwicklungsprozessen entspricht und zahlreiche Beziehungen zwischen Modellelementen nur schwer oder aufwändig mathematisch präzise erfassbar sind. Daher wird im Rahmen des vorgeschlagenen Ansatzes eine solche, umfassende Art der Beschreibung von Konsistenzbedingungen in Kombination mit manuellen Verfahren zu ihrer Prüfung und Sicherung explizit berücksichtigt. Die einzelnen Schritte eines Konsistenzsicherungsprozesses sind in diesem Zusammenhang auf folgende Weise gegeben:

Spezifikation: Konsistenzbedingungen für Produktsichten werden in Form verbaler Beschreibungen beliebiger Länge festgelegt. Somit ließe sich z.B. in Entwicklungshandbüchern oder bei einzelnen Bearbeitern vorhandenes Produktwissen mühelos und unmittelbar in ein Produktmodell integrieren.

Interpretation: Aufgrund der informell spezifizierten Konsistenzanforderung lässt sich der Schritt der Interpretation nur manuell seitens eines geeigneten Entwicklers durchführen. Dieser muss nach Aufforderung durch die Entwicklungsplattform entscheiden, ob die festgelegte Beschreibung als erfüllt anzusehen ist.

Resolution: Auch der Vorgang der Resolution einer festgestellten Inkonsistenz kann aufgrund der informell gegebenen Konsistenzanforderung ausschließlich manuell durchgeführt werden. Hierzu hat ein Entwickler geeignete Modellmodifikationen vorzunehmen und das System zu benachrichtigen, sobald diese abgeschlossen sind.

Die Kombination aus verbalen Beschreibungen von Konsistenzbedingungen und manuellen Verfahren zu ihrer Prüfung und Sicherung bietet eine umfassende Möglichkeit der Konsistenzsicherung, welche sich mühelos innerhalb des vorgeschlagenen Rahmens realisieren lässt. Sie stellt eine wichtige Alternative zu den bisher vorgestellten Techniken dar und eignet sich besonders für Fälle, in denen Konsistenzanforderungen nur schwer auf andere Weise erfassbar sind oder eine automatisierte Interpretation und Resolution ohnehin nicht angestrebt werden. Jedoch sind nicht-formalisierte Spezifikationen oftmals eine Quelle von Mehrdeutigkeiten und Fehlinterpretationen.

6.3.6 Vergleich

Die vorangehenden Abschnitte stellen eine Reihe von Ansätzen aus Mathematik und Informatik vor, die geeignet sind, Konsistenzanforderungen an Modelle zu formulieren. Trotz der Vielfältigkeit der Beschreibungsformen und Auswertungsverfahren kann jede der genannten Techniken modular in die in Abschnitt 6.2 entwickelte Konzeption integriert werden, so dass in übergreifender Weise ein stets gleichartiger und kontrollierter Ablauf eines Konsistenzsicherungsprozesses gewährleistet ist, gemäß der Schritte der Spezifikation zur Festlegung einer Konsistenzbedingung sowie der Interpretation und Resolution zu ihrer Prüfung und Sicherung.

Jede der genannten Techniken weist individuelle Anwendungsbereiche und Einschränkungen auf. Um eine gegebene Konsistenzanforderung in ein Produktmodell einzubringen, ist somit zunächst stets ein optimal angepasstes Verfahren auszuwählen unter Berücksichtigung der erläuterten Vorteile und Nachteile. Beispielsweise sind Constraints gemäß Abschnitt 6.3.3 besonders geeignet, um einfache mathematische Zusammenhänge zwischen numerischen Attributen verschiedener Produktsichten festzulegen und automatisiert zu prüfen. Zusammenfassend stellt Tabelle 6.1 die betrachteten Ansätze im Überblick dar.

Die vorgestellten Techniken zur Festlegung der Konsistenzbedingungen eines Produktmodells reichen von streng formalisierten Spezifikationen mittels logischer Kalküle bis hin zu verbalen Beschreibungsformen. Entsprechend unterschiedlich gestalten sich die Schritte der Interpretation und Resolution zur Prüfung und Sicherung der Konsistenz. Die Auswahl einer Spezifikationstechnik zur Festlegung einer Konsistenzanforderung hat somit großen Einfluss auf den weiteren Verlauf eines Konsistenzsicherungsprozesses. Insbesondere drei Faktoren spielen eine wichtige Rolle und sind daher besonders zu berücksichtigen:

- Die *Ausdruckskraft* einer Spezifikationstechnik entscheidet darüber, ob sich eine gegebene Konsistenzanforderung in adäquater Weise formulieren lässt.

- Angesichts der Vielzahl zu erwartender Konsistenzbedingungen eines komplexen Produkts stellt die *Automatisierbarkeit* von Interpretations- und Resolutionsverfahren ein wesentliches Kriterium dar.

- Schließlich ist die *Wartbarkeit* festgelegter Konsistenzbedingungen ein wichtiger Faktor, der insbesondere die Aspekte der Verständlichkeit und Modifizierbarkeit umfasst.

	Spezifikation	Interpretation	Resolution
Logik	Beliebige Formel eines Kalküls der (Prädikaten-)Logik	Zumeist interaktive Auswertung der Formel	Zumeist manuelle Modellmodifikationen
Regeln	Angabe von Prämissen und Implikation gemäß eines logischen Kalküls	Zumeist interaktive Auswertung von Prämissen und Implikation	Zumeist manuelle Modellmodifikationen
Constraints	Festlegung eines Constraint-Netzes in Form von Gleichungen oder Tabellen	Automatisierte Überprüfung festgelegter Relationen	Automatisierte Constraint-Propagierungsverfahren
Applikative Techniken	Angabe von Programmcode zur Prüfung und Sicherung der Konsistenz	Ausführung von Programmcode	Ausführung von Programmcode
Manuelle Verfahren	Verbale Beschreibung der Konsistenzanforderung	Manuelle Prüfung der Konsistenz	Manuelle Sicherung der Konsistenz

Tabelle 6.1: Techniken der Konsistenzsicherung im Überblick

Tabelle 6.2 zeigt einen qualitativen Vergleich der beschriebenen Techniken hinsichtlich ihrer Ausdruckskraft, der Automatisierbarkeit zugehöriger Auswertungsverfahren sowie der Wartbarkeit damit festgelegter Konsistenzbedingungen. Die Symbole ↑, ↗, ↘ und ↓ deuten an, in welchem Maße das betrachtete Kriterium unterstützt wird.

	Ausdrucks-kraft	Automatisier-barkeit	Wartbarkeit
Logik	↑	↘	↓
Regeln	↗	↘	↘
Constraints	↘	↗	↗
Applikative Techniken	↑	↑	↓
Manuelle Verfahren	↑	↓	↗

Tabelle 6.2: Techniken der Konsistenzsicherung im Vergleich

6.4 Strategien

Die explizite Festlegung der Konsistenzbedingungen eines segmentierten Produktmodells und ihre flexible Prüfung und Sicherung bilden einen wesentlichen Bestandteil des vorgeschlagenen Ansatzes einer integrativen modellbasierten Produktentwicklung. Die vorangehenden Abschnitte stellen die erforderlichen Teilschritte vor und erläutern eine Reihe wichtiger Techniken der Wissensrepräsentation. Darauf aufbauend bleibt zu untersuchen, auf welche Weise die beschriebenen Maßnahmen zur Sicherstellung der Konsistenz eines Produktmodells in den Ablauf eines Entwicklungsprozesses einzuordnen sind.

Wie die fachlichen Aktivitäten einer Produktentwicklung erfordern auch die Schritte der Konsistenzsicherung eine kontrollierte und transparente Vorgehensweise, ohne die erwünschte Flexibilität im Umgang mit Konsistenzeigenschaften einzuschränken. Gemäß dieser Zielsetzung werden in den folgenden Ausführungen zwei Strategien betrachtet: Abschnitt 6.4.1 zeigt zunächst, wie sich das in Abschnitt 6.2 vorgeschlagene Verfahren unmittelbar in zu-

grunde liegende Entwicklungsprozesse integrieren lässt. Ein weiter gehender Ansatz ist die Ereignis-gesteuerte Konsistenzsicherung, welche in Abschnitt 6.4.2 erläutert wird.

6.4.1 Prozesssteuerung

Produktentwicklungsprozesse bestimmen die zur Entwicklung eines komplexen technischen Produkts erforderlichen Aktivitäten und legen ihre Zusammenhänge fest (vgl. Abschnitt 2.1.2). Daher ist es nahe liegend, auch die Schritte der Konsistenzsicherung analog zu fachlichen Entwicklungsaufgaben als spezifische Bestandteile eines Entwicklungsprozesses aufzufassen. Wie in Abschnitt 3.4 erläutert wird, ist die formalisierte Erfassung von Entwicklungsprozessen in Form von Prozessmodellen eine wichtige Voraussetzung für die durchgängige Unterstützung durch Software-Lösungen im Rahmen einer übergreifenden Integrationsplattform.

Analog zu der in Kapitel 4 entwickelten Modellierungstechnik für Produkte bietet es sich an, auch die zur Modellierung von Prozessen verfügbaren Konzepte und zugehörigen Gesetzmäßigkeiten mittels eines Metamodells festzulegen (vgl. Abschnitt 4.1). Hierzu lassen sich in der Literatur unterschiedliche Ansätze finden, wie beispielsweise die im Rahmen der Unified Modeling Language (UML) vorgeschlagenen Aktivitätsdiagramme [OWS+03, UML03], das Prozessmodell der Workflow Management Coalition (WfMC) [WPD98b] oder spezifische Techniken aus dem Bereich der Modellierung betriebswirtschaftlicher Geschäftsprozesse [Sei02].

Ungeachtet ihrer unterschiedlichen Ausprägungen weisen die genannten Ansätze aufgrund der gemeinsamen Zielsetzung – der Modellierung komplexer Prozesse – gleichartige Konzepte und Darstellungsformen auf. In übergreifender Weise werden zur Erfassung von Entwicklungsprozessen die folgenden Modellelemente eingesetzt:

Aktivitäten: Aktivitäten repräsentieren die elementaren Entwicklungsaufgaben eines Prozesses, für die eine weitere Unterteilung nicht möglich oder zweckmäßig ist.

Verzweigungspunkte: An Verzweigungspunkten werden Prozesse aufgespalten, um beispielsweise die Beschreibung alternativ oder parallel durchführbarer Subprozesse zu ermöglichen.

Vereinigungspunkte: Umgekehrt zu Verzweigungspunkten dienen Vereinigungspunkte dazu, in Subprozesse aufgespaltene Prozessabläufe wieder zusammenzuführen.

Entscheidungspunkte: An Entscheidungspunkten erfolgt abhängig von einer festgelegten Bedingung eine Aufspaltung eines Prozessablaufs in alternativ durchzuführende Subprozesse.

Transitionen: Transitionen dienen schließlich dazu, die genannten Modellelemente miteinander zu verbinden und auf diese Weise eine Reihenfolge festzulegen.

Mithilfe der erläuterten Modellierungskonstrukte lassen sich beliebige Aktivitäten und Abläufe komplexer Entwicklungsprozesse in Prozessmodelle abbilden. Insbesondere können auf diese Weise die in Abschnitt 6.2 beschriebenen Schritte der Spezifikation, Interpretation und Resolution eines Konsistenzsicherungsprozesses nahtlos in ein Prozessmodell integriert werden[8]. Da Konsistenzbedingungen gemäß Kapitel 4 einen expliziten und eigenständigen Bestandteil eines Produktmodells bilden und dessen Elemente zueinander in Beziehung setzen, ergibt sich folglich eine enge Integration von Produkt und Prozess.

So sind zum einen die Schritte der Konsistenzsicherung im Rahmen eines Prozessmodells als spezifische Aktivitäten darstellbar, die sich auf ausgesuchte Konsistenzbedingungen des Produktmodells beziehen. Darüber hinaus leisten Entscheidungspunkte eines Prozessmodells einen wichtigen Integrationsbeitrag aufgrund der Möglichkeit, abhängig vom Zustand einer Konsistenzbedingung des Produktmodells Prozessabläufe zu steuern. Der dritte Ansatzpunkt einer methodischen Verknüpfung von Produkt und Prozess ist schließlich durch die in Abschnitt 4.3.1 erläuterte Zuordnung von Sichten eines Produktmodells zu Aktivitäten eines Entwicklungsprozesses gegeben.

In Anlehnung an [UML03] und [XPD02] zeigt Abbildung 6.8 ein Metamodell zur Beschreibung von Entwicklungsprozessen auf der Grundlage der erläuterten Modellierungskonstrukte und stellt zusammenfassend die Integrationspunkte zu dem in Kapitel 4 erarbeiteten Metamodell für technische Produkte dar (vgl. Abbildung 4.16). Ein Prozess (engl.: *process*) besteht demnach aus einer Menge von Aktionen (engl.: *action*) und einer Menge aus Transitionen (engl.: *transition*). Jede Transition setzt genau zwei Aktionen zueinander in Beziehung und legt so eine Ablaufreihenfolge fest. Umgekehrt kann eine Aktion an verschiedenen Transitionen beteiligt sein.

Aktionen besitzen die vier, bereits erläuterten Spezialisierungen der Aktivität (engl.: *activity*), des Verzweigungspunkts (engl.: *fork*), des Vereinigungspunkts (engl.: *join*) sowie des Entscheidungspunkts (engl.: *decision*).

[8] Die Spezifikation von Konsistenzbedingungen kann ebenso in Form vorbereitender wie auch operativer Aktivitäten eines Entwicklungsprozesses dargestellt werden.

Wie in Abschnitt 4.3.1 dargelegt, kann sich jede Aktivität auf eine feste Anzahl von Sichten (engl.: *view*) des Produktmodells beziehen, welche für ihre Durchführung erforderlich sind oder darin modifiziert werden. Bereits durch diese Zuordnung ergibt sich eine methodische und transparente Verknüpfung zwischen den Aktivitäten eines Prozesses und den Elementen eines Produktmodells bzw. seinen Produktsichten.

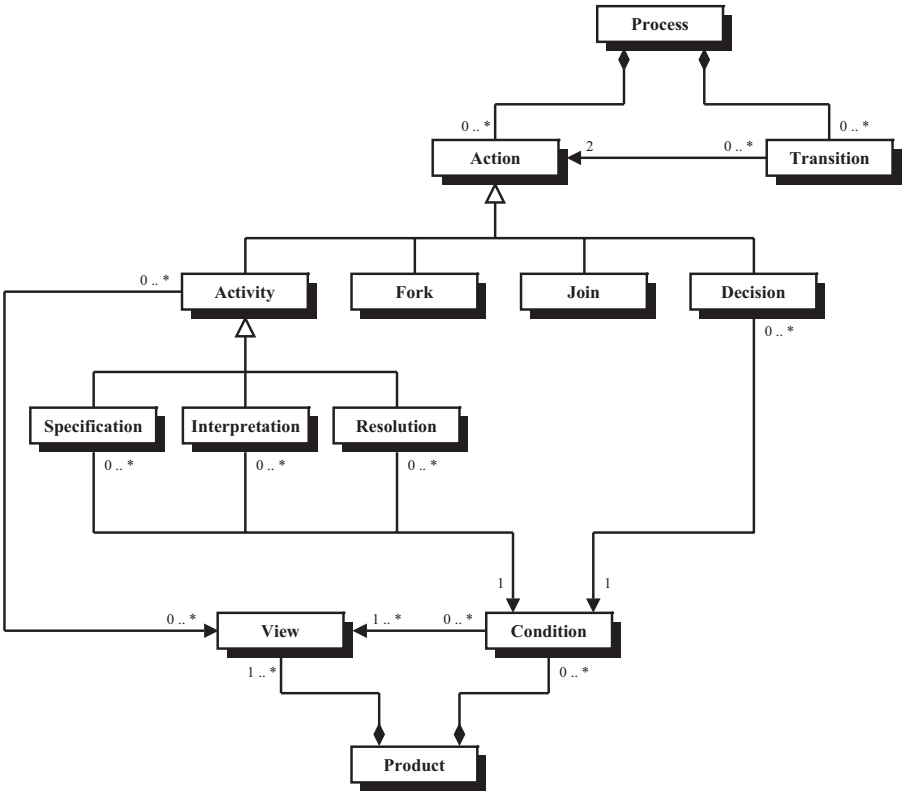

Abbildung 6.8: Integrationspunkte zwischen Prozess- und Produktmodell

Ferner lassen sich die aus Abschnitt 6.2 bekannten Schritte der Spezifikation (engl.: *specification*), Interpretation (engl.: *interpretation*) und Resolution (engl.: *resolution*) gemäß der übergreifenden Konzeption einer kontrollierten und flexiblen Konsistenzsicherung als spezielle Aktivitäten nahtlos in ein Prozessmodell einordnen. Aus ihrem jeweils gültigen Bezug auf genau eine zugeordnete Konsistenzbedingung (engl.: *condition*) des Produktmodells, welche ihrerseits verschiedene Produktsichten zueinander in Relation setzt, resultiert der zweite Ansatzpunkt einer integrativen Behandlung von Produkt- und Prozessmodell.

Schließlich leistet die Verknüpfung von Entscheidungspunkten eines Prozessmodells mit Konsistenzbedingungen eines Produktmodells einen wesentlichen Beitrag zur Integration von Entwicklungsprozessen einerseits mit den Partialmodellen eines Produkts andererseits. Jeder Entscheidungspunkt eines Entwicklungsprozesses bezieht sich auf genau eine Konsistenzbedingung. Dadurch kann mithilfe einer zugeordneten Interpretation flexibel unter Berücksichtigung des Entwicklungskontexts der Zustand der Konsistenzbedingung – und somit des Produktmodells – festgestellt und abhängig von dem ermittelten Ergebnis der weitere Prozessablauf gesteuert werden.

6.4.2 Ereignissteuerung

Der vorgestellte Ansatz der Prozesssteuerung fasst die Schritte der Konsistenzsicherung als spezifische Bestandteile eines Entwicklungsprozesses auf und verknüpft auf diese Weise Produkt- und Prozessmodell. Darüber hinaus bietet die Alternative der Ereignissteuerung erweiterte Möglichkeiten zur Durchführung von Konsistenzsicherungsprozessen. Maßnahmen zur Konsistenzsicherung werden dabei als Reaktion auf eintretende Ereignisse ausgeführt, welche im Zuge einer Produktentwicklung auftreten. Diese können beispielsweise im Ablauf des Prozesses oder in der Ausführung von Benutzeraktionen begründet sein.

Allgemein betrachtet besteht ein Ereignis-gesteuertes System, auch als Ereignis-basiertes oder Ereignis-orientiertes System bezeichnet, aus *Ereignisquellen*, *Ereignissen* sowie *Ereignisempfängern*. Im Rahmen einer übergreifenden Integrationsplattform gemäß Abschnitt 3.4 können diese Rollen wie folgt verteilt werden:

Ereignisquellen: Als Ereignisquellen werden die Verursacher von Ereignissen bezeichnet. Im Rahmen einer übergreifenden Entwicklungsplattform kommen die Komponenten der Steuerung sowie vorhandene fachliche wie technische Dienste, aber auch beteiligte Bearbeiter als mögliche Ereignisquellen in Frage.

Ereignisse: Ereignisse bezeichnen jede Art an Vorfällen, welche im Zuge eines Entwicklungsprozesses durch Ereignisquellen erzeugt werden und für Benutzer oder Komponenten einer Entwicklungsplattform von Interesse sind. In diesem Zusammenhang werden Ereignisse in drei Kategorien eingeteilt:

- *Prozess-Ereignisse* werden durch den Ablauf des Prozesses gemäß eines festgelegten Prozessmodells verursacht, beispielsweise der Beginn oder der Abschluss einer Aktivität.

- Darüber hinaus können *System-Ereignisse*, z.B. der Aufruf eines Dienstes bzw. Software-Werkzeugs, beliebige Komponenten einer Integrationsplattform als Verursacher besitzen.
- Schließlich repräsentieren *Benutzer-Ereignisse* jede Art von Interaktion zwischen einer übergreifenden Entwicklungsplattform und ihren Benutzern.

Ereignisempfänger: Die Aufgabe von Ereignisempfängern ist die Verarbeitung eingetretener Ereignisse. Analog zu Ereignisquellen sind die Komponenten der Steuerung, vorhandene Dienste oder beteiligte Benutzer mögliche Ereignisempfänger einer übergreifenden Integrationsplattform.

Ereignis-orientierte Mechanismen werden in vielerlei Bereichen eingesetzt, um eine asynchrone Kommunikation zwischen den Komponenten eines Systems zu ermöglichen. Ihre Umsetzung erfolgt meist auf der Grundlage des als Beobachter-Muster oder auch Observer-Pattern bekannten Entwurfsmusters (s. [GHJV95]). Hierbei registrieren sich Ereignisempfänger im Vorfeld bei möglichen Ereignisquellen, um über den Eintritt bestimmter Ereignisse informiert zu werden. Tritt ein Ereignis auf, so benachrichtigt eine Ereignisquelle alle registrierten Ereignisempfänger, welche daraufhin geeignete Maßnahmen ergreifen.

In Hinblick auf Konsistenzsicherungsprozesse bietet eine Ereignis-orientierte Ablaufkontrolle die Möglichkeit, die Schritte der Interpretation und Resolution gemäß der Abschnitte 6.2 und 6.3 flexibel als Reaktion auf eintretende Ereignisse durchzuführen. Auf diese Weise werden über den Ansatz der Prozesssteuerung hinaus auch System- oder Benutzer-bezogene Ereignisse, wie beispielsweise die Anfrage eines Bearbeiters oder der Aufruf eines Software-Werkzeugs, berücksichtigt. In Abbildung 6.9 wird der Ablauf einer Ereignis-gesteuerten Konsistenzsicherung schematisch durch ein Sequenzdiagramm aufgezeigt.

Zu Beginn registrieren sich Ereignisempfänger bei möglichen Ereignisquellen und bekunden so ihr Interesse, über bestimmte Ereignisse informiert zu werden. In der Folge werden sie von Ereignisquellen benachrichtigt, sobald diese ein Ereignis erzeugen. Ein Ereignisempfänger kann daraufhin die Art des Ereignisses ermitteln und den Vorgang der Interpretation einer Konsistenzbedingung anstoßen, welche ihrerseits – unter Berücksichtigung des Entwicklungskontexts – deren Zustand ermittelt. Im Falle einer Inkonsistenz wird schließlich die Resolution als letzter Schritt der Konsistenzsicherung durchgeführt.

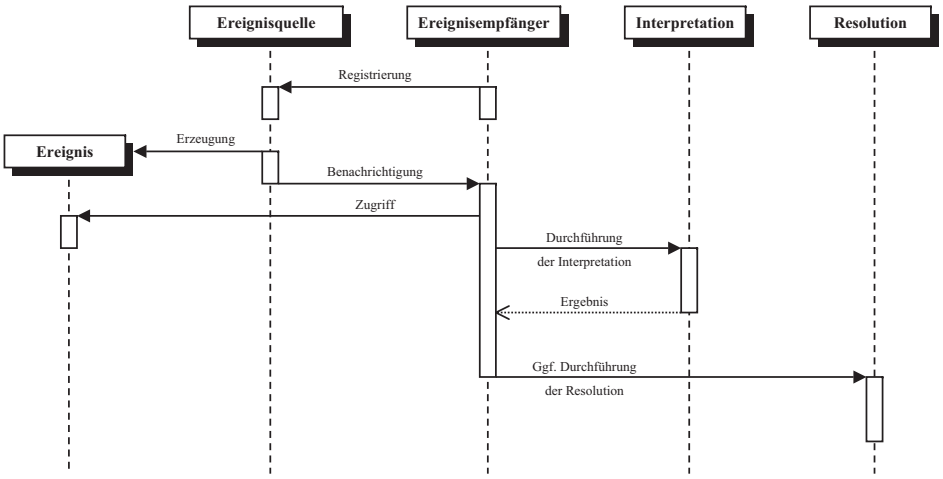

Abbildung 6.9: Prinzip der Ereignis-gesteuerten Konsistenzsicherung

6.5 Zusammenfassung

Der in Kapitel 4 vorgeschlagene Ansatz der segmentierten Produktmodellierung begreift ein Produktmodell als Reihe eigenständiger Partialmodelle in Verbindung mit explizit festgelegten Konsistenzbedingungen. Somit kann jede Produktsicht optimal an vorhandene Organisationsstrukturen, durchzuführende Entwicklungsaktivitäten und eingesetzte Software-Werkzeuge angepasst sowie individuell bearbeitet werden. Die aus dem Bezug auf ein gemeinsames Produkt resultierenden Überschneidungen, Querbezüge und Abhängigkeiten zwischen den Elementen verschiedener Produktsichten werden dagegen gesondert erfasst.

Folgerichtig ist im Sinne einer durchgängigen und integrativen Produktentwicklung explizit sicher zu stellen, dass ein Produktmodell mit dem Abschluss eines Entwicklungsprozesses in sich widerspruchsfrei ist, also ein potenziell reales Produkt beschreibt. Gemäß dieser Zielsetzung ermöglicht und unterstützt der Ansatz der segmentierten Produktmodellierung insbesondere eine praxisgerechte Behandlung der Modellkonsistenz auf der Grundlage differenzierter Konsistenzbegriffe, variabler Techniken zur Festlegung von Konsistenzbedingungen sowie flexibler Strategien zu ihrer Auswertung und Sicherung.

In Abschnitt 6.1 wird ein grundlegender Konsistenzbegriff für Produktsichten eingeführt und so auf Produktmodelle erweitert, dass eine differenzierte Unterscheidung zwischen lokalen und globalen Konsistenzeigenschaften möglich ist. Darauf aufbauend stellt Abschnitt 6.2 die Konzeption von Konsistenzsicherungsprozessen vor, welche sich stets modular in die drei elementaren Schritte der Spezifikation zur Festlegung einer Konsistenzbedingung, der Interpretation zu ihrer Überprüfung unter Berücksichtigung des Entwicklungskontexts und der Resolution für die ggf. erforderliche Wiederherstellung von Konsistenz gliedern.

Angesichts der Vielfältigkeit fachlicher Zusammenhänge zwischen den Elementen eines Produktmodells sieht die vorgeschlagene Konzeption eine gleich berechtigte Verwendung verschiedenartiger Spezifikationstechniken für Konsistenzbedingungen vor. Vor diesem Hintergrund gibt Abschnitt 6.3 eine kurze Einführung in vorhandene Ansätze der Wissensrepräsentation, zeigt ihre Einsatzmöglichkeiten innerhalb des übergreifenden Rahmens auf und vergleicht sie anhand ausgesuchter Kriterien. Informelle Beschreibungsformen finden dabei ebenso Erwähnung wie formale Techniken auf der Grundlage logischer Kalküle.

Schließlich wird untersucht, wie sich die Schritte der Konsistenzsicherung als spezifische Aufgaben einer Produktentwicklung methodisch in den Ablauf eines Entwicklungsprozesses einordnen lassen. Dazu erläutert Abschnitt 6.4 zunächst die grundlegenden Bestandteile von Entwicklungsprozessen und zeigt ihre Integrationspunkte zu dem in Kapitel 4 entwickelten Metamodell für technische Produkte auf. Die Hinzunahme Ereignis-orientierter Mechanismen erlaubt darüber hinaus die Durchführung von Konsistenzsicherungsprozessen als flexible Reaktion auf eintretende Ereignisse der Produktentwicklung.

7 Automatisierung

Die explizite Prüfung und Sicherung der Konsistenz eines Produktmodells sind wesentliche Bestandteile einer integrativen und durchgängigen Produktentwicklung. Zum einen bilden sie die folgerichtige Ergänzung zu dem in Kapitel 4 vorgeschlagenen Ansatz der Segmentierung, der es erlaubt, Produktsichten unabhängig voneinander zu erstellen und zu bearbeiten. Wie die vorangehenden Abschnitte aufzeigen, ermöglicht diese Vorgehensweise zudem einen differenzierten und praxisgerechten Umgang mit Konsistenzeigenschaften mittels variabler Spezifikationstechniken und flexibler Auswertungsstrategien.

In der vorgeschlagenen Konzeption der Konsistenzsicherung werden Querbezüge zwischen verschiedenen Produktsichten in ein integriertes Produktmodell in Form von Konsistenzbedingungen eingebracht, die individuell gemäß ihrer Interpretation und Resolution (vgl. Abschnitt 6.2) geprüft und gesichert werden können. Angesichts der Vielzahl zu erwartender Konsistenzbedingungen eines realen Produkts bzw. zugehöriger Partialmodelle ist eine weit gehende Automatisierung dieser Schritte sehr erstrebenswert. Nur so lässt sich eine praxisgerechte Unterstützung von Entwicklern in Konsistenzsicherungsprozessen gewährleisten.

Während die Automatisierbarkeit der Interpretation und Resolution einzelner Konsistenzbedingungen wesentlich von der gewählten Spezifikationstechnik abhängt (vgl. Abschnitt 6.3 und Tabelle 6.2), sind für die Automatisierung übergreifender Konsistenzsicherungsprozesse, welche verschiedene Konsistenzbedingungen betreffen, zudem mögliche Resolutionskonflikte (s. Abschnitt 6.2.3) besonders zu berücksichtigen. In diesem Zusammenhang besteht die Aufgabe eine geeignete Reihenfolge zur Durchführung erforderlicher Resolutionen zu finden, so dass Resolutionskonflikte vermieden werden und alle betrachteten Konsistenzbedingungen schließlich erfüllt sind.

Gemäß dieser Zielsetzung präzisiert Abschnitt 7.1 vorbereitend die so beschriebene Problemstellung der Automatisierung einer übergreifenden Konsistenzsicherung und zeigt vorhandene Grenzen der Automatisierbarkeit auf. Auf dieser Grundlage präsentiert Abschnitt 7.2 hinreichende Bedingungen,

die eine konfliktfreie Abfolge von Resolutionen gewährleisten. Daraus wird anschließend in Abschnitt 7.3 ein Algorithmus abgeleitet, der für eine Menge von Konsistenzbedingungen und zugehörigen Resolution eine geeignete Reihenfolge ihrer Durchführung ermittelt, so dass Resolutionskonflikte vermieden werden.

7.1 Das Resolutionsproblem

Der Vorgang der Resolution bezeichnet die ggf. erforderliche Wiederherstellung der relativen Konsistenz eines Produktmodells hinsichtlich einer ausgewählten Konsistenzbedingung und unter Berücksichtigung des Entwicklungskontexts. Verschiedene Resolutionen können sich gegenseitig negativ beeinflussen: In diesem Fall führen die im Rahmen einer Resolution zur Sicherung der aktuell betrachteten Konsistenzbedingung vorgenommenen Modellmodifikationen dazu, dass eine zuvor gesicherte Konsistenzbedingung verletzt wird. Im Ergebnis verhindert dies die Transformation eines Produktmodells in einen global konsistenten Zustand.

Vor dem Hintergrund einer stets erfüllbaren Menge an Konsistenzbedingungen (vgl. Abschnitt 6.1) können Resolutionskonflikte auf verschiedene Arten gelöst werden (s. Abschnitt 6.2.3). Ist vorausgesetzt, dass eine Reihe von Konsistenzbedingungen nicht individuell, sondern stets in ihrer Gesamtheit betrachtet wird, so besteht ein einfacher, wenn auch nicht immer möglicher Weg zur Vermeidung von Resolutionskonflikten in der Suche einer geeigneten Reihenfolge zur Durchführung zugehöriger Resolutionen. Insbesondere hinsichtlich automatisierter Resolutionsverfahren spielt dieser Lösungsansatz eine wichtige Rolle.

Zur präzisen Formulierung der so gegebenen Problemstellung greifen wir auf die in Kapitel 5 erarbeitete mathematische Fundierung des Produktmodells auf der Grundlage von Graphen und Algebren zurück. Im Folgenden sei daher eine Menge $\mathcal{M}$ zu betrachtender Produktmodelle, ein ausgewähltes Produktmodell

$$P = (S_1, \ldots, S_n) \in \mathcal{M},$$

mit n Produktsichten, $n \in \mathbb{N}$, gemäß der Definition 5.14, sowie eine endliche Menge

$$B = \{b_1, \ldots, b_m\}$$

von m Konsistenzbedingungen, $m \in \mathbb{N}$, gegeben. Für jede der Konsistenzbedingungen $b_i \in B$ stehe eine zugehörige Resolution ϱ_i, $1 \leq i \leq m$, zur Verfügung, welche sich Abschnitt 5.3 folgend als Produktmodell-Transformation $\varrho_i : \mathcal{M} \to \mathcal{M}$ auffassen lässt. Zu einer gegebenen Teilmenge $B' \subseteq B$ von

Konsistenzbedingungen vereinbaren wir zudem die Schreibweisen

$$P' \models B' \qquad \text{bzw.} \qquad P' \not\models B',$$

falls das Produktmodell $P' \in \mathcal{M}$ jede der in B' gelegenen Konsistenzbedingungen erfüllt bzw. mindestens eine dieser Bedingungen nicht erfüllt. Somit gilt für jedes Produktmodell $P' \in \mathcal{M}$ insbesondere

$$P'[\varrho_i] \models \{b_i\},$$

d.h. die Anwendung der Resolution ϱ_i auf ein Produktmodell P' führt stets dazu, dass die Konsistenzbedingung b_i, $1 \leq i \leq m$, erfüllt ist. Auf dieser Grundlage lässt sich das Resolutionsproblem wie folgt angeben:

Definition 7.1 (*Resolutionsproblem*)

Gegeben sei eine Menge $\mathcal{M}$ an Produktmodellen, ein ausgewähltes Produktmodell $P \in \mathcal{M}$ und eine Menge $B = \{b_1, \ldots, b_m\}$, $m \in \mathbb{N}$, an Konsistenzbedingungen mit zugehörigen Resolutionen $\varrho_i : \mathcal{M} \to \mathcal{M}$, $1 \leq i \leq m$. Gibt es eine Folge $i_1, \ldots, i_k$, $k \in \mathbb{N}$, von Indizes $i_j \in \{1, \ldots, m\}$, $1 \leq j \leq k$, so dass

$$P[\varrho_{i_1}, \ldots, \varrho_{i_k}] \models B$$

gilt, d.h. alle Konsistenzbedingungen erfüllt sind?

Das in Definition 7.1 beschriebene Resolutionsproblem tritt bei der Automatisierung übergreifender Konsistenzsicherungsprozesse auf: Ist jede der betrachteten Resolutionen automatisiert durchführbar, so lässt sich auch das Produktmodell stets automatisiert in einen Zustand relativer Konsistenz hinsichtlich je einer der zugehörigen Konsistenzbedingungen transformieren. Die Wahl einer ungünstigen Reihenfolge der einzelnen Resolutionen kann jedoch insgesamt dazu führen, dass aufgrund von Resolutionskonflikten letztlich kein global konsistentes Produktmodell erreicht wird. Das folgende Beispiel verdeutlicht diese Problemstellung:

Beispiel 7.1 (*Resolutionsproblem*)

Als einfaches Beispiel eines Resolutionsproblems untersuchen wir das in Abbildung 7.1 dargestellte Produktmodell P. Dieses umfasst die beiden Produktsichten S_1 und S_2, welche über je eine Komponente mit dem ganzzahligen Attribut x_1 bzw. x_2 verfügen. Ferner betrachten wir die Konsistenzbedingungen $b_1 : x_1 > 0$ und $b_2 : x_2 = x_1$. Die zugehörigen Resolutionen ϱ_1 für b_1 und ϱ_2 für b_2 seien in nahe liegender Weise durch $\varrho_1 : x_1 := 1$ und $\varrho_2 : x_2 := x_1$ definiert, d.h. ϱ_1 weise dem Attribut x_1 den Wert $x_1 = 1$ zu, während ϱ_2 dem

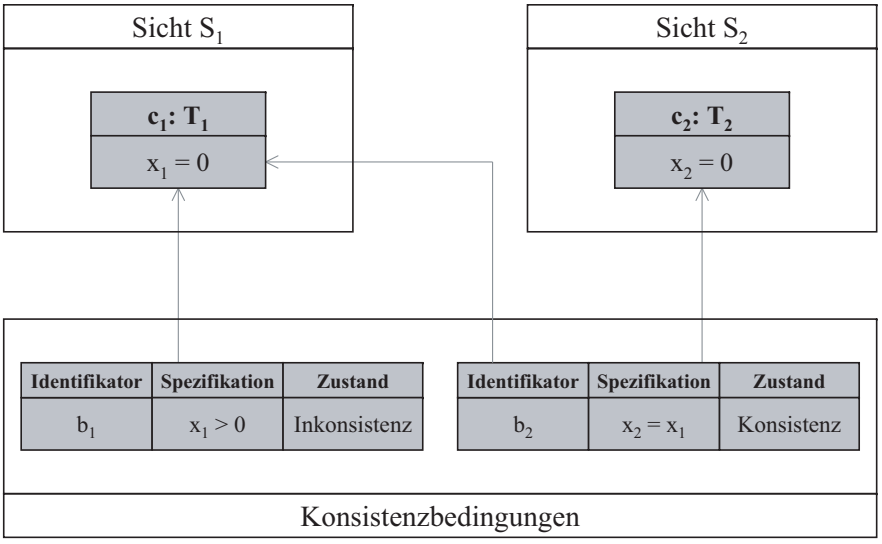

Abbildung 7.1: Beispiel eines Resolutionsproblems

Attribut x_2 den Wert des Attributs x_1 zuweise. Mit diesen Festlegungen gilt offenbar

$$P[\varrho_1] \models \{b_1\} \qquad \text{und} \qquad P[\varrho_2] \models \{b_2\}.$$

Zu Beginn besitzen beide betrachteten Attribute den Wert $x_1 = x_2 = 0$, so dass die Konsistenzbedingung b_1 verletzt und die Konsistenzbedingung b_2 erfüllt ist. Die sukzessive Anwendung der Resolutionen ϱ_1 und ϱ_2 in dieser Reihenfolge führt dazu, dass zunächst dem Attribut x_1 der Wert $x_1 = 1$ zugewiesen wird. In der Folge ist die Konsistenzbedingung b_1 erfüllt, während die Konsistenzbedingung b_2 verletzt wird. Die anschließende Ausführung der Resolution ϱ_2 führt jedoch zu dem Ergebnis $x_2 = x_1 = 1$, so dass auch die zweite Konsistenzbedingung b_2 erfüllt ist, also ein global konsistentes Produktmodell vorliegt.

Werden die betrachteten Resolutionen ϱ_1 und ϱ_2 dagegen in umgekehrter Reihenfolge angewendet, so führt die Ausführung der Resolution ϱ_2 zur Sicherung der Konsistenzbedingung b_2 zunächst zu dem Ergebnis $x_2 = x_1 = 0$, bewirkt also keinerlei Änderung des Modellzustands. Anschließend modifiziert die Resolution ϱ_1 den Wert des Attributs x_1 zu $x_1 = 1$, um die Konsistenzbedingung b_1 zu sichern. Hierbei wird jedoch die zuvor erfüllte Konsistenzbedingung b_2 verletzt, da nun $x_1 = 1$ und $x_2 = 0$ gilt. Dieser Resolutionskonflikt verhindert insgesamt die Transformation des Produktmodells in einen global

konsistenten Zustand.

Zusammenfassend gilt also für das betrachtete Produktmodell

$$P[\varrho_1, \varrho_2] \models \{b_1, b_2\},$$

jedoch

$$P[\varrho_2, \varrho_1] \not\models \{b_1, b_2\},$$

da jede Resolution nur eine Konsistenzbedingung berücksichtigt.

Wie das betrachtete Beispiel zeigt, kann die sukzessive Durchführung verschiedener Resolutionen unerwünschte Wechselwirkungen aufweisen. Eine konstruktive Lösung des Resolutionsproblems gewährleistet dagegen die Herstellung eines global konsistenten Produktmodells. Gemäß des folgenden Satzes ist eine solche jedoch im Allgemeinen nicht ermittelbar:

Satz 7.1 (*Unentscheidbarkeit des Resolutionsproblems*)

Das Resolutionsproblem ist nicht entscheidbar.

Satz 7.1 besagt, dass kein algorithmisches Verfahren existieren kann, welches das in Definition 7.1 gegebene Resolutionsproblem allgemein löst[1]. Um dies nachzuweisen, setzen wir das Resolutionsproblem in Beziehung zu dem gemeinhin als unentscheidbar bekannten (speziellen) *Post'schen Korrespondenzproblem* (s. beispielsweise [Sch97]):

Definition 7.2 (*(Spezielles) Post'sches Korrespondenzproblem*)

Gegeben sei eine endliche Folge von Wortpaaren $(x_1, y_1), \ldots, (x_k, y_k)$, $k \in \mathbb{N}$, mit $x_i, y_i \in \{0,1\}^+$, $1 \leq i \leq k$. Gibt es eine Folge $i_1, \ldots, i_l$, $l \in \mathbb{N}$, von Indizes $i_j \in \{1, \ldots, k\}$, $1 \leq j \leq l$ mit $x_{i_1} x_{i_2} \ldots x_{i_l} = y_{i_1} y_{i_2} \ldots y_{i_l}$?

Aus der Unentscheidbarkeit des angegebenen Post'schen Korrespondenzproblems lässt sich die Unentscheidbarkeit des Resolutionsproblems folgern. Hierzu genügt es allgemein, ein bereits als nicht entscheidbar bekanntes Problem – in diesem Fall das Post'sche Korrespondenzproblem – als Spezialfall in das neu zu untersuchende Problem, also das Resolutionsproblem gemäß Definition 7.1, einzubetten (vgl. [Sch97]). Ein angenommenes Entscheidungsverfahren für das Resolutionsproblem würde dann insbesondere die Entscheidbarkeit des Post'schen Korrespondenzproblem implizieren und somit zu einem Widerspruch führen.

[1] Zum Begriff der *Unentscheidbarkeit* siehe z.B. [Sch97].

Beweis 7.1 (*Unentscheidbarkeit des Resolutionsproblems*)

Im Folgenden zeigen wir die Unentscheidbarkeit des Resolutionsproblems, indem wir das Post'sche Korrespondenzproblem auf dieses reduzieren. Dazu seien gemäß Definition 7.2 k Wortpaare $(x_1, y_1), \ldots, (x_k, y_k)$, $k \in \mathbb{N}$, mit $x_i, y_i \in \{0,1\}^+$, $1 \leq i \leq k$, gegeben. Auf dieser Grundlage konstruieren wir das in Abbildung 7.2 dargestellte Produktmodell, welches aus den fünf Sichten $S_1, \ldots, S_5$ sowie $2k+2$ Konsistenzbedingungen besteht. Jede Sicht verfügt über genau eine Komponente. Dabei sollen die Komponenten c_1 bzw. c_2 der Sichten S_1 bzw. S_2 die Attribute $x_1, \ldots, x_k$ bzw. $y_1, \ldots, y_k$ gemäß des gegebenen Post'schen Korrespondenzproblems besitzen.

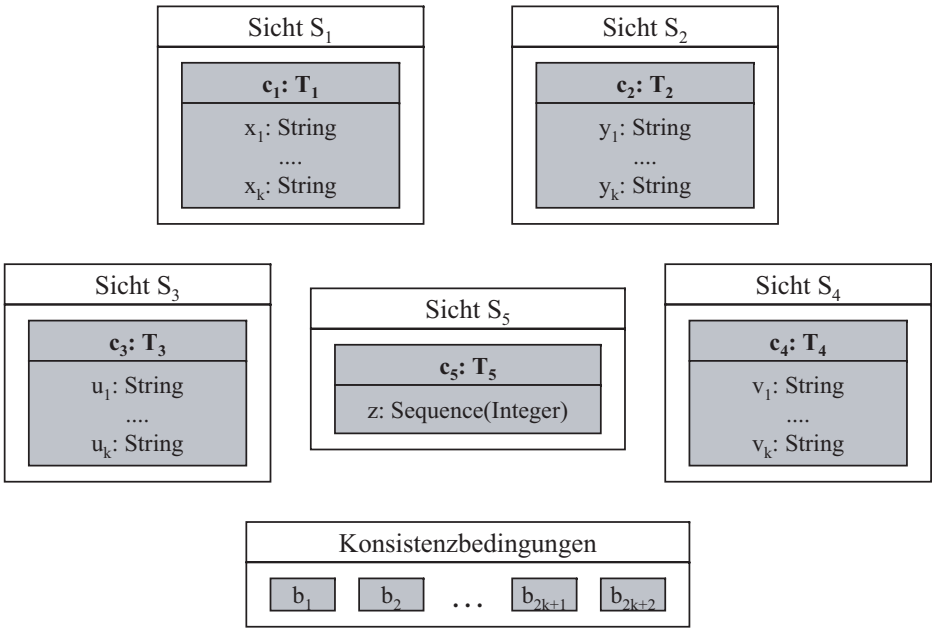

Abbildung 7.2: Produktmodell für das Post'sche Korrespondenzproblem

Analog dazu existieren die beiden Sichten S_3 und S_4, deren Komponenten c_3 bzw. c_4 ebenfalls über je k Attribute $u_1, \ldots, u_k$ bzw. $v_1, \ldots, v_k$ verfügen. Initial sollen die Beziehungen $u_i = x_i$ und $v_i = y_i$, $1 \leq i \leq k$, erfüllt sein. Jedes der genannten Attribute repräsentiert eine Zeichenreihe $w \in \{0,1\}^+$ von Binärziffern gemäß des Post'schen Korrespondenzproblems und besitzt aufgrund der ggf. erforderlichen Berücksichtigung führender Nullen die Sorte *String* (s. Abschnitt 4.3.4). Schließlich dient das Attribut z der Komponente c_5 in Sicht S_5 dazu, eine Indexfolge ganzer Zahlen $z = (i_1, \ldots, i_l)$ mit $i_1, \ldots, i_l \in \{1, \ldots, k\}$, $l \in \mathbb{N}$, aufzunehmen.

Zu den beschriebenen Sichten S_1 bis S_5 werden $2k+2$ Konsistenzbedingungen betrachtet, welche sich zum einen auf die Gleichheit der Attribute x_i und u_i bzw. y_i und v_i, $1 \leq i \leq k$, gemäß folgender Festlegungen beziehen:

$$\begin{array}{lcllcl} b_1 & : & u_1 = x_1, & b_{k+1} & : & v_1 = y_1, \\ & & \vdots & & & \vdots \\ b_k & : & u_k = x_k, & b_{2k} & : & v_k = y_k. \end{array}$$

Die Konsistenzbedingungen b_{2k+1} und b_{2k+2} seien zudem wie folgt definiert:

$$b_{2k+1} \;:\; x_1 = x_1, \qquad\qquad b_{2k+2} \;:\; u_{i_1} \ldots u_{i_l} = v_{i_1} \ldots v_{i_l},$$

falls das Attribut z die Indexfolge $z = (i_1, \ldots, i_l)$ mit $i_1, \ldots, i_l \in \{1, \ldots k\}$ und $l \in \mathbb{N}$ repräsentiert. Die Konsistenzbedingung b_{2k+1} ist offenbar stets erfüllt, während b_{2k+2} genau der Bedingung des Post'schen Korrespondenzproblems mit den Attributen u_i und v_i anstelle von x_i und y_i, $1 \leq i \leq k$, sowie der in z enthaltenen Indexfolge $z = (i_1, \ldots, i_l)$ entspricht.

Jeder gegebenen Konsistenzbedingung $b_1, \ldots, b_{2k+2}$ ist schließlich eine Resolution ϱ_i, $1 \leq i \leq 2k+2$, d.h. eine Produktmodell-Transformation gemäß Abschnitt 5.3 zuzuordnen:

$$\begin{array}{lcllcl} \varrho_1 & : & u_1 := x_1, & \varrho_{k+1} & : & v_1 := y_1, \\ & & \vdots & & & \vdots \\ \varrho_k & : & u_k := x_k, & \varrho_{2k} & : & v_k := y_k. \end{array}$$

Die Resolutionen $\varrho_1, \ldots, \varrho_{2k}$ weisen also den Attributen u_i bzw. v_i den Wert der Attribute x_i bzw. y_i zu $(1 \leq i \leq k)$. Ferner seien die Resolutionen ϱ_{2k+1} und ϱ_{2k+2} durch

$$\varrho_{2k+1} : z = (i_1, \ldots, i_l) \mapsto \begin{cases} z = (i_1, \ldots, i_l + 1), & \text{falls } i_l < k, \\ z = (i_1, \ldots, i_l, 1), & \text{falls } i_l = k \end{cases}$$

sowie

$$\varrho_{2k+2} : v_1 := u_1, \ldots, v_k := u_k$$

gegeben. Die sukzessive Anwendung der Resolution ϱ_{2k+1} erlaubt es demnach, alle möglichen Indexfolgen $z = (i_1, \ldots, i_l)$ mit Indizes $i_1, \ldots i_l \in \{1, \ldots, k\}$, $l \in \mathbb{N}$, zu generieren. Dagegen gewährleistet die Resolution ϱ_{2k+2} eine triviale Lösung der Konsistenzbedingung b_{2k+2}, welche jedoch im Regelfall zu Resolutionskonflikten mit den Bedingungen $b_1, \ldots, b_{2k}$ führt.

Insgesamt betrachtet lässt sich auf die genannte Weise zu jedem vorgegebenen Post'schen Korrespondenzproblem gemäß Definition 7.2 mittels einer (totalen und berechenbaren) Transformation ein Produktmodell P konstruieren, welches aus den in Abbildung 7.2 dargestellten Sichten S_1 bis S_5 besteht

und die beschriebenen Konsistenzbedingungen $b_1, \ldots, b_{2k+2}$ aufweist. Werden die festgelegten Resolutionen $\varrho_1, \ldots, \varrho_{2k+2}$ individuell betrachtet, so gilt offenbar

$$P[\varrho_i] \models \{b_i\}$$

für jeden Index $i \in \{1, \ldots, 2k+2\}$. Vor diesem Hintergrund besteht folgender Zusammenhang zwischen einem vorgegebenen Post'schen Korrespondenzproblem gemäß Definition 7.2 und dem in Definition 7.1 erläuterten Resolutionsproblem für das gemäß der dargelegten Ausführungen konstruierte Produktmodell P:

Das Resolutionsproblem besitzt eine Lösung für das konstruierte Produktmodell P

$\Leftrightarrow$ Es gibt eine Indexfolge $(i_1, \ldots, i_l)$, $i_1, \ldots, i_l \in \{1, \ldots, 2k+2\}$, $l \in \mathbb{N}$, mit $P[\varrho_{i_1}, \ldots, \varrho_{i_l}] \models \{b_1, \ldots, b_{2k+2}\}$

$\Leftrightarrow$ Es gibt eine Indexfolge $(i_1, \ldots, i_l)$, $i_1, \ldots, i_l \in \{1, \ldots, 2k+2\}$, $l \in \mathbb{N}$, mit

$$\begin{array}{ccccccccc} u_1 = x_1 & \wedge & u_2 = x_2 & \wedge & \ldots & \wedge & u_k = x_k & \wedge \\ v_1 = y_1 & \wedge & v_2 = y_2 & \wedge & \ldots & \wedge & v_k = y_k & \end{array}$$

und

$$u_{i_1} u_{i_2} \ldots u_{i_l} \;=\; v_{i_1} v_{i_2} \ldots v_{i_l}$$

$\Leftrightarrow$ Es gibt eine Indexfolge $(i_1, \ldots, i_l)$, $i_1, \ldots, i_l \in \{1, \ldots, 2k+2\}$, $l \in \mathbb{N}$, mit

$$x_{i_1} x_{i_2} \ldots x_{i_l} \;=\; y_{i_1} y_{i_2} \ldots y_{i_l}$$

$\Leftrightarrow$ Das Post'sche Korrespondenzproblem besitzt eine Lösung

Somit ist nachgewiesen, dass sich das Post'sche Korrespondenzproblem als Spezialfall in das Resolutionsproblem gemäß Definition 7.1 einbetten lässt. Setzt man die Unentscheidbarkeit des Post'schen Korrespondenzproblems als bekannt voraus, so folgt daraus unmittelbar die Unentscheidbarkeit des Resolutionsproblems.

Satz 7.1 besagt, dass kein – weder manuelles noch automatisiertes – Entscheidungsverfahren für das Resolutionsproblem existiert, das mit der gemeinhin akzeptierten Auffassung von Berechenbarkeit vereinbar ist[2]. Analog zum Post'schen Korrespondenzproblem ist das Resolutionsproblem jedoch (positiv) semi-entscheidbar.

[2]Zum Begriff der Berechenbarkeit siehe z.B. [Sch97] oder [Bro95].

7.2 Der Resolutionssatz

Die Unentscheidbarkeit des Resolutionsproblems verhindert automatisierte Resolutionsverfahren, welche simultan mehrere Konsistenzbedingungen umfassen. Als Ursache sind potenziell auftretende Wechselwirkungen zwischen einzelnen Resolutionen zu nennen, welche dazu führen können, dass im Verlauf eines solchen, übergreifenden Resolutionsprozesses gesicherte Konsistenzbedingungen durch später angewendete Resolutionen wiederum verletzt werden. Angesichts des Umfangs und der Komplexität realistischer Produktmodelle und zugehöriger Konsistenzbedingungen erfordert eine praxisgerechte Lösung jedoch den Einsatz automatisierter Verfahren.

Vor diesem Hintergrund sind für eine Automatisierung übergreifender Resolutionsprozesse zusätzlich die Wechselwirkungen zwischen einzelnen Resolutionen zu betrachten, um Resolutionskonflikte – sofern möglich – auszuschließen. Lässt sich eine Reihenfolge für die Durchführung vorhandener Resolutionen finden, so dass einmal gesicherte Konsistenzbedingungen durch darauf folgende Resolutionen nicht verletzt werden, so kann eine sukzessive Transformation des Produktmodells in einen global konsistenten Zustand erreicht werden. Gemäß dieser Zielsetzung wird zunächst der Begriff der *Neutralitätsmenge* einer Resolution eingeführt:

Definition 7.3 (*Neutralitätsmenge einer Resolution*)

Gegeben sei eine Menge $\mathcal{M}$ an Produktmodellen, ein ausgewähltes Produktmodell $P \in \mathcal{M}$ und eine Menge $B = \{b_1, \ldots, b_m\}$, $m \in \mathbb{N}$, an Konsistenzbedingungen mit zugehörigen Resolutionen $\varrho_i : \mathcal{M} \rightarrow \mathcal{M}$, $1 \leq i \leq m$. Dann heißt

$$N(\varrho_i) := \{b' \in B : \forall P' \in \mathcal{M} : P' \models \{b'\} \rightarrow P'[\varrho_i] \models \{b'\}\}$$

die Neutralitätsmenge der Resolution ϱ_i $(1 \leq i \leq m)$.

Die Neutralitätsmenge einer Konsistenzbedingung umfasst demnach alle die Konsistenzbedingungen, welche durch die Anwendung einer Resolution nicht negativ beeinflusst werden, also erfüllt bleiben. Aus Gründen der Einfachheit wird im Folgenden stets davon ausgegangen, dass die Neutralitätsmengen betrachteter Resolutionen bekannt und vorgegeben sind. Abhängig von der gewählten Spezifikationstechnik einer Konsistenzbedingung könnten sich diese jedoch ebenso automatisiert ermitteln lassen oder explizit im Rahmen der Spezifikation einer Konsistenzbedingung (s. Abschnitt 6.2.1) definiert werden.

Aus den Neutralitätsmengen betrachteter Resolutionen lassen sich offenbar Rückschlüsse auf ihre Wechselwirkungen ziehen. Liegt eine Konsistenzbedingung $b_i \in B$ in der Neutralitätsmenge einer Resolution ϱ_j, also $b_i \in N(\varrho_j)$ $(1 \leq i, j \leq m)$, so bleibt diese nach Anwendung der Resolution ϱ_j erhalten, sofern sie zuvor erfüllt ist. Zudem bewirkt die Resolution ϱ_j die ggf. erforderliche Wiederherstellung relativer Konsistenz hinsichtlich der Konsistenzbedingung b_j, so dass schließlich beide Konsistenzbedingungen erfüllt sind. Die folgende Definition verallgemeinert und präzisiert diesen Begriff der *Verträglichkeit* für eine endliche Anzahl an Resolutionen:

Definition 7.4 (*Verträglichkeit*)

Gegeben sei eine Menge $\mathcal{M}$ an Produktmodellen, ein ausgewähltes Produktmodell $P \in \mathcal{M}$ und eine Menge $B = \{b_1, \ldots, b_m\}$, $m \in \mathbb{N}$, an Konsistenzbedingungen mit zugehörigen Resolutionen $\varrho_i : \mathcal{M} \to \mathcal{M}$, $1 \leq i \leq m$. Dann heißen die betrachteten Resolutionen $\varrho_1, \ldots, \varrho_m$ verträglich, wenn eine Permutation $\pi : \{1, \ldots, m\} \to \{1, \ldots, m\}$ existiert, so dass die Verträglichkeitsbedingungen

$$\begin{aligned}
b_{\pi(1)} &\in N(\varrho_{\pi(2)}) \cap \ldots \cap N(\varrho_{\pi(m)}),\\
b_{\pi(2)} &\in N(\varrho_{\pi(3)}) \cap \ldots \cap N(\varrho_{\pi(m)}),\\
&\vdots\\
b_{\pi(m-2)} &\in N(\varrho_{\pi(m-1)}) \cap N(\varrho_{\pi(m)}),\\
b_{\pi(m-1)} &\in N(\varrho_{\pi(m)})
\end{aligned}$$

erfüllt sind.

Die so beschriebenen Verträglichkeitsbedingungen geben eine Reihenfolge der Resolutionen vor, so dass gesicherte Konsistenzbedingungen in den Neutralitätsmengen aller später auftretenden Resolutionen liegen. Folgerichtig führt die Anwendung der Resolutionen in dieser Reihenfolge zu einem global konsistenten Produktmodell.

Satz 7.2 (*Resolutionssatz*)

Gegeben sei eine Menge $\mathcal{M}$ an Produktmodellen, ein ausgewähltes Produktmodell $P \in \mathcal{M}$ und eine Menge $B = \{b_1, \ldots, b_m\}$, $m \in \mathbb{N}$, an Konsistenzbedingungen mit zugehörigen Resolutionen $\varrho_i : \mathcal{M} \to \mathcal{M}$, $1 \leq i \leq m$. Sind die Resolutionen $\varrho_1, \ldots, \varrho_m$ verträglich, so existiert eine Permutation $\pi : \{1, \ldots, m\} \to \{1, \ldots, m\}$ mit

$$P[\varrho_{\pi(1)}, \ldots, \varrho_{\pi(m)}] \models B.$$

Beweis 7.2 (*Resolutionssatz*)

Da die Resolutionen $\varrho_1, \ldots, \varrho_m$ als verträglich vorausgesetzt sind, existiert eine Permutation $\pi : \{1, \ldots, m\} \to \{1, \ldots, m\}$, so dass die in Definition 7.4 angegebenen Verträglichkeitsbedingungen erfüllt sind. Im Folgenden wird induktiv gezeigt, dass mit dieser Permutation π die Eigenschaft

$$P[\varrho_{\pi(1)}, \ldots, \varrho_{\pi(k)}] \models \{b_{\pi(1)}, \ldots, b_{\pi(k)}\}$$

für jedes $k = 1, \ldots, m$ erfüllt ist:

(1) $k = 1$:

Im Fall $k = 1$ gilt offenbar $P[\varrho_{\pi(1)}] \models \{b_{\pi(1)}\}$, da $\varrho_{\pi(1)}$ die der Konsistenzbedingung $b_{\pi(1)}$ zugeordnete Resolution ist.

(2) $k \to k + 1$ für $k < m$:

Nach Induktionsvoraussetzung gilt

$$P' := P[\varrho_{\pi(1)}, \ldots, \varrho_{\pi(k)}] \models \{b_{\pi(1)}, \ldots, b_{\pi(k)}\}$$

und somit insbesondere $P' \models \{b_{\pi(i)}\}$ für jedes $i \in \{1, \ldots, k\}$. Aus den Verträglichkeitsbedingungen folgt zudem $b_{\pi(i)} \in N(\varrho_{\pi(k+1)})$ und daher $P'[\varrho_{\pi(k+1)}] \models \{b_{\pi(i)}\}$ für jedes $i \in \{1, \ldots, k\}$, also

$$P'[\varrho_{\pi(k+1)}] \models \{b_{\pi(1)}, \ldots, b_{\pi(k)}\}.$$

Andererseits gilt $P'[\varrho_{\pi(k+1)}] \models \{b_{\pi(k+1)}\}$, da $\varrho_{\pi(k+1)}$ die der Konsistenzbedingung $b_{\pi(k+1)}$ zugeordnete Resolution ist. Insgesamt ergibt sich daher

$$P[\varrho_{\pi(1)}, \ldots, \varrho_{\pi(k+1)}] \models \{b_{\pi(1)}, \ldots, b_{\pi(k+1)}\}$$

da $P'[\varrho_{\pi(k+1)}] = P[\varrho_{\pi(1)}, \ldots, \varrho_{\pi(k)}][\varrho_{\pi(k+1)}] = P[\varrho_{\pi(1)}, \ldots, \varrho_{\pi(k+1)}]$ gilt.

Für den Fall $k = m$ folgt somit die Behauptung

$$P[\varrho_{\pi(1)}, \ldots, \varrho_{\pi(m)}] \models B.$$

Die zusätzliche Voraussetzung der Verträglichkeit betrachteter Resolutionen gemäß Definition 7.4 schränkt das im vorhergehenden Abschnitt 7.1 beschriebene Resolutionsproblem offenbar hinreichend ein, um stets eine Lösung zu gewährleisten. Wie aus Beweis 7.2 hervorgeht, geben die Verträglichkeitsbedingungen sogar unmittelbar die Reihenfolge einer konfliktfreien Durchführung vorhandener Resolutionen vor. Allerdings sei darauf hingewiesen,

dass es keineswegs immer möglich ist, Resolutionen in einer verträglichen Reihenfolge anzuordnen. In diesem Fall können geeignete Partitionierungen jedoch zur Lösung von Teilproblemen genutzt werden.

Beispiel 7.2 (*Resolutionssatz*)

Im Folgenden betrachten wir ein Produktmodell P, das über drei ganzzahlige Attribute x, y, z sowie die Konsistenzbedingungen $B = \{b_1, b_2, b_3\}$ mit

$$b_1 : x = 0, \qquad b_2 : y = 0, \qquad b_3 : z = 0$$

verfüge. Auf dieser Grundlage seien die drei Resolutionen

$$\begin{aligned} \varrho_1 &: x := 0,\ y := y + 1,\ z := y, \\ \varrho_2 &: x := x + 1,\ y := 0,\ z := x, \\ \varrho_3 &: x := x - 1,\ y := x,\ z := 0 \end{aligned}$$

gegeben. Aufgrund der Wechselwirkungen zwischen den Resolutionen ϱ_1, ϱ_2 und ϱ_3 gilt offenbar

$$N(\varrho_1) = \{b_1\}, \qquad N(\varrho_2) = \{b_2\}, \qquad N(\varrho_3) = \{b_3\},$$

so dass $\varrho_1, \varrho_2, \varrho_3$ nicht verträglich sind. Dennoch führt ihre Durchführung in dieser Reihenfolge zu einem global konsistenten Produktmodell $P[\varrho_1, \varrho_2, \varrho_3]$, das jede der drei Konsistenzbedingungen b_1, b_2, b_3 erfüllt.

Wie das betrachtete Beispiel zeigt, lassen sich die Wechselwirkungen zwischen Resolutionen nur allgemein und ohne Berücksichtigung des konkreten Zustands eines Produktmodells aus ihren Neutralitätsmengen ableiten. Die in Definition 7.4 vorgestellten Verträglichkeitsbedingungen stellen somit nur eine hinreichende, jedoch nicht eine notwendige Voraussetzung für die Lösbarkeit des Resolutionsproblems dar. Durch eine zusätzliche Analyse des Verhaltens vorhandener Resolutionen vor dem Hintergrund eines gegebenen Produktmodells und ihrer potenziellen Konflikte ließen sich diese weiter abschwächen.

7.3 Der Resolutionsalgorithmus

Der Resolutionssatz benennt ein hinreichendes Kriterium für die Entscheidbarkeit und Lösbarkeit des in Abschnitt 7.1 vorgestellten Resolutionsproblems durch Hinzunahme weiter gehender Informationen über die Wechselwirkungen betrachteter Resolutionen. Setzt man ihre Verträglichkeit und ihre

Neutralitätsmengen als gegeben voraus, so lässt sich das Resolutionsproblem auf die Ermittlung einer Reihenfolge der Resolutionen zurückführen, welche die in Definition 7.4 vorgestellten Verträglichkeitsbedingungen erfüllt. Die sukzessive Anwendung vorhandener Resolutionen in dieser Reihenfolge führt insgesamt zu einem global konsistenten Produktmodell.

Auf dieser Grundlage lassen sich folglich Algorithmen entwickeln, welche unter der Annahme der Verträglichkeit und ausgehend von gegebenen Neutralitätsmengen eine geeignete Reihenfolge der Resolutionen berechnen. Als Grundidee solcher Algorithmen kann die sukzessive Konstruktion einer Reihenfolge durch Analyse der Neutralitätsmengen dienen. Dabei ist zunächst zu erwarten, dass dies den Einsatz des häufig genutzten Prinzips der Rückverfolgung (engl.: *backtracking*, s. z.B. [Lex01]) erfordert, um nicht weiter verfolgbare Teillösungen vermeiden zu können. Das folgende Ergebnis zeigt jedoch, dass solche Fälle nicht auftreten:

Satz 7.3 (*Fortsetzungssatz*)

Gegeben sei eine Menge $\mathcal{M}$ an Produktmodellen, ein ausgewähltes Produktmodell $P \in \mathcal{M}$ und eine Menge $B = \{b_1, \ldots, b_m\}$, $m \in \mathbb{N}$, an Konsistenzbedingungen mit zugehörigen Resolutionen $\varrho_i : \mathcal{M} \to \mathcal{M}$, $1 \leq i \leq m$. Sind die Resolutionen $\varrho_1, \ldots, \varrho_m$ verträglich, so gibt es zu jeder Folge paarweise verschiedener Indizes $i_1, \ldots, i_k \in \{1, \ldots, m\}$, $k < m$, mit

$$\begin{aligned} b_{i_1} &\in \bigcap_{1 \leq l \leq m,\ l \neq i_1} N(\varrho_l), \\ b_{i_2} &\in \bigcap_{1 \leq l \leq m,\ l \neq i_1, i_2} N(\varrho_l), \\ &\vdots \\ b_{i_k} &\in \bigcap_{1 \leq l \leq m,\ l \neq i_1, \ldots, i_k} N(\varrho_l) \end{aligned}$$

stets einen Index $i \in \{1, \ldots, m\} \setminus \{i_1, \ldots, i_k\}$ mit

$$b_i \in \bigcap_{1 \leq l \leq m,\ l \neq i_1, \ldots, i_k, i} N(\varrho_l)^3.$$

Satz 7.3 impliziert demnach die überraschende Aussage, dass sich bei der Suche nach einer verträglichen Resolutionsreihenfolge jede gefundene Teillösung stets zu einer vollständigen Lösung fortsetzen lässt.

[3] Im Fall $\{i_1, \ldots, i_k, i\} = \{1, \ldots, m\}$ sei $\bigcap_{1 \leq l \leq m,\ l \neq i_1, \ldots, i_k, i} N(\varrho_l) = B$.

Beweis 7.3 (*Fortsetzungssatz*)

Ohne Beschränkung der Allgemeinheit kann davon ausgegangen werden, dass $i_1 = 1, \ldots, i_k = k$, also

$$\begin{aligned} b_1 &\in N(\varrho_2) \cap \ldots \cap N(\varrho_m), \\ &\vdots \\ b_k &\in N(\varrho_{k+1}) \cap \ldots \cap N(\varrho_m) \end{aligned}$$

gilt. Durch eine Permutation der gewählten Indizes lässt sich dies stets erreichen. Wir nehmen nun an, dass die Aussage des Fortsetzungssatzes nicht zutrifft und leiten daraus einen Widerspruch ab:

Annahme: Für jedes $i \in \{k+1, \ldots, m\}$ gilt $b_i \notin \bigcap_{k+1 \leq l \leq m,\ l \neq i} N(\varrho_l)$.

Aus der Annahme folgt unmittelbar, dass zu jedem $i \in \{k+1, \ldots, m\}$ mindestens ein Index $\mu(i) \in \{k+1, \ldots, m\} \setminus \{i\}$ mit der Eigenschaft $b_i \notin N(\varrho_{\mu(i)})$ existiert.
Da $\varrho_1, \ldots, \varrho_m$ andererseits als verträgliche Resolutionen vorausgesetzt sind, gibt es eine Permutation $\pi : \{1, \ldots, m\} \to \{1, \ldots, m\}$, so dass die Verträglichkeitsbedingungen

$$\begin{aligned} b_{\pi(1)} &\in N(\varrho_{\pi(2)}) \cap \ldots \cap N(\varrho_{\pi(m)}), \\ b_{\pi(2)} &\in N(\varrho_{\pi(3)}) \cap \ldots \cap N(\varrho_{\pi(m)}), \\ &\vdots \\ b_{\pi(m-2)} &\in N(\varrho_{\pi(m-1)}) \cap N(\varrho_{\pi(m)}), \\ b_{\pi(m-1)} &\in N(\varrho_{\pi(m)}) \end{aligned}$$

erfüllt sind. Darauf aufbauend, lässt sich für jedes $l \in \{1, \ldots, k\}$ induktiv zeigen, dass $\pi(1), \ldots, \pi(l) \in \{1, \ldots, k\}$ gilt:

(1) $l = 1$:

Die Annahme $\pi(1) \in \{k+1, \ldots, m\}$ impliziert die Existenz eines Index $\mu(\pi(1)) \in \{k+1, \ldots, m\} \setminus \{\pi(1)\}$ mit $b_{\pi(1)} \notin N(\varrho_{\mu(\pi(1))})$. Dies steht jedoch im Widerspruch zu der angegebenen Verträglichkeitsbedingung $b_{\pi(1)} \in N(\varrho_{\pi(2)}) \cap \ldots \cap N(\varrho_{\pi(m)})$, so dass $\pi(1) \in \{1, \ldots, k\}$ gelten muss.

(2) $l \to l+1$ für $l < k$:

Aus der Annahme $\pi(l+1) \in \{k+1, \ldots, m\}$ folgt wiederum die Existenz eines Index $\mu(\pi(l+1)) \in \{k+1, \ldots, m\} \setminus \{\pi(l+1)\}$ mit der Eigenschaft $b_{\pi(l+1)} \notin N(\varrho_{\mu(\pi(l+1))})$. Wegen $b_{\pi(l+1)} \in N(\varrho_{\pi(l+2)}) \cap \ldots \cap N(\varrho_{\pi(m)})$ folgt daraus

$$\mu(\pi(l+1)) \in \{k+1, \ldots, m\} \setminus \{\pi(l+1), \ldots, \pi(m)\}.$$

Da π eine Permutation ist, gilt $\{\pi(1), \ldots, \pi(m)\} = \{1, \ldots, m\}$ aufgrund der Bijektivität, sowie $\pi(1), \ldots, \pi(l) \in \{1, \ldots, k\}$ nach Induktionsvoraussetzung. Somit folgt $\{k+1, \ldots, m\} \subseteq \{\pi(l+1), \ldots, \pi(m)\}$, also

$$\{k+1, \ldots, m\} \setminus \{\pi(l+1), \ldots, \pi(m)\} = \emptyset.$$

Dies stellt jedoch einen Widerspruch zur Existenz des Index $\mu(\pi(l+1))$ dar, so dass im Gegensatz zur Annahme $\pi(l+1) \in \{1, \ldots, k\}$ gilt.

Insgesamt folgt $\pi(1), \ldots, \pi(k) \in \{1, \ldots, k\}$ und somit, da π bijektiv ist, $\{\pi(1), \ldots, \pi(k)\} = \{1, \ldots, k\}$. Mit $i := \pi(k+1) \in \{1, \ldots, m\} \setminus \{1, \ldots, k\}$ gilt offenbar

$$b_i \in N(\varrho_{\pi(k+2)}) \cap \ldots \cap N(\varrho_{\pi(m)}) = \bigcap_{k+1 \leq l \leq m,\ l \neq i} N(\varrho_l),$$

im Widerspruch zur getroffenen Annahme.

Der in Abschnitt 7.2 vorgestellte Resolutionssatz besagt, dass ausgehend von verträglichen Resolutionen stets eine Reihenfolge ihrer Anwendung existiert, die zu einem global konsistenten Produktmodell führt. Ergänzend zeigt Satz 7.3, dass eine solche Reihenfolge sukzessive konstruiert werden kann, da jede gefundene Teillösung stets zu einer vollständigen Lösung fortsetzbar ist. Die Kombination beider Aussagen bietet die Grundlage für die Entwicklung eines entsprechenden Algorithmus, der ausgehend von einer Menge an Konsistenzbedingungen und Resolutionen aus zugehörigen Neutralitätsmengen eine geeignete Reihenfolge der Resolutionen ermittelt.

Algorithmus 7.1 (*Resolutionsalgorithmus*)

Gegeben sei eine Menge $\mathcal{M}$ an Produktmodellen, ein ausgewähltes Produktmodell $P \in \mathcal{M}$ und eine Menge $B = \{b_1, \ldots, b_m\}$, $m \in \mathbb{N}$, an Konsistenzbedingungen mit zugehörigen Resolutionen $\varrho_i : \mathcal{M} \to \mathcal{M}$, welche als verträglich vorausgesetzt werden und über die Neutralitätsmengen $N(\varrho_i) \subseteq B$ verfügen ($1 \leq i \leq m$). Dann berechnet folgender Algorithmus eine Reihenfolge $v = (i_1, \ldots, i_m)$ paarweise verschiedener Indizes $i_l \in \{1, \ldots, m\}$, $1 \leq l \leq m$, mit der Eigenschaft

$$P[\varrho_{i_1}, \ldots, \varrho_{i_m}] \models B.$$

Die Anwendung der betrachteten Resolutionen in der durch v vorgegebenen Reihenfolge ergibt also ein global konsistentes Produktmodell, das alle festgelegten Konsistenzbedingungen erfüllt.

```
// Initialisierung
I := {1, ..., m};
v := emptySequence();

// Sukzessive Berechnung von v
while I ≠ ∅ do
   I' := I;
   repeat
      // Nicht-deterministische Auswahl eines Index i ∈ I'
      choose i ∈ I';
      I' := I' \ {i};
   until b_i ∈ ⋂_{l∈I\{i}} N(ϱ_l);
   v := append(v, i);
   I := I \ {i};
end;
```

Der so gegebene Algorithmus zur Berechnung einer Resolutionsreihenfolge benutzt die Hilfsoperationen `emptySequence()` und `append()`, um initial eine leere Sequenz zu erzeugen bzw. ein Element am Ende einer vorhandenen Sequenz einzuordnen. Seine Funktionsweise wird durch das folgende Beispiel veranschaulicht:

Beispiel 7.3 (*Resolutionsalgorithmus*)

Wir betrachten ein gegebenes Produktmodell P mit fünf Konsistenzbedingungen $b_1, \ldots, b_5$ und entsprechenden Resolutionen $\varrho_1, \ldots, \varrho_5$, welchen die Neutralitätsmengen

$$N(\varrho_1) = \{b_1, b_3, b_5\}, \quad N(\varrho_2) = \{b_1, b_3, b_4, b_5\}, \quad N(\varrho_3) = \{b_3\},$$

$$N(\varrho_4) = \{b_1, b_3, b_5\}, \quad N(\varrho_5) = \{b_3, b_5\}$$

zugeordnet seien. Die Durchführung des Resolutionsalgorithmus initialisiert die Indexmenge I mit $I := \{1, \ldots, 5\}$. Sukzessive wird aus I nicht-deterministisch ein Index i ausgewählt mit der Eigenschaft $b_i \in \bigcap_{l \in I \setminus \{i\}} N(\varrho_l)$. Wegen

$$\bigcap_{l \in \{2,3,4,5\}} N(\varrho_l) = \bigcap_{l \in \{1,3,4,5\}} N(\varrho_l) = \bigcap_{l \in \{1,2,3,5\}} N(\varrho_l) = \bigcap_{l \in \{1,2,3,4\}} N(\varrho_l) = \{b_3\}$$

und

$$\bigcap_{l \in \{1,2,4,5\}} N(\varrho_l) = \{b_3, b_5\}$$

stellt im ersten Schritt nur der Index $i = 3$ eine geeignete Wahl dar. Die Indexmenge I verkürzt sich daher zu $I := \{1, 2, 4, 5\}$. Auf analoge Weise ist im nächsten Schritt des Algorithmus daraus der Index $i = 5$ zu wählen, so dass sich die Indexmenge I zu $I := \{1, 2, 4\}$ und die Teillösung v zu $v := (3, 5)$ ergeben. Die darauf folgende Wahl von $i = 1$, $i = 4$ und $i = 2$ führt schließlich zu der vollständigen Lösung $v = (3, 5, 1, 4, 2)$, welche die Verträglichkeitsbedingungen

$$\begin{aligned} b_3 &\in N(\varrho_1) \cap N(\varrho_2) \cap N(\varrho_4) \cap N(\varrho_5), \\ b_5 &\in N(\varrho_1) \cap N(\varrho_2) \cap N(\varrho_4), \\ b_1 &\in N(\varrho_2) \cap N(\varrho_4), \\ b_4 &\in N(\varrho_2) \end{aligned}$$

erfüllt. Folgerichtig ist nach Satz 7.2

$$P[\varrho_3, \varrho_5, \varrho_1, \varrho_4, \varrho_2] \models \{b_1, b_2, b_3, b_4, b_5\}$$

gewährleistet, da Resolutionskonflikte durch die gegebene Reihenfolge der Resolutionen ausgeschlossen sind.

7.4 Zusammenfassung

Der vorgeschlagene Ansatz eines segmentierten Produktmodells zur Unterstützung einer integrativen Produktentwicklung begreift Produktsichten als eigenständige und zunächst unabhängige Partialmodelle, welche über explizite Konsistenzbedingungen zueinander in Beziehung gesetzt werden. Ihre individuelle Interpretation und Resolution bilden entscheidende Bestandteile eines durchgängigen Entwicklungsprozesses. Aufgrund der Vielzahl und Vielfältigkeit zu erwartender Konsistenzbedingungen eines realen Produktmodells erfordert eine praxisgerechte Lösung den Einsatz automatisierter Verfahren.

Vor diesem Hintergrund ist das Auftreten potenzieller Resolutionskonflikte in besonderer Weise zu berücksichtigen. So ist im Zuge der Transformation eines Produktmodells in einen global konsistenten Zustand mittels einer sukzessiven Anwendung von Resolutionen zu vermeiden, dass gesicherte Konsistenzbedingungen durch Modellmodifikationen später ausgeführter Resolutionen wiederum verletzt werden. Vielmehr besteht die Aufgabe, Resolutionen – sofern möglich – in einer geeigneten Reihenfolge durchzuführen, um unerwünschte Wechselwirkungen im Rahmen übergreifender Resolutionsprozesse auszuschließen.

Diese, als Resolutionsproblem bezeichnete Aufgabenstellung unterliegt jedoch der prinzipiellen Einschränkung, dass es im allgemeinen Fall kein Verfahren gibt, welches eine geeignete Reihenfolge zur konfliktfreien Durchführung von Resolutionen berechnet. Zur Entwicklung eines hinreichenden Kriteriums für die Existenz einer Lösung werden daher zusätzlich die Wechselwirkungen einzelner Resolutionen betrachtet und ein Verträglichkeitsbegriff eingeführt. Auf dieser Grundlage lässt sich im Rahmen einer weiter gehenden Untersuchung der Problemstruktur schließlich ein algorithmisches Lösungsverfahren ableiten.

Epilog

8 Zusammenfassung

Technische Produkte bilden einen bedeutenden Bestandteil des Wirtschaftslebens moderner Gesellschaften. Ihre Entwicklung ist geprägt durch umfangreiche, vielschichtige und anspruchsvolle Arbeitsabläufe, die ein koordiniertes Zusammenwirken zahlreicher Bearbeiter in einer hochgradig interdisziplinären Umgebung erfordern. Über alle Wirtschaftsbereiche hinweg führen steigende Anforderungen an Funktionsumfänge und Leistungsmerkmale gefertigter Produkte, kontinuierlich verbesserte Produktionsverfahren sowie gleich bleibend hohe Qualitätserwartungen zu einer signifikant anwachsenden Komplexität der Produktentwicklung.

Dem Einsatz moderner Informations- und Kommunikationstechnologien kommt unter diesen Gesichtspunkten eine wichtige Rolle zu. Sie bieten vielfältige Lösungen, um die Effizienz und Effektivität verschiedenster Teilabläufe einer Produktentwicklung zu erhöhen und ihre Komplexität zu beherrschen. Mithilfe von Software-Werkzeugen lassen sich insbesondere digitale Modelle von Produkten bilden, die im Vergleich zu realen Modellen erheblich schneller und kostengünstiger zu bearbeiten sind. Entsprechend existiert auch im Ingenieursbereich seit geraumer Zeit ein umfangreicher Markt zahlreicher kommerzieller Software-Lösungen.

Software-Werkzeuge sind jedoch in aller Regel auf die optimale Unterstützung fachspezifischer Einzelaktivitäten ausgerichtet. Ihre zunehmende Verwendung verursacht eine Reihe an Integrationsdefiziten in dreierlei Bereichen, deren Bewältigung rasch an Bedeutung gewinnt und für zahlreiche Unternehmen eine strategische Herausforderung darstellt:

- Auf der Ebene des Prozesses fehlt eine methodische Verknüpfung von Entwicklungsschritten mit eingesetzten Software-Werkzeugen bzw. zugehörigen Modellen.

- Erstellte Teilmodelle eines Produkts sind überwiegend isoliert und oftmals inkompatibel. Fachliche Überschneidungen, Querbezüge und Abhängigkeiten werden nicht ausreichend berücksichtigt.

- Schließlich fehlt die technische Integration genutzter Software-Werkzeuge aufgrund unterschiedlicher Technologien, Systemplattformen und Entwicklungsumgebungen.

Zu jedem der genannten Punkte existieren Lösungsansätze, die jedoch jeweils nur einen spezifischen Ausschnitt der auftretenden Integrationsdefizite überwinden. Die vorliegende Arbeit verbindet sie zu einem übergreifenden, ganzheitlichen und methodischen Integrationsansatz, der Teilmodelle eines technischen Produkts, zugehörige Schritte des zugrunde liegenden Entwicklungsprozesses und eingesetzte Software-Werkzeuge gleichermaßen berücksichtigt. Die Entwicklung dieses Ansatzes erfolgt gemäß der Gliederung der Arbeit in drei Teilen, welche aufeinander aufbauen und sukzessive ausgearbeitet werden (vgl. Abschnitt 1.2):

- Im ersten Teil wird eine Software-technische *Integrationsplattform* entworfen, die als übergreifende Entwicklungsumgebung Prozessabläufe steuert, Modelle zentral erfasst und Software-Werkzeuge in einheitlicher Weise anbindet.

- Der zweite Teil konzipiert ein *integriertes Produktmodell*, welches als wesentlicher und entscheidender Bestandteil der Integrationsplattform Produktinformationen über alle Phasen eines Entwicklungsprozesses hinweg aufnimmt.

- Im dritten Teil werden Techniken und Strategien betrachtet, um Überschneidungen, Querbezüge und Abhängigkeiten zwischen Elementen eines integrierten Produktmodells zu spezifizieren und seine *Konsistenz* zu gewährleisten.

Mit der Konzeption einer Integrationsplattform werden mehrere Zielsetzungen erreicht: Sie sieht zum einen ein *Prozessmodell* in Verbindung mit einem *Organisationsmodell* vor, um Entwicklungsabläufe zusammen mit zugrunde liegenden Organisationsstrukturen durchgängig zu erfassen, zu koordinieren und zu steuern. In gleicher Weise werden bisher isolierte Teilmodelle eines technischen Produkts als individuelle Produktsichten im Rahmen eines integrierten *Produktmodells* zusammengeführt. Darüber hinaus dient die Integrationsplattform der technischen Integration von Datenquellen und Software-Werkzeugen.

Insgesamt überwindet dieser Ansatz die wesentlichen Integrationsdefizite rechnergestützter Produktentwicklungsverfahren in ihrer heutigen Form und bildet eine solide Grundlage für eine methodische Produktentwicklung: Einzelne Entwicklungsaufgaben werden anhand des Prozessmodells verfügbaren

Bearbeitern gemäß freier Ressourcen des Organisationsmodells zugewiesen, zugehörige Teilmodelle des Produkts werden zusammen mit ihren Querbezügen und Abhängigkeiten an zentraler Stelle verwaltet und auf Software-Werkzeuge sowie Datenquellen wird mittels standardisierter Schnittstellen auf einheitliche Weise zugegriffen.

Als entscheidendes Kernstück der Integrationsplattform präsentiert die Arbeit ein integriertes Produktmodell, das Produktinformationen aus allen Phasen einer Produktentwicklung erfasst und zueinander in Beziehung setzt. Dieses begreift die sehr zahlreichen, vielfältigen und unterschiedlichen Teilmodelle eines Produkts – eine gerade für den Bereich komplexer technischer Produkte besonders charakteristische Eigenschaft – als eigenständige und optimal angepasste Produktsichten gemäß einer gemeinsamen und übergreifenden Modellierungstechnik. Darauf aufbauend werden fachliche Querbezüge in Form expliziter Konsistenzbedingungen festgelegt.

Aus den vier Basiskonzepten der *Segmentierung*, der *Strukturierung*, der *Typisierung* und der *Attributierung* wird ein Metamodell abgeleitet, das Begriffe, Abstraktionen und Gesetzmäßigkeiten vorgibt, um Produktsichten zu definieren, ihre individuellen Produktstrukturen in das Produktmodell abzubilden, Produktbestandteile zu klassifizieren und elementare Produkteigenschaften zu erfassen. Die Einführung einer graphischen Beschreibungstechnik für Produktsichten sowie die formale Fundierung der gewählten Modellabstraktionen mit Methoden der Graphentheorie und der algebraischen Spezifikation ergänzen den Entwurf des Produktmodells.

Der vorgeschlagene Ansatz eines integrierten Produktmodells als Kombination eigenständiger Produktsichten unterstützt eine flexible und differenzierte Behandlung der Modellkonsistenz. Fachliche Überschneidungen, Querbezüge und Abhängigkeiten zwischen Produktsichten, welche sich aus dem Bezug auf ein gemeinsames Produkt ergeben, werden in Form expliziter Konsistenzbedingungen als eigene Modellelemente in das Produktmodell eingebracht. Zu ihrer Spezifikation sind verschiedenartige Techniken ausdrücklich zugelassen, welche formale Ansätze der Wissensrepräsentation ebenso umfassen wie informelle Beschreibungsformen.

Auf dieser Grundlage werden zwei Strategien der methodischen Überprüfung und Sicherstellung von Konsistenzanforderungen vorgestellt, die diese einerseits als spezifische Aktivitäten eines Entwicklungsprozesses begreifen oder – allgemeiner – als Reaktion auf eintretende Ereignisse der Produktentwicklung verstehen. In beiden Fällen ist es insbesondere vorgesehen, über den Zustand des Produktmodells hinaus den vollständigen Entwicklungskontext in die Prüfung und Sicherung von Konsistenzbedingungen einzubeziehen. Die Untersuchung der Automatisierbarkeit übergreifender Konsistenzsicherungsprozesse und ihrer prinzipiellen Grenzen rundet die Arbeit ab.

Literatur

[AMD96] R. Anderl, R. Mendgen und B. Daum. *Produktdatentechnologie – Grundlagen und DV-Systeme*. Technische Hochschule Darmstadt, 1996.

[And03] R. Anderl. Produktdatentechnologie A – CAD-Systeme und CAx-Prozessketten. Vorlesungsskript, Technische Universität Darmstadt, Fachgebiet Datenverarbeitung in der Konstruktion, 2003.

[ART03a] Architektur und Implementierung der Engineering Workbench, 2003. Abschlussbericht des Projekts ARTWORK.

[ART03b] Das Produkt- und Prozessmodell der Engineering Workbench, 2003. Abschlussbericht des Projekts ARTWORK.

[ART05] ARTWORK Homepage. `http://artwork.in.tum.de/`, 2005.

[Asp05] AspectJ Homepage. `http://eclipse.org/aspectj/`, 2005.

[Aut05] AutoFocus Homepage. `http://autofocus.informatik.tu-muenchen.de/`, 2005.

[Bal94] C. Ballarin. *Algorithmische Schritte in formalen Beweisen – Entwurf und Implementierung der Anbindung eines Computeralgebrasystems an einen Theorembeweiser*. Diplomarbeit, Universität Karlsruhe, 1994.

[Bal99] H. Balzert. *Lehrbuch der Objektmodellierung*. Spektrum Akademischer Verlag, 1999.

[Bec96] M. Becker. *Praxis des Workflow-Managements*, Kapitel Workflow Management – Szenarien und Potentiale. Vieweg-Verlag, 1996.

[Ben01] K. Bender. PDM und Engineering-Informationssysteme. Vorlesungsskript, Technische Universität München, Lehrstuhl für Informationstechnik im Maschinenwesen, 2001.

[Biz05] Microsoft BizTalk Server Homepage. `http://www.microsoft.com/biztalk/`, 2005.

[BK02] J. Becker und D. Kahn. Der Prozess im Fokus. In J. Becker, M. Kugeler und M. Rosemann, Hrsg., *Prozessmanagement – Ein Leitfaden zur prozessorientierten Oganisationsgestaltung*. Springer-Verlag, 2002.

[BKS+96] S. Brandner, M. Kelch, H. Stengele, M. Eigner, A. Suhm und G. Reinhart, Hrsg. *EDM – Engineering Data Management*. iwb Seminarberichte, Herbert Utz Verlag, 1996.

[BKZ+97] S. Brandner, M. Kelch, R. Zahn, J. Parzinger, D. Bald und G. Reinhart, Hrsg. *Engineering Data Management (EDM) – Erfahrungsberichte und Trends*. iwb Seminarberichte, Herbert Utz Verlag, 1997.

[BM02] P. Braun und F. Marschall. Transforming Object Oriented Models with BOTL. In P. Bottoni und M. Minas, Hrsg., *International Workshop on Graph Transformation and Visual Modeling Techniques*. Elsevier Science, 2002.

[BM03] P. Braun und F. Marschall. BOTL – The Bidirectional Object Oriented Transformation Language. Forschungsbericht TUM-I0307, Technische Universität München, 2003.

[Bom99] B. Bomsdorf. Ein kohärenter, integrativer Modellrahmen zur aufgabenbasierten Entwicklung interaktiver Systeme. Dissertation, Universität Paderborn, 1999.

[Bra03a] D. Braess. *Finite Elemente*. Springer-Verlag, 3. Auflage, 2003.

[Bra03b] P. Braun. *Metamodellbasierte Kopplung von Werkzeugen in der Softwareentwicklung*. Dissertation, Technische Universität München, 2003.

[BRJ99] G. Booch, J. Rumbaugh und I. Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley, 1999.

[Bro92] M. Broy. *Informatik – Eine grundlegende Einführung – Teil I*. Springer-Verlag, 1992.

[Bro95] M. Broy. *Informatik – Eine grundlegende Einführung – Teil IV.* Springer-Verlag, 1995.

[BS98] U. M. Borghoff und J. H. Schlichter. *Rechnergestützte Gruppenarbeit – Eine Einführung in Verteilte Anwendungen.* Springer-Verlag, 2. Auflage, 1998.

[CAT04] CATIA Homepage. `http://plm.3ds.com/10+M52c1d207f79.0.html`, 2004.

[CBC99] S. J. Culley, O. P. Boston und A. Christopher. Suppliers in New Product Development – Their Information and Integration. *Journal of Engineering Design*, 1999.

[Cha85] G. Chartrand. *Introductory Graph Theory.* Dover Publications, 1985.

[Cha92] H. J. Charwat. *Lexikon der Mensch-Maschine-Kommunikation.* Oldenbourg-Verlag, 1992.

[Che76] P. P.-S. Chen. The Entity-Relationship Model – Toward a Unified View of Data. In *ACM Transactions on Database Systems*, 1976.

[CHOT99] S. Clarke, W. Harrison, H. Ossher und P. Tarr. Subject-Oriented Design – Towards Improved Alignment of Requirements, Design and Code. In *Proceedings of the 14th ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications*, 1999.

[CJB99] B. Chandrasekaran, J. R. Josephson und V. R. Benjamins. What are Ontologies, and why do we need them? *IEEE Intelligent Systems*, 1999.

[CM85] E. Charniak und D. McDermott. *Introduction to Artificial Intelligence.* Addison-Wesley, 1985.

[COR00] Object Management Group. *The Common Object Request Broker: Architecture and Specification*, 2000.

[CQ69] A. M. Collins und M. R. Quillian. Retrieval Time from Semantic Memory. *Journal of Verbal Learning and Verbal Behavior*, 1969.

[Cum02] F. A. Cummins. *Enterprise Integration: An Architecture for Enterprise Application and Systems Integration.* John Wiley & Sons, 2002.

[CW02] T. Clark und J. Warmer. *Object Modeling with the OCL – The Rationale behind the Object Constraint Language*. Springer-Verlag, 2002.

[Die00] R. Diestel. *Graphentheorie*. Springer-Verlag, 2. Auflage, 2000.

[DW99] W. Dröschel und M. Wiemers. *Das V-Modell 97*. Oldenbourg-Verlag, 1999.

[DWD87] W. Dangelmaier, H. J. Warnecke und Deutsches Institut für Normung (Kommission Computer Integrated Manufacturing). *Normung von Schnittstellen für die rechnerintegrierte Produktion (CIM) – Standortbestimmung und Handlungsbedarf*. Beuth-Verlag, 1987.

[Dyl02] A. Dyla. Modell einer durchgängig rechnerbasierten Produktentwicklung. Dissertation, Technische Universität München, 2002.

[EFKN94] S. Easterbrook, A. Finkelstein, J. Kramer und B. Nuseibeh. Coordinating Distributed ViewPoints – The Anatomy of a Consistency Check. Technical Report 94/7, Imperial College London, Department of Computing, 1994.

[EFT96] H.-D. Ebbinghaus, J. Flum und W. Thomas. *Einführung in die mathematische Logik*. Spektrum Akademischer Verlag, 1996.

[EM85] H. Ehrig und B. Mahr. *Fundamentals of Algebraic Specification 1: Equations and Initial Semantics*. Springer-Verlag, 1985.

[EXP94] International Organization for Standardization (ISO). *Industrial Automation Systems and Integration – Product Data Representation and Exchange – Part 11: Description Methods: The Express Language Reference Manual*, 1994. ISO Standard 10303-11.

[FCF02] J. Farley, W. Crawford und D. Flanagan. *Java Enterprise in a Nutshell*. O'Reilly, 2. Auflage, 2002.

[FKG90] A. Finkelstein, J. Kramer und M. Goedicke. ViewPoint Oriented Software Development. In *Proceedings of the 3rd International Workshop on Software Engineering and its Applications*, 1990.

[FKN+92] A. Finkelstein, J. Kramer, B. Nuseibeh, L. Finkelstein und M. Goedicke. ViewPoints – A Framework for Integrating Multiple Perspectives in System Development. In *International Journal of Software Engineering and Knowledge Engineering*, 1992.

[Fow03] M. Fowler. *UML Distilled - A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley, 3. Auflage, 2003.

[Gal92] T. Gal, Hrsg. *Grundlagen des Operations Research 2 – Graphen und Netzwerke, Netzplantechnik, Transportprobleme, Ganzzahlige Optimierung*. Springer-Verlag, 3. Auflage, 1992.

[GAP93] H. Grabowski, R. Anderl und A. Polly. *Integriertes Produktmodell*. Beuth-Verlag, 1993.

[GG84] J. Grimm und W. Grimm. *Deutsches Wörterbuch*. Deutscher Taschenbuch Verlag, 1984.

[GHJV95] E. Gamma, R. Helm, R. Johnson und J. Vlissides. *Design Patterns – Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[Gün01] A. Günzler. Das ARTWORK Constraint-Konzept. Forschungsbericht, Technische Universität München, 2001.

[Güs89] H. W. Güsgen. *CONSAT: A System for Constraint Satisfaction*. Morgan Kaufmann Publishers, 1989.

[Göt97] K. Götzer. *Workflow – Unternehmenserfolg durch effizientere Arbeitsabläufe*. Computerwoche Verlag GmbH, 1997.

[GV03a] H. Giese und A. Vilbig. Separation of Non-Orthogonal Concerns in Software Architecture and Design. Bericht tr-ri-03-238, Universität-Gesamthochschule Paderborn, 2003.

[GV03b] A. Günzler und A. Vilbig. Integrating Product and Process in Model-based Engineering. In M. H. Hamza, Hrsg., *Proceedings of the 7th IASTED International Conference on Software Engineering and Applications*. ACTA Press, 2003.

[Har93] R. L. Harmon. *Das Management der Neuen Fabrik – Lean Production in der Praxis*. Campus-Verlag, 1993.

[Har99] F. Harary. *Graph Theory*. Perseus Publishing, 1999.

[HJ01] S. Horn und S. Jablonski. *Von der Anwenderanalyse zu ersten Systemkonzepten für Workflow-Management-Lösungen*. Gruner Druck GmbH, 2001.

[HO93] W. Harrison und H. Ossher. Subject-Oriented Programming – A Critique of Pure Objects. In *Proceedings of the 8th ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications*, 1993.

[Hol95] D. Hollingsworth. The Workflow Reference Model. The Workflow Management Coalition Specification 1.1, The Workflow Management Coalition, 1995. Dokument TC00-1003.

[Hom96] K. Homann. *Symbolisches Lösen mathematischer Probleme durch Kooperation algorithmischer und logischer Systeme.* Dissertation, Universität Karlsruhe, 1996.

[HS00] A. Heuer und G. Saake. *Datenbanken – Konzepte und Sprachen.* mitp-Verlag, 2. Auflage, 2000.

[HY99] C. Homes und B. Yazdani. Internal Drivers for Concurrent Engineering – Industrial Case Study. In *5th International Conference on Concurrent Enterprising*, 1999.

[Hyp05] alphaWorks HyperJ Homepage. `http://www.alphaworks.ibm.com/tech/hyperj/`, 2005.

[IBM05] IBM Product Lifecycle Management Homepage. `http://www.ibm.com/software/applications/plm/`, 2005.

[ISO94] International Organization for Standardization (ISO). *Information Technology – Open Systems Interconnection – Basic Reference Model: The Basic Model*, 1994. ISO Standard 7498-1.

[ISO01] International Organization for Standardization (ISO). *Software Engineering – Product Quality – Part 1: Quality Model*, 2001. ISO Standard 9126-1.

[J2E05] Java 2 Platform – Enterprise Edition (J2EE) Homepage. `http://java.sun.com/j2ee/`, 2005.

[Jec00] M. Jeckle. Konzepte der Metamodellierung – Zum Begriff Metamodell. `http://www.jeckle.de/files/swt_2000.pdf`, 2000.

[Kai02] M. Kaib. *Enterprise Application Integration – Grundlagen, Integrationsprodukte, Anwendungsbeispiele.* Deutscher Universitäts-Verlag, 2002.

[Kei99] F. Keienburg. Modellierung und Organisation technischer Daten am Beispiel einer Turbinenscheibe. Diplomarbeit, Technische Universität München, 1999.

[Kel02] W. Keller. *Enterprise Application Integration – Erfahrungen aus der Praxis*. dpunkt-Verlag, 2002.

[KG02] M. Kofler und H.-G. Gräbe. *Mathematica – Einführung, Anwendung, Referenz*. Addison-Wesley, 2002.

[Küh02] C. Kühberger. Von der Prozess-Analyse zur Workflow-Unterstützung des Produktentwicklungsprozesses im Maschinenbau. Diplomarbeit, Technische Universität München, 2002.

[Kla04] H. Klaeren. *Konzepte höherer Programmiersprachen*. Universität Tübingen, 2004.

[KLM+97] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier und J. Irwin. Aspect-Oriented Programming. In *Proceedings of the European Conference on Object-Oriented Programming*, 1997.

[KP88] G. E. Krasner und S. T. Pope. A Cookbook for using the Model-View Controller User Interface Paradigm in Smalltalk-80. In *Journal of Object-Oriented Programming*, 1988.

[Kru99] P. Kruchten. *Der Rational Unified Process – Eine Einführung*. Addison-Wesley, 2. Auflage, 1999.

[Ler00] B. Lercher. Konzeption einer Integration von EXPRESS- und UML-Modellen. Diplomarbeit, Technische Universität München, 2000.

[Lex01] Meyers Lexikonredaktion, Hrsg. *Duden Informatik*. Dudenverlag, 3. Auflage, 2001.

[Lin99] D. S. Linthicum. *Enterprise Application Integration*. Addison-Wesley, 1999.

[Lus90] M. Lusti. *Wissensbasierte Systeme – Algorithmen, Datenstrukturen und Werkzeuge*. BI-Wissenschaftsverlag, 1990.

[MC99] L. Mandel und M. V. Cengarle. On the Expressive Power of the Object Constraint Language OCL. *World Congress on Formal Methods*, 1999.

[MF96] D. McNeill und P. Freiberger. *Fuzzy Logic*. Droemer Knaur Verlagsanstalt, 1996.

[MH02] R. Monson-Haefel. *Enterprise JavaBeans*. O'Reilly, 3. Auflage, 2002.

[MOF02] Object Management Group. *Meta Object Facility (MOF) Specification, Version 1.4*, 2002. OMG document formal/02-04-03.

[NCA99] T. Nelson, D. Cowan und P. Alencar. Towards a Formal Model of Object-Oriented Hyperslices. In *First Workshop on Multi-Dimensional Separation of Concerns in Object-Oriented Systems*, 1999.

[NCEF02] C. Nentwich, L. Capra, W. Emmerich und A. Finkelstein. xlinkit: A Consistency Checking and Smart Link Generation Service. In *ACM Transactions on Internet Technology*, 2002.

[NEFE03] C. Nentwich, W. Emmerich, A. Finkelstein und E. Ellmer. Flexible Consistency Checking. In *ACM Transactions on Software Engineering and Methodology*, 2003.

[Net05] SAP NetWeaver Homepage. `http://www.sap.com/solutions/netweaver/`, 2005.

[NX04] UGS NX Homepage. `http://www.ugs.com/products/nx/`, 2004.

[Oes04] B. Oestereich. *Objektorientierte Softwareentwicklung – Analyse und Design mit der UML 2.0*. Oldenbourg Verlag, 6. Auflage, 2004.

[OH98] R. Orfali und D. Harkey. *Client/Server Programming with Java and CORBA*. John Wiley & Sons, 2. Auflage, 1998.

[OMG04] Object Management Group Homepage. `http://www.omg.org`, 2004.

[OT00] H. Ossher und P. Tarr. Multi-Dimensional Separation of Concerns and the Hyperspace Approach. In *Proceedings of the Symposium on Software Architectures and Component Technology*, 2000.

[OWS+03] B. Oestereich, C. Weiss, C. Schröder, T. Weilkiens und A. Lenhard. *Objektorientierte Geschäftsprozessmodellierung mit der UML*. dpunkt-Verlag, 2003.

[Poh03] S. Pohlmann. Konzeption von Organisationsmodellen zur Workflow-Unterstützung von Produktentwicklungsprozessen im Maschinenbau. Diplomarbeit, Technische Universität München, 2003.

[PR91] A. Picot und R. Reichwald. *Industriebetriebslehre – Entscheidungen im Industriebetrieb*. Gabler-Verlag, 1991.

[Pup88] F. Puppe. *Einführung in Expertensysteme*. Springer-Verlag, 1988.

[RJB99] J. Rumbaugh, I. Jacobson und G. Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1999.

[Roz97] G. Rozenberg, Hrsg. *Handbook of Graph Grammars and Computing by Graph Transformations – Volume 1: Foundations*. World Scientific, 1997.

[San03] M. Sandberg. Knowledge Based Engineering in Product Development. Technischer Bericht, Lulea University of Technology, 2003.

[Sch94] A. Schürr. Specification of Graph Translators with Triple Graph Grammars. In G. Tinhofer, Hrsg., *Proceedings of the 20th International Workshop on Graph-Theoretic Concepts in Computer Science*, 1994.

[Sch95] U. Schöning. *Logik für Informatiker*. Spektrum Akademischer Verlag, 4. Auflage, 1995.

[Sch97] U. Schöning. *Theoretische Informatik – kurzgefaßt*. Spektrum Akademischer Verlag, 3. Auflage, 1997.

[Sch98] G. Schreyögg. *Organisation*. Gabler-Verlag, 2. Auflage, 1998.

[Sch99] J. Schöttner. *Produktdatenmanagement in der Fertigungsindustrie*. Fachbuchverlag Leipzig, 1999.

[Sch03] A. B. Schmidt. Eine formale Modellierung von Verbindungsstrukturen in Kommunikationssystemen. Dissertation, Technische Universität München, 2003.

[Sei02] H. Seidlmeier. *Prozessmodellierung mit ARIS – Eine beispielorientierte Einführung für Studium und Praxis*. Vieweg-Verlag, 2002.

[SI03] A. Saaksvuori und A. Immonen. *Product Lifecycle Management*. Springer-Verlag, 2003.

[SOA03] SOAP Version 1.2 Part 0: Primer. `http://www.w3.org/TR/soap12-part0/`, 2003.

[Sow83] J. F. Sowa. *Conceptual Structures – Information Processing in Mind and Machine*. Addison-Wesley, 1983.

[SS80] G. J. Sussmann und G. L. Steele. Constraints – A Language for Expressing Almost-Hierarchical Descriptions. *Artificial Intelligence Journal*, 1980.

[SS97] C. Schulz und P. Schaeffer. *Informationstechnik für Manager*. Carl Hanser Verlag, 1997.

[SSU01] G. Schwabe, N. Streitz und R. Unland. *CSCW-Kompendium – Lehr- und Handbuch zum computerunterstützten kooperativen Arbeiten*. Springer-Verlag, 2001.

[Stü97] A. Stübinger. *Eine algebraische Beschreibung planarer Graphen und ihre Anwendung auf Petri-Netze*. Shaker-Verlag, 1997.

[Sta83] H. Stachowiak, Hrsg. *Modelle – Konstruktion der Wirklichkeit*. Wilhelm Fink-Verlag, 1983.

[STE94] International Organization for Standardization (ISO). *Industrial Automation Systems and Integration – Product Data Representation and Exchange – Part 1: Overview and Fundamental Principles*, 1994. ISO Standard 10303-1.

[Ste01] A. Steger. *Diskrete Strukturen – Band 1: Kombinatorik, Graphentheorie, Algebra*. Springer-Verlag, 2001.

[Str98] S. Strahringer. Ein sprachbasierter Metamodellbegriff und seine Verallgemeinerung durch das Konzept des Metaisierungsprinzips. In K. Pohl, A. Schürr und G. Vossen, Hrsg., *CEUR Workshop Proceedings zur Modellierung*, 1998.

[Tea05] UGS Teamcenter Homepage. `http://www.ugs.com/products/teamcenter/`, 2005.

[Tel05] Telelogic TAU UML Suite Homepage. `http://www.telelogic.com/products/tau/uml/`, 2005.

[Tog05] Borland Together Homepage. `http://www.borland.de/together/`, 2005.

[TOHS99] P. Tarr, H. Ossher, W. Harrison und S. M. Sutton. N Degrees of Separation – Multi-Dimensional Separation of Concerns. In *Proceedings of the 21st International Conference on Software Engineering*, 1999.

[UG96] M. Uschold und M. Grüninger. Ontologies – Principles, Methods and Applications. *Knowledge Engineering Review*, 1996.

[UML03] Object Management Group. *OMG Unified Modeling Language Specification, Version 1.5*, 2003.

[Vli98] J. Vlissides. Subject-Oriented Design. *C++ Report*, 1998.

[Vog96] P. Vogler. *Praxis des Workflow-Managements*, Kapitel Chancen und Risiken von Workflow-Management. Vieweg-Verlag, 1996.

[VV87] A. Voß und H. Voß. Formalizing Local Constraint Propagation Methods. In *Arbeitspapiere der GMD 248*. Gesellschaft für Mathematik und Datenverarbeitung, 1987.

[W3C99] World Wide Web Consortium (W3C). *XML Path Language (XPath)*, 1999. W3C Recommendation.

[W3C01] World Wide Web Consortium (W3C). *XML Linking Language (XLink)*, 2001. W3C Recommendation.

[W3C04] World Wide Web Consortium (W3C). *Extensible Markup Language (XML)*, 2004. W3C Recommendation.

[WD02] C. Weber und T. Deubel. Ein neuer Ansatz zur Modellierung von Produkten und Produktentwicklungsprozessen. In M. Broy und M. Sihling, Hrsg., *Tagungsband zum Workshop Produktmodellierung*. Technische Universität München, Institut für Informatik, 2002.

[Wei02] P. Weinert. *Organisation*. Vahlen-Verlag, 2002.

[Wes03] T. Westermann. *Mathematische Probleme lösen mit Maple*. Springer-Verlag, 2003.

[WfM98] The Workflow Management Coalition. *Workflow Management Application Programming Interface Specification*, Juli 1998. Dokument WFMC-TC-1009.

[WfM04] The Workflow Management Coalition Homepage. `http://www.wfmc.org/index.html`, 2004.

[WK98] J. B. Warmer und A. G. Kleppe. *The Object Constraint Language – Precise Modeling with UML*. Addison-Wesley, 1998.

[WPD98a] Workflow Management Coalition (WfMC). *Interface 1: Process Definition Interchange – Organisational Model*, 1998. Document Number WfMC TC-1016-O, Document Status – Draft 6.99.

[WPD98b] Workflow Management Coalition (WfMC). *Interface 1: Process Definition Interchange – Process Model*, 1998. Document Number WfMC TC-1016-P, Document Status – 7.04 (Official Release).

[XPD02] Workflow Management Coalition (WfMC). *Workflow Process Definition Interface – XML Process Definition Language, Version 1.0*, 2002. Document Number WFMC-TC-1025, Document Status – 1.0 Final Draft.

[Zhu01] W. Zhuang. Konzeption der verteilten Ingenieursarbeit auf Basis des Common Product Model (CPM). Diplomarbeit, Technische Universität München, 2001.