

Laufzeit-Modellierung objektorientierter interaktiver Prozesse in der Produktion

Von der Fakultät
Konstruktions-, Produktions- und Fahrzeugtechnik
der Universität Stuttgart
zur Erlangung der Würde eines Doktor-Ingenieurs (Dr.-Ing.)
genehmigte Abhandlung

vorgelegt von
Dipl.-Inf. Thomas Schlegel
aus Ludwigsburg

Hauptberichter:
Prof. Dr.-Ing. Dr.-Ing. E.h. Dieter Spath

Mitberichter:
Prof. Dr.-Ing. Prof. E.h. Dr.-Ing. E.h. Dr. h.c. mult. Engelbert Westkämper

Tag der Einreichung:
15.02.2008

Tag der mündlichen Prüfung:
18.11.2008

IPA-IAO Forschung und Praxis

Berichte aus dem
Fraunhofer-Institut für Produktionstechnik und
Automatisierung (IPA), Stuttgart,
Fraunhofer-Institut für Arbeitswirtschaft und
Organisation (IAO), Stuttgart,
Institut für Industrielle Fertigung und
Fabrikbetrieb (IFF), Universität Stuttgart
und Institut für Arbeitswissenschaft und
Technologiemanagement (IAT), Universität Stuttgart

Herausgeber:

Univ.-Prof. Dr.-Ing. Prof. E.h. Dr.-Ing. E.h. Dr. h.c. mult. Engelbert Westkämper
und

Univ.-Prof. Dr.-Ing. habil. Prof. E.h. mult. Dr. h.c. mult. Hans-Jörg Bullinger
und

Univ.-Prof. Dr.-Ing. Dieter Spath



I·A·T Institut
Arbeitswissenschaft und
Technologiemanagement
Universität Stuttgart



Fraunhofer Institut
Arbeitswirtschaft und
Organisation

Thomas Schlegel

Laufzeit-Modellierung objektorientierter interaktiver Prozesse in der Produktion

Nr. 477

JOST-JETTER VERLAG
Fachverlag · 71296 Heimsheim

Dr.-Ing. Dipl.-Inf. Thomas Schlegel

Fraunhofer-Institut für Arbeitswirtschaft und Organisation (IAO), Stuttgart

Univ.-Prof. Dr.-Ing. Prof. E.h. Dr.-Ing. E.h. Dr. h.c. mult. Engelbert Westkämper

ord. Professor an der Universität Stuttgart

Fraunhofer-Institut für Produktionstechnik und Automatisierung (IPA), Stuttgart

Univ.-Prof. Dr.-Ing. habil. Prof. E.h. mult. Dr. h.c. mult. Hans-Jörg Bullinger

ord. Professor an der Universität Stuttgart

Präsident der Fraunhofer-Gesellschaft, München

Univ.-Prof. Dr.-Ing. Dieter Spath

ord. Professor an der Universität Stuttgart

Fraunhofer-Institut für Arbeitswirtschaft und Organisation (IAO), Stuttgart

D 93

ISBN (10) 3-939890-35-9, ISBN (13) 978-3-939890-35-5

Jost Jetter Verlag, Heimsheim

Dieses Werk ist urheberrechtlich geschützt. Die dadurch begründeten Rechte, insbesondere die der Übersetzung, des Nachdrucks, des Vortrags, der Entnahme von Abbildungen und Tabellen, der Funksendung, der Mikroverfilmung oder der Vervielfältigung auf anderen Wegen und der Speicherung in Datenverarbeitungsanlagen, bleiben, auch bei nur auszugsweiser Verwertung, vorbehalten. Eine Vervielfältigung dieses Werkes oder von Teilen dieses Werkes ist auch im Einzelfall nur in den Grenzen der gesetzlichen Bestimmungen des Urheberrechtsgesetzes der Bundesrepublik Deutschland vom 9. September 1965 in der jeweils gültigen Fassung zulässig. Sie ist grundsätzlich vergütungspflichtig. Zuwiderhandlungen unterliegen den Strafbestimmungen des Urheberrechtsgesetzes.

© Jost Jetter Verlag, Heimsheim 2008.

Printed in Germany.

Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Werk berechtigt auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürften.

Sollte in diesem Werk direkt oder indirekt auf Gesetze, Vorschriften oder Richtlinien (z. B. DIN, VDI, VDE) Bezug genommen oder aus ihnen zitiert worden sein, so kann der Verlag keine Gewähr für die Richtigkeit, Vollständigkeit oder Aktualität übernehmen. Es empfiehlt sich, gegebenenfalls für die eigenen Arbeiten die vollständigen Vorschriften oder Richtlinien in der jeweils gültigen Fassung hinzuzuziehen.

Druck: printsystem GmbH, Heimsheim

Geleitwort der Herausgeber

Über den Erfolg und das Bestehen von Unternehmen in einer marktwirtschaftlichen Ordnung entscheidet letztendlich der Absatzmarkt. Das bedeutet, möglichst frühzeitig absatzmarktorientierte Anforderungen sowie deren Veränderungen zu erkennen und darauf zu reagieren.

Neue Technologien und Werkstoffe ermöglichen neue Produkte und eröffnen neue Märkte. Die neuen Produktions- und Informationstechnologien verwandeln signifikant und nachhaltig unsere industrielle Arbeitswelt. Politische und gesellschaftliche Veränderungen signalisieren und begleiten dabei einen Wertewandel, der auch in unseren Industriebetrieben deutlichen Niederschlag findet.

Die Aufgaben des Produktionsmanagements sind vielfältiger und anspruchsvoller geworden. Die Integration des europäischen Marktes, die Globalisierung vieler Industrien, die zunehmende Innovationsgeschwindigkeit, die Entwicklung zur Freizeitgesellschaft und die übergreifenden ökologischen und sozialen Probleme, zu deren Lösung die Wirtschaft ihren Beitrag leisten muss, erfordern von den Führungskräften erweiterte Perspektiven und Antworten, die über den Fokus traditionellen Produktionsmanagements deutlich hinausgehen.

Neue Formen der Arbeitsorganisation im indirekten und direkten Bereich sind heute schon feste Bestandteile innovativer Unternehmen. Die Entkopplung der Arbeitszeit von der Betriebszeit, integrierte Planungsansätze sowie der Aufbau dezentraler Strukturen sind nur einige der Konzepte, welche die aktuellen Entwicklungsrichtungen kennzeichnen. Erfreulich ist der Trend, immer mehr den Menschen in den Mittelpunkt der Arbeitsgestaltung zu stellen - die traditionell eher technokratisch akzentuierten Ansätze weichen einer stärkeren Human- und Organisationsorientierung. Qualifizierungsprogramme, Training und andere Formen der Mitarbeiterentwicklung gewinnen als Differenzierungsmerkmal und als Zukunftsinvestition in *Human Resources* an strategischer Bedeutung.

Von wissenschaftlicher Seite muss dieses Bemühen durch die Entwicklung von Methoden und Vorgehensweisen zur systematischen Analyse und Verbesserung des Systems Produktionsbetrieb einschließlich der erforderlichen Dienstleistungsfunktionen unterstützt werden. Die Ingenieure sind hier gefordert, in enger Zusammenarbeit mit anderen Disziplinen, z. B. der Informatik, der Wirtschaftswissenschaften und der Arbeitswissenschaft, Lösungen zu erarbeiten, die den veränderten Randbedingungen Rechnung tragen.

Die von den Herausgebern langjährig geleiteten Institute, das

- Fraunhofer-Institut für Produktionstechnik und Automatisierung (IPA),
- Fraunhofer-Institut für Arbeitswirtschaft und Organisation (IAO),
- Institut für Industrielle Fertigung und Fabrikbetrieb (IFF), Universität Stuttgart,
- Institut für Arbeitswissenschaft und Technologiemanagement (IAT), Universität Stuttgart

arbeiten in grundlegender und angewandter Forschung intensiv an den oben aufgezeigten Entwicklungen mit. Die Ausstattung der Labors und die Qualifikation der Mitarbeiter haben bereits in der Vergangenheit zu Forschungsergebnissen geführt, die für die Praxis von großem Wert waren. Zur Umsetzung gewonnener Erkenntnisse wird die Schriftenreihe „IPA-IAO - Forschung und Praxis“ herausgegeben. Der vorliegende Band setzt diese Reihe fort. Eine Übersicht über bisher erschienene Titel wird am Schluss dieses Buches gegeben.

Dem Verfasser sei für die geleistete Arbeit gedankt, dem Jost Jetter Verlag für die Aufnahme dieser Schriftenreihe in seine Angebotspalette und der Druckerei für saubere und zügige Ausführung. Möge das Buch von der Fachwelt gut aufgenommen werden.

Engelbert Westkämper Hans-Jörg Bullinger Dieter Spath

Vorwort

„Es war schön und ist nun doch umso schöner, dass es geschafft ist.“ – Eine Doktorarbeit ist immer eine große Aufgabe, begleitet von großer Freiheit, aber auch vom „Kampf an vielen Fronten“. Dies ist sicher ein Aspekt, der sie so interessant macht und zu einem hohen Ansehen in unserer Gesellschaft, aber auch zu den aktuellen Änderungsbestrebungen geführt hat.

Meine Arbeit ist – ein wenig wie ich selbst auch – ein Wanderer zwischen den Welten. Informatik, Modellierung, Interaktion, Produktion und weitere interessante Wissenschaftsbereiche haben ihre Spuren hinterlassen. Ich denke, dass Interessantes häufig an solchen Grenzen zwischen den Fächern entsteht.

Ich hoffe und wünsche mir, dass diese Arbeit und ihre Ergebnisse ihren Teil dazu beitragen, die Welt noch etwas interessanter zu machen und dass meine Gedanken auf Ihr Interesse stoßen.

Ich danke meinem Doktorvater Herrn Prof. Dr.-Ing. Dr.-Ing. E.h. Dieter Spath aufs Allerherzlichste für seine Unterstützung und Förderung, die auch über die Arbeit hinausgeht.

Frau Dr.-Ing. habil. Anette Weisbecker danke ich ganz herzlich für die intensive Betreuung und Unterstützung auch über den Rahmen der Arbeit hinaus. Ihren fundierten Korrekturen und Hinweisen hat diese Arbeit ihre Qualität mit zu verdanken.

Herrn Prof. Prof. E.h. Dr.-Ing. Dr.-Ing. E.h. Dr.-Ing. h.c. mult. Engelbert Westkämper danke ich für die Übernahme des Mitberichts.

Ein großer Dank geht an meine Eltern, die mir erst meinen Werdegang ermöglicht haben und mich zu jeder Zeit voll unterstützt und begleitet haben. Auch meinem Bruder Matthias Schlegel danke ich dafür.

Meiner Partnerin möchte ich für ihre Begleitung, ihre Unterstützung und ihr Verständnis in dieser arbeitsintensiven Zeit danken.

Danken möchte ich auch meinen Kollegen wie Simon Thiel und Silke Lotterbach sowie meinen wissenschaftlichen Hilfskräften und Studenten, die mich in Diskussionen und Projektbeiträgen oft auf neue Ideen gebracht und mir die Arbeit einfacher gemacht haben.

Dem Fraunhofer IAO danke ich für die Dissertationsförderung und das förderliche Umfeld für die Arbeit.



Stuttgart, im November 2008

Thomas Schlegel

Inhaltsverzeichnis

Inhaltsverzeichnis.....	9
Abbildungsverzeichnis.....	13
Tabellenverzeichnis.....	16
Abkürzungsverzeichnis.....	17
Deutsche Zusammenfassung.....	21
English Summary.....	23
1 Einleitung.....	28
1.1 Dynamik, Flexibilität und Virtualisierung in der Fabrik der Zukunft.....	28
1.2 Dynamische Prozesse.....	29
1.3 Komponenten und Integrationstechnologien.....	29
1.4 Aufbau der Arbeit.....	30
2 Grundlagen, Vorarbeiten und Stand der Technik.....	32
2.1 Darstellungskonventionen objektorientierter Modelle.....	33
2.2 Aktuelle Konzepte IT-basierter Produktion.....	33
2.3 Flexibilität dynamischer Fertigungs- und Unternehmensstrukturen.....	34
2.3.1 Flexibilität.....	34
2.3.2 Verteilung von Abläufen.....	38
2.3.3 Verteilte Intelligenz – Agentensysteme.....	39
2.3.4 Service-orientierte Architekturen und Web-Services.....	39
2.4 Workflows und Prozesse.....	42
2.5 Abgrenzung der Prozesstypen im Zielsystem.....	44
2.6 Prozess-Modellierungstechnologien.....	44
2.6.1 Ablauforientierte Metamodelle.....	46
2.6.2 Prozess-Austauschsprachen.....	49
2.6.3 Petrinetze.....	50
2.7 Ontologien und Metamodellierung.....	53
2.7.1 Ontologie-Repräsentationen.....	54
2.7.2 Objektorientierung.....	55
2.8 Transformative und generative Modellierungsverfahren.....	57
2.9 Integration von Objekt- und Ablauforientierung.....	59
3 Ansatz und Aufgabenstellung.....	62
3.1 Vision.....	62
3.2 Ansatz.....	62
3.3 Aufgabenstellung.....	63

3.4	Anforderungen	65
3.5	Ausrichtung des Ansatzes	68
4	Ein Modellierungskonzept für objektorientierte Prozesse	70
4.1	Komponenten	70
4.2	Artefakte	72
4.3	Relationen	73
4.3.1	Typisierung von Relationen	73
4.3.2	Graphstruktur	76
4.4	Vererbung	78
4.5	Referenzierung und Referenzen	81
4.6	Instanziierung und Typen	84
4.6.1	Instanziierung über mehrere Ebenen	84
4.6.2	Prozess-Metamodelle	85
4.6.3	Polymorphie	86
4.6.4	Konstanten und Defaultwerte	87
5	Komplexe Komponenten und Abläufe im Metamodell	89
5.1	Konkatenation	89
5.1.1	Prozess-Konkatenation mit Hilfe der Prozess-Flussrelation	89
5.1.2	Implizite Konkatenation	90
5.2	Komposition und Aggregation	91
5.2.1	Aggregierte Ein- und Ausgangsbereiche von Teilkomponenten	93
5.2.2	Komponentensignaturen: Eingang, Ausgang, Typ	94
5.2.3	Polymorphie bei Prozessen	96
5.3	Vererbung komplexer Komponenten	97
5.3.1	Sub-Komponenten in der Vererbung	98
5.3.2	Mehrfachvererbung komplexer Komponenten	101
5.3.3	Variantenbildung und Versionen	103
5.4	Modelle und Modelltypen	104
5.4.1	Relationen im Diagrammtyp	105
5.4.2	Integration von Diagrammtypen	107
5.4.3	Implizite Modelltypen	108
5.5	Sichten	108
6	Das OMICRON Meta-Metamodell	111
6.1	Das Multi-Ebenen-Modell	112
6.2	Relationen im Gesamtmodell	113
6.3	Metamodelle	113

6.4	Basiselemente.....	115
6.5	Formalisierung der Zusammenhänge des Meta-Metamodells.....	116
7	Prozessmodellierung und Ablaufkonzepte im Modell.....	121
7.1	Modellanforderungen und Komponententypen.....	121
7.2	Kontrollfluss.....	122
7.2.1	Aktivierungseigenschaft des Kontrollflusses.....	122
7.2.2	Ereignisse, Trigger und Listener.....	123
7.2.3	Übergeordnete und zeitbezogene Ereignisse.....	125
7.2.4	Ausnahmen.....	125
7.2.5	Tokens.....	126
7.2.6	Daten- und Materialflüsse.....	128
7.2.7	Verzweigungen und Vereinigungen.....	130
7.3	Aufrufkonzept: Externalisierung von Befehlen.....	131
7.4	Generierung.....	133
7.4.1	Interaktionsgenerierung.....	135
7.4.2	Rollen.....	137
7.5	Ressourcen: Einführung einer Ontologie.....	138
8	Laufzeitmodellierung und Modellausführung zur Laufzeit.....	139
8.1	Laufzeitinstanziierung.....	139
8.2	Laufzeitveränderung.....	139
8.3	Metamodell-Konsistenz: Überprüfung von Änderungen.....	141
8.4	Veränderungen innerhalb der Hierarchie: Eine Fallunterscheidung.....	141
8.5	Laufzeitmodellierung Just in time: Rapid Prototyping für Prozesse.....	144
9	Modell-Verteilung: Ein verteiltes semantisches System.....	146
9.1	Dezentralisierung durch Peer-to-Peer-Konzepte.....	146
9.1.1	Caching des Modells.....	147
9.2	Nachrichtenbasierung – das Semantic-Messaging-Konzept.....	147
9.3	Hinzufügen neuer Elemente zum System.....	150
9.3.1	Nicht-integrierte Geräte.....	151
9.3.2	Hierarchien im dezentralen Peer-Netzwerk.....	151
10	Szenarien in der Produktion.....	154
10.1	Einsatzszenario: Verteilte Ausführung in existierenden Systemen.....	154
10.2	Variantenfertigung: Flexibilisierung und Effizienzsteigerung.....	156
10.3	Detailszenario.....	159
11	Implementierung.....	162
11.1	Komponenten-Server.....	162

11.2	Modellsystem	162
11.3	Grundfunktionen	163
11.4	Optimierung und Verteilung	164
11.5	Externalisierung von Aufrufen	165
12	Evaluation	166
12.1	Überprüfung der Anforderungserfüllung	166
12.2	Klassifikation möglicher Transformationsansätze	168
12.3	Effizienz des Ansatzes	169
12.4	Evaluation der Formalisierung und Automatisierbarkeit	172
12.5	Diskussion der Ergebnisse	173
13	Zusammenfassung	175
14	Ausblick	176
	Literaturverzeichnis	178
	ANHANG A: Grundbegriffe und ihre Herkunft	203
	ANHANG B: PlatformPeer-Systemkonzept	205
	ANHANG C: Prozess-Metamodelle	207
	ANHANG D: Entwicklung der Metamodelle	208
	ANHANG E: Kritik und Erweiterung des Vier-Ebenen-Modells	211
	ANHANG F: Repräsentation von Teilkomponenten bei komplexer Vererbung	216
	ANHANG G: Segmentierung: Kontext, Namespaces und Projektbereiche	219
	ANHANG H: Beispielhafte Artefaktklassifikation	221
	ANHANG I: Interpretation von Basistypen, Zeit, Kosten, metrische Begriffe	222
	ANHANG J: Mengenverarbeitung und Sammlungen	224
	ANHANG K: Ausnahme-Klassifikation und Fehlerbehandlung	225
	ANHANG L: Komponenteneigenschaften	227
	ANHANG M: MDA und generative, modellbasierte Softwareentwicklung	228
	ANHANG N: WRITE: Visualisierungen und Update	230
	ANHANG O: Klassifikation möglicher Transformationsansätze	231
	ANHANG P: Modellstruktur im Code: Klassendiagramm	235
	ANHANG Q: Code von BuildSubComponentHash	236
	ANHANG R: Aufbau und Ergebnisse der Testläufe	237
	ANHANG S: Externe Daten und Persistenz	242

Abbildungsverzeichnis

Abbildung 1: Aufbau der Arbeit	31
Abbildung 2: Überblick zur Entwicklung des Stands der Technik und Integrationsdefizit	32
Abbildung 3: Darstellungskonventionen in Diagrammen, basierend auf UML	33
Abbildung 4: Rationalisierung / Flexibilität: Zielbereich integrierter Objekt- und Prozessorientierung (grau).....	34
Abbildung 5: Entwicklung von Sprachkonzepten- und Verteilungstechnologien.....	41
Abbildung 6: „Wissenspyramide“ in Anlehnung an [Müller 2007] und [Wolf et al. 1999].....	53
Abbildung 7: Spannungsfeld Objektorientierung und Ablauforientierung	64
Abbildung 8: OMICRON Komponenten-Konzept.....	71
Abbildung 9: Eine Relation c zwischen zwei Komponenten A und B ist immer Instanz eines Relationtyps C	74
Abbildung 10: Erben von Beziehungen in der Metamodellierung – Beispiel, vgl. [VarróPataricza 2002].....	74
Abbildung 11: Integration der Port- und RelationEnd-Semantik in das Komponenten-Relationen-Modell.....	75
Abbildung 12: Relationsvererbung aus dem Metamodell	75
Abbildung 13: Komplexe Relationen als semantisch verknüpfte, binäre Relationen.....	77
Abbildung 14: Spezialisierende Vererbung im Beispiel in Anlehnung an [Mili et al. 2004].....	78
Abbildung 15: Mehrfachvererbung durch Integration von Fähigkeiten	78
Abbildung 16: Verwendung von Feldern bei der Vererbung	81
Abbildung 17: Inkompatibilität bei Vererbung als Verwendungsrelation – Notwendigkeit der Referenzierung.....	82
Abbildung 18: Vererbung als Lösung für Verwendung scheidet aus	82
Abbildung 19: Verwendung mit Hilfe der Instanziierung verursacht eine Durchmischung von Instanzebenen	83
Abbildung 20: Referenzierung als Lösung für die mehrfache Verwendung von Prozessschritten in Aktivitäten	83
Abbildung 21: Konkatenation – Verkettung von Komponenten anhand kompatibler Ein- und Ausgangsparameter ..	89
Abbildung 22: Kontextabhängige Definition von Ein- und Ausgängen bei der Konkatenation	90
Abbildung 23: Synthese komplexer Komponenten aus verknüpften Teilkomponenten: Ein- und Ausgänge.....	91
Abbildung 24: Aggregation in UML- und Container-Darstellung	92
Abbildung 25: Typkontrolle und -kompatibilität für Parameter.....	94
Abbildung 26: Definition Vor- und Nachbereich („Parameter“) einer komplexen Komponente	95
Abbildung 27: Vererbung einer komplexen Komponente	98
Abbildung 28: Anwendung der SubInheritance zur Referenzierung der geerbten Komponenten	98
Abbildung 29: Änderungskontext bei Änderung einer Teilkomponente.....	98
Abbildung 30: Vollständiger Kontext durch SubInheritance, Änderung von y durch Vererbung	99
Abbildung 31: Zusätzliche Aktivitäten in der Spezialisierung B erfordern einen vollständigen Kontext.....	99
Abbildung 32: Vererbung des vollständigen Kontexts mit Hilfe der SubInheritance	99
Abbildung 33: Vererbung und Änderungen auf mehreren Ebenen	100
Abbildung 34: Instanziierung mit vollständigem Kontext	100
Abbildung 35: Mehrfachvererbung ohne Änderung bedeutet unpriorisierte Aggregation aller Teilkomponenten....	101
Abbildung 36: AND-Sequenzierung der Aktivität A und Aktivität B in ihrer Spezialisierung AB.....	102
Abbildung 37: Mehrfachvererbung einer Teilkomponente über mehrere Pfade in der Vererbungshierarchie	102
Abbildung 38: Verschmelzung einer mehrfach geerbten Komponente x zu einer einzigen	103

Abbildung 39: Einfachvererbung und Einbindung benötigter Komponenten	103
Abbildung 40: Möglichkeiten der Variantenbildung.....	104
Abbildung 41: Zwei Möglichkeiten der Versionierung mittels einer speziellen Versions-Vererbung	104
Abbildung 42: Instanziierung von Bestandteilen eines Modelltyps auf Modellebene	106
Abbildung 43: Polymorphie bei der Modelltyp-Instanziierung.....	106
Abbildung 44: Polymorphe Verwendung und externe Relationen in Modellen.....	107
Abbildung 45: Aggregierte Metamodelle erlauben modellübergreifende Relationen.....	107
Abbildung 46: Zuordnung von Komponententypen zu Ansichtstypen über individuelle ElementViewTypes.....	109
Abbildung 47: Zugehörigkeit zu mehreren Sichten c und d auf dasselbe Modell, das a und b enthält	110
Abbildung 48: Vererbung von Sichten an Spezialisierungen.....	110
Abbildung 49: Selbstbezügliche Definition der Vererbung im Meta-Metamodell	113
Abbildung 50: Entwicklung einer endlosen Instanziierungrekursion im Meta-Metamodell	113
Abbildung 51: Das Meta-Metamodell als oberste Schicht des OMICRON-Gesamtmodelles	115
Abbildung 52: Ausschnitt aus dem Meta-Metamodell von OMICRON mit einem beispielhaften Metamodell.....	116
Abbildung 53: Daten-/Objekt- und Kontrollflüsse in und zu einer Aktivitätskomponente	122
Abbildung 54: Ereignisse in UML und OMICRON: Trigger (Send) und Listener (Receive)	123
Abbildung 55: Ausnahmen in OMICRON und UML2	126
Abbildung 56: Endknoten-Typen.....	127
Abbildung 57: Ereignisse und Informationen entstehen und werden vernichtet.....	128
Abbildung 58: Objektknoten – Flussdaten in der UML2	128
Abbildung 59: Objektknoten in der UML2 Objektknoten- (oben) und Pin-Notation (unten).....	129
Abbildung 60: Aktivitäten in Pin-Darstellung der UML2 ohne angebundene Quelle / Ziel	129
Abbildung 61: Alternative Notationsformen für Streams in UML2	130
Abbildung 62: Vergleichskomponenten als externe Funktion – in Komponenten und Aggregatdarstellung	131
Abbildung 63: Schematischer Aufbau einer Generatorkomponente	134
Abbildung 64: Beispiel für einzubeziehende Komponenten bei der Erzeugung einer komplexen Interaktion.....	136
Abbildung 65: Rollenabbildung in Anlehnung an [Caetano et al. 2005]	137
Abbildung 66: Beispielhafte, klassifizierende Ontologie für ein Produktionssystem	138
Abbildung 67: Production Service Bus (PSB), basierend auf semantischen Nachrichten	149
Abbildung 68: Beispielhafte Aggregationshierarchie einer Produktionsinfrastruktur	152
Abbildung 69: Beispielhafte organisationale Strukturierung auf Instanzebene	152
Abbildung 70: Prototypische Klassifikation von Struktur- beziehungsweise Organisationseinheiten.....	153
Abbildung 71: Einsatzszenario für OMICRON in einer bestehenden Fertigungsinfrastruktur.....	155
Abbildung 72: Bilder vom Test der Peer-Plattform von INT-MANUS, November 2007	156
Abbildung 73: Hinzufügen einer NC-Steuerung bei existierender Definition und Instanz des Steuerungstyps	159
Abbildung 74: Spezifizierung eines Sondermodells – vereinfachte Darstellung	159
Abbildung 75: Erweiterung eines bestehenden Grundprozesses.....	160
Abbildung 76: Blockierende Instanzen bei Änderungen mit existierenden Varianten.....	161
Abbildung 77: Effizienz der Einfügeoperation im Komponentenmodell des Laufzeitsystems.....	170
Abbildung 78: Rechenaufwand für den Modellexport – Zeit in Sekunden, abhängig von der Modellgröße	170
Abbildung 79: Exportdateigröße in Abhängigkeit von der Anzahl Komponenten im Modell.....	171

Abbildung 80: Meta-Ebenen des UML-Metamodells, angewandt auf die Produktion	211
Abbildung 81: Beispiel einer Vier-Ebenen-Hierarchie in Anlehnung an [OMG 2007a]	213
Abbildung 82: Strict Metamodeling in Anlehnung an [AtkinsonKühne 2002].....	214
Abbildung 83: Konzept der UML für die Instanziierung nach [AtkinsonKühne 2002].....	214
Abbildung 84: Swimlanes in UML	220
Abbildung 85: UML2 Notation von Schleifen in Aktivitäten	224
Abbildung 86: Die OMICRON-Schleifensemantik als Schleifenkonstrukt ohne semantisch fragwürdige Muster ...	224

Tabellenverzeichnis

Tabelle 1: Paradigmen-Entwicklung IT in der Fertigungstechnik (vgl. [Kurbel 2005])	33
Tabelle 2: Vergleich der Produktionsplanungs-, -steuerungs-, -überwachungs- und -integrationswerkzeuge	37
Tabelle 3: Zunehmende Dezentralisierung der Steuerungszintelligenz	38
Tabelle 4: Agentensysteme als Lösungsansatz für Verteilungsprobleme in der Fertigung [ShenNorrie 1999]	39
Tabelle 5: Verteilte Prozesse – aktuelle Klassen von Ansätzen	42
Tabelle 6: Prozessstypenvergleich für produzierende Unternehmen	44
Tabelle 7: Prozessmodellierungssprachen, Klassifikation nach [Zamli 2001] anhand von [ZamliLee 2001]	46
Tabelle 8: Prozessmodellierungssprachen und rechnergestützte Animationskonzepte	48
Tabelle 9: Austauschsprachen und deren Fähigkeiten, basierend auf [Mendling et al. 2004]	49
Tabelle 10: Petrinetz-basierte Ansätze zur Verwendung von Objekten	52
Tabelle 11: Kategorien von Sprachen mit Objekten	55
Tabelle 12: Konzepte der Objektorientierung	56
Tabelle 13: Generative Ansätze – Tauglichkeit für die Animation objektorientierter Prozessmodelle	58
Tabelle 14: Vergleich relevanter Konzepte für die Laufzeitmodellierung objektorientierter Prozesse	61
Tabelle 15: Anforderungen an das zu entwickelnde System	67
Tabelle 16: Graphtypen in OMIRCON	76
Tabelle 17: Definition der Flusstypen von Komponenten relativ zu einem Kontext K	94
Tabelle 18: Vergleich von Konzepten der Prozessmodellierung	121
Tabelle 19: Anforderungserfüllung anhand der Anforderungsgruppen	168
Tabelle 20: Tauglichkeit des Modellierungsansatzes für generative Verfahren nach [CzarneckiHelsen 2003]	169
Tabelle 21: Evaluation der Formalisierung und Automatisierbarkeit nach den Kriterien von [Müller 2007]	172
Tabelle 22: Laufzeit Modellerzeugung mit $i = 500$, $r = 15\%$	240
Tabelle 23: Laufzeit Modellerzeugung mit $i = 100$, $r = 15\%$	240
Tabelle 24: Laufzeit Modellerzeugung mit $i = 500$, $r = 85\%$	240
Tabelle 25: Laufzeit Modellerzeugung mit $i = 100$, $r = 85\%$	240
Tabelle 26: Laufzeit Modellabfrage auf transitive Instanzen (IsTransitiveInstanceOf)	241
Tabelle 27: Laufzeit Modellabfrage auf transitive Subtypen / Vererbung (IsTransitiveSubtypeOf)	241
Tabelle 28: Laufzeit Modellabfrage auf Instanzen und Subtypen, Vergleich der Geschwindigkeit	241
Tabelle 29: Laufzeit und Modellgröße bei Modelllexport	241

Abkürzungsverzeichnis

.NET	(Keine Abkürzung, Microsoft CLI Plattform)
4GL	Fourth Generation (Programming) Languages
APS	Advanced Planning and Scheduling /Advanced Planning Systems
AUML	Agent Unified Modeling Language
B/E-Netz	Bedingung-Ereignis-Netz
BDE	Betriebsdatenerfassung
BPEL	Business Process Execution Language
BPEL4WS	Business Process Execution Language for Web Services
BPML	Business Process Modeling Language
BPMN	Business Process Modeling Notation
BPSS	Business Process Specification Schema
CAD	Computer Aided Design
CIM	(in der Fertigung:) Computer Integrated Manufacturing
CIM	(in der IT / MDA:) Computation Independent Model
CLI	Common Language Infrastructure
CMM	Capability Maturity Model [®]
CORBA	Common Object Request Broker Architecture
CPN	Colored Petri Nets
DAG	Directed Acyclic Graph
DAML	DARPA Agent Markup Language
DAML+OIL	s. DAML, s. OIL, Vereinigung von DAML und OIL
DAML-ONT	DARPA Agent Markup Language Ontology
DAML-S	DARPA Agent Markup Language Schema
DARPA	Defense Advanced Research Projects Agency
DFD	Data Flow Diagrams
DIN	Deutsches Institut für Normung
DNC	Distributed Numerical Control
DNS	Domain Name Service
DSL	Domain Specific Language
DTD	Document Type Definition
DV	Datenverarbeitung
ebXML	Electronic Business XML
EFQM	European Foundation for Quality Management
EPK	Ereignisgesteuerte Prozesskette

EPML	EPK-Markup-Language
EPML	Enterprise Process Modeling Language
EPOS	Expert System for Program and System Development
ERP	Enterprise Resource Planning
ERP II	Enterprise Resource Planning II
ESB	Enterprise Service Bus
FOOP	Fraktale, objektorientierte Prozesse
HMS	Holonic Manufacturing Systems
HTML	Hypertext Markup Language
IDEAL	Independently Developable End-User Assessable Logical (Components)
IML	Interaction Markup Language (IML, [Schlegel 2002][Schlegel et al. 2004])
INT-MANUS	Intelligent Networked Manufacturing System (Projektname)
ISO	International Organization for Standardization
IT	Informationstechnologie
J2EE	Java 2 Enterprise Edition
KAON	Karlsruhe Ontologie
MB-UIDE	Model-Based User Interface Development Environment
MDA	Model-driven Architecture
MDE	Maschinendatenerfassung
MDS	Model-Driven Software Development
MES	Manufacturing Execution System
MIME	Multipurpose Internet Mail Extensions
MM	Metamodell
MMM	Meta-Metamodell
MOF	Meta Object Facility
MPL	Material- und Produktionslogistik
MRP	Manufacturing Resource Planning
NC	Numeric Control
OBA	Object Behavior Analysis
OIL	Ontology Inference Layer
OMG	Object Management Group
OMICRON	Objektorientiertes Metamodell für interaktive computergestützte Prozesse in Produktionsnetzwerken
OMT	Object-Modeling Technique
OO	Objektorientierung / objektorientierte
OOA	Objektorientierte Analyse
OOA&D	Object-Oriented Analysis and Design

OOCNP	Object-oriented Colored Petri-Nets
OOP	Objektorientierte Programmierung
OOSE	Object-Oriented Software Engineering
OPC	Open Process Control
OPC UA	Open Process Control Unified Architecture
OPN	Object Petri-Nets
OWL	Web Ontology Language
OWL-DL	OWL Description Logic
OWL-Full	s. OWL
OWL-Lite	s. OWL
OWL-S	Web Ontology Language Service Ontology
PEE	(SPADE) Process Enactment Environment
PIM	Platform Independent Model
PML	Process Modeling Language (Prozessmodellierungssprache)
PNML	Petri-Net Markup Language
PPS	Produktions-Planungs-System
Pr/T-Netz	Prädikat-Transitions-Netz
PSB	Production Service Bus
PSM	Platform Specific Model
RAID	Redundant Array of Independent Disks
RDF	Resources Description Framework
RDFS	Resources Description Framework Schema
RDFS(FA)	RDF-Schema Fixed Layer Metamodeling Architecture
RFID	Radio Frequency Identification
RPC	Remote Procedure Calls
S/T-Netz	Stellen-Transitionen-Netz
SCI	SPADE Communication Interface
SCM	Supply Chain Management
SDCG	Secure Device Control Gateway
SESAM-2	Software Engineering Simulation by Animated Models 2
SGML	Standard Generalized Markup Language
SLANG	Spade LANGuage
SOA	service-orientierte Architektur / Service-Oriented Architecture
SOAP	Simple Object Access Protocol
SOM	Semantisches Objektmodell
SOP	Start of Production

SPADE	Software Process Analysis, Design and Enactment
SPADE-1	Software Process Analysis, Design and Enactment 1
SPS	Speicherprogrammierbare Steuerung
SVG	Scalable Vector Graphics
SWRL	Semantic Web Rule Language
UDDI	Universal Description, Discovery and Integration
UIDE	User Interface Development Environment
UIE	(SPADE) User Interaction Environment
UIML	User Interface Markup Language
UML	Unified Modeling Language
UML2	Unified Modeling Language Version 2
UMLi	Unified Modeling Language for Interactive Applications
URI	Uniform Resource Identifier
VKD	Vorgangs-Ketten-Diagramm
VTT	Virtuelle Typ Tabelle
WfMS	Workflow (Modellierungs- und) Management System
WRM	Werkzeug- und Ressourcenmanagement
WS-CDL	Web Services Choreography Description Language
WSCl	Web Service Choreography Interface
WSCL	Web Services Conversation Language
WSDL	Web Service Definition Language
WSFL	Web Services Flow Language
WSIA	Web Services Interactive Applications
WSUI	Web Service User Interface
WSXL	Web Service Experience Language
XLANG	(Erweiterung von WSDL – keine Abkürzung)
XML	Extensible Markup Language
XPDL	XML Process Definition Language

Deutsche Zusammenfassung

Moderne Organisationsformen und globale Marktfaktoren im produzierenden Gewerbe stellen neue Herausforderungen an die Flexibilität und Dynamik von Produktionssystemen. Eine statische Automatisierung und Prozessimplementierung kann mit dieser Entwicklung weder strategisch noch operativ Schritt halten.

Diese Arbeit stellt daher ein integriertes Konzept für eine Prozessmodellierung vor, das aktuelle Konzepte des objektorientierten Paradigmas wie Ontologien und Vererbung nutzt. Ergänzt man diese um eine Laufzeitkomponente und eine Methodik zur verteilten, dezentralen Anwendung, wird es möglich, Prozesse dynamisch zu modellieren, auszuführen und zu adaptieren.

Hierzu werden zunächst existierende Konzepte IT-basierter Produktionssysteme analysiert. Zudem fließen Erkenntnisse aus dem Workflowmanagement und der Prozessmodellierung sowie informationstechnische Methoden der Prozessausführung wie erweiterte Petrinetz-Konzepte mit ein.

Die Objektorientierung bietet für Prozesse eine Reihe von Vorteilen, wie konsistente Variantenbildung und -pflege durch Vererbung und individuelle Produktions-/Ablaufverfolgung mit Hilfe von Prozessinstanzen. Um diese Vorteile objektorientierter Konzepte für Prozesse in der Produktion nutzen zu können, müssen jedoch Ablauf- und Objektorientierung zunächst integriert werden. Hierzu wird ein Komponenten-Relationen-Modell entworfen, das beide Aspekte vereint.

Während dies für einzelne Komponenten weitgehend mit existierenden Methoden möglich ist, können komplexe, aggregierte Prozesse derzeit nicht von Vererbung, Instanziierung und Typsemantik profitieren. Daher werden die objektorientierten Prozesse gezielt für komplexe Komponenten erweitert, so dass Vererbung und Polymorphie nun auch für komplexe Prozesse eingesetzt werden können. Dabei wird die hierarchische Aggregation und Schnittstellenbildung als modellinhärentes Konzept eingeführt. Mit der Definition und Formalisierung des Meta-Metamodells von OMICRON entsteht so ein objektorientiertes Gesamtkonzept für objektorientierte komplexe Prozesse.

Dieses bildet die Grundlage für eine Prozessentwicklung und -ausführung zur Laufzeit. Durch geeignete Kontrollflusskonzepte wie Aktivierung, Tokens und Ereignisbehandlung sowie Ansätze für generative Konzepte kann das Modellsystem um eine Laufzeitkomponente ergänzt werden, die das Modell interpretiert und ausführt. So wird es möglich, Prozesse und deren Varianten objekt- und ablauforientiert zu beschreiben und zur Systemlaufzeit zu adaptieren. Zudem können nun Prozessinstanzen erzeugt, animiert und weitergegeben werden.

Um dies auch in verteilten, dezentralen Systemen zu erreichen, wird eine dezentrale, nachrichtenbasierte Infrastruktur geschaffen, die eine verteilte Ausführung des Modells in einem dynamisch veränderlichen Produktionssystem ermöglicht. So können flexible, heterogene Organisationsformen wie Betreibermodelle und virtuelle Unternehmen gemeinsame Prozesse ohne Neukonfiguration und Stillstandszeiten nutzen.

Dies wird in den Szenarien erläutert. Zunächst wird hierzu eine beispielhafte Systemkonstellation gezeigt, die existierende Teilsysteme wie MES an die OMICRON-Prozessinfrastruktur anbindet. Eine Übertragung auf die Fertigung der BSH Bosch und Siemens Hausgeräte GmbH und ein Variantenszenario im OMICRON Modell zeigen die Einsatzmöglichkeiten und Systemgrenzen auf. Die Implementierung und Evaluation zeigen in der Folge die Realisierung ebenso wie Grenzen in der Umsetzbarkeit.

So liefert diese grundlegende Arbeit eine von den Basiskonzepten bis zur Anwendungsebene durchkonzipierte Basis für die weitere Forschung und Systementwicklung dezentraler, flexibler Systeme in der modernen Produktion. Sie zeigt zudem weiteren Forschungsbedarf und neue Fragestellungen auf – beispielsweise im dezentralen Scheduling und über semantische Modelle hinausgehende System-Organisationsstrukturen – die durch die vorgestellten Konzepte erst entstehen.

English Summary

Production research is currently experiencing the rise of a paradigm-shift from time and cost optimization towards flexibility of processes and customization. Modern organizational forms and global market factors in the manufacturing industry require higher flexibility and greater dynamics of production systems to be able to compete in today's highly competitive and globalized market environment.

Keeping pace with these developments is difficult when employing static automation and process implementation. For this reason, new concepts have emerged, including rapid reconfiguration of the factory, mass customization, holonic enterprise and many others. One of the goals is to increase flexibility. Flexibility makes it possible to respond to customer's needs and to overcome dead-lock situations on runtime while preserving the advantages of current mass production systems in performance, cost and quality.

In digital production, model-based automation, control and supervision of processes are a basic requirement. To be able to act flexibly in the market and to respond to increasing individualization of products and their adjacent processes, more and ad-hoc variants have to be created and supported by production systems. This reduces down-, start-up and cycle times for changes and variants.

In addition, organisations as a whole become more and more dynamic. Virtual enterprises and logistic chains – production as well as transport logistics – are not restricted to formal company borders anymore, requiring systems like ERP II to integrate data and processes. Even within factories, different companies work together. This makes it necessary to support clear separation and profiles of stakeholder organisations as well as individual stakeholders by intelligent models.

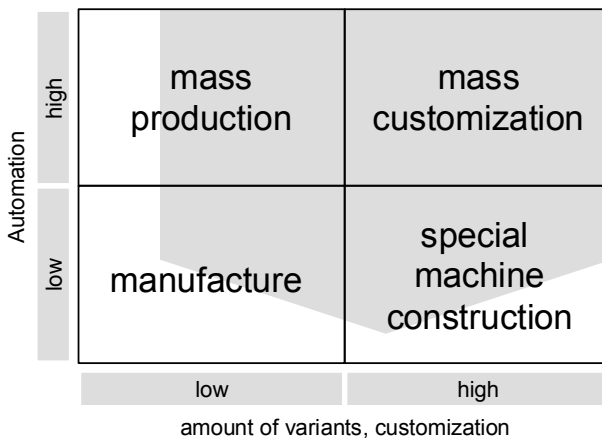


Figure 1: Automation and flexibility: applicability of object-oriented process models (grey)

Decentralized, flexible processes show their strength in production environments that are determined by a high amount of variants and a high degree of automation.

For environments where no product is similar to another, the application of a model-based process framework only makes sense if two conditions are given: high automation requires a clear model-based and executable definition of processes and variant processes are still required.

In fields with no process variants and a very limited number of product types, cost will often be higher than benefit. In all other fields, especially where mass customization shows a high degree of automation but also requires flexibility for customization, object-oriented processes can be of great benefit.

The main goal is therefore to integrate static structural models and dynamic flow models into a joint model for object-oriented processes in production:

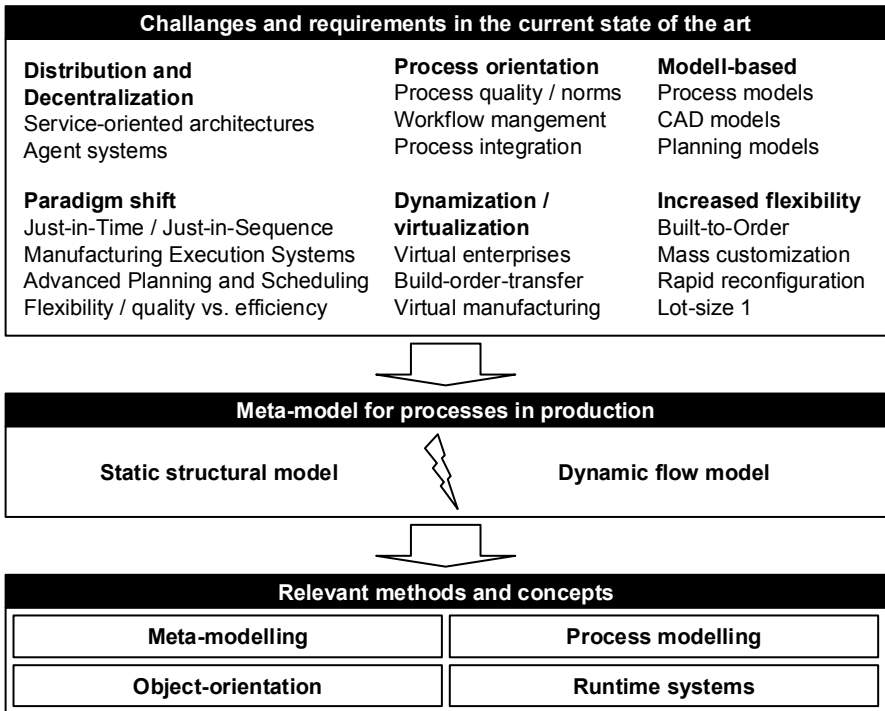


Figure 2: Overview on state of the art, challenges and approach of the thesis.

The word process is used in the sense of „a partially ordered set of tasks or steps undertaken towards a specific goal“ [Curtis et al. 1992].

In the context of object-oriented processes in production, processes contain steps, sub-processes, flow information and data, forming an executable description of action to achieve a specific goal – or perform a defined task.

Automation has developed from automation on one machine (computerised numerical controls) to static automation of whole production lines and even factories.

Distribution of control and intelligence is the next step and has become a major issue in the transition from central control systems via client server to agent-based and peer-to-peer

approaches. This development has even sped up with the rise of service-oriented architectures (SOA). When control is distributed, every peer in the network needs to have access to every model information that has an implication on its operation. In addition, each peer has to synchronize with others to keep the shared model consistent.

Process modelling languages have developed in application fields like business workflow, automation and embedded systems but process modelling languages have also been generalized using modelling methods like object-oriented Petri nets and process enactment languages.

Object-oriented concepts have been introduced to the modelling domain, e.g. by ontologies and semantic approaches in general. While these concepts have been researched in-depth for programming languages, object-orientation in workflows has not been extended to runtime model execution, meta-modelling and deep inheritance of processes.

Hence, this thesis introduces an integrated concept for process modelling that builds on – and incorporates – concepts of the object-oriented paradigm like ontologies, inheritance, instantiation, aggregation, semantic relations and classification. These basic (meta-)modelling concepts are complemented by a runtime component and a method for distributed and decentralized application. This extended model achieves the goal of being able to model, execute, modify and adapt processes dynamically while the production is running.

The developed concept and system is called OMICRON – object-oriented meta-model for interactive computer-based processes.

The developed meta-meta model has to make consistent use of the object-oriented concepts' inheritance, instantiation and aggregation. The meta-meta model has additionally been formalized using concepts from set theory.

At top level, a basic model element is defined, which is then instantiated into the (meta)Component and (meta)Relation. Each element instantiated from Component forms a component. Each element instantiated from Relation is a RelationType. When a RelationType is instantiated, it forms a relation. This enables the model to be extended by new component types and new relation types that can be specialized using inheritance. The thesis solves the problem of cyclic relations, relation type definitions and their inheritance and instantiation.

New models are described by meta-models that define components and relation types available in a model of this type. The meta-model defines all applicable component types and the types of relations that can be used between them when modelling by the meta-model's structure of aggregation and connection. Each model is an instance of one meta-model. In the thesis, it is shown that models can be treated as components whose concepts can be harmonized and integrated with normal components. This rules out the necessity of specific model component types. Where necessary, model-external concepts like views – which in their semantics are not a part of the core model – extend the model still using concepts of meta-meta model, e.g. the aggregation of views.

Inheritance of atomic components, e.g. roles, does not show many problems in implementation – even at runtime.

Complex components are created by aggregating atomic or even other complex components creating hierarchic structures. When a complex component, e.g. a process, is being specialized, the new specialized component has to inherit all steps and data parts of the above general/source component.

This is done by using reference relations from the child component's part (specialization) to the parent component. A changed sub-component in a child inherits from the parent's sub-component instead of referencing it.

In the thesis it is shown that for runtime models using a full reference – i.e. duplicating the full component structure with all sub-components and relations between them – is the more suitable way compared to other approaches, e.g. change coverage or algorithmic projection. This is because the full context is accessible in the model when a process is changed or extended.

Changes in the parent component cause transitive adaptations in unchanged children. When steps have been overwritten in a child, the overriding step is always considered, because the overwriting component is closer and therefore has priority in the inheritance graph.

To maintain a hierarchy in the graph, the inheritance graph forms a directed acyclic graph (DAG) because of multiple inheritance. Single inheritance would form a tree structure. It is shown that multiple inheritance is needed to represent real environments, e.g. to define integrated roles like “Manager and Administrator”.

Variants are created by inheritance, which means that each variant A’ of a basic process A inherits all elements of its parent process A. On design the advantages are that fundamental changes can be applied to all variants by modifying the basic process only. At runtime, object-oriented polymorphism enables the use of variants instead of the basic process using the type compatibility between the basic process and its variants.

All atomic input and output sub-components form the signature of a complex component, which can be a process for example. Input, steps and output of a complex component are characterized by the input, steps and output of their sub-components:

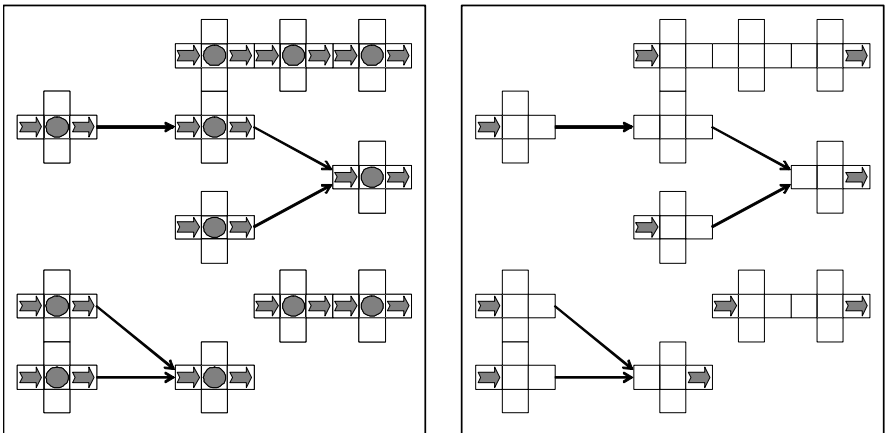


Figure 3: Input and output signature of a component created from the flow model

Processes can be instantiated on runtime in OMICRON. This means that every process instance – also called project – is an object-oriented instance of its process. Changes in the process have direct influence on all its instances and often also instances of its specializations, while the instance in turn is executed and typed according to the process defined.

The sequence of process steps is identified by flow relations – instances of the RelationType flow and its specializations – connecting the process steps in the desired sequence.

Once a process instance is created, all sub-components with data and input are created and initialized immediately. Process steps are created in line with the request of their execution: while

the process is executed, steps in the instance are created determining which are the next to be executed from the current state.

Flow relations in the instance are activated if they lead from a currently activated step to the next step. A token concept similar to that of the Unified Modelling Language 2 is used for description and execution.

To benefit from these enhanced models and execute processes in decentralized environments, a message-based, decentralized infrastructure is created that enables a decentralized execution of the model in a dynamic, changing production environment. In this way, flexible and heterogeneous organisational forms like build-operate-transfer (BOT) and virtual enterprises can use common processes, share semantically defined datasets and use common semantic models and classifications. Process instances can flow from one peer to another carrying their semantic model with them that describes how they are interpreted and executed.

Scenarios describe and detail the requirements and side-effects when implementing the OMICRON approach. An exemplary system constellation shows how existing partial systems like Manufacturing Executions Systems (MES) can be connected to and integrated with an OMICRON system that is responsible for the process infrastructure.

An application scenario at BSH Bosch und Siemens Hausgeräte GmbH and a detailed scenario on variants show the approach's applications, capabilities and limits of applicability.

This fundamental work describes a broad concept from the basis to the application. It delivers a foundation for further work in research and system development of decentralized and flexible systems in modern production. This work highlights the need for further research and introduces new questions that arise from a different view on production systems and new possibilities created by advancing intelligent technologies. This includes decentralized scheduling, new organizational structures of systems that go beyond semantic models and – evolving from the “Web 2.0” paradigm – new technologies and concepts for “Production 2.0”.

1 Einleitung

1.1 Dynamik, Flexibilität und Virtualisierung in der Fabrik der Zukunft

In der Produktionsforschung kündigt sich derzeit ein Paradigmenwechsel von Geschwindigkeits- und Kostenoptimierung hin zur Flexibilisierung von Abläufen an. Aktuelle Studien wie [I*PROMS 2006a] zu innovativen, vernetzten Produktionssystemen haben die Rekonfigurierbarkeit [I*PROMS 2006b] und Flexibilität [Constantinescu et al. 2006] als Kernziel hervor. Die zunehmende Virtualisierung [Westkämper et al. 2005] von Prozessen (beispielsweise [Shaofei et al. 2007]), Unternehmen und Produktion [Neugebauer et al. 2007] ist dabei nur mit neuen, semantischen Modellen möglich. Speziell Virtual Manufacturing [I*PROMS 2006a] und virtuelle Fabriken in der digitalen Produktion [Westkämper 2007] müssen sich hier auf hochwertige Modelle stützen. Semantische Modelle sollen daher starre Prozesse und datenzentrierte Lösungen zukünftig ersetzen. Unter dem Stichwort „Holonic Enterprise“ [Ulmer et al. 2002][Ulmer et al. 2001] zielt die Forschung daher seit einigen Jahren anstelle bisheriger Komplexitäts- und damit Flexibilitätsreduktion zunehmend auf die Flexibilisierung bei Vermeidung von Unter- oder Überflexibilität [Aurich et al. 2006]. Auf die hierzu entwickelten intelligenten Modelle stützt sich dann die Individualisierung von Produkten und Prozessen (Individualised Manufacturing), die sich durch flexible Adaption an Märkte und Kunden gegen Preisdumping in der Massenproduktion durchsetzen soll.

Zur Erhaltung von Qualität und Effizienz trotz Flexibilität in der digitalen Produktion erlangt die Automatisierung von Abläufen sowie deren Regelung und Kontrolle eine hohe Bedeutung [Spath 2007]. Um flexibel am Markt agieren zu können und der steigenden Individualisierung von Produkten Rechnung zu tragen, werden daher im Zuge steigenden Variantenreichtums und größerer Auftragsabhängigkeit Konzepte wie Mass Customisation [BullingerSchweizer 2005], Collaborative Business [WettkloSchultze 2003] und unternehmensübergreifende Workflows [MeierHomuth 2006] [MeierHomuth 2007] wie ERP II Systeme diskutiert und umgesetzt [Kurbel 2005]. Mit Konzepten der Massenproduktion und verstärktem Einsatz von Informationstechnologie soll dabei eine möglichst effiziente Fertigung bei hoher Qualität gewährleistet werden, während intelligente Modelle einen Flexibilitätsvorsprung erzielen helfen sollen. Einen ersten Schritt stellte dabei die Integration von Daten und Funktionalität mit Manufacturing Execution Systems (MES) [Kletti 2006] dar. Diese ermöglichen eine zeitnahe Planung und Überwachung in Produktionssystemen.

Die Integration in MES trägt der Individualisierung und Flexibilisierung jedoch nur begrenzt Rechnung. Musste bisher ein Fertigungsprozess möglichst exakt und schnell wiederholt werden, treten nun Prozessvarianten auf, die trotzdem effizient und weitgehend automatisiert ablaufen müssen. Was im Maschinen- und Anlagenbau von kleinen und mittleren Unternehmen an Anpassung ermöglicht wird, ist zunehmend auch Richtschnur für große Produktionssysteme. Diese sind jedoch für einen Menschen nicht mehr überschaubar und in ihrer Komplexität kaum mehr beherrschbar. Änderungen an Abläufen oder Strukturen können hier ohne systemtechnische Unterstützung nicht mehr manuell und ad-hoc durchgeführt werden. Vielmehr wird eine technische Unterstützung benötigt, die über eine reine Automatisierung weit hinaus geht: Das System muss die Semantik seines Zustands und seiner Einflussgrößen selbst erkennen und interpretieren können.

Hinzu kommt, dass immer häufiger nicht nur Produkte und Dienstleistungen sondern auch Unternehmen selbst dynamischen Veränderungen unterworfen sind. Änderungen dürfen daher nicht mehr als tiefe Einschnitte in die Fertigung und IT-Landschaft wirken. Unternehmenszusammenschlüsse, globalisierte oder sogar virtuelle Unternehmen, und Betreibermodelle [SpathDemuß 2001] erfordern neue Systemkonzepte [SpathSchuster 2004], die sich flexibel und mit geringen Anpassungskosten auf neue Strukturen und Anforderungen einstellen.

1.2 Dynamische Prozesse

Gerade die IT-Konzepte können hier treibende Kraft und Vorreiter sein – durch hohe Kosten im negativen, durch schnelle und günstige Anpassung im positiven Fall. Auch in der Informationstechnologie hat sich im Laufe der Jahre der Erkenntnis durchgesetzt, dass monolithische Architekturen, vollständige Planung und das Fixieren von Unternehmensstrukturen in Systemen häufig Ausfälle und hohe Kosten für Anpassungen nach sich ziehen.

Neue, dynamisch veränderliche Kooperationsformen vom Partizipations- bis zum Betreibermodell und eine neue Rollenverteilung zwischen den Beteiligten [SpathKoch 2007] erfordern eine schnelle Anpassung von Prozess- und Informationsstrukturen, die nicht mehr durch Umprogrammieren vorgenommen werden können. Soll ein neuer Betreiber eingebunden werden benötigt dieser teilweise auch interne Produktionsinformationen zur Eingliederung in den Produktionsprozess. Allerdings dürfen nur für ihn bestimmte Informationen sichtbar sein. Eine manuelle Selektion ist hier ebenso unmöglich wie eine andauernde Adaption von Filtern. Bereits das Prozess- und Informationsmodell muss also die „Intelligenz“ enthalten, die nun bestimmte Informationen ohne explizite Anweisung weiterleitet und andere verwirft.

Dynamisierung von Abläufen einerseits, und die Flexibilisierung von Fertigungsstrukturen andererseits, kann mit den existierenden Strukturen nicht mehr bewältigt werden. Daher setzen Unternehmen verstärkt auf informationstechnische Vernetzung und den Einsatz verteilter Strukturen, wie sie derzeit in den Bemühungen um service-orientierte Architekturen (SOA, [Erl 2005][Spath et al. 2006]) zum Ausdruck kommen.

Während einfache Abläufe an einer Maschine weitgehend durch Programmierung automatisiert werden können, scheitert die dynamische Ansteuerung und Rekonfiguration von Fertigungszellen oder ganzen Fabriken an der mit existierenden Methoden nicht beherrschbaren Heterogenität und den mit Änderungen verbundenen Seiteneffekten. Der Grund hierfür liegt in der derzeitigen Notwendigkeit, selbst dezentrale Fertigungsstrukturen durch einen zentralisierten Prozess zu steuern und notwendige Änderungen – speziell an der Benutzungsschnittstelle – immer manuell nachzuführen. Selbst ein einfaches Umleiten von Information ist durch nicht durchgängige oder fehlende Klassifikation von Artefakten, Nachrichten und Elementen der Produktion (beispielsweise Werkstücken oder Werkzeugen) unmöglich. Mit dem Aufkommen von komplexen Netzwerken und Identifizierungstechnologien wie RFID-basierten Smarttags wird zudem eine Virtualisierung in Gang gesetzt, die nur mit geeigneten Modellen ihre Vorteile ausspielen und beherrschbar bleiben kann.

Die Separierung von Abläufen in Workflows, SPS oder NC einerseits, und datenzentrierten Beschreibungen wie Stücklisten oder ERP-Daten andererseits, führt zudem zu einer Unvereinbarkeit von Teilmodellen für Abläufe und Daten – und damit zu Inselfösungen, die einer Dynamisierung und Flexibilisierung entgegenstehen und eine Weiterentwicklung und dynamische Anpassung verhindern. Im Gegenteil wird mit jeder Rekonfiguration eine manuelle Anpassung vieler Systembestandteile notwendig.

1.3 Komponenten und Integrationstechnologien

Initiativen zur Entwicklung von flexiblen Systemen auf Maschinenseite [SpathKoch 2005] und die Bestrebungen zur direkten Verbindung von ERP-Systemen mit der Produktions-Steuerung zeigen die Wichtigkeit teilsystemübergreifender Konzepte. Zukünftige integrierte Systeme werden auf Prozessebene eine modulare, modellbasierte Entsprechung benötigen, die Strukturen wie Anlagen, Maschinen und Anwendungen ebenso wie Komponenten, Sensoren und Aktoren mit ihren Fähigkeiten und Schnittstellen im Fertigungs- und Prozesssystem abbildet, integriert und Rekonfigurationen modellbasiert im Sinne eines „Plug-and-Produce“ [SpathKoch 2005] [Feldmann et al. 2007a][Feldmann et al. 2007b] handhabt. Informationstechnologie spielt hier auch als Komponente mechatronischer Systeme [SpathKoch 2007] mittlerweile eine wichtige Rolle.

Eine dynamische Anpassung von Prozessen und Interaktionen an unterschiedliche Anforderungen und Kontexte ist nur mit Modellen möglich, die in Klassen von Systemen operieren, also nicht eine spezifische Maschine oder ein bestimmtes Werkstück als Grundlage benutzen, sondern die Konzepte und Zusammenhänge hinter den Prozessen und Artefakten der Produktion „verstehen“.

Ähnliche Problemstellungen traten in der Softwareentwicklung aufgrund deren weitgehender Virtualität bereits vor längerer Zeit auf. Entsprechend existieren hier bereits vielfältige Ansätze zur Prozessmodellierung und Objektorientierung [Breu et al. 2005]. Trotzdem wurde auch hier eine Verschmelzung der eigentlich orthogonalen Konzepte von Objekt- und Ablauforientierung nur teilweise durchgeführt.

Erste Ansätze zur Integration objektorientierter Methoden in der Fertigung, beispielsweise die objektorientierte Fabrikplanung [Bergholz 2005][Laubscher 1997] in Modellierung [Schuh et al. 2007a] und Vorgehensmodell [Schuh et al. 2007b] sowie in Montage [Westkämper/Schreiber 2006] und Leitständen [Otterbein 1994], zeigen bereits die Anwendbarkeit des Paradigmas in der Produktion. Hier standen allerdings Kapselung von Produktionseinheiten, deren temporäre Vernetzung und die Reduktion auf notwendige Schnittstellen im Vordergrund.

Werden diese Ansätze zu einer „objektorientierten Prozessmodellierung“ [Shams-Aliee/Warboys 1995] für die Produktion weiterentwickelt, könnten Abläufe in der Fertigung mit demselben Modellierungskonzept beschrieben und ausgeführt werden, das auch die Strukturen der Fertigung modelliert und klassifiziert.

Soll eine neue Variante gefertigt, eine neue Vorschrift umgesetzt oder ein neuer Zulieferer oder Betreiber integriert werden, sind immer direkt die laufenden Prozesse betroffen. Es müssen neue Schritte eingefügt, existierende angepasst oder neue Teile und Anweisungen eingepflegt werden.

Im laufenden Betrieb, sogar während bereits Teilschritte für einen Auftrag ausgeführt wurden ist eine automatisch abgewickelte Prozessänderung dieser Form bisher nicht denkbar – auch wenn beispielsweise in der Automobilindustrie in vordefinierter und eingeschränkter Form noch Kundenwünsche während des Fertigungsprozesses berücksichtigt werden können.

Gerade in der Produktion mit ihren Varianten und komplexen Prozessen bietet sich also im Rahmen der neuen Entwicklungen und Anforderungen ein optimales Anwendungsgebiet für integrierte, objekt- und prozessorientierte Modelle.

1.4 Aufbau der Arbeit

Die vorliegende Arbeit zeigt daher in Kapitel 2 zunächst relevante Konzepte und Defizite von Prozessen in der Fertigung. Hieraus wird in Kapitel 3 der Ansatz entwickelt. Vor der Anwendung auf die Produktion steht zunächst die vollständige Integration der Prozessmodellierung mit der Objektorientierung in einem konsistenten Modellansatz (Kapitel 4). Eine Entwicklung von Mechanismen unter anderem der Vererbung, Instanziierung und Aggregation von komplexen Komponenten und Abläufen in Kapitel 5 führt zum Meta-Metamodell in Kapitel 6, das die entwickelten Konzepte in ein stringentes Modell integriert. Das Prozessmodellierungskonzept in Kapitel 7 zeigt die Umsetzung umfangreicher Prozessmodellierungskonzepte basierend auf dem objektorientierten Meta-Metamodell und vermittelt gleichzeitig wie existierende Konzepte wie die UML2-Aktivitäten [OMG 2005b] vom objektorientierten Gesamtmodell profitieren.

In Kapitel 8 vorgestellte Laufzeitkonzepte sollen dem System die Anwendung der entwickelten Konzepte in einem laufenden Produktionssystem ermöglichen. Das Modell soll gerade in verteilten, dezentralen Systemen seine Stärke zeigen, wozu besondere Konzepte der Informationsverteilung und -übermittlung in Kapitel 9 vorgestellt werden.

Das Szenario in Kapitel 10, die Implementierung in Kapitel 11 und die folgende Evaluation in Kapitel 12 zeigen die Praktikabilität und Umsetzung des Konzepts, so dass im Ausblick (Kapitel 12) weiterführende Themen aufgezeigt werden können.

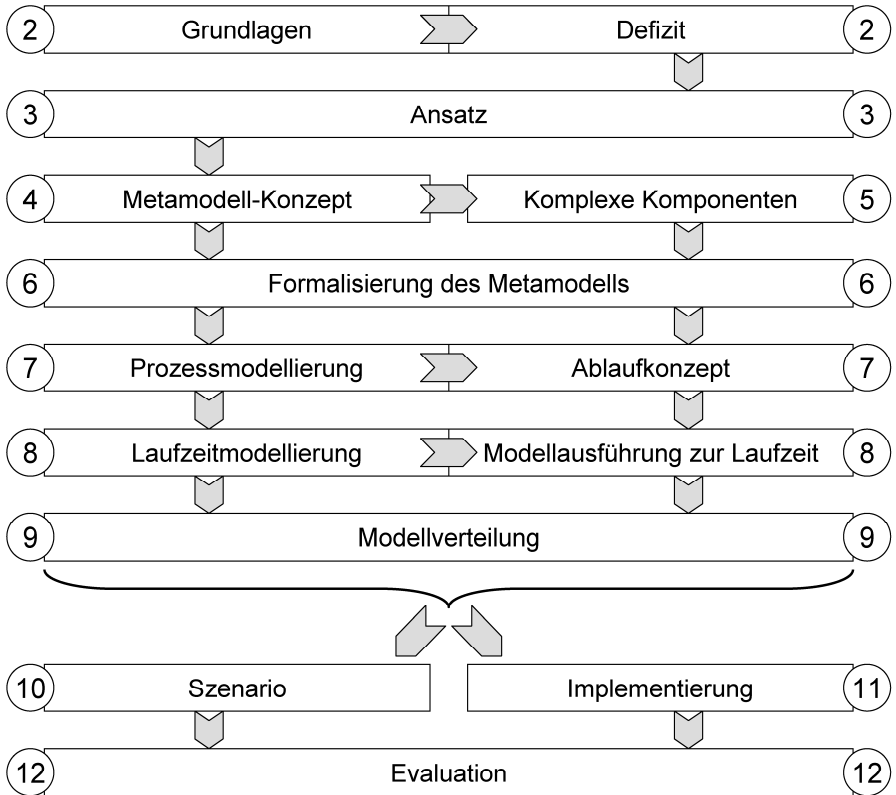


Abbildung 1: Aufbau der Arbeit

2 Grundlagen, Vorarbeiten und Stand der Technik

In diese Arbeit fließen unter anderem Erkenntnisse aus der Softwareentwicklung, Prozessmodellierung, Metamodellierung und Produktion sowie der Benutzungsschnittstellen-Modellierung und -Generierung ein. Hierbei soll durch Nutzung von Stärken und Verringerung von Defiziten der unterschiedlichen Disziplinen eine verbesserte Methode entstehen. Diese für die Arbeit speziell relevanten Themen werden im Rahmen dieses Kapitels untersucht.

Zunächst werden Besonderheiten, aktuelle Konzepte und Anforderungen aus dem Bereich der **IT-basierten Produktion** beschrieben. Diese liefern die Anwendungsgrundlage für die folgenden Betrachtungen.

Zunehmende **Flexibilität und dynamische Fertigungs- und Unternehmensstrukturen** haben zu vielfältigen Ansätzen geführt, die bisher starre Systeme flexibilisieren sollen.

Workflows und Prozesse wurden bereits in unterschiedlichen Gebieten wie der Workflow-Technologie näher untersucht. Eine **Abgrenzung der Prozesstypen im Zielsystem** ermöglicht eine Einordnung und Identifikation von spezifischen Defiziten. Existierende **Prozessmodellierungstechnologien** zeigen dabei den aktuellen Stand, aber auch Defizite für modellgestützte Abläufe in der Produktion, ergänzt um dezentrale Technologien. Mit **Ontologien und Metamodellierung** wurden bereits Erfolge hinsichtlich einer besseren Klassifikation und Modellintegration erzielt. Die Anwendbarkeit für das vorliegende Problem muss jedoch überprüft werden.

Aktive Modelle, die **transformative oder generative Modellierungsverfahren** ermöglichen, bedürfen aussagekräftiger und automatisierbarer Modelle, um flexible Produktionssysteme durch die Erzeugung von Artefakten zu unterstützen.

Diese für eine **Integration von Objekt- und Ablauforientierung** zu nutzen ist Anliegen dieser Arbeit. Existierende Ansätze aus unterschiedlichen Forschungsdomänen werden hierzu untersucht und ausgewertet.

Die Entwicklung der Problemstellung und des Stands der Technik aus den unterschiedlichen Forschungsfeldern zeigt die folgende Grafik:

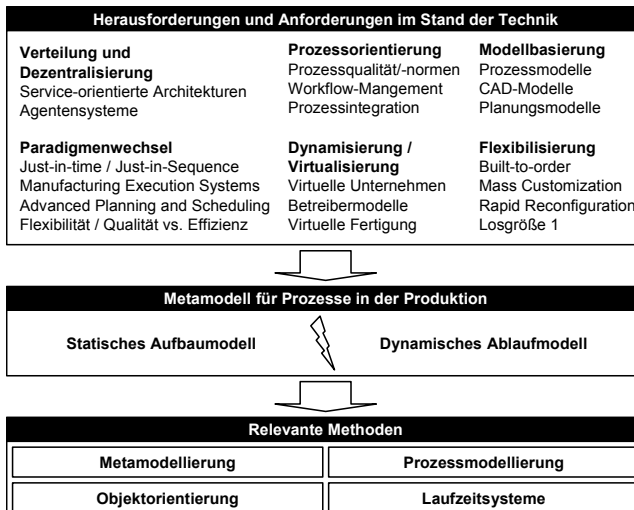


Abbildung 2: Überblick zur Entwicklung des Stands der Technik und Integrationsdefizit

2.1 Darstellungskonventionen objektorientierter Modelle

Im Rahmen dieser Arbeit werden häufig Modellkonstellationen und objektorientierte Zusammenhänge mit grafischen Modellen verdeutlicht. Die UML 2 [OMG 2005a][OMG 2005b] bietet für die Vererbung, Instanziierung, Aggregation und Komposition eine entsprechende Notation, die breite Verwendung in Forschung und Praxis erfährt.

Daher wird in Diagrammen soweit nicht anderes angegeben die UML als grundlegende Notation verwendet. Auf eine abweichende Notation oder Semantik wird entsprechend hingewiesen. Komponenten werden dabei als vereinfachte Klassen dargestellt, Relationen abhängig von ihrem Typ. Die folgende Grafik zeigt die Darstellung der grundlegenden Relationstypen.

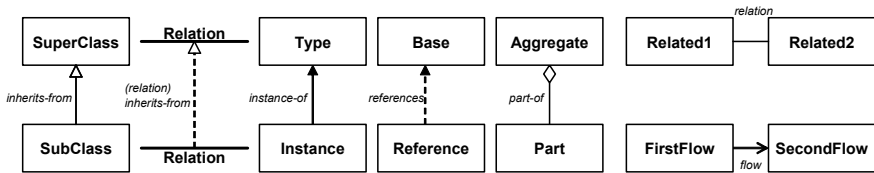


Abbildung 3: Darstellungskonventionen in Diagrammen, basierend auf UML

2.2 Aktuelle Konzepte IT-basierter Produktion

Kennzeichnend ist für Software, dass sie immateriell ist. Diese immateriellen Modelle und Softwarebausteine sind jedoch speziell in der Produktion an ein System gekoppelt, das sowohl für die Betriebsmittel (wie Werkzeuge) als auch für die Ergebnisse (Produkte) vollständig von physikalischen und materiellen Rahmenbedingungen geprägt wird. Die Immaterialität der Software macht daher eine explizite und detaillierte Modellierung der materiellen Zusammenhänge eines Produktionssystems notwendig, um eine Konvergenz von Software und Produktionssystem sicherzustellen.

Beispiel 1: So kann beispielsweise eine geänderte CAD-Konstruktion nicht ohne weiteres in die Produktion übergeführt werden, da mit CAM-Systemen Fertigungsstrategien definiert und Randbedingungen überprüft werden müssen, die nur für materielle Produkte gelten.

Klassische Modelle in der Produktion sind meist funktional und damit an Transformationen orientiert. Wie in anderen Ingenieursdisziplinen entwickelt sich die Fertigungstechnik von rein funktionalen zu komplexeren Modellen:

Paradigmen	Charakteristika
Statische Sicht , ausgehend von der Aufbauorganisation (vgl. [Kurbel 2005])	Statische Ausrichtung an den Organisationseinheiten, statisches Bild der Produktion
Funktionale Modelle basierend auf „Manufacturing Execution Functions“ [Engelke et al. 1983]	Funktionen wie Rest (Store/Wait), Move (Transfer/Handle), Make (Fabricate, Assemble), Verify (Inspect, Measure, Test)
Kommunizierende Funktionsblöcke , beispielsweise Funktionsblock-Modell (vgl. [Gehnen 1999] S. 104)	Netzwerk-, Hardware- und Konfigurations-Input, die Netzwerk- und Hardware-Output erzeugen
Objektorientierte Systeme , beispielsweise auf CORBA [OMG 2004b] aufbauende verteilte Systeme	Objekte senden Nachrichten, beispielsweise ein durch Software modelliertes Werkstück „schwarz lackieren“ an das Produktionssystem
Kombiniert prozess- und objektorientierte Modelle	Abläufe werden mit objektorientierten Konzepten modelliert, jedes Element (ob Prozessschritt oder Artefakt) ist objektorientiert beschrieben.

Tabelle 1: Paradigmen-Entwicklung IT in der Fertigungstechnik (vgl. [Kurbel 2005])

Objektorientierte Prozesse können ein gutes Kosten-Nutzen-Verhältnis besonders dann erreichen, wenn Variantenvielfalt und Automatisierung zusammenfallen, da funktionale und rein prozessorientierte Konzepte zu statisch sind, individuelle, klassifikatorische Ansätze jedoch einer Automatisierung entgegenstehen. Bei hoher Prozessautomatisierung und gleichzeitiger Flexibilität [Jackson 2000] durch Varianten- und Produktvielfalt sowie Auftragsabhängigkeit

bieten objektorientierte Prozesse ein hohes Optimierungspotenzial speziell für Mass Customization und alle Bereiche in denen Flexibilität bei gleichzeitiger Prozessstreuung und -unterstützung notwendig ist. Eine reine Massenproduktion eines Produkts profitiert dabei nicht von Flexibilität – ebenso wie eine Produktion von Einzelstücken ohne Gemeinsamkeiten nicht vom Prozessmodell profitiert. Alle anderen Fertigungstypen gehören jedoch zum Zielbereich:

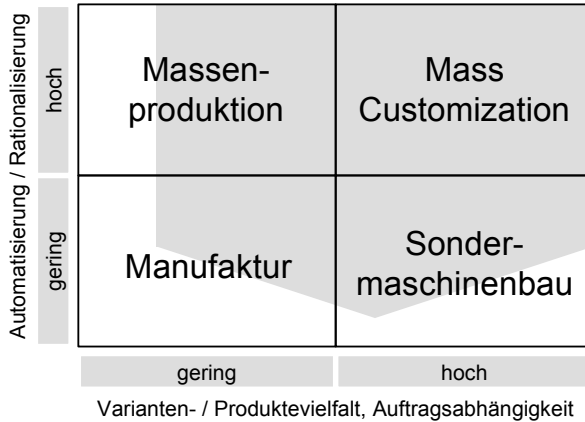


Abbildung 4: Rationalisierung / Flexibilität: Zielbereich integrierter Objekt- und Prozessorientierung (grau)

Ist die Variantenvielfalt hoch, gleichzeitig jedoch eine prozessorientierte Automatisierung erforderlich, entfalten objektorientierte Prozesse den optimalen Nutzen. Handelt es sich im Extrem um reine Einzelanfertigungen bei geringer Prozessautomatisierung, fällt der Vorteil jedoch ebenso gering aus wie bei sehr kleiner Variantenzahl.

2.3 Flexibilität dynamischer Fertigungs- und Unternehmensstrukturen

2.3.1 Flexibilität

Die Forderung nach Flexibilität (vgl. [Kurbel 2005]) und dynamischen Unternehmensstrukturen bei gleichzeitiger Integration und Automatisierung erzeugt ein Spannungsfeld zwischen **Autonomie und Integration**. Häufig wird Autonomie der Produktionseinheiten gefordert, um flexibel reagieren zu können. Den gleichzeitig für eine digitale Produktion [Westkämper 2007] notwendigen Integrationsgrad (vgl. [Spath 2007]) können jedoch nur Technologien erreichen, die neben der Autonomie in der operativen Ausführung eine Begrenzung, Überprüfung und Integrationen mit Informationen aus einem Gesamtmodell ermöglichen, das eine individuelle, digitale Fertigung erfordert.

Eine **Integration** wird deshalb stark thematisiert, weil für unterschiedliche Aufgaben in der Produktion in der Vergangenheit dedizierte, abgegrenzte Systeme (**Insellösungen**) geschaffen wurden, die nun integriert werden müssen (vgl. [Kletti 2006]), unter anderem Betriebsdatenerfassung (BDE), Maschinendatenerfassung (MDE), Leitstand/Plantafel, Werkzeug- und Ressourcenmanagement (WRM), Einstelldatenübertragung (DNC) sowie Material- und Produktionslogistik (MPL). Systeme der Prozessautomatisierung wie Open Process Control (OPC, <http://www.opceurope.org>) werden bisher meist losgelöst von diesen eher datenzentrierten Systemen betrachtet.

Bestrebungen eine Integration dieser Teilssysteme zu ermöglichen führten zu Produktionsleitständen und in neuerer Zeit zu **Manufacturing Execution Systems** (MES, [Kletti 2006]). MES sind hochintegrierte, dedizierte Softwarelösungen für die Produktionstechnik, die

Daten vormals getrennter Systeme integrieren und so eine bessere Übersicht und Planung der Produktion ermöglichen sollen. Der Begriff MES ist allerdings sehr weit gefasst, die Systeme je nach Ausprägung proprietär und beschränkt oder flexibel und mächtig. Trotzdem folgen auch sie einem zentralen, datenbankbasierten Paradigma, das noch immer als Herzstück der Informationsarchitektur in Unternehmen betrachtet wird (vgl. beispielsweise [Davenport 1998]). Die zentrale Konzeption ermöglicht keine Partitionierung und Hierarchisierung des Systems. Auch bereichsabhängige Sichtbarkeitsbeschränkungen wie sie für ein virtuelles Unternehmen notwendig sind können daher ebenso wenig vorgenommen werden, wie System-Rekonfigurationen im Betrieb bei laufenden Prozessen.

Ob ERP-System für die Querschnittsbereiche oder MES für die Fertigung – seit der Entwicklung der Client-Server-Systeme ist Zentralisierung ein wichtiges Grundkonzept.

Um zumindest die dynamische Verwendung von Funktionen zu ermöglichen, schlagen [LastraDelamer 2005] eine MES-Lösung mit Geräte-Agenten auf Basis von **semantischen Web-Services** vor. Hierzu wird neben einer semantischen Kennzeichnungsschicht (Knowledge Layer) auch eine Ambient-Intelligence-Schicht eingeführt, auf deren Basis die Geräte kommunizieren sollen. Das zentrale MES-Konzept bleibt dabei jedoch erhalten, so dass die Flexibilität fehlt.

Während ERP-Systeme im Wochenbereich planen, können PPS¹- und APS²-Systeme auch detaillierter planen. Selbst MES erreichen allerdings selten die Minutenebene. Eine genaue Planbarkeit entsteht hier nur durch Taktungen und genaues Design von Fertigungssystemen. Wo dies bei individueller Fertigung nicht möglich ist versagen daher auch die Planungssysteme, da diese meist zentrale Vorplanung und Simulation betreiben und eine situierte Planung [Thies 2003] aus dem Echtzeitkontext heraus nicht ermöglichen.

PPS-Systeme arbeiten zudem mit fixen Produktdatenstrukturen, so dass konzeptuelle Änderungen immer eine Systemveränderung hervorrufen. Mit ihrer Orientierung nach außen gingen die PPS-Systeme in den ERP-Systemen auf oder entwickelten sich zu solchen.

Der aktuell geläufige Ansatz für das Enterprise Resource Planning (ERP), das sich an den Geschäftsprozessen orientiert [HammerChampy 1993][Davenport 1993], basiert auf proprietären, monolithischen jedoch fachgebietsspezifisch komponentisierten Lösungen, die mit Hilfe einer Programmiersprache angepasst werden können.

Enterprise Resource Planning soll alle vorhandenen Ressourcen eines Unternehmens möglichst effizient in den betrieblichen Ablauf einplanen. *„Als ERP-System bezeichnet man hierbei zentrale, integrierte Informationssysteme, welche die vorwiegend produktionsbezogenen Funktionalitäten der MRP II- (Manufacturing Resource Planning-) beziehungsweise PPS (Produktionsplanung und -steuerung)-Programme [...] auf alle Kernbereiche eines Unternehmens erweitern und in einem ganzheitlichen Ansatz vereinen. Im Unterschied zu PPS-bilden ERP-Systeme nicht nur den Produktionsprozess eines Unternehmens ab, sondern definitionsgemäß alle relevanten Bereiche.“* [AlbertFuchs 2007]

Auch im ERP-Bereich entwickelt sich eine neue Sichtweise, weg von der zentralen, monolithischen Unternehmenssteuerung hin zu zwischenbetrieblichem Austausch. Daher entwickelt sich unter der Bezeichnung **ERP II** die Erweiterung klassischer ERP-Systeme um Funktionen zur Unterstützung **unternehmensübergreifender Prozesse**, die über das ebenfalls diskutierte Supply Chain Management (SCM)[KuhnHellingrath 2002] hinausgehen. Daher *„sind*

¹ Fälschlicherweise häufig als Produktions-Planungs- und -Steuerungs-Systeme bezeichnet, obwohl diese im Allgemeinen keine steuernde Komponente aufweisen.

² Advanced Planning and Scheduling oder Advanced Planning Systems [CorstenGössinger 2001]; deckt vielfältige Aspekte und Methoden der fortgeschrittenen Produktionsplanung mit Hilfe von ERP ab, fokussiert sich jedoch stark auf das Gebiet der Planung, weniger auf Prozesse etc.

flexible Systemstrukturen erforderlich, so genannte service-orientierte Architekturen, die mit Hilfe standardisierter Komponenten sowie Web Services die durchgängige Prozessabbildung und -unterstützung sicherstellen.“ [AlbertFuchs 2007]

Auch das Unternehmen selbst ist heute dynamischen Änderungen durch Unternehmenszusammenschlüsse, Umstrukturierungen und Verkäufe ausgesetzt. **Virtuelle Unternehmen, Konsortien, verlängerte Werkbank und Betreibermodelle** [SpathDemuß 2001] stellen Produktionsinformations- und -steuerungssysteme vor neue Herausforderungen. Die beschriebenen zentralen Ansätze erreichen maximal in der Logistik³ eine horizontale Integration für den Material- und Datenfluss. Sie sehen nur eine monolithische Vollintegration oder reine Schnittstellen vor. Mehrere gleichberechtigte Unternehmen, die mit wechselnder Informations- und Integrationstiefe verbunden werden sollen, stellen existierende Systeme daher vor unlösbare Probleme. Zentrale, nicht modellbasierte Strukturen führen hier häufig zu Informationsverlusten und immensen wirtschaftlichen Nachteilen, weil Daten nicht weitergegeben, integriert oder vertrauliche Informationen nicht von weiterzugebenden getrennt werden können.

So müssen **Betreiber** mit allen notwendigen Daten (Stücklisten, Konstruktionen, interne Aufträge) versorgt und in das Produktionssystem eingegliedert werden, gleichzeitig aber Informationen anderer Bereiche (externe Auftragsverwaltung, andere Teams) vor diesen verborgen werden.

³ Dezentrale Systeme für eine auf Web-Services basierende Logistik wurden beispielsweise im Projekt KOMPASS (<http://www.webservice-kompass.de/>) entwickelt.

K	Systemtyp Unterstützung für	APS/PPS	MES	ERP	ERP II	Automatisierung / OPC
S	Steuerung Wochen	●	●	●	●	○
S	Steuerung Tage / Schichten	●	●	○	○	○
S	Steuerung Stunden / Minuten	○	●	○	○	◐
S	Steuerung Sekunden	○	◐	○	○	●
P	Prozesse	◐	◐	◐	◐	◐
P	Prozess-Dokumentation	◐	◐	◐	◐	○
P	Prozesssteuerung	◐	◐	○	○	●
P	Prozessklassifikation	○	◐	◐	◐	◐
F	Varianten / Flexibilisierung	◐	◐	○	○	◐
F	Rekonfiguration der Produktion	◐	◐	○	○	◐
F	Mass Customization	◐	◐	◐	◐	◐
F	Individualisierung und „Losgröße 1“	◐	◐	◐	◐	◐
G	Segmentierung	○	◐	○	○	◐
G	Dezentrale Prozesse	○	○	○	○	○
G	Split-on-run (Aufteilung)	○	◐	○	○	◐
G	Virtuelle Unternehmen	◐	◐	◐	●	○
I	Unternehmensübergreifende Integration	○	○	◐	◐	○
I	Betreiberintegration (Sichten, PNP...)	○	◐	○	◐	○
I	CAM Integration	○	◐	○	○	◐
I	Integration	○	●	◐	◐	○
B	Produktionsbezug	●	●	◐	◐	●
B	Produktionsdaten	◐	●	●	●	◐

Legende: ● trifft voll zu, ◐ trifft weitgehend zu, ◑ trifft teilweise zu, ◒ trifft kaum zu, ○ trifft nicht zu;

Unterscheidung nach Kriterienclustern K:

zeitnahe Steuerung S, Prozessorientierung P, Flexibilität F, Segmentierung G, Integration I, Bezug zur Produktion B

Tabelle 2: Vergleich der Produktionsplanungs-, -steuerungs-, -überwachungs- und -integrationswerkzeuge

Wie das Unternehmen ist auch ein Produktionssystem neben Arbeitsabläufen und Materialflüssen meist hierarchisch strukturiert. Einzelne Arbeitsplätze und Maschinen werden zu Zellen und Produktionslinien zusammengefasst. Diese bilden wiederum Fabriken, Unternehmen und Lieferketten.

Dabei sind Kriteriencluster K (unter anderem aus [I*PROMS 2006a]) wie die zeitnahe Steuerung S (Wochen bis Sekunden), Prozessorientierung P (Dokumentation, Steuerung, Klassifikation), Flexibilität F mit Segmentierung G und Integration I sowie der direkte Bezug zur Produktion B wichtig. Ein wichtiges Element der Flexibilität ist dabei die Rekonfigurierbarkeit, die Systeme durch Modularität, Skalierbarkeit, Integrierbarkeit, Umwandelbarkeit, kundenspezifische Anpassung und Diagnostizierbarkeit kennzeichnet [Koren 2005][SpathKoch 2007], die sich ebenfalls auf die Prozesse auswirken.

Ein unterschiedlicher Fokus der untersuchten Werkzeugtypen führt zu unterschiedlichen Stärken und Schwächen. Den größten Bereich decken noch immer MES ab, wenngleich auch hier nur Steuerung und Produktionsbezug umfassend gegeben sind. Sowohl kurzfristige und langfristige Flexibilität [Gehnen 1999] sind eng mit dem Themenfeld „Rapid Reconfiguration of the Factory“ [I*PROMS 2006a] verknüpft. Selbst MES sind hier nur bedingt einsetzbar. Alle anderen Systemklassen verfügen nicht über ein umfassendes Modell der Produktion, so dass Systemgrenzen und -klassifikationen nicht zur Verfügung stehen.

2.3.2 Verteilung von Abläufen

Die Fertigungssteuerung in der Fertigungsleitebene umfasst die Planung von Fertigungsaufträgen, die Steuerung von Fertigungssystemen und die Überwachung der Durchführung von Fertigungsaufträgen. [Warnecke 1984]

Um diese Aufgaben zu erfüllen, müssen unterschiedliche Systeme wie Maschinen und Terminals überwacht und gesteuert werden. Der ursprünglich rein zentrale und hierarchische Ansatz wird jedoch der bereits oben beschriebenen, zunehmenden Flexibilisierung nicht mehr gerecht. Zusammenfassend erklärt [Gehnen 1999]: „Die Hierarchien der Fertigungssteuerung, die klassische CIM-Pyramide also, wird verschwinden und durch ein flaches Netzwerk ersetzt werden.“ Nach [Gehnen 1999] stellt so bereits das integrierte Netzwerk mit seiner flachen Vernetzungsstruktur eine Abkehr von der fixen Hierarchie der CIM-Pyramide dar. Webserver für Feldgeräte [Spath et al. 2002] stellen dabei eine Möglichkeit der Zugriffsdezentralisierung dar.

Diese „Dezentralisierung von Steuerungszintelligenz“ [TöpferSommerlatte 1991] wird auch durch die folgenden Generationen der Kommunikation im Feld [Gehnen 1999] charakterisiert, die folgende Eigenschaften aufweisen:

Generation	Technologie	Geräte	Leitfunktion
1.	Prozessrechner / Prozessrechnen	zentral	zentral
2.	Speicherprogrammierbare Steuerungen (SPS)	segmentiert	zentral
3.	Feldbussystemen wie CAN-Bus	Teilsystem	zentral
4.	Dezentrale intelligente Automatisierung	Peer	dezentral

Tabelle 3: Zunehmende Dezentralisierung der Steuerungszintelligenz

Peers sind dabei intelligente, dezentrale und vernetzte Elemente wie Softwareagenten die Systembestandteile unterschiedlicher Granularität vom Sensor bis zur Fabrik überwachen und steuern.

Nach [Brambilla et al. 2006] können sowohl Prozesse als auch deren Kontrolle zentral oder dezentral sein. Bei verteilten Prozessen mit zentraler Kontrolle existiert für jede Prozessinstanz ein Case-Manager, der alle Entscheidungen und die Ablaufkontrolle übernimmt. Bei der dezentralen Ablaufkontrolle ist neben der verteilten Modellhaltung ein Koordinationsmechanismus notwendig, der sich in einem Kontinuum zwischen den zwei Extremen *Nested Coordination* und *Generalized Coordination* bewegen kann. Bei *Nested Coordination* werden (Sub-)Prozesse mit genau einem Ein- und Austrittspunkt an andere Peers vergeben. Bei der *Generalized Coordination* muss mit Hilfe von Nachrichtenaustausch die Konsistenz sichergestellt werden, um beispielsweise zwei parallele Stränge wieder zusammenzuführen. [Brambilla et al. 2006] sieht auch hier einen speziellen „Main Peer“ vor, der die Koordination und Statusabfrage übernimmt. Für ein dezentrales System, das auch während der Prozessausführung Veränderungen unterliegen kann, reicht allerdings auch dieser Ansatz noch nicht aus.

Für die Realisierung einer verteilten Kontrolle besteht daher nach wie vor ein großes Defizit in der fehlenden Koordination. Diese ist nicht möglich, weil keine hinreichende Semantik in Software und Prozessmodellen existiert, die eine dezentrale Interpretation des Prozesses erlauben würde.

2.3.3 Verteilte Intelligenz – Agentensysteme

Verteilte Intelligenz auf der Basis von Agentensystemen führte zur Bildung des Begriffs Holon [van Brussel et al. 1998][Colombo et al. 2001] für intelligente Geräte und Produkte und deren Bezeichnung als Physical Agents⁴. Die folgende Tabelle zeigt existierende Agentenansätze:

Anwendungsdomäne	Ansätze / Projekte [ShenNorrie 1999]
Enterprise Integration and Supply Chain Management (SCM)	Agentensysteme für die Integration mit externen Systemen der Produktion. Beispiele: ABMA, ADE, AIMS, ATP, CIIMPLEX, CLAIM, IA framework, IAO, iDCSS, ISCM, KRAFT, Madefast, MADEsmart, MetaMorph I / II
Manufacturing Planning, Scheduling and Control	Agentensysteme zur Fertigungsplanung und -steuerung. Beispiele: AARIA, ABACUS, ADDYMS, AMACIOA, AMC, ARMOSE, CAMPS, CORTES, DAS, I-Control, IFCF, LMS, MAPP, MASCADA, MASCOT, Reagere, Sensible Agents, SFA, YAMS
Holonic Manufacturing Systems (HMS)	Einsatz des Paradigmas „Holon“ unter Nutzung existierender Agentenansätze. „Holon“ ist nach [Koestler 1967] ein Kunstwort aus „holos“ (das Ganze) und „on“ (Teilchen): „an autonomous and cooperative building block of a manufacturing system for transforming, transporting, storing and/or validating information and physical objects“ [Van LeeuwenNorrie 1997]

Tabelle 4: Agentensysteme als Lösungsansatz für Verteilungsprobleme in der Fertigung [ShenNorrie 1999]

Eine vollständig flache, möglicherweise sogar dynamische Netzstruktur der vierten Generation [Gehnen 1999] kann mangels fest programmierbarer Verfahren auch als Agenten- oder Peer-to-Peer-System nicht ohne ordnende Strukturen auskommen. Hieraus leitet sich das Defizit der fehlenden Modellsemantik für diese Klassen von verteilten Ansätzen ab. Während der Verteilungsaspekt und die funktionale Dezentralisierung von Agentensystem bewältigt wird, ist die prozessbasierte Koordination bisher nur über prozessunabhängige Sprechakte oder fixe Prozessmuster möglich.

2.3.4 Service-orientierte Architekturen und Web-Services

Service-orientierte Architekturen (SOA, [Dostal et al. 2005][Erl 2005]) werden derzeit häufig mit dem Ziel eingeführt, neue Systembestandteile zur Laufzeit zu integrieren und die Funktionalität an neue und veränderte Anforderungen und Prozesse dynamisch anzupassen. Auch für Embedded Systems in der Produktion wurden bereits SOA vorgestellt, beispielsweise das SIRENA System [JammesSmit 2005].

Konzepte wie Service-Discovery mit UDDI⁵, inhaltliche Service-Beschreibungssprachen wie OWL-S [W3C 2004f] und Ablaufbeschreibungen wie BPEL [IBM et al. 2003] können eine grundlegende Anbindung und Ansteuerung von Diensten in Prozessen sicherstellen. Allerdings fehlt ein Konzept, das es ermöglicht in verteilten Umgebungen zur Laufzeit dynamische Ablaufbeschreibungen und auf diesen basierende Benutzungsschnittstellen zur Verfügung zu stellen, da nur automatisierte Abläufe und Aufrufe ohne Interaktionen vorgesehen sind.

Gerade auch für dynamische Strukturen wie sie durch service-orientierte Architekturen vorherrschen sind modellbasierte Konzepte wie Model-driven Architecture (MDA) [OMG 2001] [OMG 2003] für Web-Services geeignet. [Santos et al. 2006]

Nach [Santos et al. 2006] lassen sich Kompositionsstrategien für Web-Services nach dem Planungsvorgang (manuell bis automatisch) unterteilen. Da für eine automatische Planung meist nicht genügend Modellinformationen vorliegen, erfolgt der Planungsvorgang entweder nur teil-automatisch oder der Ablauf wird bereits zur Entwicklungszeit vollständig fixiert. Neben der Differenzierung frühe (Entwicklung) und späte (zur Laufzeit) Bindung (early/late binding, [Santos et al. 2006]) muss unterschieden werden, ob bereits die Zuordnung von Funktionen zu

⁴ vgl. Foundation for Intelligent Physical Agents (FIPA), <http://www.fipa.org/>

⁵ Universal Description, Discovery and Integration, siehe www.uddi.org

Prozessschritten bei der Entwicklung oder zur Laufzeit geschieht. Selbst Web-Services sehen hier nur eine Zuordnung zur Entwicklungszeit vor.

So ermöglicht die Verwendung von Prinzipien der MDA zwar die „dynamische“ Komposition von Web-Services. [Santos et al. 2006] zeigt jedoch, dass **Dynamik** unterschiedlich interpretiert wird: Modellbasierte Generierung (MDA) versus direkte Animation des Modells zur Laufzeit. Im Rahmen dieser Arbeit wird vom Ziel einer direkten Modellanimation ausgegangen, während existierende MDA-Ansätze meist von einem vorgelagerten generativen Verfahren ausgehen.

Service-orientierte Architekturen, wie sie gestützt auf Web-Services erzeugt werden können, entwickelten sich als Antwort auf ein Defizit, das auch diese Arbeit adressiert: Monolithische und zentrale System sind einer dynamischen und verteilten Umgebung nicht gewachsen und führen zu hohen Kosten und Bruchstellen.

So entwickelte sich das verteilte, zwei(t)stufige Programmieren (2-Level-Programming) [Leymann 2003], das eine getrennte Entwicklung von Funktionen und Prozessen ermöglicht. Services müssen erst im zweiten Schritt durch die Servicekomposition als vollständige Prozesse aus einem Baukasten von Services realisiert werden.

BPEL [IBM et al. 2003] ist ein wichtiges Beispiel für eine hierfür verwendete Prozessbeschreibungssprache. Es entstand aus der Fusion und Weiterentwicklung anderer Ablaufsprachen für auf Web-Services basierende Prozesse, beispielsweise WSFL [Snell 2001].

Obwohl Web-Services eine dezentrale Technologie sind erfolgt eine sogenannte **Web-Service-Orchestrierung** mit BPEL zentral: Es werden dezentral verfügbare Dienste (Web-Services) im Rahmen einer zentralen Prozessplanung und -ausführung aufgerufen. So können zwar Dienste auf unterschiedlichen Systemen zur Verfügung gestellt werden, jedoch erfolgt die Ausführung des Prozesses zentral durch eine alle Services kontrollierende Workflow Engine. Die sich ergebenden Probleme erinnern dabei an zentrale Ansätze, wengleich Funktionen verteilt werden können. Ansätze zur notwendigen Umsetzung von Modellen auf BPEL Workflows wurden beispielsweise mit einer Abbildung von Zustandsdiagrammen auf BPEL [BenyoucefRinderle 2005] eingeführt.

Um auch mehreren Service-Anbietern die Integration von Web-Services in einem gemeinsamen Prozess zu ermöglichen, wird die **Web-Service-Choreographie** eingesetzt, die im Gegensatz zur -Orchestrierung nicht nur für einen einzelnen Service-Anbieter durchgeführt wird. Hierbei erfolgt eine Integration verschiedener Service-Anbieter, so dass ein Zugriff auf alle Servicedaten nicht möglich ist. Es müssen vielmehr komplexe Services aus einfachen Services unterschiedlicher Anbieter erzeugt und deren Zusammenspiel gesichert werden. Eine detaillierte Abgrenzung der Choreographie gegen die Orchestrierung liefert [Peltz 2003].

Die Verbindung von Services wird nötig, um aus einzelnen Diensten komplexe zu erzeugen und Prozesse auf den Services abzubilden. Ein Schwachpunkt ist, dass Änderungen von Dienstespezifikationen – selbst bei einer Änderung der Server-Adresse – Änderungen des Modells, beispielsweise der BPEL-Beschreibung, nach sich ziehen. Die eigentliche Idee, passende Services beispielsweise mit UDDI zu finden und direkt aufzurufen, also von konkreten Endpunkten und Funktionen zu abstrahieren wird damit torpediert und eine flexible typbasierte (abstrahierte) Ablaufsteuerung unmöglich gemacht.

Durch die dezentrale Ausrichtung funktioniert die Datenübertragung nach dem Buskonzept, so dass für Service-orientierte Architekturen (SOA) der Begriff Enterprise Service Bus (ESB) geprägt wurde, der für entsprechende Systeme der Produktion von [SchlegelThiel 2007a] als Production Service Bus (PSB) übernommen wird: „*The foremost task is the composition and coordination of workflows across the distributed components attached to both the enterprise service bus and the production service bus. In order to get an execution framework for business process automation and automated production workflows, a run-time environment has to be provided that drives the business services attached to the ESB and triggers the production steps according to*

the component-spanning process descriptions, similar to semantic web-services concepts in automation." [Schlegel et al. 2007].

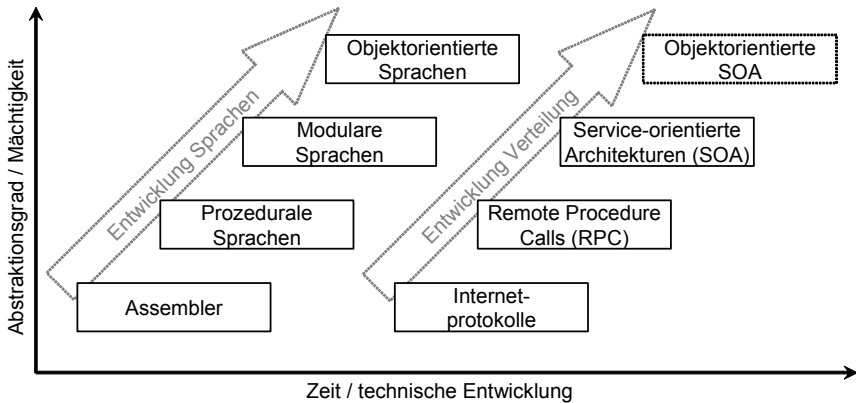


Abbildung 5: Entwicklung von Sprachkonzepten- und Verteilungstechnologien

Für verteilte System ist eine parallele Entwicklung zu den Paradigmen der Softwareentwicklung zu beobachten: Proprietärer Datenaustausch im Internet (Assembler), RPC und Web-Service-Aufrufe (funktional/prozedural), BPEL [IBM et al. 2003] / SOA (prozedural/modular, Architektur) zu Semantic Web-Services (Objektorientierung).

Menschen sollen in BPEL 2.0 [OASIS 2007] Prozesse durch das nun auf Web Services Human Task (WS-HumanTask, [Active Endpoints et al. 2007a]) basierende BPEL4People [IBMSAP 2005][Active Endpoints et al. 2007b] einbezogen werden. Es soll Nachteile von BPEL in der Einbindung von Interaktionen und menschlichen Arbeitsschritten im Vergleich zu anderen Prozessmodellierungs- und Enactment-Ansätzen verringern. Enactment bedeutet dabei „Act of applying a methodology to a particular project.“ [Gonzalez-PerezHenderson-Sellers 2006].

Das komplexe Thema der Benutzungsschnittstellen in service-orientierten Architekturen (SOA) führte zu unterschiedlichen Initiativen wie Web Services User Interface (WSUI, [Epicentric 2001]), WSXL [IBM 2002] und WSIA⁶. Nur Web Services for Remote Portlets (WSRP) [OASIS 2006b] zielen stärker auf Web-basierte Interaktion und sind neben BPEL4People eine der wenigen derzeit noch existierenden Web-Service-Initiativen mit Fokus auf Interaktion.

Das Defizit von Ansätzen wie WSUI liegt vor allem in der fehlenden Modellintegration und semantischen Kontrolle über die Abläufe. Wegen der zentralen Orchestrierung kann auch BPEL4People dieses Problem nur begrenzt lösen. – Ein dezentral nutzbares Interaktionsmodell, das den erforderlichen Modellkontext bietet, existiert jedoch bisher nicht.

Derzeit existieren neben der Orchestrierung und Choreographie von Web-Services auch agentenbasierte Abläufe. In der folgenden Tabelle werden diese im Problemraum anhand von Kriterien für verteilte Prozesse ergänzt um Objektorientierung und Interaktion charakterisiert.

⁶ TC geschlossen, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsia

Unterstützte Konzepte \ Ansatz	WS Orchestrierung	WS Choreographie	Agentenbasierte Abläufe	Peerbasierte OO-Prozesse
Prozesse	●	●	◐	●
Prozesse zentral	●	●	◐	◐
Prozesse dezentral	◐	◐	◐	●
Flexibel	◐	◐	●	●
Dynamische Netzstruktur	◐	◐	●	●
Ereignisverarbeitung	◐	◐	●	●
Prozessvarianten	◐	◐	◐	●
Objektorientierung	○	○	○	●
Interaktion / Benutzer	○	○	○	◐

Legende: ● trifft voll zu, ● trifft weitgehend zu, ◐ trifft teilweise zu, ◐ trifft kaum zu, ○ trifft nicht zu

Tabelle 5: Verteilte Prozesse – aktuelle Klassen von Ansätzen

Während Web-Service-Ansätze zentral gesteuerte, verteilte Prozesse gut unterstützen, sind Agentensysteme in der Netzflexibilität überlegen. Nur eine Verbindung der Vorteile beider Ansätze und die Ergänzung durch Objektorientierung und Interaktion würden alle Bereiche abdecken – hier skizziert durch „peerbasierte OO-Prozesse“, die semantische und dezentrale Aspekte vereinigen sollen.

Semantic Web Services (SWS) für verteilte die Nutzung (Discovery, Klassifikation, Komposition und Orchestrierung) in der Fabrikautomatisierung und Rekonfiguration werden in [LastraDelamer 2006] bereits beschrieben und stellen – sofern die Übertragung in die Praxis gelingt eine gute Ausführungsbasis für die in dieser Arbeit entwickelte objektorientierte Prozessschicht dar.

Interaktion von Benutzern mit dem Prozess ist dabei meist nicht vorgesehen. Erste Ansätze für eine interaktive Modellierung von Abläufen durch Komposition von Web-Services zur Laufzeit mit Hilfe einer grafischen Umgebung und einer Inferenz-Maschine [Talib et al. 2005] sind zwar möglich, erlauben jedoch keine Interaktion mit dem Prozess.

Eine Konzentration auf service-orientierte Architektur wirft zudem viele Probleme auf, die speziell in der Produktion Wirkung zeigen. [Howerton 2007] zeigt auf, dass aufgrund der verwendeten Technologien und Konzepte SOA schlecht skalieren und die Reaktionszeit nur für Nicht-Echtzeit- oder „Near-Realtime“-Anwendungen ausreichend ist. Zudem ist eine Migration von existierenden Technologien und Architekturen oftmals schwierig und kostenintensiv, so dass eine Integration von SOA-Technologien für Teile der Struktur Sinn macht, jedoch gerade im Prozessbereich nicht ausreicht. Eine Peer-basierte Struktur mit objektorientierten Prozessen ist daher im Produktionsbereich überlegen.

2.4 Workflows und Prozesse

„A process-oriented model is a description of the sequence of processing steps these entities experience as they flow through the system.“ [HealyKilgore 1997]

Die Notwendigkeit, weitgehend standardisierte Abläufe für die Ablauforganisation einzusetzen wurde früh erkannt. Modellierungsmethoden wie die Ereignisgesteuerte Prozesskette (EPK) [Keller et al. 1992] und ausführende Workflow Management Systeme sollten Wiederholbarkeit, Verfolgbarkeit gleich bleibende Qualität gewährleisten. In der Produktion werden bisher kundenindividuelle Auftragsabwicklungen maximal in Form von Geschäftsprozessbibliotheken ermöglicht. [Zäh et al. 2006]

Ein Prozess wird häufig in folgender Form definiert *„a partially ordered set of tasks or steps undertaken towards a specific goal“* [Curtis et al. 1992], teilweise auch mit einer Ordnung *„a set of partially ordered steps intended to reach a goal“* [HumphreyFeiler 1992], wo häufig Kanten

„Kanäle“ (Datenflüsse, Eingaben) mit „Instanzen“ (Aktivitäten) verbinden [BeckerRosemann 1996]. Ein Prozessschritt wird dabei definiert als „*an atomic action of a process that has no externally visible substructure.*“ [HumphreyFeiler 1992]. Dabei ist jede Komponente eines Prozesses ein Prozesselement [Curtis et al. 1992].

Dies greift jedoch zu kurz und macht deutlich, dass aufgrund fehlender Konzepte der Komposition häufig nur Prozesse als Ganzes und atomare Prozessschritte, jedoch keine aggregierten Zwischenebenen existieren. Dies führt auch zu einer semantisch kaum zu erfassenden Unterscheidung von Prozesselementen und -schritten: „*Determining that a process element is a process step depends in part on whether any further decomposition of the element's structure is needed to support the objectives of the process model. In ordinary usage a task is often synonymous with a process and an activity is synonymous with a process element or step.*“ [Curtis et al. 1992]

Ein einheitliche Sichtweise auf Abläufe von atomarer (Prozessschritte) bis hin zu hoch-aggregierter Ebene (Prozess) fehlt daher bisher ebenfalls.

Die **Workflow-Modellierung** wurde bereits in den frühen 1990er Jahren diskutiert, um die Ausführung von Anwendungsprozessen informationstechnisch zu unterstützen [JablonskiBussler 1996]. Ein **Workflow** stellt dabei nach [Müller 2007] die technische Umsetzung eines Anwendungsprozesses dar: „*The automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules.*“ [WfMC 1999]

Auf operativer Ebene geht das Prozessmanagement in Workflow-Management [Gadatsch 2005] mit Workflow-Modellierung, Workflow-Ausführung und Prozess-Monitoring über.

In der Literatur wird die Prozessmodellierung von der Workflow-Modellierung getrennt betrachtet. Der Bruch zwischen Prozessmodellierung auf der konzeptionellen Ebene und Workflows auf der operativen Ebene ist hier sowohl Ursache als auch Wirkung des Defizits.

Prozesse können dabei sowohl transzendent als reale Abläufe (Prozesse nach [Gadatsch 2005]), als auch konkret und automatisierbar im Modell als Workflows vorliegen. Eine integrative Betrachtung von Workflows als (automatisierte) Prozesse einschließlich integrierter Modellierung und Ausführung fehlt bisher.

Eine **Prozess- oder Workflow-Instanz** beziehungsweise ein **Projekt** ist dabei die Repräsentation eines ausgeführten Prozesses. Von Prozessmanagement- und Workflow-Systemen wird häufig nur eine Prozesskopie erstellt oder nur der Prozess mit Hilfe von basalen Instanzinformationen animiert, was der Definition von objektorientierten Instanzen vollständig zuwiderläuft.

Wenn auch noch nicht an Abläufen orientiert, führen [Henderson-SellersGonzalez-Perez 2005a] zumindest einen korrekten Begriff des Projekts als Instanz der Methodenebene eine, die ihrerseits dem Metamodell untergeordnet ist. Jedes Projekt folgt dabei einem Prozess, der es strukturell und in seinen Abläufen definiert sowie entsprechend klassifiziert.

Prozess- und Workflow-Management-Systeme verwalten dabei Prozesse und deren Instanzen. Wobei **Adaptionen von Abläufen** in Prozessmanagementsystemen meist auf Instanz- und Schemaebene ausgeführt [Ly et al. 2006] oder auf semantisch flache Modelle wie BPEL [IBM et al. 2003] angewandt werden [ReichertRinderle 2006].

Da oft mehrere, nicht vollständig zugreifbare Systeme beteiligt sind, existieren als verteilte Ansätze neben Web-Service- und Middleware-Ansätzen auch agentenbasierte Workflow Execution Systeme [Stormer 2003].

2.5 Abgrenzung der Prozessstypen im Zielsystem

Prozesse spielen in der Produktion und den mit ihr verzahnten Bereichen wie der Entwicklung eine tragende Rolle. So ist das Anwendungsgebiet der Fertigung und des Maschinenbaus beispielhaft für **wissensintensive, strukturierte** Prozesse [Müller 2007]. Der Grad der **Automatisierungsfähigkeit** unterteilt Prozesse dieser Felder weiter.

[Thies 2003] führt zur Charakterisierung von Prozessen die **Periodizität** (Synthese aus Wiederholungshäufigkeit und Wiederholungsintervalltreue), Schemavolatilität und Schemadetermination ein. In der folgenden Klassifikation industrieller Prozessstypen werden diese Kriterien um die Automatisierung und die Bedeutung von Wissensartefakten ergänzt.

Charakteristikum	Prozesstyp	Kreative Prozesse	Geschäftsprozesse	Sondermaschinenbau	Mass Customization	Massenproduktion
Wiederholungshäufigkeit		○	◐	◐	●	●
Wiederholungsintervalltreue		○	◐	◐	◐	●
Strukturierungsgrad		○	●	◐	●	●
Schemadetermination		○	●	◐	●	●
Schemavolatilität		●	○	◐	◐	○
Automatisierung		○	○	◐	●	●
Wissensartefakte in der Fertigung		●	◐	●	●	◐

Legende: ● trifft voll zu, ◐ trifft weitgehend zu, ◑ trifft teilweise zu, ◒ trifft kaum zu, ○ trifft nicht zu

Tabelle 6: Prozessstypenvergleich für produzierende Unternehmen

Für kreative und ad-hoc Prozesse in Unternehmen ist eine geringe Periodizität, hohe Schemavolatilität und geringe Schemadetermination charakteristisch. Für Prozesse in der Massenproduktion gilt dagegen eine hohe Periodizität, sehr geringe Schemavolatilität und eine hohe Schemadetermination als charakteristisch. Im Fall einer Individualisierung im Rahmen von Mass Customization oder sogar Sondermaschinenbau ist eine mittlere Periodizität im Fall von weitgehend ähnlichen Prozess- und Auftragsstypen vorhanden, die Schemavolatilität ist mittel bis hoch je nach Typunterschieden und Variantenvielfalt.

Aufgrund der Qualitätserfordernisse muss jedoch die Schemadetermination hoch sein, da sonst Nachvollziehbarkeit und Wiederholbarkeit nicht gewährleistet sind. Daher ist durch ein Prozess-System für die Produktion trotz der Variantenbildung eine hohe Definiertheit – zusammengesetzt aus Periodizität, Schemavolatilität und Schemadetermination – zu gewährleisten.

Dies legt Verwendung eines deskriptiven Komponentenkonzepts – aufgrund der Periodizität und hohen Schemadetermination – und die Verwendung eines variablen Verkettungskonzepts für diese Komponenten – aufgrund der Schemavolatilität auf höherer Aggregationsebene – nahe. Um jedoch eine integrierte Modellierung und Modellesemantik zu ermöglichen, müssen beide Aspekte mit einem gemeinsamen Metamodell abgedeckt werden und Prozesse exakt definiert sein.

Mit Ansätzen wie Model-Integrated Mechatronics [Thramboulidis 2005] wird bereits versucht, über mehrere Ebenen von mechatronischem Prozess über Ressourcen und Anwendung bis hin zur eigentlichen Mechatronik eine Integration herbeizuführen. Die Modellintegration findet jedoch auch hier nur in der Entwicklung statt und mündet nicht in eine modellintegrierte Ausführung und Adaption zur Laufzeit, wie sie für eine flexible Produktion notwendig ist.

2.6 Prozess-Modellierungstechnologien

Die Problematik der Modellierung von Prozessen ohne ein konsistentes Metamodell liegt in der „ad-hoc“ Erstellung von Prozessen und damit der Abhängigkeit von der Erfahrungsdomäne des Modellierers [Rolland 1998], was nur durch ein gemeinsames, meist domänen-spezifisches Metamodell gelöst werden kann.

Existieren definierte Prozessmodelle, so dienen diese in der Praxis meist der Beschreibung, Analyse und Ausführung. Das Ziel bei der Anwendung zur (Produktions-)Ablaufsteuerung liegt dabei meist auf Prozessanimation [Ould 1995], für das eine eindeutige und verständliche Beschreibungsmethode und Spezifikation, das heißt ein automatisierbares Modellierungskonzept, notwendig ist [Mili et al. 2003].

Metamodelle in der Prozessmodellierung verfügen meist über unterschiedliche Teilmodelle oder Modellperspektiven. Ein Metamodell für Prozesse hat beispielsweise nach [Curtis et al. 1992] eine funktionale Perspektive, die die auszuführenden Abläufe umfasst. Wer diese tatsächlich ausführen soll – ob Agenten oder Menschen wird von der organisationalen Perspektive beleuchtet. Um zu beschreiben wann, das heißt beispielsweise in welcher Reihenfolge, und wie, das heißt unter welchen Bedingungen/Entscheidungen und mit welchen Wiederholungen, Aktionen durchgeführt werden gibt die Verhaltensperspektive wieder. Die „informationale“ Perspektive zeigt, welche Informationen erzeugt oder verändert werden: Daten, Artefakte, Produkte, etc.

Aspektororientierte Prozessmodellierung [Jablonski 1994] unterteilt Prozessmodelle in Bereiche wie funktionale, verhaltensbezogene, organisatorische, operationale und datenorientierte sowie historische, qualitätsbezogene und sicherheitsbezogene Aspekte [Müller 2007]. Wenngleich diese Orientierung an Aspekten dem Verständnis zugute kommt, sollte jedoch ein Metamodell nicht hieran orientiert werden, wenn so das Kriterium der Konsistenz verletzt wird: Unabhängig von Aspekten müssen grundlegende Modellierungskonzepte gelten, auf deren Basis dann eine aufgabenabhängige Zusammenfassung nach Aspekten erfolgen sollte. Auch hier wird also ein Modellierungsaspekt ungünstig bereits auf höchster Abstraktionsebene verankert. Problematisch ist, dass die unterschiedlichen Perspektiven meist nicht Sichten auf ein Modell sondern disjunkte Modelle darstellen (vgl. [ListKorherr 2006]).

[Müller 2007] unterscheidet beim **Vorgehen zur Prozessmodellierung** zwischen der Prozessanalyse, der eigentlichen Prozessmodellierung und der Prozessausführung, da zunächst ein Prozess durchlaufen, dokumentiert und dann (erneut) ausgeführt werden muss. Für die Produktion können jedoch nur Prozesse für größere Stückzahlen einzeln geplant und durchlaufen werden, bevor ein Modell erzeugt wird – beispielsweise im Rahmen einer Vorserie. Der Einsatz von Varianten muss daher für eine flexible Fertigung ohne die Einhaltung eines vollständigen Prozess-Entwicklungszyklus möglich sein. Diese Ausführbarkeit ohne einen vorgelagerten Modellierungsprozess stellt noch immer ein Defizit der Prozessmodellierungsansätze dar.

Die weit verbreitete UML verfügt über eine informale, nur teilweise beschriebene Semantik, hat sich jedoch vor allem wegen ihrer grafischen Ausdrucksmöglichkeiten und dem weitgehenden Einverständnis über die grundlegenden Konzepte durchgesetzt.

Aktivitäten (Activities) wurden als Nachfolger der Aktivitätendiagramme (Activity Diagrams) für die Version 2 der UML um viele Prozessmodellierungskonzepte erweitert und damit zu einer ausgewachsenen Ablaufmodellierungsmethode. Die UML 2 betont eine integrative Modellierung stärker als die UML 1.x. Andere Diagrammtypen sollen die Möglichkeit einer Verbindung mit den modellierten Abläufen zu erhalten und so das Modell-Integrationsdefizit verringern.

Eine Formalisierung von UML-Aktivitätendiagrammen kann beispielsweise mit Hilfe einer Abbildung auf Petrinetze [Gehrke et al. 1998][Hagen 2005], auf die Prozessalgebra CSP [BoltonDavies 2000] oder auf abstrakte Zustandsautomaten [Börger et al. 2000] erfolgen. Auch eine Spezifikation mit Sprachen wie LOTOS [Pinheiro Da Silva 2001] ist möglich.

Eine Zusammenfassung der Akzeptanzentwicklung für die Software-Prozessmodellierung in der industriellen Praxis gibt [GruhnUrbainczyk 1998] wieder. Sie folgt einer Art Kondratjew-Zyklus.

2.6.1 Ablauforientierte Metamodelle

[BretonBézivin 2001] unterscheiden für Prozessmodelle Spezialisierungen des generischen Prozesses nach den Anwendungsbereichen Software Process, Workflow, Manufacturing und Business Process. „Manufacturing meta-model may thus define mechanisms to handle resources consumption, whereas software process introduces iteration.“ [BretonBézivin 2001]

Für Prozessmodelle aus der Praxis der betrieblichen Informationssysteme wurden traditionell stärker Diagramme mit informaler Semantik oder informale Notationen wie natürliche Sprache verwendet. [Rolland 1998]

Im Software Engineering werden hingegen häufig formalisierte Prozessmodelle eingesetzt (vgl. [Armenise et. al. 1993], [Curtis et al. 1992], [Finkelstein et al. 1994]). Diese hauptsächlich aus dem Requirements Engineering und der Forschung zu modellgestützten Softwareentwicklungsprozessen stammenden Ansätze bieten ein hohes Potential für die Problemstellung und fließen daher in den Vergleich ein.

Zu unterscheiden sind dabei nach [Zamli 2001] reine Prozessmodellierungssprachen (Process Modeling Languages, PML), simulierte und ausführbare Sprachen, die durch unterschiedliche Eigenschaften charakterisiert werden:

Unterstützungsgebiete \ PML Typ	Non-Enactable PML	Simulated PML	Enactable PML
Modellierungsunterstützung	●	●	●
Evaluationsunterstützung	○	●	●
Prozessanimation („Enactment“)	○	○	●
Weiterentwicklungsunterstützung („Evolutions“)	○	○	●
Benutzerunterstützung („Human Dimension Support“)	○	○	●

Legende: ● trifft zu, ○ trifft nicht zu

Tabelle 7: Prozessmodellierungssprachen, Klassifikation nach [Zamli 2001] anhand von [ZamliLee 2001]

Ereignisgesteuerte Prozessketten (EPK [Keller et al. 1992]) wurden mit dem Ziel eines besseren Verständnisses von Geschäftsprozessen entwickelt. Die Grundelemente sind dabei Funktionen und Ereignisse. Funktionen stellen Aktivitäten dar, während Ereignisse von Funktionen oder Aktoren ausgelöst werden. Mit der EPK Markup Language (EPML, [MendingNüttgens 2003]) wurde auch eine XML-basierte Repräsentation für EPK entwickelt. Da der Hauptzweck die grafische Repräsentation und Analyse durch Benutzer ist, fehlt eine weitergehende Typ- oder Ablaufsemantik, die eine Animation komplexer Geschäftsprozesse ermöglicht. Die direkte Abbildung von EPK auf S/T-Netze [MoldtRodenhagen 2000] ist möglich.

Für EPK und andere Ablaufkonzepte existieren Meta-Modell-Ansätze, die sich jedoch meist darauf beschränken eine Klassen- und eine Instanz-Ebene für ein existierendes Modellierungskonzept wie EPK einzuführen oder einen Meta-Prozess in Form eines Vorgehensmodells zur Entwicklung bestimmter Prozesse zu definieren [Keller et al. 1992].

Die UML 2 Aktivitäten sind ein klar mächtigeres und fortgeschritteneres Konzept, wie der Vergleich zwischen EPK und UML 2 Aktivitäten von [Störrle 2007] zeigt.

Wie die EPK verfügt auch die Business Process Modeling Notation (BPMN, [OMG 2006a]) über Ereignisse und Aktivitäten (in EPK Funktionen genannt). Bei den Abläufen selbst wird zwischen Sequence Flow und Message Flow unterschieden (vgl. [Brambilla et al. 2006]). Die Trennung zwischen Aktivierung und Informationen ist ein wichtiges Konzept und wird in Kapitel 7.2 näher erläutert und umgesetzt.

Mit der Business Process Modeling Language steht eine XML-basierte Repräsentation des Notationskonzepts zur Verfügung. Die Grundkonzepte ähneln denen der Aktivitätendiagramme

in UML 1.x (vgl. [ListKorherr 2006]). Noch mehr als die UML 2 lässt allerdings die BPMN eine definierte Semantik, Klassifikationskonzepte und Möglichkeiten der Automatisierung vermissen.

Ein aussichtsreiche und verbreitete Klasse von Prozessmodellierungssprachen sind die **zustandsbasierten** Sprachen, die komplexe Abläufe, Nebenläufigkeit (mit Operatoren zur Teilung und Zusammenführung von Kontrollflüssen) und Nichtdeterminismus wiedergeben. Hierunter finden sich häufig netzbasierte Sprachen. **Objektorientierte** Prozessmodellierungssprachen konzentrieren sich dem Paradigma folgend bislang meist auf die statischen Aspekte der Prozesse, das heißt deren Objekte wie beispielsweise Artefakte, oder basieren auf objektorientierten Sprachen wie Smalltalk. **Regelbasierte** Sprachen bauen häufig auf Prolog oder vergleichbare prädikatenlogische Sprachen auf und lassen daher Daten- und Objektkonzepte vermissen. **Funktionale** Sprachen führen eine Transformation durch, die Eingaben fordert und Ergebnisse erzeugt, scheitern jedoch an komplexen Zusammenhängen und dynamischen Verkettungen. **Prozedurale** Sprachen betonen dagegen den Ablaufaspekt in Form der Verkettung von Aktionen, so dass komplexe Abläufe wiedergegeben werden, jedoch die Variabilität und Flexibilität eingeschränkt wird.

Frühere objektorientierte Modellierungsmethoden wie OMT [Rumbaugh 1991], OBA [Rubin 1992], OOA [Coad 1991], OOA&D [Martin 1992] und OOSE [Jacobson 1992] wurden ebenso wie Zustands- [Harel 1987] und Datenflussdiagramme (DFD, [DeMarco 1979]) zu UML und ähnlichen Ansätzen weiterentwickelt und werden daher in dieser Arbeit nicht mehr gesondert betrachtet. Kommunikationsstrukturen in der Fertigung analysiert die Kommunikationsstrukturanalyse (CIM-KSA, [KrallmannScholz-Reiter 1990]).

Die folgende Tabelle liefert einen Vergleich von Prozessmodellierungssprachen, unter anderem basierend auf [Zamli 2001], [Hagen 2005], [Neumann 2002], [Zamliisa 2004], [Rolland 1998] und [ListKorherr 2006], ergänzt um Konzepte objektorientierter Prozesse wie objektorientierte Variantenbildung, Prozessinstanzierung und Teilprozesspolymorphie, Ausführbarkeit [ZamliLee 2001], integratives OO-Gesamtmodell / Meta-Meta-Modell, sowie speziell der Optimierung für Prozesse in der Fertigung.

Bezüglich der Verwendung objektorientierter Konzepte für die Prozessmodellierung wurden mehrere Studien durchgeführt (unter anderem [Hahn et al. 1997]), die meist einen geringen Vorteil für objektorientierte Ansätze erbrachten, jedoch zu stark von Vorkenntnissen abhängen. Hier wurde nur das bessere Verständnis der Modellierer, jedoch nicht die Problemlösungs-Mächtigkeit der Ansätze untersucht. Eine einfachere Adaption der Modelle an Änderungen wird jedoch konstatiert. Zusätzlich merken [Hahn et al. 1997] an, dass objektorientierten Modellierungsmethoden meist die für Geschäftsprozesse grundlegenden Dimensionen wie Kosten, Qualität und teilweise sogar Zeit fehlen.

Weil die UML nicht alle objektorientierten Konzepte konsistent umsetzt, wird sie vor allem vor der Version 2 auch als hybrid-objektorientiert und C++-orientiert bezeichnet [Henderson-SellersFiresmith 1999]. Fehlende Konzepte zu einer durchgängigen Vererbung auf allen Abstraktions- und Aggregationsebenen führen häufig zu einer nur teilweisen Objektorientierung, die in Behelfskonzepte wie Katalogsystem mit einer modellexternen Klassifizierung mündet [Mili et al. 2004] und gerade im Workflow-Bereich häufig zu Kopien statt objektorientierten Varianten führt.

Während für alle Modelltypen Beispiele existieren, die zudem meist über eine gute Ausführbarkeit verfügen, ist eine vollständige Prozessinstanzierung mit Teilprozesspolymorphie ebenso wie die Ausrichtung auf die Fertigung nicht enthalten. Objektorientierte Varianten sowie ein Gesamtmodell mit übergreifendem Meta-Meta-Modell existieren kaum.

Prozessmodellierungssprache (PML)	Kriterium	Klassifikation des Ansatzes						①	②	③	④	⑤	⑥
		①	②	③	④	⑤	⑥						
ADELE-PML [BelkhatirMelo 1994], ADELE-TEMPO [Belkhatir et al. 1994]						●	●	○	●	○	○	○	
APEL [Dami et al. 1998]					●	●	●	○	●	○	○	○	
APPL/A [Sutton et al 1995] / Arcadia				●			○	○	●	○	○	○	
BORM [Knott et al. 2003]					●	●	●	○	○	○	○	○	
BPDM [OMG 2007c] & BPEL4WS			●		●	●	○	○	○	●	●	○	
BPEL als Petrinetz [Lauer 2006]			●		●		○	○	●	○	○	○	
BPEL4WS / BPEL [IBM et al. 2003]			●		●		○	○	●	○	○	○	
BPEL + EPK [Simon et al. 2006] [Kopp et al. 2006]			●		●		○	○	○	○	○	○	
EPK als Ontologie [ThomasFellmann 2006]			●		●	●	○	○	○	●	●	○	
BPMN & BPML [OMG 2006a] / BPEL4WS			●		●		○	○	○	○	○	○	
CSPL [Chen 1997]						●	○	○	○	○	○	○	
DesignNets [LiuHorowitz 1989]					●		○	○	○	○	○	○	
E3 [Finkelstein et al. 1994] [Jaccheri et al. 1998]						●	○	○	○	○	○	○	
ebXML [OASIS 2001], ebBP / ebXML [OASIS 2006a]			●		●		○	○	○	○	○	○	
Entity Model [HumphreyKellner 1994]					●		○	○	○	○	○	○	
EPK / EPML [MendingNüttgens 2003] [MendingNüttgens 2005]					●		○	○	○	○	○	○	
ESCAPE+ [Neumann et al. 1996] / Merin TIC [Neumann 2002]	●				●	●	○	○	○	○	○	○	
EVPL (VPL [Swenson 1993]) / Serendipity [GrundyHosking 1998]					●		○	○	○	○	○	○	
FUNSOFT [DeitersGruhn 1994]					●		○	○	○	○	○	○	
Grapple [HuffLesser 1988]	●	●					○	○	○	○	○	○	
HFSP [Katayama 1989]			●				○	○	○	○	○	○	
IDEF3 [Mayer et al. 1995]				●		●		○	○	○	○	○	
JIL / Julia [SuttonOsterweil 1997]				●			○	○	○	○	○	○	
Little JIL / Juliette [Cass et al. 2000]				●			○	○	○	○	○	○	
Marvel Strategy Language (MSL) / Marvel [Kaiser et al. 1988][Kaiser et al. 1990]	●	●				●	○	○	○	○	○	○	
MASP/DL / Alf [Benali et al. 1989]	●						○	○	○	○	○	○	
Melmac [DeitersGruhn 1990]					●		○	○	○	○	○	○	
MERLIN/PML [Emmerich et al. 1991], ESCAPE [Junkermann 1995]	●				●		○	○	○	○	○	○	
MVP / MVPL [Finkelstein et al. 1994]		●					○	○	○	○	○	○	
OML/OPF [FiresmithHenderson-Sellers 2002][Firesmith et al. 1998]						●		○	○	○	○	○	
Pate / Oikos [AmbriolaJaccheri 1991][MontangeroAmbriola 1994]	●						○	○	○	○	○	○	
Peace [Finkelstein et al. 1994] / PDL [Inoue et al. 1986]	●		●				○	○	○	○	○	○	
PIF [Lee et al. 1996]				●			○	○	○	○	○	○	
PROGRESS / DYNAMITE [Jäger et al. 1999]					●	●	○	○	○	○	○	○	
PROMENADE (UML) [FranchRibo 1999b]					●	●	○	○	○	○	○	○	
PS-Algol / PWI [Finkelstein et al. 1994]	●						○	○	○	○	○	○	
PWI PML / PADM [Bruynooghe et al. 1994]						●	○	○	○	○	○	○	
RAD [Holt et al. 1983] [Ould, 1995]							○	○	○	○	○	○	
SLANG / SPADE [Bandinelli et al. 1994]					●		○	○	○	○	○	○	
SOCCA [Engels et al. 1994]					●	●	○	○	○	○	○	○	
SPELL [Jaccheri et al. 1992b] / EPOS [Jaccheri et al. 1992]	●				●	●	○	○	○	○	○	○	
UML 2.0 Aktivitäten & BPEL4WS					●	●	○	○	○	○	○	○	
Wf-XML [Swenson et al. 2004]			●		●		○	○	○	○	○	○	
① Regelsbasiertes Modell ② Techniken aus dem Bereich der künstlichen Intelligenz ③ Funktionales Modell ④ Prozedurales Modell ⑤ Zustandsbasiertes Modell ⑥ Objektbasiertes / objektorientiertes Modell ● Objektbasierte Variantenbildung (Vererbung / Polymorphie) ● Vollst. Prozessinstanzierung und Teilprozesspolymorphie	① Anmierbare, ausführbare Prozessmodellierungssprache (Enactable Process Modeling Language). ② Objektorientiertes Gesamtmodell, das alle Artefakte, Interaktionen, Ereignisse etc. enthält. ● bedeutet hier „könnte“. ③ Übergreifendes Meta-Meta-Modell erlaubt Definition, Hinzufügen und Veränderung des Metamodells. ④ Optimierung oder spezifisches Metamodell für Prozesse in der Fertigung vorhanden.												

Legende: ● trifft voll zu, ● trifft weitgehend zu, ○ trifft teilweise zu, ○ trifft kaum zu, ○ trifft nicht zu

Tabelle 8: Prozessmodellierungssprachen und rechnergestützte Animationskonzepte

2.6.2 Prozess-Austauschsprachen

Für die meisten Konzepte existieren mittlerweile Austauschformate [Mendling et al. 2004]. Vorgangskettendiagramme (VKD, [Scheer 1996]) sind abgesehen von der stärkeren Aspektorientierung durch die Swimlanes äquivalent zu den (e)EPK zu betrachten. Weitere Ansätze für Web-Service-Flusssprachen wie XLANG, WSFL, EDOC gingen in anderen Standards auf [GötzLiddle 2003].

Für ein verteiltes System ist der Datenaustausch von Prozessen besonders wichtig. Die folgende Tabelle vergleicht daher existierende Prozess-Austauschsprachen.

In der folgenden Tabelle werden existierende Austauschformate hinsichtlich ihrer Leistungsfähigkeit verglichen. Dabei finden zunächst die Kriterien von [Mendling et al. 2004] Anwendung. Diese werden um Kriterien der Objektorientierung (OO Prozessinstanzen, Prozessvererbung und starke Typisierung) sowie Verteilbarkeit und Eindeutigkeit (Instanzidentität, Weiterleitung von Prozessinstanzen) ergänzt.

Austauschsprache vgl. [Mendling et al. 2004]	BP4WS [IBM et al. 2003]	BPML [OMG 2006a]	BPMN [OMG 2006a]	BPSS / ebXML [OASIS 2001]	EPML [Mendling Nüttgens 2005]	OWL-S [W3C 2004f]	FNML [Weber 2002]	JML Aktivitäten [OMG 2005b]	WS-GDL [W3C 2004g]	WSC1 [W3C 2002a]	WSCL [W3C 2002b]	WSFL [Snell 2001]	XLANG [Thate 2001]	XPDL [WfMC 2005]
Kontrollfluss	●	●	●	●	●	●	●	●	●	●	●	●	●	●
Data Handling	●	●	●	○	○	○	○	●	●	○	○	●	○	●
Rollen	●	●	●	●	○	●	○	●	●	●	○	●	●	●
Ereignisse	●	●	●	○	●	○	○	○	○	○	○	●	●	●
Ausnahmen	●	●	●	●	○	○	○	●	●	●	○	○	●	●
Statistische Daten	○	○	○	○	○	○	○	○	○	○	○	○	○	●
Instanzidentität	●	●	○	○	○	○	○	○	○	●	○	●	●	○
Weiterleitung von Prozessinstanzen	○	○	○	○	○	○	○	○	○	○	○	○	○	○
OO Prozessinstanzen	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Prozessvererbung	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Starke Typisierung (Ontologie)	○	○	○	○	○	●	○	○	○	○	○	○	○	○

Legende: ● trifft zu, ○ trifft teilweise zu, ○ trifft nicht zu

Tabelle 9: Austauschsprachen und deren Fähigkeiten, basierend auf [Mendling et al. 2004]

Während die Abbildung klassischer Elemente von Prozessmodellen meist möglich ist, können objektorientierte Prozesskonzepte in keiner der Modellierungssprachen umgesetzt werden. Einzige die OWL-S erlaubt eine umfangreiche Klassifikation, weist dafür aber fundamentale Schwächen in fast allen anderen Bereichen auf.

Minimalvoraussetzung ist unter anderem die Instanzidentität für Prozessinstanzen, die nur von wenigen Austauschsprachen unterstützt wird [Mendling et al. 2004]. Allerdings sind bei allen Sprachen selbst grundlegende Forderungen eines objektorientierten, verteilten Laufzeit-Prozessmodells für die Produktion nicht erfüllt. Wechselbeziehungen zwischen Geschäfts- und Produktionsprozessen werden nicht berücksichtigt: Produktion behandelt die Transformation von Gütern auf einer niederen Abstraktionsebene, Geschäftsprozesse Markt- und finanzielle Strategien, obwohl eine Integration notwendig wäre: „To be effective, process design, control, and improvement demand the use of modeling methods with scalable and dynamic properties providing seamless links between business and technical process issues.“ [Dalal et al. 2004], S. 85. Es fehlen also ein gemeinsames (Meta-)Modell und integrierte Prozesse.

2.6.3 Petrinetze

Ein Petrinetz [Reisig 1986] $N = (P, T, F, m_0)$ besteht aus zwei disjunkten Mengen P (Places) und T (Transitions) und einer Menge von Flussrelationen $F \subseteq (P \times T) \cup (T \times P)$. Die Anfangsmarkierung m_0 bestimmt, welche $p \in P$ markiert sind. Petrinetze verfügen über einen einfachen aber vollständigen Formalismus und wurden in vielen Bereichen [Murata 1989] bis hin zur (formalisierten) Geschäftsprozessmodellierung verwendet und weiterentwickelt (vgl. beispielsweise [Frenkler 2005][Dalal et al. 2004]). [Heimann et al. 1996] verwenden in DYNAMITE Aufgabennetze für das Prozessmanagement.

Aus Bedingungs-Ereignis-Netzen (B/E-Netze) wurden die Stellen-Transitions-Netze (S/T-Netze)⁷ entwickelt. Es folgten Pr/T-Netze mit unterscheidbaren Tokens und Prädikaten an Stellen und Transitionen, hierarchische Netze, gefärbte und andere High-Level-Petrinetze. **Gefärbte Netze**⁸ (Colored Petri Nets, CPN) [Jensen 1992] können Tokens nach Farben unterschieden, allerdings sind nicht mehrere Instanzen möglich.

Petrinetze werden unter anderem in EPOS [Jacherri et al. 1992][Conradi et al. 1994] und SPADE (Software Process Analysis, Design and Enactment) beziehungsweise der zugehörigen Sprache SLANG [Bandinelli et al. 1993] eingesetzt. Auch die Enterprise Process Modeling Language (EPML, [Chaugule 2001][Dalal et al. 2004]) nutzt mehrere petrinetz-basierte Diagrammtypen, wie Basic Process Model, Task Specification Diagram und Task Execution Diagram darzustellen.

Ein Ansatz, Petrinetze mit Komponenten zu koppeln, wurde unter dem Namen CNet für die Automatisierung in [WurmusWagner 2000] entwickelt. Die Verhaltensbeschreibung der CNet-Komponenten erfolgt grafisch durch die Petri-Netz-Klasse PNet (vgl. [Jensen 1992]). PNet verwendet informationstragende Marken, um zum Beispiel verschiedene Werkstücksorten in einem Fertigungsprozess kompakt zu modellieren. Ein ähnliches Modell führte zum IEC Standard 61499 [IEC 2005]. Dieses wird in [HaggeWagner 2005] mit CNet verglichen.

Die Modularisierung wird teilweise sogar durch objektorientierte Konzepte ergänzt, wodurch Transport und Instanziierung von Netz-Komponenten ermöglicht werden [Esser 1996] [JanneckNaegele 1999]. Ansätze zur Integration von Objektorientierung und Petrinetzen können je nach Integrationsrichtung wie folgt kategorisiert werden [Bastide 1995][ZapfHeinzl 1998]:

- Ansätze zur Einbettung von Objekten in Petri-Netze,
- Ansätze zur Einbettung von Petri-Netzen in Objekte und
- Ansätze zur beidseitigen Integration von Objekten und Petri-Netzen

Die Verwendung von **Petrinetzobjekten als Marken** durch ein Systemnetz, das Objektnetze transportiert, oder zur Modularisierung von Petrinetzen [Philippi 1999a] beziehungsweise zur Beschreibung objektorientierter Zusammenhänge mit Hilfe von Petrinetzen [Philippi 1999b] führt zu Objekt-Petrinetzen (Object Petri Nets, OPN, [Valk 1998][Lakos 1994]) mit einem sehr komplexen Formalismus [Lakos 1995]. Diese sind eine besondere Form von High-Level-Netzen, transportieren als komplexe Marken oder Objekte ganze Graphen, verfügen jedoch nicht über alle Eigenschaften der Objektorientierung. Ein Objekt-Petrinetz besteht aus einem Systemnetz und mehreren Objektnetzen. Objekt-Petrinetze werden verwendet, um sowohl das System mit seinen Ressourcen als auch Aufgaben des Systems zu modellieren. Systemnetze modellieren

⁷ Ein S/T-Netz wird als 6-Tupel $N=(S,T,F,K,E,M)$ beschreiben, das Stellen (S), Transitionen (T), Kanten (F), Stellenkapazitäten (K), Kantenausdrucksfunktionen (E) für die Definition der transportierten Marken und eine Initialisierungsfunktion M enthält, die den Stellen die Startmenge von Marken zuordnet (vgl. beispielsweise [Martens 2002]).

⁸ Die Einführung von gefärbten Netzen [Jensen 1997], ihre Definition [Jensen 1994], ihre Nutzung und CPN Modellierungssprache [Jensen 1998] sowie praktische Anwendung [Kristensen et al. 1998] wird in der existierenden Literatur bereits detailliert beschrieben.

beispielsweise eine Maschinenstrecke und die Objektnetze die Werkstücke, die in der Maschinenstrecke unterschiedlich bearbeitet werden. [Martens 2002]

Werden Petrinetze dagegen objektorientiert aufgebaut, entstehen **Objektorientierte Gefärbte Petrinetze** (Object-Oriented Coloured Petri Nets – OOCPN), ein Gesamtsystem aus nach bestimmten Kriterien aufgebauten Objekt-Netzen. Dabei werden Methoden gekapselt und Nachrichten zwischen Objekten ausgetauscht. Über eine Object ID prüft das Netz, ob die Nachricht für das vorliegende Objektnetz bestimmt ist. Jede Methode eines OOCPN entspricht einer Transition oder einer entsprechenden Verfeinerung, die Zugriff auf vollständig gekapselte objektinterne Daten hat. [Moldt 1996]

Für eine Formalisierung der Ablaufsemantik müssen gefärbte Petrinetze (Colored Petrinets, CPN) eingesetzt werden. Die Marken können typisiert werden, indem jede Färbung einem Typ entspricht. Es wird mit der Funktion C jeder Stelle ein Typ aus der Typmenge Σ zugeordnet.

Allerdings müssen in einem vollständig objektorientierten Gesamtmodell auch Instanzen von Subtypen zur Laufzeit polymorph eingesetzt werden können. Die Färbung der Stelle ist damit nicht ausreichend. Vielmehr müssen zusätzlich die Marken gefärbt werden und eine Transition nicht nur bei Äquivalenz des Markentyps möglich sein, sondern auch wenn ein kompatibler Subtyp vorliegt. Die Mehrfachvererbung erschwert die Formalisierung dabei weiter und passt nicht in das Konzept der High-Level Petrinetze, wengleich die Graphbasierung sinnvoll ist.

[Wurmus 2002] kritisiert jedoch die hohe Modellierungskomplexität und den Fokus auf die Generierung von Anwendungen. Für eine vollständig objektorientierte Laufzeitumgebung reichen diese Konzepte auch mangels Typisierung nicht aus.

[Moldt 1996] bildet für OOCPNs zunächst Objekte, Klassen und Nachrichten ab und führt dann die Vererbung ein. Die Semantik muss entsprechend in das Petrinetz-Konzept eingepasst werden. So wird beispielsweise ein Objekt durch eine New-Nachricht an die gewünschte Klasse erzeugt. Ziel ist eine Nachbildung der programmiersprachlichen Vererbung und nicht die Umsetzung einer Objektorientierung mit Prozessvererbung für die Animation von Prozessen mit Hilfe von objektorientiert typisierten Marken. Die (Mehrfach-)Vererbung der Prozesse selbst lässt sich selbst mit Objekt-Petrinetzen nicht sinnvoll formalisieren.

Mehrere der High-Level-Ansätze stammen auch aus dem Geschäftsprozess- und Workflowmanagement. Mit der Petri Net Markup Language (PNML) [Weber 2002][Billington 2003] steht auch eine Markup-Sprache für die Übertragung von Petrinetzen sowie mit dem Petrinetz-Kern (PNK) [Weber 2002] eine Systemumgebung zur Verfügung.

Während komplexe Zustände, Abläufe und Automatisierungen gut beschrieben und teilweise um Objekt-Funktionalitäten ergänzt werden, zeigen auch die „objektorientierten“ Petrinetz-Ansätze klare Schwachpunkte in objektorientierten Bereichen wie Polymorphie, Vererbung und Varianten. Die Stärke der Petrinetze in einer überprüf- und automatisierbaren Ablaufsemantik ist gleichzeitig die Schwäche im Hinblick auf Flexibilität im Sinne der Objektorientierung und Varianten-Produktion.

Die in der folgenden Tabelle verwendeten Kriterien lehnen sich an [ZapfHeinzl 1998] an und werden um dort implizit verwendete Kriterien und zusätzlich um das Kriterium Automatisierung / Produktion erweitert.

Es existieren Beispiele für jeden Netztyp ① bis ⑧. Abgesehen von der Konzeption für oder Abdeckung von Produktionsaspekten erfüllen die Ansätze CO-OPN/2, COOs, LOOPN++, OOPN und PN-TOX PN grundlegende Kriterien der Objektorientierung für eine petrinetzbasierte Ablaufsprache. Allerdings lässt sich keiner der meist sehr theorielastigen Ansätze auf die Aufgabenstellung im Bereich der Produktion vollständig anpassen oder übertragen. Das für die Automatisierung konzipierte CNet hat dagegen Schwächen in fast allen anderen Bereichen.

Ansatz	OO-Notation	Kriterien						Netztyp basiert auf / transform. in									
		①	②	③	④	⑤	⑥	⑦	⑧	⑨	⑩	⑪	⑫	⑬	⑭		
Petrinetze [Petri 1962][DeselOberweis 1996]	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○
Stellen-Transitions-Netze (S/T-Netze) [Reisig 1986]	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Prädikaten-Transitions-Netze (Pr/T) [Genrich 1987]	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Gefärbte Netze (Colored Petri Nets, CPN) [Jensen 1992]	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○
CNet [Wurmus 2002]	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Referenznetze (RN) [Kummer 2002]	Java	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Protob-Netze [Bruno 1994]	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Agentennetze [Rölke 1999]	eigene	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Objekt-Petrinetze (OPN) [Lakos 1994] [Valk 1998]	eigene	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
LOOPN Language for Object-Oriented Petri Nets [LakosKeen 1991]	eigene	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Macronet [Keller et al. 1994]	ER, eigene	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
MOBY Modellierung von Bürosystemen [FleischhackLichtblau 1993]	Smalltalk	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
NetCASE Petri Net Based CASE [Dahr et al. 1994]	OMT	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
OBJSA Nets [Battiston et al. 1988]	OBJ2	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Object-Oriented Coloured Petri Nets (OOCPN) [Englisch 1993] [Moldt 1996]	EIFFEL	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Objekt-Prozess-Modell (OPM) [Burkhardt 1994]	UML	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
SimCon [Verkoulen 1993][ZapfHeinzl 2000]	eigene	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
THORN Timed Hierarchical Object-Related Nets [Schöf et al. 1995]	C++	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
CO-OPN/2 Concurrent Object-Oriented Petri Nets [BiBu94; BiBG96]	eigene	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
COOs Cooperative Objects [BaPa93; Sibe93; Sibe94]	eigene	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
HOON Higher-Order Object Nets [LöWH95]	OMT / ER	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
LOOPN++ Language for Object Petri Nets [Lako94; LaKe94a; LaKe94b; Lako97]	Eigene	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
OOPN Object-Oriented Petri Nets [CeJa96; CeJa97]	Smalltalk, eigene	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
OOPNL Object-Oriented Petri Net Language [Esse97]	eigene	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
OOPrT Nets Object-Oriented Pr/T-Nets [Phil97]	OMT	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
PN-TOX PN Tools for Object Concurrency Specification [HolvoetVerbaeten 1995]	eigene	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○

<ul style="list-style-type: none"> ① Objektorientiertes Vorgehensmodell ② Werkzeug für die Sprache vorhanden ③ Marken im Netz sind unterscheidbar ④ Objekte in Petrinetzen möglich ⑤ zusätzlich Petrinetze in Objekten möglich ⑥ Automatisierung / Produktion abgedeckt 	<ul style="list-style-type: none"> ① Petrinetzbasiert ② S/T-Netzbasiert ③ Pr/T-Netzbasiert ④ CPN-basiert ⑤ OPN-basiert ⑥ OOCPN-basiert 	<ul style="list-style-type: none"> ⑦ Referenznetzbasiert ⑧ Eigenes Konzept <p>● Trifft zu, ○ trifft nicht zu</p> <p>Ansätze zur Verwendung von Objekten in Petrinetzen unter anderem nach [ZapfHeinzl 1998].</p>
---	--	--

Tabelle 10: Petrinetz-basierte Ansätze zur Verwendung von Objekten

2.7 Ontologien und Metamodellierung

Ein gegenteiliges Konzept stellen Ontologien und Metamodellierung (vgl. Anhang A) dar, die Kernprinzipien der Objektorientierung wie Vererbung oder Klassifikation einsetzen.

Die (Computer-)Linguistik liefert für die Vererbungsbeziehung auf Begriffsebene eine semantische Definition (vgl. [Lyons 1995]). Ein Unterbegriff (Hyponym) steht mit seinem Oberbegriff (Hyperonym) in einer Vererbungsbeziehung. Der Begriffsumfang des Hyponyms ist dabei kleiner als der des Hyperonyms, das heißt weniger Begriffe fallen in die Klasse der Spezialisierung, verglichen mit dem allgemeineren Hyperonym. Dagegen ist der Begriffsinhalt (Intension) des Hyponyms größer, weil durch die Spezialisierung weitere semantische Merkmale hinzukommen. Die Präzisierung eines Objekts als Hyponym A impliziert dabei die Präzisierung des Objekts als Hyperonym B – jedoch nicht umgekehrt (Asymmetrie). [Lyons 1995]. Im objektorientierten Terminus ist also ein Objekt der Subklasse (hier des Hyponyms) automatisch Objekt der Superklasse (hier des Hyperonyms). Auf diesem Sprachmechanismus fußt die Polymorphie. Aus der Tatsache, dass ein Begriff Hyponym mehrerer Hyperonyme erklärt sich die Mehrfachvererbung, die teilweise erst eine korrekte semantische Abbildung komplexer Produktionssysteme ermöglicht.

IT-gestützte, formalisierte Ontologien ergeben semantische Modelle, die für eine rechnergestützte Modellierung sowie vor allem Modellinterpretation und -ausführung benötigt werden.

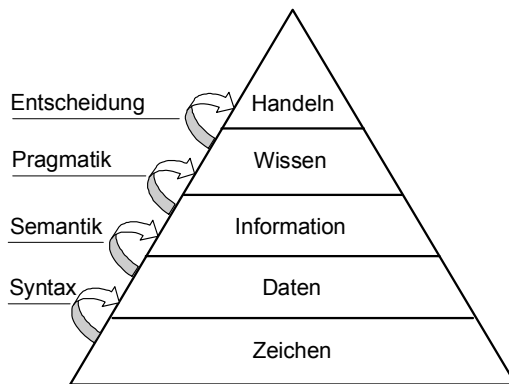


Abbildung 6: „Wissenspyramide“ in Anlehnung an [Müller 2007] und [Wolf et al. 1999]

Gewöhnlich enthalten Komponentenmodelle nur Komponenten und eine Verbindungs- (beispielsweise in TopicMaps [Rath 2003]) oder Aggregationsfunktion (beispielsweise in MetaChart, [Janssen et al. 2003]), die es ermöglicht Komponenten zu verbinden und – im Falle der Aggregation auch hierarchisch zur strukturieren.

In der IT werden Ontologien [Gomez-Perez et al. 2004] für die Repräsentation semantischer Informationen verwendet. Sie bestehen meist aus nicht weiter zerlegbaren Konzepten (Begriffen, meist Darstellung als Ellipse) aufgebaut, die über Relationen (Darstellung durch Pfeile oder Kanten) verbunden sind. Sind unterschiedliche Konzepte über eine gleichnamige Relation verbunden (beispielsweise *is_a*), können entsprechende – auch transitive – Schlussfolgerungen getroffen werden. Auf diese Weise entstehen auch grundlegende Relationstypen, die jedoch nicht wie Konzepte über eine eigene Typhierarchie verfügen.

Eine Problematik der Ontologie liegt bereits in ihrer weiten Definition. Bereits die gesonderte Modellierung von Entitäten (Substantiven) und Aktivitäten (Verben) wird als Ontologie interpretiert [Caetano et al. 2005]. Es gibt unterschiedliche Beispiele für eine praktische

Anwendung [Gomez-Perez et al. 2004], wobei Ontologien als Metamodelle häufig einen eingeschränkten Geltungsbereich haben [Terrasse et al. 2006].

Die Einführung und Umsetzung von Regelmodellen wie SWRL [W3C 2004e] ist noch wenig fortgeschritten und nur in proprietären Regel- oder Reasoner-Ansätzen wie in Jena⁹ oder KAON2¹⁰ verfügbar.

Wie sich ihm Rahmen von [Wagner 2007] zeigt, können reine Ontologien nur eingeschränkt zur Prozessmodellierung und -steuerung eingesetzt werden. Ihre Ausrichtung auf Klassifikationen und Zusammenhänge macht eine Animation schwierig, die zudem der Semantik widerspricht.

2.7.1 Ontologie-Repräsentationen

In der derzeitigen Praxis werden meist Repräsentationen eingesetzt, die auf W3C-Standards wie XML basieren.

Das Resource Description Framework (RDF) [W3C 2004a] [W3C 2004c] ermöglicht eine Repräsentation von Ontologien, ergänzt um RDF Schema (RDFS) [W3C 2004b], das weitere Beschreibungskonzepte zur Verfügung stellt. RDF wird auch als Basis des Semantic Web bezeichnet [Grau 2004]

Für RDFS wurde mit der RDF-Schema Fixed Layer Metamodeling Architecture (RDFS(FA), [PanHorroks 2003]) eine Semantik und Metamodellierungsarchitektur entwickelt, die RDF(S) und OWL DL (siehe unten) integrieren soll [PanHorroks 2007].

Während bei der DARPA Agent Markup Language (DAML) mehr die Agentenklassifikation und -beschreibung mit dem Profil DAML-S im Vordergrund stand, war Ontology Inference Layer (OIL) direkt auf die Formulierung von Ontologien ausgelegt, DAML jedoch erst mit der speziellen Version DAML-ONT. Große Ähnlichkeiten der Sprachen führten dann zu einer Verschmelzung von DAML und OIL unter dem Namen DAML+OIL (letzter Language Release 2001).

OIL ist kompatibel mit RDF-Schema, besitzt jedoch keine XML-Konformität. Es besteht aus vier aufeinander aufbauenden Stufen: Die Schnittmenge¹¹ des Standard OIL mit RDF-Schema heißt **Core OIL**. **Standard OIL** enthält alle Modellkonstrukte und ermöglicht Inferenz, **Instance OIL** verfügt über Instanzen in RDF und ist datenbanktauglich. **Heavy OIL** sollte zusätzliche Darstellungs- und Reasoning-Möglichkeiten besitzen, wurde jedoch nie spezifiziert

Die W3C Web Ontology Working Group führte auf der Basis von **DAML+OIL** die **Web Ontology Language OWL** [W3C 2004d][Patel-Schneider et al. 2004] ein. Der definierte Anwendungsbereich der OWL ist die Nutzung durch Maschinen, das heißt die Automatisierung, um deren Interpretation von Web-Content zu vereinfachen. OWL-Spezifikationen nutzen ebenfalls RDF zur Repräsentation. OWL-S [W3C 2004f] ermöglicht darüber hinaus die Beschreibung von Web-Services mit Hilfe einer OWL-basierten Ontologie.

Die OWL unterteilt sich dabei in drei (Unter-)Sprachen ähnlich der OIL-Klassifikation: OWL Lite, OWL DL und OWL Full. Die Beziehung zwischen OWL-Lite und OWL-DL ist ähnlich der zwischen den OIL-Dialekten: OWL-DL basiert auf dem gleichen semantischen Formalismus, besitzt aber eine höhere Ausdrucksmächtigkeit. Die Beziehung zwischen OWL-DL und OWL-Full ist wesentlich komplexer und unklar. [Grau 2004]

Die Problematik von OWL wird folgendermaßen charakterisiert [Grau 2004]: Während OWL-Lite entscheidbar und gut automatisierbar ist begrenzen viele Einschränkungen die

⁹ <http://jena.sourceforge.net/>

¹⁰ <http://kaon2.semanticweb.org/>

¹¹ mit Ausnahme der Konkretisierung („reification“), Interpretation durch einfache RDF-Schema-Agenten möglich

Ausdrucksmächtigkeit. Obwohl OWL-DL entscheidbar ist, gilt: „*The entailment problem is computationally intractable in the worst-case.*” [Grau 2004]. OWL und RDFS basieren zudem auf unterschiedlichen Formalismen, was die praktische Anwendung weiter erschwert. OWL Full kommt für eine automatisierte Anwendung in der Produktion trotz ausreichender Mächtigkeit nicht in Frage: „*OWL Full is too expressive (namely undecidable).*” [Grau 2004]

Die Ausrichtung auf (semantische) Web-Inhalte schafft viele weitere Nachteile, die einem Einsatz für die Prozessbeschreibung in der Produktion entgegenstehen. Als Austauschformat für Klassifikationen wird OWL jedoch eine zunehmende Bedeutung zu kommen.

Um ontologiebasiert Unternehmen zu modellieren, existieren bereits Ansätze wie die Unified Enterprise Modelling Language (UEML) [OpdahlBerio 2006]: „*The common ontology is organised hierarchically. Ontology classes are organised using class-subclass (ProductionEquipment is-subclass-of Equipment) and part-whole (Equipment iscomposed-of EquipmentParts) relationships.*”

2.7.2 Objektorientierung

Während Ontologien bereits Teilkonzepte der Objektorientierung einsetzen, ist für flexible, objektorientierte Prozesse in der Produktion eine weitergehende Übertragung und Anwendung notwendig.

Der Begriff **objektorientiert** wird durch die [ISO 1998] definiert als: „*Pertaining to a technique or a programming language that supports objects, classes, and inheritance.*”

Charakteristisch ist das Konzept der Klasse zur integrierten Kapselung von Daten und Abläufen in der Objektorientierung: „*In an object-oriented approach, the association between the state of an object and it's set of behaviors is made explicit via encapsulation whereby both are defined in an integral, self-contained unit called a class.*“ [HealyKilgore 1997]

Die grundlegende Definition der **objektorientierten Sprachen** durch [Wegner 1987] ist weitgehend anerkannt und unterscheidet für die Definition von Objekt-Sprachen drei Klassen von Sprachen, denen [Bergin 1997] eine vierte hinzufügt:

Objektbasierte Sprachen (B) [Wegner 1987]	enthalten Objekte als Konzept. Sie können also nicht nur mit atomaren Einheiten, sondern auch aggregierten „Komponenten“ umgehen.
Klassenbasierte Sprachen (K) [Wegner 1987]	sind eine Teilmenge der Objektbasierten Sprachen. Neben den oben genannten Kennzeichen lässt sich hier jedes Objekt einer Klasse zuordnen und verfügt somit über einen Typ.
Vererbungsaktivierte (V) Sprachen [Bergin 1997]	sind eine Teilmenge der klassenbasierten Sprachen. Sie ermöglichen zusätzlich eine Vererbung von Klassen und somit die Bildung von Subtypen und Hierarchien. Teilweises Instanziierungskonzept.
Objektorientierte (O) Sprachen [Wegner 1987]	setzen das Konzept der Instanzierung von Objekten vollständig um, also auch Polymorphie und virtuelle Methoden. [Bergin 1997] $O \subset K \subset V \subset B$

Tabelle 11: Kategorien von Sprachen mit Objekten

Eine formale Theorie zur Objektorientierung in Programmiersprachen liefern [AbadiCardelli 1996]. Für Prozesse gestaltet sich die Objektorientierung durch die hierarchische Aggregation von Prozessschritten und anderen Modellelementen sowie das Überschreiben von Teilen allerdings anders, so dass die Theorie nicht direkt auf Prozesse abzubilden ist.

Bis auf wenige Ausnahmen [Kay 2003] wird Vererbung auch heute als das grundlegende Konzept der Objektorientierung angesehen [Armstrong 2006]. Zur objektorientierten Modellierung von Prozessen müssen jedoch alle Konzepte umgesetzt werden. Die folgende Tabelle verwendet daher die von [Armstrong 2006] eingeführten Konzepte bzw. Kriterien für objektorientierte Ansätze.

Objektorientiertes Konzept	Ansatz								
	① OOP Basis	② OOP heute	③ OOD	④ OOM UML+	⑤ Ver- wndg.	① WfMS	② OO WfMS	③ Relevanz OO-Proz.	
Vererbung	●	●	●	●	81 %	○	○	●	
Objekt	●	●	●	●	78 %	○	○	●	
Klasse / Typ	●	●	●	●	71 %	○	●	●	
Kapselung / Information Hiding (32 %)	●	●	●	○	63 %	○	○	○	
Methoden / Operationen	●	●	●	●	57 %	○	○	○	
Nachrichten	●	●	●	●	56 %	○	○	○	
Polymorphie	○	○	●	○	53 %	○	○	●	
(Daten) Abstraktion (ADT 8%)	●	○	●	●	51 %	○	●	●	
Instanziierung (i.S.d. OO)	●	●	●	●	35 %	○	○	●	
Attribute	●	○	○	●	33 %	○	●	●	
Polymorphe Methoden, Dynam. Bindg.	○	●	○	○	15 %	○	○	○	
Spezialisierung / Hierarchie	●	●	○	○	10 %	○	○	●	
Aggregation	○	○	○	○	5 %	○	○	●	
Graphbasierung	○	○ (DAG)	○	●	-	● / ○	● / ○	○	
Reflection	○	○	○	● (MOF)	-	○	○	○	

① [Wegner 1987] Kernkonzepte objektorientierter Programmiersprachen, Basisdefinition
 ② [Bergin 1997] Kernkonzepte der objektorientierten Programmierung (OOP)
 ③ [Armstrong 2006] Kernkonzepte („quarks“) der objektorientierten Entwicklung (OOD)
 ④ [AtkinsonKühne 2002] Kernkonzepte der objektorientierten Modellierung (UML + Verbesse- rungen)
 ⑤ Häufigkeit der Verwendung in 239 publizierten OOD-Methoden nach [Armstrong 2006]
 ① Erfüllung durch Workflow-Modellierung und ausführbare Prozessmodellierungsansätze
 ② Erfüllung durch Workflow-Modellierung und ausführbare Prozessmodellierungsansätze mit objektorientierten Teilkonzepten (vgl. Kap. 7 Prozessmodellierung). Einen Vergleich von drei aktuellen Workflow-Systemen bietet [PesciVan der Aalst 2006].
 ③ Relevanz des Konzepts für objektorientierte Prozesse (Varianten, Flexibilität, Hierarchie etc.) in der Produktion
 Legende: ● trifft voll zu, ○ trifft teilweise zu, ○ trifft nicht zu

Tabelle 12: Konzepte der Objektorientierung

Die Tabelle zeigt, dass sowohl in der Definition als auch in der tatsächlichen Verwendung die Konzepte Vererbung, Objekt, Klasse / Typ, Operationen, Nachrichten und Instanziierung anerkannt sind. Diese müssen also übernommen werden. Auch Kapselung und Abstraktion sind wichtige Konzepte. Während die Definition Polymorphie eher vernachlässigt, zeigen die praktischen Umsetzungen eine hohe Verwendungsrates in ⑤. Hieraus kann geschlossen werden, dass die tatsächliche Umsetzung meist eine Verwendung erfordert, deren Wichtigkeit jedoch in der Theoriebildung nur teilweise erkannt wurde. Die Umsetzung im Bereich der WfMS und selbst der OO-WfMS ist im Vergleich zur Relevanz der Konzepte ③ gering.

Die Objektorientierung wird auch bereits zur Klassifikation von Software und Hardware der Produktion gesetzt (beispielsweise OONEIDA [Vyatkin et al. 2005]), so dass mit geeigneten objektorientierten Prozessmodellierungsmethoden auch eine konzeptuelle Integration erfolgen könnte.

Oft werden einzelne Konzepte der Objektorientierung herausgegriffen und dann – gewissermaßen – ohne Berücksichtigung der Semantik – durch eine behelfsmäßige „Verpackung“ zugänglich gemacht: „Der Grundgedanke ist hier, das objektorientierte Konzept der Wiederverwendung zu nutzen und es im Wesentlichen durch "Spezifikation" neu zu verpacken.“ [GötzLiddle 2003]

Eine integrierte Methode, die konsistent die in Spalte ③ aufgeführten Konzepte für ein Prozess-System in der Produktion umsetzt fehlt daher.

Sowohl bei den Programmiersprachen als auch bei den Modellierungssystemen in der Softwareentwicklung hat sich ein Paradigmenwechsel hin zur Objektorientierung vollzogen. Dieser ist aufgrund der optimalen Eigenschaften objektorientierter Modelle im Hinblick auf Klassifikationen, Varianten und Entitäten teils auch auf die Prozessmodellierung übergegangen. Zunächst in Prozessen der Softwareentwicklung verwendet und für diese weiterentwickelt, existieren heute unter anderem für Geschäftsprozesse auch Modellierungssysteme, die Implikationen des objektorientierten Paradigmas realisieren.

Die UML ist heute noch immer eine der wenigen allgemeinen, weitgehend objektorientierten Modellierungssprachen, die zur Prozessmodellierung verwendet werden können (siehe Anhang D), auch wenn sie wegen der semi-informalen Semantik häufig kritisiert wird [VarróPataricza 2002][Patel-Schneider et al. 2004][Thomas 2003].

Zudem impliziert die reine Objektorientierung auch das Konzept der (Prozess-) **Meta-Modellierung**: Ein Meta-Modell gibt die Regeln, Begrifflichkeiten/Typen und Strukturen vor, die zur Prozessmodellierung eingesetzt werden. Modelle beziehungsweise Prozesse werden dann durch Instanziierung der Meta-Modelle erzeugt und ihrerseits zu Prozessinstanzen weiterinstanziiert.

Prozess-Metamodelle stellen die Typebene von modellierten Prozessen (Prozessmodellen) dar. Tatsächlich ausgeführte Prozessinstanzen befinden sich auf der Instanzebene. So ergibt sich eine (Instanziierungs-)Hierarchie von Modellen:

1. Meta-Metamodell
2. Prozess-Metamodell
3. Prozess(modell)
4. Prozessinstanz / Projekt

Ein Meta-Metamodell kann Regeln, Strukturen und Beschreibungsmechanismen für alle Prozess-Metamodelle definieren.

Ein konsistentes Metamodellierungskonzept, das über eine proprietäre Vorgabe von Modellierungselementen und Regeln hinausgeht, lassen jedoch die meisten Ansätze vermissen. Gerade für Prozesse fehlen Metamodelle, die flexibel an unterschiedliche Domänen angepasst werden können und so zu domänenspezifischen Modellen führen. Fehlende Metamodellsemantik führt daher meist zu einer direkten Implementierung und damit Fixierung von Regeln in Software, die die Modellierungssoftware auf ein Anwendungsgebiet oder -problem beschränken.

2.8 Transformative und generative Modellierungsverfahren

Ebenso wie die Softwareentwicklung und Entwicklungsprozesse benötigen Prozessmodelle im Produktionsumfeld Konzepte zur modellbasierten Erzeugung von Systemen. Aus diesem Grund sollen diese Ansätze aus der Softwaretechnik im Folgenden näher untersucht werden.

Generative Programmierung [CzarneckiEisenecker 2000] ermöglicht die Erzeugung von Systemen durch Generierung aus Modellen und führt den Ansatz der Programmiersprachen der vierten Generation (Fourth Generation Languages, 4GL) fort: Aus vordefinierten Komponenten werden hier komplexe Systeme erzeugt. (vgl. auch [StahlVölter 2005], [Müller 2007]).

Model Integrated Computing (MIC) [Sprinkle 2004] hat einen erweiterten Fokus und betrachtet den gesamten Software-Produktlebenszyklus. Bei der domänenspezifischen Modellierung wird viel Wert auf Simulation und Verifikation gelegt – erklärbar aus der ursprünglichen Konzeption für die Entwicklung von verteilten Echtzeitsystemen. Derartig umfassende, richtlinienorientierte Ansätze erlauben eine Transformation in andere Gebiete wie die Produktion nur rudimentär, da Domänenspezifika nicht übertragen werden können und ein Basismodell unnötig kompliziert machen.

Bei **Software Factories** [GreenfieldShort 2004] stand ursprünglich die Konfiguration neuer Softwaresysteme aus Komponenten in Form einer „Massenproduktion“ im Vordergrund. Wie in der Produktion auch hat sich hier der Fokus auf eine Variantenfertigung beziehungsweise Build-to-Order [Anderson 2003] verschoben [Müller 2007]. Während das Konzept also den neuen Anforderungen an die Softwareentwicklung besser gerecht wird, kann der Transfer in die Domäne der Produktionsprozesse nur bedingt erfolgen. Grundvoraussetzung für eine Anwendung zur Laufzeit ist, dass Prozessänderungen automatisierte Änderungen nach sich ziehen, also die Anpassung ohne weitere Konfiguration und Einwirkung durch Entwickler stattfindet.

Model Driven Architecture (MDA) [OMG 2001][KempaMann 2005] wurde schnell als spezielle Form der modellgetriebenen Softwareentwicklung **Model-Driven Software Development (MDS)** [StahlVölter 2005] bekannt und in Projekten angewandt, beispielsweise MDA im Enterprise Computing [Frankel 2003]. Bei MDA (vgl. Anhang M) beschreibt ein Computation Independent Model (CIM) auf fachlicher Ebene die Systemfunktionalität, unabhängig von Struktur und informationstechnischer Umsetzung, das Platform Independent Model (PIM) die Funktionalität einer Komponente unabhängig von der Zielplattform, jedoch bereits mit einer definierten Semantik, die eine spätere Umsetzung ermöglicht. Während dabei noch von plattformspezifischen Konzepten wie J2EE abstrahiert wird, realisiert Platform Specific Model (PSM) ein PIM unter Nutzung der durch die Zielplattform zur Verfügung gestellten Schnittstellen und Konstrukte. Es kann dabei mehrere kaskadierende Transformationsebenen geben, die erst als letzte Stufe den ausführbaren Code enthalten. [StahlVölter 2005] Die vorgesehene Transformation zwischen PIM und PSM folgt Transformationsregeln, die nicht zwangsläufig automatisierbar sein müssen, jedoch im „Record of Transformation“ als Ablaufprotokoll aufgezeichnet werden [KempaMann 2005].

	① OO	② Poly	③ MB	④ Gen	⑤ GenAb	⑥ ProzM	⑦ Prod	⑧ LZG	⑨ LZV	⑩ Interakt
Generative Programmierung [CzarneckiEisenecker 2000]	●	○	●	●	○	○	○	○	○	●
Model Integrated Computing (MIC) [Sprinkle 2004]	●	○	●	●	○	○	●	○	○	○
Software Factories [GreenfieldShort 2004]	●	○	●	●	○	○	○	○	○	●
Model Driven Architecture (MDA) [OMG 2001]	●	○	●	●	●	●/○	○	○	○	●
Model Driven Software Development (MDS), [StahlVölter 2005]	●/●	○	●	●	●	●/○	○	●	○	●
User Interface Interpreter wie ITS ¹⁾	●/○	○	○	●	●	●/○	○	●	●	●
UIMS-Generatoren wie FUSE ¹⁾	●	○	●	●	●	●/○	○	●	○	●
Quellcode-Generatoren wie JANUS ¹⁾	●	○	●	●	●	●/○	○	○	○	●
¹⁾ Analyse der Ansätze siehe [Schlegel 2002] ① OO: Unterstützung für objektorientierte Modelle vorhanden ② Poly: Prozess-Polymorphie möglich ③ MB: Modellbasierung ④ Gen: Generierung von Artefakten, Code oder Interaktionen ⑤ GenAb: Generierung aus ablauforientierten Modellen möglich					⑥ ProzM: Prozessmodelle als Basis für die Generierung ⑦ Prod: Produktionsspezifische Modelle verfügbar ⑧ LZG: Laufzeitgenerierung ⑨ LZV: Laufzeitänderung des Prozessmodells möglich ⑩ Interakt: Interaktionen können aus einem Modell direkt generiert werden					

Legende: ● trifft voll zu, ● trifft teilweise zu, ○ trifft nicht zu

Tabelle 13: Generative Ansätze – Tauglichkeit für die Animation objektorientierter Prozessmodelle

Die untersuchten generativen oder transformativen Ansätze zeigen teilweise eine Ausrichtung auf objektorientierte Modellierung und ermöglichen eine Generierung von Abläufen. Allerdings sind Konzepte der Polymorphie und der Laufzeitveränderung nicht in ausreichendem Maße verfügbar. Auch wenn ablauforientierte Modelle generiert werden, erfolgt die Erzeugung nicht zur Laufzeit.

Generative Ansätze im Interaktionsbereich werden in [Schlegel 2002] ausführlich beschrieben und bewertet. Bereits hier zeigte sich, dass ein konsistentes, typologisches Metamodell zusammen mit Ablauforientierung in keinem der Ansätze vorliegt. Meist werden ein oder mehrere spezifische Modelltypen entworfen, auf deren Basis ein Generator entwickelt wird. Dieser ist vollständig an das Modellierungskonzept gebunden und kann keine Spezialisierungen von Schritten verarbeiten oder zur Laufzeit Änderungen an den Abläufen ermöglichen.

Reinen Modellierungsansätzen wie UIML 3.1 [OASIS 2004], das auch Mehrgeräte-Schnittstellen (Multi Device User Interfaces) mit UIML erlaubt [Luyten et al. 2005] oder daten- und layoutorientierten Modellierungsansätzen wie WebML¹² [Ceri et al. 2000] fehlen aufgrund der Ausrichtung auf die Interaktion Konzepte der Prozessmodellierung und -animation. So ist es zwar möglich für Leitstände portable und skalierbare, auf Scalable Vector Graphics (SVG) [W3C 2003] basierende Benutzungsschnittstellen [Li et al. 2006][Li 2006] zu erzeugen. Für die Prozessschicht müssen jedoch andere Modelle verwendet werden.

Trotzdem liefern diese Ansätze zum Teil eine gute Grundlage für die Entwicklung eines Laufzeitgenerators für interaktive Prozesselemente. Die Modellierungskonzepte berücksichtigen dabei teilweise die geforderte Generierung von passenden Schnittstellen ohne Einwirkung von Entwicklern mit Hilfe von Layoutregeln und Element-Transformationen. Daher existiert meist nur der sinnvollere, umgekehrte Weg einer Generierung von Interaktionsmodellen (wie WebML) aus Prozessmodellen [Brambilla 2006].

Die vorgestellten Konzepte wurden in der Hauptsache für die Softwareentwicklung konzipiert. Eine Anwendung zur Laufzeit ist meist aus Mangel an Manipulationsmöglichkeiten und Automatisierung nicht möglich.

Für dynamisch veränderbare Prozesse müssen jedoch Abläufe und Interaktionen rein modellabhängig erzeugt werden, womit die aufwändigen mehrschrittigen und alle interaktiven Generierungs- und Transformationsverfahren ausgeschlossen sind.

2.9 Integration von Objekt- und Ablauforientierung

Unter der Integration von Objekt- und Prozessorientierung wird in der Literatur meist entweder die Verwendung von ablauforientierten Techniken wie EPK für die Entwicklung objektorientierter Software (beispielsweise [LoosFettke 2001]) oder die Beschreibung von Prozessen mit eigentlich für die objektorientierte Modellierung entwickelten Methoden verstanden. Beide Ansätze können dabei Objekt- und Ablauforientierung nur bedingt integrieren, weil das Gesamtmodell nicht über eine vollständig objektorientierte Semantik verfügt.

Die integrierte Modellüberführung beispielsweise vom idealen zum realen Objektmodell aber auch die Integration beider Sichten beispielsweise von ablauforientierten Anwendungsfällen (Use-Cases) und Objektmodellen müssen durch ein integratives Modell erst noch unterstützt und verbessert werden. Wo bestehende Modelle nicht ausreichen muss es möglich sein neue und erweiterte Metamodelle zu ergänzen sowie neue und verbindende Modellierungsansätze und Methoden wie beispielsweise Business Conversations [Hupe et al. 1998] zu integrieren.

Abläufe und Klassifikationen liegen aufgrund ihrer Definition weitgehend orthogonal zueinander, während die beschriebenen Prozessmodellierungsansätze die Dynamik eines Systems oder Modells wiedergeben, beschreibt die Klassifikation Statik und Entitäten. Trotzdem gibt es für die Produktion bereits Konzepte, Prozesspläne mit Hilfe einer Ontologie semantisch zu beschreiben [LastraDelamer 2005]. Produktionsabläufe werden dabei mit der NIST Process Specification Language (PSL) [Schlenoff et al. 1999] mit Hilfe der Prädikatenlogik 1. Stufe beschrieben und auf Semantic Web Services abgebildet [Grüninger 2003]: „*Rather than utilizing standard or*

¹² WebML als UML 2.0 Profil [Moreno et al. 2006] bietet immerhin eine Integration in die UML 2.0

proprietary formats, intelligent Production Scheduling or Manufacturing Execution Systems can reason about semantic workflows.“ [LastraDelamer 2005]

Meist finden bisher jedoch rein ablaufstrukturelle (wie im Workflow-Management) oder rein artefaktstrukturelle Beschreibungen wie Document Type Definitions (DTD) und Dokumenten-Templates bei der Modellierung Verwendung. Interaktive Abläufe in der Produktion basieren jedoch meist auf Materialflüssen und Informationen, die in Abläufen verändert und transportiert werden [Thies 2003], so dass hier beide Konzepte Anwendung finden.

Für die Modellierung allein existieren allerdings bereits integrative Ansätze wie die Integration von UML (objektorientiert) und EPK (rein ablauforientiert) [LoosAllweyer 1998], wo jedoch zunächst nur eine Abbildung auf die UML stattfindet. Die Integration von EPK und Klassendiagrammen [Dandl 1999] ermöglicht für EPK-Abläufe eine Definition von Typen. Die Klassen werden zudem im Klassendiagramm dargestellt. Hier fehlt die entsprechende Modellintegration über ein gemeinsames Metamodell, jedoch ist der Ansatz auf Einzeldiagrammebene in der Auswirkung ähnlich wie der in dieser Arbeit angestrebte. In unterschiedlichen Stufen des Prozesses sollen hier Teilsichten erzeugt werden können, auf der Makroebene in der Analysephase mit Hilfe der Verbindung von Klassen und Prozessschritten, auf der Mikro-Ebene im Entwurf mit Hilfe der Verbindung von Funktionen und Prozessschritten.

Auch die UML selbst deckt in den Version 2.0 und 2.1 die Ablaufmodellierung nun wesentlich besser ab, versucht auch hinsichtlich der Objektorientierung ein konsistentes Metamodell zur Verfügung zu stellen und bietet mit den Aktivitäten bereits einen Ansatz zur Integration der statischen Objektorientierung und der dynamischen Ablaufmodellierung: „*Objektknoten sind das Bindeglied zwischen der dynamischen Flussmodellierung mit Aktivitäten und der statischen Modellierung von Strukturelementen.*“ [Jeckle et al. 2004]

Allerdings lassen alle Ansätze eine Laufzeitunterstützung für die Modellierung vermissen, die es ermöglicht, Modelle zur Laufzeit zu erstellen, zu verändern und auszuführen.

Die folgende Tabelle zeigt, die zur Laufzeitentwicklung und -ausführung von integriert objektorientierten Prozessen notwendigen Kriterien. Die vorgestellten Klassen von Ansätzen werden hierbei – teilweise anhand von typischen Stellvertretern – an diesen Kriterien gemessen. Hier haben meist nicht alle Vertreter der Methodenklasse die markierten Funktionalitäten, so dass einzelne Ansätze teilweise schlechter zu bewerten wären als ihre Gruppe.

Der Vergleich zeigt, dass sehr unterschiedliche Typen von Ansätzen und Systemen vorliegen, die einen unterschiedlichen Ursprung und eine unterschiedliche Zielrichtung aufweisen. Ansätze aus der Informationstechnologie betonen Modelle, formalisieren Abläufe und führen diese aus. Produktions- und geschäftsprozessorientierte Ansätze zeigen eine bessere Ausrichtung auf die Erfordernisse der Produktion, arbeiten allerdings meist mit flachen Datenmodellen und sind stärker auf Planung sowie auf Automatisierung (MES) oder Unternehmensdaten (ERP) ausgerichtet. Stärken und Schwächen bestehen daher in differierenden Teilbereichen. Insgesamt bestehen jedoch bei den existierenden Ansätzen folgende Defizite:

- Fehlende Integration von Ablauf- (Dynamik) und Objektorientierung (Statik)
- Mangelnde semantische Aussagekraft und Variabilität von Prozessmodellen
- Mangelnde Modellintegration und fehlende Abdeckung der Produktionsspezifika
- Mangelnde Laufzeitänderbarkeit und Animationsfähigkeiten von Prozessen
- Fehlende Dezentralisierbarkeit auch bei verteilten Systemen
- Mangelnde Generierungs- und Interaktionsfähigkeiten

Ansätze / Kriterien	Software-entwicklung		Modellierungsmethoden					Systemkonzepte			
	M D A	Software-Prozessmodelle	Ontologien	Petrinetze	Objektorientierte Petrinetze	Proz.mod.sprachen EPK, BPMN, etc.	U M L 2	WFMS	PPS	MES	ERP / ERP II
Typisierung	●	●	●	○	●	●	●	●	○	●	●
Semantische Varianten	●	●	○	○	●	○	●	○	○	○	○
Mehrfachvererbung	○	○/○	●	○	●	○	●	○	○	○	○
Komponenteninterfaces	●	●	●	○	●	○	●	○	○	○	○
Unterst. generativer Ansätze	●	●	●	●	○	●	●	○	○	○	○
Modellanimation (möglich)	●	●	○	●	●	●	●	●	○	●	●
Laufzeitadaption	○	●	●	○	○	○	○	●	○	●	●
OO Laufzeitadaption	○	●	●	○	○	○	○	○	○	○	○
Laufzeitentwicklung Typen	○	○	●	○	○	○	○	○	○	○	○
Metamodelle / DSL	●	●	●	○	○	○	●	○	○	●	○
Datenaustausch	●	●	●	○	●	○	○	●	●	●	●
Dezentrale Verteilung (Modell) mögl.	○	●	●	○	○	○	○	●	○	●	●
Produktion in einem Modell	○	○	●	○	○	○	○	○	○	●	○
Integratives Gesamtmodell	○	○	●	○	○	○	○	○	○	●	○
Modellperformanz	○	○	○	●	●	○	○	○	○	●	○
Darstellungsunabhängigkeit	●	●	●	●	●	○	●	○	○	○	●
Ebenenübergreifende Konzepte	●	●	●	○	●	●	●	○	○	○	○
Wiederverwendung	●	●	●	○	○	●	●	●	○	○	○
Externe Artefakte	●	○	○	○	●	○	○	●	○	●	●
Konkretisierungsebenen	●	○	●	○	●	○	○	○	○	●	○
Hierarchische Komponenten	●	●	●	○	●	●	○	○	○	○	○
Typisierte Komponenten	●	●	●	○	●	○	●	○	○	○	○
Ablaufkomponenten	●	●	○	●	●	○	●	○	○	○	○
Integration Abläufe / Objekte	○	●	○	●	●	○	○	○	○	○	○
Integration Abläufe / Typen	●	●	○	○	○	○	○	○	○	○	○
Statische Prozessklassifikation	○	○	●	○	○	○	○	○	○	○	○
Dynamische Prozessklassifikation	○	○	●	○	○	○	○	○	○	○	○
Automatisierung Prozess	●	●	○	●	●	●	○	○	○	○	○
Gesch.Proz.-Unterstützung	○	○	○	○	○	○	○	○	○	○	○
Laufzeitw. Prozesse	○	○	○	○	○	○	○	○	○	○	○
Implizite Interaktion	○	○	○	○	○	○	○	○	○	○	○
Explizite Interaktion	●	○	○	○	○	○	○ ¹⁾	○	○	○	○
Produktions-Spezifika	○	○	○	○	○	○	○	○	○	○	○

¹⁾ beispielsweise UMLi

Legende: ● trifft voll zu, ○ trifft teilweise zu, ○ trifft nicht zu

Tabelle 14: Vergleich relevanter Konzepte für die Laufzeitmodellierung objektorientierter Prozesse

3 Ansatz und Aufgabenstellung

„On the one hand comprehensive models are required, which cover extensive production areas and permit detailed questions. On the other hand the models should be rapid enough, to be able to use the responses actually accompanying to the production process. Using only one modelling technique, these requirements cannot be met at one time.” [ReinhartLulay 1998]

Diese Defizite sollen im Folgenden durch Anforderungen adressiert werden, die im Ansatz dieser Arbeit zusammengefasst sind. Der Abschnitt „Vision“ zeigt dabei den grundlegenden Rahmen auf, in dem sich die Arbeit bewegt. Der folgende Ansatz enthält die Ziele der Arbeit, die durch die Aufgabenstellung so konkretisiert werden, dass darauf folgend die Anforderungen als konkrete, zu lösende Aspekte dargestellt werden können.

3.1 Vision

Funktionale Sichten fokussieren und modellieren die Transformationen von Artefakten – ob Werkstücke oder Daten – und wurden daher in der Produktion lange Zeit bevorzugt verwendet.

In der Softwareentwicklung wird seit Jahren das Paradigma der Objektorientierung als Sicht auf komplexe Systeme genutzt. Die Klassifikation mit einem semantischen Modell hilft dabei Systeme und ihre Bestandteile für Menschen verständlicher und für Maschinen besser nutzbar zu machen und der Realwelt anzunähern. Dabei wurden objektorientierte Konzepte bereits in andere Domänen übertragen und verwendet wie bei der Erstellung von Business Objects.

Die vorliegende Arbeit soll durch die Integration von ablauforientierten, dynamischen Prozessmodellen mit objektorientierten, statischen Klassifikationen helfen, eine semantische Produktion zu erschaffen. Semantische Produktion bedeutet dabei eine Überlagerung der Produktion mit semantischen Informationen wie Klassifikationen von Artefakten, Nachrichten und Beziehungen zwischen diesen. In Anlehnung an das Web 2.0 Paradigma auch als Produktion 2.0 bezeichnet, wird ein objektorientiertes, integratives Gesamtmodell angestrebt, das aussagekräftiger und flexibler ist als herkömmliche Modelle.

Während bei funktionalen Systemen jede Änderung auch eine Anpassung der Funktion in Software nach sich zieht, sollen Systemänderungen mit dem hier vorgestellten Ansatz durch Modelländerungen zur Laufzeit ermöglicht werden. Das Abschalten und aufwändige Reinitialisieren von Fertigungssystemen bei Veränderungen soll durch flexible Zellen und Peers (vgl. Kapitel 2.3.2) sowie das semantische Prozessmodell nicht mehr nötig sein.

Vielmehr soll zur Laufzeit eine Umstrukturierung von Produkten, Fertigung und sogar Unternehmen möglich sein: **Rapid Model-based Reconfiguration** – flexible Produktion der Zukunft.

3.2 Ansatz

Das primäre Ziel ist die Flexibilisierung und gleichzeitig abstrakte Systematisierung von Abläufen mit Hilfe objektorientierter Konzepte. Hierzu wird ein multidisziplinärer Ansatz gewählt, der Methoden und Technologien aus anderen Bereichen nutzt und integriert.

Hierbei tritt die weitere Spezialisierung bereits etablierter Domänen wie der Fertigungsplanung in den Hintergrund. Dagegen zielt die Arbeit auf die Erschließung und Verbesserung der Prozess- und Produktionsmodellierung mit Hilfe objektorientierter Konzepte aus der Modellierung, dem Software Engineering, der Metamodellierung, der User-Interface-Generierung und der Workflow-Domäne.

Die Übertragung und Anwendung dieser Konzepte auf Produktionssysteme soll zu neuen Erkenntnissen und einem integrativen Modell führen, das möglichst viele Elemente eines

Produktionssysteme im Modell abbilden und klassifizieren kann und so neue Möglichkeiten für die prozessorientierte Steuerung von Abläufen in der Produktion bietet.

Das Modell soll sich während der Laufzeit dynamisch an neue Situationen im Prozess jedoch auch im gesamten Unternehmen anpassen. Rekonfigurierbarkeit, individualisierte Produktion, Lösgröße 1, Anwendung von Individualfertigungskonzepten auf eine Massenproduktionsinfrastruktur und umgekehrt sind dabei Beispiele aktueller Forderungen und Entwicklungen in der Produktion. Diese sollen mit Hilfe der Fusion objektorientierter Konzepte mit ablauforientierten, funktionalen Konzepten ermöglicht werden.

Da eine vollständig freie Produktion keine Qualität sichern kann, ist das Ziel, prozessorientierte Kontrolle und Qualität mit Flexibilität und zeitnaher Prozessdefinition zu koppeln – bei gleichzeitiger Erhöhung des Prozessreifegrades (wie CMMI, EFQM, ISO9000ff.) statt Beliebigkeit einer vollständig individuellen, prozesslosen Fertigung.

3.3 Aufgabenstellung

Gegenstand dieser Arbeit ist daher die Zusammenführung der ablauforientierten Prozessmodellierung mit der objektorientierten, klassifikatorischen, statischen Modellierung unter einer weitgehenden Wahrung der Vorteile beider Paradigmen mit einem Fokus auf interaktive, verteilte Prozesse in der Produktion.

Artefakte der Produktion, Daten und auch Prozessschritte können mit einem objektorientierten Ansatz – wie er in Ontologien zum Einsatz kommt – klassifiziert werden. Die vollständige Objektorientierung soll dabei für alle an den modellierten Prozessen beteiligten Elemente eine Klassifikation und Identität liefern. Gerade in einem heterogenen und dynamischen Produktionsumfeld ist die korrekte Zuordnung von Artefakten der Produktion ohne entsprechende Modellsemantik ein aufwändiger und fehlerträchtiger Vorgang, der schnell zu einer Abweichung von Modell und Realität führt, die dann durch Inventuren etc. revidiert werden muss.

So soll jeder Prozessschritt über eine Typzuordnung verfügen, die dank Konzepten der Mehrfachvererbung auch komplex und mehrdimensional sein kann. Für die Ausführung eines solchen Prozessschritts benötigte Eingaben, Daten, Maschinen und Werkzeuge sollen ebenfalls mit Hilfe der Objektorientierung klassifiziert werden. So soll für alle Vor- und Nachbedingungen eine klassifikatorische Korrektheitsprüfung ermöglicht werden, die bereits ohne explizite Vorgabe von weiteren Regeln allein auf Basis der Modellsemantik durchgeführt werden kann.

Für proprietäre, nicht klassifizierte Datenformate für Produktionsdaten soll es dabei möglich sein, diese mit einer Klassifikation zu überlagern, die auch eine dezentrale und heterogene Datennutzung über Anwendungs- und Organisationsgrenzen hinweg ermöglichen sollen.

Die Vererbung ermöglicht dabei die Definition von Standardprozessen, von denen für Organisationseinheiten und Produkte spezifische Varianten abgeleitet werden können. Hierbei soll ein Metamodell automatisch die Kompatibilität zum Standardprozess sichern und durch veränderte Rahmenbedingungen notwendig gewordene Anpassungen des Standardprozesses an die Varianten weitergeben.

Um eine schnelle Rekonfiguration zur Laufzeit zu ermöglichen, müssen Änderungsmöglichkeiten zur Laufzeit geschaffen werden, die auch bereits in Ausführung befindliche Prozesse betreffen können. Hierzu wird ein Laufzeitkonzept mit einer entsprechenden Modellarchitektur benötigt, der Modellierung und Modelländerungen noch während der Systemausführung ermöglicht. Dabei ist zu berücksichtigen, dass Konzepte der Vererbung und Instanziierung erhebliche Seiteneffekte bei Änderungen zur Laufzeit auslösen. Diese treten nicht in der für die objektorientierte Programmierung und Modellierung typischen, generativen Verwendung auf, sondern während der hier angestrebten direkten Modellausführung.

Änderungen müssen dabei vorgegebenen Regeln folgen, die aufgrund der angestrebten Flexibilität des Ansatzes nicht individuell programmiert sondern durch ein Metamodell vorgegeben werden müssen.

Gegenstand der Arbeit ist die Entwicklung eines durchgängigen Modells und Laufzeitkonzepts für die integrative Beschreibung von Prozessen und Klassifikationen. Der Schwerpunkt liegt auf der Integration von ablauf- und typorientierten Beschreibungen zu einem Gesamtmodell, das so Statik und Dynamik eines interaktiven Fertigungssystems erfasst. Das Gesamtmodell wird zur Spezifikation von Prozessen herangezogen, wobei eine dynamische Anpassung und Interpretation zur Laufzeit ermöglicht wird. Die hierdurch mögliche Qualitätssicherung, Typsicherheit und Wiederverwendung unterstützen eine schnelle Variantenbildung und Prozessanpassung bei hoher Qualität und systemimmanenter Konsistenzsicherung.

Zu lösen bleibt jedoch das Spannungsfeld zwischen Objektorientierung und Ablauforientierung:

Modelltyp	Objektorientierung	Ablauf-/Prozessmodellierung
Betrachtungsgegenstand		
Modellaspekt	Modellierung des Produktionssystems	Modellierung der Produktionsabläufe
Ausdrucksmittel / Konzepte	Komponenten, Relationen	Abläufe
Modellcharakteristik	Statik	Dynamik
Verkettung	Konkatenation der Struktur	Folgebeziehung im Ablauf
Modellstruktur	Hierarchien: Typen, Aggregate	flaches Modell
Modellanimation durch	Nachrichten	Folgen von Aktionen
Veränderung der Modellsemantik	Metamodell-Semantik erweiterbar	Vorgegebene Semantik
Ablaufverifikation	Nichtdeterminismus	Überprüfbare Abläufe
Konsistenzprüfung	Typsichere Verwendung von Elementen	Typ nur durch Modellelement vorgegeben
Wiedergabe von Abläufen	Weitgehend unklare Abläufe	Animierbare Abläufe
Verwendung von Laufzeitartefakten	Typsichere Auswahl	Verwendung beliebiger Artefakte
Modellfokus	Strukturen	Manipulationen
Wiedergabe von Daten	Datenstruktur	Datenfluss
Wiedergabe von Veränderungen	Elemente die verändert werden	Auf Elemente angewandte Veränderungen

Abbildung 7: Spannungsfeld Objektorientierung und Ablauforientierung

Des Weiteren sollen so sekundäre Methoden wie Wissensmanagement leicht in das System einführbar werden. Modellierungssprachen wie K-Modeler [Gronau et al. 2003] beziehungsweise die Weiterentwicklung KMDL 2 [Fröming et al. 2005] zur Abbildung wissensintensiver Geschäftsprozesse verbinden bereits den Prozess mit dem Wissensmanagement. Prozessintegrierte Ansätze des Wissensmanagements fehlen jedoch. Die Schlüssel zur Überlagerung solcher externer Konzepte sind Typ (statisch) und Prozess-Kontext (dynamisch), die für jedes Artefakt dessen Art, Entstehung und Kontext beschreiben sollen.

Durch die Auslegung auf Laufzeit- und Service-Orientierung sollen in einem produktiven, interaktiven System direkt Erweiterungen vorgenommen werden können, die nun nicht mehr zu Unterbrechungszeiten führen, sondern neue Funktionalitäten schnell und sicher in den laufenden Herstellungsprozess einbetten.

Die Arbeit entwickelt daher ein integratives, objektorientiertes Prozess-Metamodell zur Modellierung und Animation von flexiblen Prozessen in einem verteilten, service-orientierten Produktionssystem.

Die Integration von Abläufen und Klassifikationen mit Hilfe einer objekt- und ontologiezentrierten Darstellung berücksichtigt auch Interaktionen, wo Modellinformationen nicht vollständig sind.

Somit legt die Arbeit die Modell-Grundlage für den Einsatz modellbasierter, service-orientierter, interaktiver Architekturen in der Produktion, die sich technologisch bereits abzeichnen beginnen.

3.4 Anforderungen

Im Hinblick auf ein laufzeitbasiertes Prozessmodell ist ein auf Service-orientierte Architekturen zugeschnittenes Ablauf- und Typmodell zu entwickeln. Hierzu muss zunächst ein übergreifendes Metamodell entwickelt werden, das alle Teilaspekte enthält und Ansätze für eine Modellverteilung zur Laufzeit liefert. Basierend auf diesem soll es ermöglicht werden, interaktive objektorientierte Prozesse in der Produktion zu instanzieren und zu animieren, so dass generativ zur Laufzeit entsprechende Aufrufe in einem verteilten System und die Erzeugung von Interaktionen möglich wird.

Für die beschriebenen Problemfelder konnten bereits einzelne Defizite im aktuellen Stand von Forschung und Technik identifiziert werden. Um eine Lösung der Probleme und eine Umsetzung erreichen zu können, sind folgende Anforderungen zu erfüllen, die die genannten Defizite beheben sollen:

A1 Integration von Ablauforientierung (Dynamik) und Typ-/Objektorientierung (Statik) stellt das grundlegende Defizit und dessen Behebung damit auch die grundlegende Anforderung an diese Arbeit dar. Existierende Arbeiten konzentrieren sich entweder auf eine Modellierung von Abläufen (EPK, UML-Sequenzdiagramme) oder auf eine Modellierung von Strukturen und Klassifikationen (UML-Klassendiagramm, Stücklisten etc.). Obwohl grundlegende Mechanismen zumindest für die grafische Modellierung (beispielsweise in der neuen UML 2) zur Verfügung stehen, wird eine diagrammübergreifende Kopplung von Typen und Abläufen nicht vorgenommen. Das integrierte Basismodell muss also sowohl Abläufe in typisierten Komponenten kapseln als auch solche Komponenten in Abläufe einbinden können.

A2 Dabei soll mit einem **integrativen Gesamtmodell** die Abdeckung der gesamten Produktion mit dem Modell möglich sein. Bestandteile eines Produktionssystems wie Prozesse, Prozessarten, Artefakte, Maschinen, Werkzeuge, Daten und Interaktionen müssen im selben Gesamtmodell zusammengefasst, klassifiziert und in Verbindung gebracht werden können, um Inferenz zu ermöglichen, also automatisiert Schlüsse aus der Systemkonfiguration zu ziehen. Dieses konsistente, generische Gesamtmodell soll eine integrierte Typologie der Abläufe und der intern und extern erzeugten Artefakte enthalten.

A3 Zur Komplexitätsreduktion, Hierarchisierung beziehungsweise Partitionierung und Wiederverwendung wird ein Konzept benötigt, das einzelne Komponenten wie Prozessschritte zu einem größeren Ganzen zusammenfassen kann. Die **hierarchische Modularisierung und Verwendung von Komponenten** setzen allerdings voraus, dass die Konzepte auf allen Ebenen der Aggregationshierarchie und in allen Modellen und Instanzebenen einheitlich sind und dass jede Komponente die in ihr aggregierten Komponenten automatisch als Schnittstelle nach außen zur Verfügung stellt. Die Modulbildung orientiert sich dabei nicht nur an mechatronischen Strukturen [SpathKoch 2007] sondern ergänzt die übergeordnete Prozessebene.

A4 Ein **konsistentes Typ- und Spezialisierungskonzept** bildet die Grundlage für ein kohärentes und erweiterbares objektorientiertes Modell. Dabei ist eine Typunabhängigkeit der Komposit-/Aggregattypen von den aggregierten Typen wichtig. Produkte werden während ihrer Entstehung verändert, so dass sie nicht mehr (nur) als Summe ihrer Einzelteile aufzufassen sind. Die starke Typisierung aller Modellelemente und Typabhängigkeit bei Spezialisierung stellt im Gegenzug sicher, dass eine einmal eingeführte, objektorientierte Semantik immer verfügbar ist: Wird ein spezielles Produkt „Kfz A mit 2,3l Hubraum“ erzeugt, handelt es sich um eine Spezialisierung von „Kfz A“, die den Basistyp übernimmt und durch Spezifizierung eines Hubraums eine Einschränkung vornimmt. Für das System muss also erkennbar sein, dass auch

„KfZ A mit 2,3l Hubraum“ ein KfZ A und damit ein KfZ ist. Dies erleichtert das Auffinden und Identifizieren von Prozessen, Modellelementen und Artefakten. Die Bildung und Verwendung von semantischen Varianten ist ein zentraler Bestandteil individualisierter und flexibler Produktion, da bisherige Verfahren des Kopierens und Veränderns zusätzlich meist eine Variantenverwaltung benötigen, die Varianten, Neuentwicklungen und Versionen¹³ identifiziert. Dieses System ist unsicher und redundant. Zudem erlaubt es nicht, Änderungen automatisch auf abgeleitete Varianten anzuwenden. Das zu entwickelnde Konzept soll hier Abhilfe schaffen. Baumförmige Hierarchien erreichen zwar eine eindeutige Unterteilung existierender Konzepte, jedoch gehen moderne Rollenmodelle und komplexe Produkte häufig mit Typfusionen (Mehrfachvererbung) einher. So können Werkzeuge und Maschinen Produkt und Produktionsmittel in einem sein. Das Konzept soll auch diese Sachverhalte abbilden können.

A5 Ein **Laufzeitkonzept** soll das gesamte Modell laufzeitfähig machen, das heißt Mechanismen der Objektorientierung und Animation sollen nicht nur in einer Konzeptionsphase sondern im laufenden Betrieb verfügbar sein. Dazu muss das Modellsystem von existierenden Instanzen ausgehen sowie Prozessschritte instanzieren, gezielt aktivieren und Artefakte erzeugen können. Eine besondere Herausforderung stellen dabei dynamische Modelladaptionen zur Laufzeit dar. Diese werden notwendig, wenn Prozesse in einer flexiblen und auftragsbasierten Produktion häufigen Anpassungen unterliegen. Eine Systemabschaltung zur Integration neuer Prozesse ist dabei aus Effizienzgründen undenkbar. Änderungskonflikte und Variantenbildung müssen daher zur Laufzeit gelöst werden. Da sich bei der Analyse existierender ontologiebasierter Systeme bereits ohne Prozessanimation hohe Antwortzeiten zeigten, soll dabei auf eine schnelle interne Antwortzeit zur Laufzeit für Prozessanimation und Klassifikationsanfragen geachtet werden.

A6 Häufig weisen theoretisch erzeugte Prozesse Schwachstellen auf oder können erst mit Verfahren wie Service Blueprinting [Shostack 1982] erzeugt werden, die ein Durchlaufen des Prozesses während seiner Erzeugung vorsehen. Die Möglichkeit einer **Laufzeitentwicklung von Prozessen** soll dieser Forderung Rechnung tragen. Das Modell muss so konzipiert werden, dass quasi „just in time“ der folgende Prozessschritt während des (meist prototypischen) Durchlaufens definiert und dann instanziiert wird. Sowohl die Auswahl als auch die Überprüfung der Schritte muss dabei typbasiert erfolgen können. Das System muss also unvollständige Instanzen und Systemadaption zur Laufzeit (A5.2) unterstützen. Flexibilität führt zu einer erhöhten Änderungshäufigkeit – meist mit der Notwendigkeit, neue Anforderungen direkt für alle Prozesse eines bestimmten Typs durchzuführen. Diese automatische Kaskadierung von Änderungen über die Vererbungshierarchie ist die Voraussetzung für konsistente Veränderungen.

A7 Ein **flexibles Modell** muss möglichst frei in Software umsetzbar sein, das heißt nur ein Minimum der Modellbestandteile soll implementiert, der Rest durch Definition und Interpretation des Modells ergänzt werden. Nur so kann eine gute Wartbarkeit und Flexibilität des Produktionssystems sichergestellt werden. Es sollen dabei auch unterschiedliche Modellierungstechniken und Perspektiven unterstützt werden, da die mit einem betrieblichen Prozesssystem interagierenden Personen häufig unterschiedliche Kenntnisse oder Sichtweisen haben und damit für denselben Prozess auch unterschiedliche Sichten und Darstellungen benötigen. Auch die Modellierung teilmodellübergreifender Zusammenhänge, das heißt sichten- und diagrammunabhängige Beziehungen, und die Integration unterschiedlicher Metamodelltypen und Modellierungstechniken im selben Gesamtmodell sind daher wünschenswert.

A8 **Integriertes Interaktionskonzept:** Durch die geforderte Flexibilität des Modells kann es bewusst oder zufällig zu fehlenden Werten oder Freiheitsgraden in einem Prozess kommen. Hier soll ein implizites Interaktionskonzept für nicht existierende Werte eingesetzt werden. Ergänzend

¹³ Eine Variante führt eine Veränderung gleichberechtigt ein, eine Version ersetzt das vorhergehende Modell, eine Neuentwicklung kennt keine Vorversion oder Varianten

soll ein explizites Interaktionskonzept für vorgesehene Eingaben im Hinblick auf eine dynamische Erzeugung von prozessbezogenen Interaktionen mit diesem integriert werden.

Anforderungsgruppe	Nr. / Gruppenname	Anforderung
A1	Integration von Ablauforientierung (Dynamik) und Typ-/Objektorientierung (Statik)	
	A1.1	Kapselung von Abläufen in typisierten Komponenten
	A1.2	Konkatenation und Aggregation von Komponenten zu Abläufen
A2	integratives Gesamtmodell	
	A2.1	Typologische Abdeckung der gesamten Produktionsabläufe und Artefakte möglich
	A2.2	Modell aller Bestandteile eines Produktionssystems im selben Gesamtmodell
	A2.3	Interaktionen, Daten und Abläufe in einem Modell
	A2.4	Automatisierte Auswertung des Gesamtmodells
	A2.5	Einbindung von klassifizierten externen Artefakten
	A2.6	Modellkompatibilität zur Geschäftsprozess-Unterstützung (ERP etc.)
A3	hierarchische Modularisierung und Verwendung von Komponenten	
	A3.1	Hierarchisierung und Partitionierung
	A3.2	Wiederverwendungskonzepte für Teilmodelle
	A3.3	Einheitliche Konzepte auf allen Modellebenen
	A3.4	Automatisch auswertbare Selbstbeschreibungsfähigkeit der Komponenten
A4	Konsistentes Typ- und Spezialisierungskonzept	
	A4.1	Typunabhängigkeit der Komposit-/Aggregatypen von den aggregierten Typen
	A4.2	Typabhängigkeit der Spezialisierungen von übergeordneten Komponenten
	A4.3	starke Typisierung aller Modellelemente
	A4.4	Bildung und Verwendung von (semantischen) Typ-Varianten
	A4.5	Typologische Prozess-Komponentenfusionen (Mehrfachvererbung)
A5	Laufzeitkonzept	
	A5.1	Animationskonzept für Prozessmodelle zur Laufzeit
	A5.2	dynamische Modelladaption zur Systemlaufzeit
	A5.3	Effizientes Typ- und Laufzeitmodell: kurze interne Antwortzeiten
	A5.4	Polymorphie: Typkompatible Verwendung von Spezialisierungen / Varianten
A6	Laufzeitentwicklung von objektorientierten Prozessen	
	A6.1	just in time: unvollständige Instanzen während der Prozessmodellierung animieren
	A6.2	typbasierte Auswahl und Überprüfung von Prozessschritten bei der Modellierung
	A6.3	automatische Kaskadierung von Änderungen über die Vererbungshierarchie
A7	Flexibles Modell	
	A7.1	Das Modell soll möglichst frei in Software umsetzbar sein
	A7.2	Umsetzung durch minimales Basismodell, Rest durch Modellerweiterung
	A7.3	Darstellungsunabhängigkeit des Modells
	A7.4	differenzierte Modellsichten
	A7.5	Modellierung teilmodellübergreifender Beziehungen
	A7.6	Integration unterschiedlicher Modellierungstechniken / Metamodeltypen
A8	Integriertes Interaktionskonzept	
	A8.1	implizites Interaktionskonzept für nicht existierende Werte
	A8.2	explizites Interaktionskonzept für vorgesehene, modellierte Eingaben
	A8.3	integriertes Laufzeitkonzept für implizite und explizite Interaktionen
A9	Unterstützung von Verteilung und Generierung	
	A9.1	direkte Unterstützung von Semantic Messaging Konzepten
	A9.2	Modellunterstützung von transformativen und generativen Verfahren
	A9.3	dezentrales, dynamisches und partitionierbares Zielsystem
	A9.4	Basismodell für Prozesse in service-orientierten Architekturen (SOA)
	A9.5	Modelldatenaustausch durch Abbildung auf ein XML-Format
	A9.6	Modell ermöglicht die Erzeugung von User Interfaces verteilt und zur Laufzeit

Tabelle 15: Anforderungen an das zu entwickelnde System

A9 Neben der Prozessmodellierung und -animation ist die **Unterstützung von Verteilung und Generierung** zu beachten. Die direkte Unterstützung von Semantic-Messaging-Konzepten [Schlegel et al. 2008][Schlegel et al. 2007] schließt den standardisierten Modelldatenaustausch mit ein und soll eine Verteilung von Daten und die Klassifizierung von Nachrichten mit Hilfe des Modells ermöglichen. Dies soll durch die Auslegung für ein dezentrales, dynamisches und beliebig partitionierbares Zielsystem erreicht werden, so dass eine Einbindung in ein dezentrales Produktionssystem ermöglicht werden kann. Es soll also ein Basismodell für Prozesse in Service-orientierten Architekturen (SOA) entstehen. Nach außen soll das Modell durch die modellinhärente Unterstützung von transformativen und generativen Verfahren zur Erzeugung von Interaktionen und Artefakten aus dem Gesamtmodell leicht in die IT-Systemlandschaft integrierbar sein. Herausforderungen sind dabei immer noch die Modellintegration und die direkte Erzeugung zur Laufzeit auf Anforderung, so dass Modelländerungen wie das Hinzufügen neuer Prozessschritte auch direkte Auswirkungen auf die Benutzungsschnittstelle haben. Mit dem Aufkommen der SOA rücken zudem aktuelle Defizite in der Laufzeit- und Verteilungstauglichkeit immer mehr in den Vordergrund: Soweit möglich soll das Modell für die Erzeugung von User Interfaces in SOA und zur Laufzeit ausreichende Kontextinformationen zur Verfügung stellen.

3.5 Ausrichtung des Ansatzes

Ontologische Konzepte haben in den letzten Jahren an Bedeutung gewonnen, weil sie in der Lage sind, die Welt zu strukturieren und Zusammenhänge einer maschinellen Erfassung der Semantik zugänglich zu machen.

In der Softwareentwicklung löste die Objektorientierung einen Paradigmenwechsel aus, weil diese eine der menschlichen Wahrnehmung verwandte Konzeption ermöglichte und sich so als wirklichkeitsnahes Modellierungsparadigma darstellte. Auch die Komponentenorientierung zur Beherrschung der Komplexität durch Partitionierung verbesserte sich bedeutend.

Auch die Produktion orientiert sich stark an Objekten wie Maschinen, Werkzeugen und Produkten. Gleichwohl können rein objektorientierte Konzepte auf moderne, stark prozessorientierte Systeme nicht übertragen werden. Hier stehen Transformationen und Abläufe im Vordergrund, die einer Abbildung als nichtdeterministische Kommunikation zwischen Objekten entgegenstehen.

Während also die Statik eines Produktionssystems objektorientiert modelliert werden kann, ist die Dynamik nur über Prozesse abzubilden.

Die Trennung beider Aspekte – beispielsweise in Steuerungssysteme und Datenbanken – oder die starre Verbindung spezifischer Daten und Abläufe in MES führt zu einem Verlust von Kontrolle oder Flexibilität.

Der Ansatz soll beide Welten integrieren, indem Konzepte integrativ und interdisziplinär angewandt und auf Besonderheiten der Produktion adaptiert werden. Erst die vollständige Integration aller Prozesse einschließlich aller Objekte wie Nachrichten, Daten und Ereignisse in ein Gesamtmodell macht das System einer objektorientierten Auswertung zugänglich.

Eine besondere Herausforderung stellt dabei die Vererbung und Instanziierung komplexer Prozesse dar. Gelingt diese, sind weitreichende Auswirkungen auf die Variantenbildung (mit Hilfe der Vererbung) und dynamische Anpassung von Prozessen und Artefakten zur Laufzeit zu erwarten.

Wo bisher Regelsystem und statische Überprüfungen den Einsatz neuer Komponenten auf ein Mindestmaß reduzieren und kostenintensive Systemanpassungen nach sich ziehen, wird ein objektorientiertes System die Zusammenhänge direkt erfassen und inkompatible Änderungen verhindern. Systemflexibilität wird damit zur Laufzeit möglich.

Ein zusätzlicher Vorteil ist, dass objektorientierte Metamodelle wie die UML bereits Einzug in die Prozessmodellierung halten und daher im Ansatz nicht mehr fremd sind. Durch ein laufzeitfähiges Modell könnten diese Darstellungen zur direkten Ausführung gelangen.

Service-orientierte, verteilte Architekturen werden zunehmend als Schlüsseltechnologien angesehen, kranken aber bisher an statischen Prozessen, die die Flexibilität wieder auf ein Minimum reduzieren. Gelingt die dynamische Prozessmodellierung zur Laufzeit gesichert durch ein ontologisches Klassifizierungskonzept, können dynamische Produktions- und Unternehmensstrukturen auf der Basis von dezentralen Architekturen betrieben werden, die ein enormes Potential für Flexibilität und Dynamik bieten.

4 Ein Modellierungskonzept für objektorientierte Prozesse

Diese Arbeit führt ein integriertes objekt- und ablaforientiertes Modellierungskonzept für Prozesse und Artefakte ein. Die Gesamtheit des entsprechenden Modells zur Laufzeit und aller beschriebenen Konzepte wird im Folgenden als „Objektorientiertes Metamodell für interaktive computergestützte Prozesse in Produktionsnetzwerken“ OMICRON bezeichnet. Es umfasst daher Komponenten, Relationen, Metamodell, Konzepte (Vererbung, Instanziierung) und Laufzeitsystem.

Das gesamte OMICRON-Konzept ist auf Komponenten und Relationen aufgebaut, die abwechselnd miteinander verbunden werden können und durch Typisierung und Instanziierung über eine unterschiedliche Semantik verfügen. Während Komponenten (je nach Metamodell zumindest temporär) ohne Relationen existieren können, sind Relationen nur zwischen zwei Komponenten beziehungsweise an einer Komponente reflexiv möglich.

4.1 Komponenten

Modularisierung ist zunächst eine Maßnahme des Komplexitätsmanagements [SpathKoch 2007]. „Die Möglichkeit, verschiedene Module so zu kombinieren, dass sie Anpassungen an unterschiedliche Marktbedürfnisse mit geringem Aufwand erlauben, macht modulare Produktarchitekturen zu einem wirkungsvollen Werkzeug des Variantenmanagements.“ [SpathKoch 2007]. In gleichem Maße kann eine komponentenbasierte Prozessmodellarchitektur für komplexe Variantenlandschaften einen Ablaufrahmen zur Verfügung stellen, der sowohl Flexibilität als auch Prozessorientierung offeriert.

Die Idee Softwaresysteme in Module zu zerlegen (vgl. beispielsweise [LudewigLichter 2007]) und damit zu kapseln (Information Hiding [Parnas 1972]) ist nicht neu, hat sich aber vielfach bewährt, wo komplexe Zusammenhänge zu verstehen/analysieren, zusammenzufügen und wieder zu verwenden waren. Die Modularisierung transferiert Konzepte aus der materiellen (Fertigungs-)Welt in die immaterielle Welt der Software und Modelle. Module stellen die Vorstufe der Komponenten dar: Aufgeteilt, gekapselt und gegebenenfalls mit weiteren Informationen versehen ermöglichen sie eine Zusammenstellung größerer Systeme aus existierenden, adäquat beschriebenen Bauteilen.

Die Tragfähigkeit von Komponentenkonzepten gilt im Softwarebereich als gesichert [Schwichtenberg 2003]. Unter anderem dienen Komponenten hier zur Verbesserung der Wiederverwendung in Software-Produktlinien [van Ommering 2005].

Für zuverlässige und sichere Prozesssteuerungs-Softwaresysteme wurden bereits **Formalisten** entwickelt, um bestimmte Aspekte dieser Systeme modellieren und konsistent halten zu können [Wang et al. 2005]. Hierbei sollen Inkonsistenzen aufgedeckt werden. Zur Modellierung werden Independently Developable End-User Assessable Logical (IDEAL) Components verwendet. Der Systemzustand und seine Transitionen können so modelliert und überprüft werden.

Während für programmierte sicherheitsrelevante Systeme ein solcher aspektorientierter und logischer Formalismus sinnvoll ist, kommt er für ein Prozessmodell zunächst nur als zusätzlicher (aufwändiger) Mechanismus zur formalen Evaluation der Prozess-Vollständigkeit und Konsistenzsicherung in Frage. Ein derartiges Komponentenkonzept ist daher für die praxisrelevante Modellierung zu formal und komplex, so dass auf einen solchen Formalismus im Rahmen dieser Arbeit zu Gunsten der Anwendbarkeit verzichtet wird.

Komponenten oder Module werden oft zum Zusammenfassen einzelner Teile oder Elemente genutzt. Softwaremodule dienen dabei der Kapselung von Softwarefunktionen, Prozessmodule ([Klein et al. 2004], S.11) der Abstraktion beziehungsweise Zusammenfassung von Teilprozessen

oder Funktionen. Eine einfache Modularisierung¹⁴ erzeugt so Aggregate. Eine maschinelle Unterstützung, Auswertung oder sogar Wiederverwendung ist erst möglich, wenn überlagerte, semantische Beschreibungen und festgelegte Schnittstellen zur Verfügung stehen. Daher ist eine Automatisierung oder Unterstützung bei der Modellierung und Entwicklung nur möglich, wenn die Module zu Komponenten werden, die durch ihre Semantik als Ganzes verstanden und eingesetzt werden können. [Szyperki 2001] definiert daher **Komponenten** als **groß genug zur Separierung** und als **geschlossene Einheit mit einer definierten Schnittstelle**.

Die in Anlehnung an [Höb 2005] definierten wesentlichen Eigenschaften einer Software-Komponente (vgl. auch [Weisbecker 2002], S. 151) treffen auf eine Prozesskomponente wie sie in OMICRON definiert wird ebenfalls zu. Eine Komponente

- ist eine technisch, typologisch und funktional abgeschlossene Einheit
- bietet ihre Zugang ausschließlich über genau spezifizierte Schnittstellen (Kapselung)
- greift auf andere Komponenten über genau spezifizierte Schnittstellen zu
- kann zu größeren Komponenten aggregiert oder in kleinere aufgeteilt werden
- wird (auch) zur Wiederverwendung entwickelt

Das Komponentenkonzept weist den Vorteil einer exzellenten Skalierbarkeit auf: Ein mächtiges Konzept zur Komponentenbildung kann es erlauben, von einer einfachen Aggregationsfunktion über eine Hierarchisierung bis hinzu einer beliebigen, semantisch kontrollierten Teilkomponenten-Synthese Komponenten über beliebig viele Ebenen zu einem größeren Ganzen zusammenfassen oder diese bis auf eine atomare Ebene zerlegen.

Um eine lückenlose Abdeckung durch das Metamodell sicherzustellen, werden alle Modellbestandteile bis zur atomaren Ebene durch Komponenten realisiert. Atomare Ebene heißt in diesem Fall jedes Attribut, jeder Wert, jedes Statement etc., das nicht mehr weiter unterteilt werden kann.

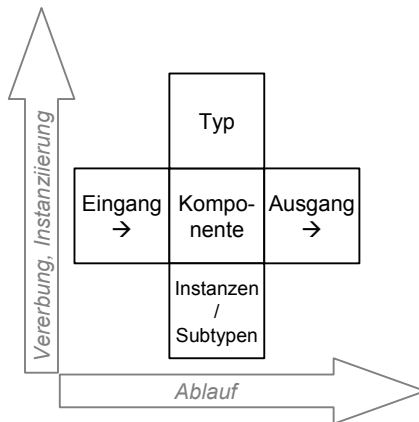


Abbildung 8: OMICRON Komponenten-Konzept

¹⁴ Die Definition von physischen Modulen als „funktional und physisch weitgehend vom Rest des Produkts unabhängige Einheit“ [SpathKoch 2007] kann dabei für Prozessmodule angepasst werden: Funktional vom Rest des Prozesses unabhängiger Teilprozess.

Das grundlegende Konzept muss für Komponenten im Hinblick auf die Anforderungen zunächst einen vertikalen, statischen Aspekt in Form einer Vererbungshierarchie mit Subtypisierung (Spezialisierung) und Instanziierung enthalten.

Das horizontale Ablaufkonzept muss ebenfalls integriert sein. Die obige Abbildung verdeutlicht diesen Zusammenhang der zwei Dimensionen.

Dies ist notwendig, um Vererbung und Komposition/Aggregation mit allen Modellartefakten betreiben zu können. Eine Ausgrenzung einzelner Bestandteile würde eine automatisierte Überprüfung und integrierte Generierung sofort ausschließen, da diese Artefakte nicht mehr im selben Repository beziehungsweise unter demselben Metamodell gespeichert werden könnten.

Die Leistungsfähigkeit von Komponenten ergibt sich aus den im Folgenden beschriebenen Eigenschaften.

4.2 Artefakte

Eine spezielle Form der Komponenten, sind Artefakte, da deren Inhalt nicht vom System direkt interpretiert werden kann.

Artefakte sind **inhaltlich extern zu interpretierende Erzeugnisse**, die als eingebrachte oder vom System erzeugte Komponenten nicht weiter mit Hilfe des Komponentenmodells untergliedert und typisiert werden können oder wurden. Artefakte können allerdings sowohl atomar als auch komplex, das heißt aus atomaren Artefakten zusammengesetzt, sein.

Hierbei kann es sich um Texte, HTML, Präsentationen oder ähnliche Binärdaten **ohne modell-strukturierten Inhalt** handeln, ebenso aber auch um Erzeugnisse des Prozesses.

Die Klassifikation stellt dabei die richtige Interpretation durch das System sicher. Beispielsweise kann so festgelegt werden, welche Applikation zur Anzeige oder Bearbeitung aufgerufen werden muss oder welche Artefakte kompatibel sind. Dagegen unterliegt der Inhalt des Artefakts nicht der Modellstruktur, sondern ist beispielsweise binär- oder XML-codiert ist.

Für Prozesse ist die Klassifikation von verwendeten und entstehenden Artefakten besonders wichtig, um für jeden folgenden Prozessschritt die erwarteten Artefakttypen zu definieren.

Eine mögliche Klassifikationshierarchie hängt vom Anwendungszweck ab und könnte für Dokumenttypen beispielsweise auf den MIME-Typen [IETF 1996] aufbauen. Allerdings ist diese Klassifikation zunächst flach (Nominalskala) und nicht hierarchisch. Daher müssen weitere Abstraktionsebenen eingeführt werden. Eine beispielhafte Klassifikation findet sich in Anhang H.

Je genauer ein Typ das Artefakt beschreibt, desto besser kann die Verarbeitung und Weitergabe erfolgen. Falls keine spezifische Behandlung definiert wurde, ermöglicht die Vererbung auch eine Interpretation als allgemeineren Typ.

Beispiel 2: Soll ein „Spezifikationsentwurf“ geöffnet werden, für den keine Interpretation vorliegt, kann dieser auch allgemein als „Spezifikation“ behandelt werden: „Spezifikationsentwurf“ und „Finale Spezifikation“ greifen so auf dieselbe Interpretationsroutine für die allgemeinere „Spezifikation“ zurück.

Selbst in der UML wird versucht externe Daten in die Modelle einzubinden. So lässt UML 2.1.1 [OMG 2007a] beispielsweise eine Formatangabe und das serialisierte Bild als Attribute von bestimmten Modellelementen zu.

Das Konzept, Artefakte mit Meta-Daten anzureichern wird in unterschiedlichen Domänen wie dem verteilten Software Engineering bereits genutzt [Boldyreff et al. 2002], allerdings findet hier keine Modellintegration sondern eine Speicherung in Repositories statt.

4.3 Relationen

Im Komponenten-Relationen-Modelle stellen Relationen Verknüpfungen zwischen den oben beschriebenen Komponenten dar.

Sei K die Menge der existierenden Komponenten, so ist eine Relation r

$$r \in (K \times K): (k, l) \text{ mit } k \in K, l \in K$$

Transitive Relationen werden nicht (explizit) modelliert: Bei $A \rightarrow B \rightarrow C$ wird $A \rightarrow C$ bereits transitiv erschlossen.

Bisher werden selbst in Ontologien und der Metamodellierung¹⁵ Komponenten und Relationen vollständig getrennt definiert: Die Folge ist, dass Relationen nur künstlich typisiert werden und keine Verbindung zu ihrer Typdefinition haben.

Bei den Relationen einer Ontologie handelt sich um eine Menge von prädikatenlogischen [Schöning 1995] Definitionen anstelle einer Typhierarchie. Relationen zwischen Konzepten können in Ontologien durch Relationen zwischen ihren Spezialisierungen allerdings „erweitert“ werden. Eine Bildung von Subtypen für Relationen ist jedoch nicht möglich.

In der Literatur finden sich auch in aktuellen Ansätzen nur Definitionen, die zunächst Relationen und Objekte korrekt definieren, dann jedoch nur Einfachvererbung für Objekte und keiner Vererbung für Relationstypen zulassen. Eine solche Formalisierung von Metamodellen findet sich beispielsweise in [VarróPataricza 2002]. Hier werden Instanz (Identifier) und Klasse (Type) zur leichteren Formalisierung in einen „Record“ zur besseren mathematischen Beschreibung zusammengefasst. Ein Vererbungskonzept für Relationen fehlt jedoch auch hier.

4.3.1 Typisierung von Relationen

Methoden für eine Subklassifikation (Bildung von Subtypen) von Relationen sind allerdings für eine Differenzierung von Relationen bei gemeinsam ererbter Funktionalität unabdingbar. Das OMICRON Modell muss in dieser Hinsicht also auch dieses Konzept umfassen.

Daher werden in OMICRON ähnlich wie in manchen anderen objektorientierten Modellierungsmethoden typisierte Relationen eingesetzt, die einem Relationstyp angehören.

Sei T die Menge aller Relationstypen, das heißt aller Komponenten, die von der basalen **Relationstypkomponente** e_R im Meta-Metamodell instanziiert werden, so ist

$$r \in (K \times K \times T): (k, l, t) \text{ mit } k \in K, l \in K, t \in T$$

Dabei ist $V \subset T$ die Menge der Vererbungstypen ausgehend von der basalen Vererbungskomponente $v \in V$ und damit $v \in T$. Eine Formalisierung des vollständig entwickelten Modells erfolgt in Kapitel 6.5.

Um eine Typ-Hierarchie von Relationstypen zu erzeugen, können **Subtypen von Relationstypen** durch das Vererbungskonzept gebildet werden, so dass Relationstypen in einem Spezialisierungsverhältnis zu einander stehen können. Dabei kann dem objektorientierten Paradigma folgend an allen Stellen, wo ein Relationstyp A gefordert ist auch eine Spezialisierung A' des Relationstyps A verwendet werden.

Die **Relationstypen** werden als Instanz des **Relations-Metatyps** definiert. Hierdurch treten die Relationstypen wie spezielle Komponenten (Relationstypkomponenten) auf, die aufgrund der Komponenteneigenschaft mit Hilfe der Vererbung weiter differenziert werden können. Von jedem Basis-Relationstyp werden per Vererbung Sub-Typen beispielsweise der Vererbung,

¹⁵ vgl. beispielsweise IkoSem-Metamodell, Abbildung 3 in [Terrasse et al. 2006]

Aggregation, Flussrelation oder Instanziierung gebildet, die auch die entsprechende Semantik im Laufzeitsystem erhalten.

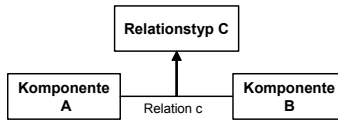


Abbildung 9: Eine Relation *c* zwischen zwei Komponenten A und B ist immer Instanz eines Relationstyps C

Werden Relationstypkomponenten instanziiert, so entstehen Relationen. Diese können aufgrund der (transitiven) Abstammung vom Relations-Metatyp als solche identifiziert werden. Alle **Instanzen von Relationstypen sind damit Relationen**.

Die Definition des Meta-Metamodells in Kapitel 6 stellt die Zusammenhänge zwischen Relationen und Relationstypen im Gesamtmodell dar. Die besonderen Eigenschaften der Relationstypkomponenten erklären sich aus der unterschiedlichen Definition im Metamodell. Relationstypen und Relationen fehlen Eigenschaften wie die Aggregationsfähigkeit, so dass sie nicht als komplexe Komponenten genutzt werden können.

Nach [VarróPataricza 2002] (Fig. 1) können Relationen von übergeordneten Komponenten geerbt werden. Relationen, die für eine allgemeinere Komponente zulässig sind, können auch in den Spezialisierungen verwendet werden. Die Abbildung unten zeigt daher einen Spezialfall. Wurde auf einen Flow-Typ reflexiv eine Relation definiert, so gilt diese auch zwischen seinen Spezialisierungen, da nur vorgeschrieben ist, dass Flow-Typen mit Flow-Typen verbunden werden können:

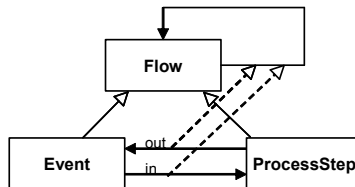


Abbildung 10: Erben von Beziehungen in der Metamodellierung – Beispiel, vgl. [VarróPataricza 2002]

Die stringente Umsetzung des objektorientierten Relationen-Konzepts führt nicht nur zu einer Polymorphie¹⁶ der Komponenten, sondern aufgrund des Relationstypkonzepts auch zu einer **Polymorphie von Relationen**. Es ist also möglich, typkompatible Instanzen von Relationstypen zu verwenden und so beispielsweise spezialisierte Aggregationsbeziehungen zu realisieren.

Basierend auf Modellierungsansätzen wie UML oder WSDL liegt es nahe, für Verbindungen RelationEnd- (UML) oder Port-Konzept (WSDL) zu definieren: Ein **Port** gibt an, welche Verbindungen (Relationen) zu anderen Komponenten bestehen. Die Frage, welche Beziehungen – mit Relationen welches Typs – überhaupt eingegangen werden dürfen, regelt dann ein **Port-Typ**, der sowohl den geforderten Relationstyp als auch weitere Regeln wie die Kardinalität beschreibt. Der Port-Typ regelt also die Art und Anzahl möglicher Ports.

Für die Bearbeitung im Speicher ebenso wie zur Abbildung auf ein XML- oder datenbankgestütztes Konzept erhöht die Verwendung einer derartigen Struktur allerdings die Komplexität. Zudem ist eine Verwendung des Konzepts über mehrere Instanziierungsebenen kaum möglich, da auf jeder Ebene wieder neue Port-Typen definiert werden müssten, so dass das

¹⁶ Verwendung einer Instanz einer Spezialisierung anstelle des geforderten Typs (Polymorphie)

Port-Konzept meist nur in flachen oder Zwei-Ebenen-Modellen verwendet wird. Daher ist das Ziel bei OMICRON, dieses Konzept durch eine entsprechende Metamodellierungssemantik zu ersetzen.

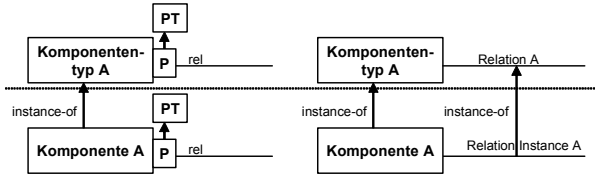


Abbildung 11: Integration der Port- und RelationEnd-Semantik in das Komponenten-Relationen-Modell

Schritt 1: Werden Ports der Ebene E2 mit den Port-Typen für die untergeordnete Instanzebene E1 gleichgesetzt, definieren existierende Relationen und Ports auf Ebene E2 bereits die möglichen Relationen und Ports auf Ebene E1. Ports auf Ebene E1 sind dann quasi Instanzen der Ports auf Ebene E2.

Im zweiten Schritt können die Konzepte der Ports vollständig in die Relation verschoben werden: Kardinalität etc. für die nächste Ebene E1 werden nun direkt mit der Relation A verbunden. Damit definiert Relation A auf Ebene E2 die Rahmenbedingungen für Relationsinstanzen RelationInstance A auf Ebene E1. Die Instanziierung von Relation A liefert alle Modellinformationen wie Typen und Beschränkungen, die sonst mit zusätzlichen Modellbestandteilen definiert werden müssten.

Über das Metamodell kann wesentlich einfacher die Konnektivität und Verbindungstypologie beschrieben werden, die nun zudem auch alle Vorteile der objektorientierten Modellierung aufweist.

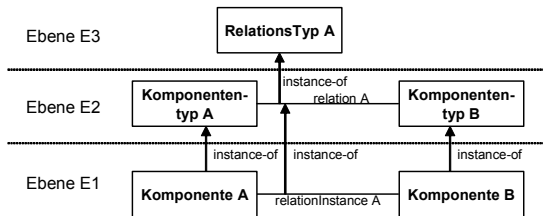


Abbildung 12: Relationsvererbung aus dem Metamodell

Die Definition aller Relationstypen folgt im Gegensatz zu anderen Ansätzen der oben bereits erwähnten integrierten Definition im Metamodell mit Hilfe von **Relationstypkomponenten**. Diese ermöglichen es, alle Konzepte der Komponentenvererbung einschließlich der Mehrfachvererbung auf die Relationstypen zu übertragen. Eine detaillierte Typsemantik für Relationstypen ermöglicht dann eine differenzierte typabhängige Behandlung von Relationen.

Eine Relation erster Ordnung ist Instanz einer Relationstypkomponente. Relationen aller weiteren Ordnungen werden von Relationen instanziiert. Dies ist notwendig, um in Metamodellen verwendete Relationen in Modellen instanziierten zu können und so eine eindeutige Zuordnung zu Kardinalitäten und Restriktionen des Metamodells zu treffen. Relationen erster Ordnung sind auf allen Ebenen möglich.

Relationen können daher in OMICRON weiter instanziiert werden. Der Typ von Relationen kann allerdings nur auf der Ebene der Relationstypkomponenten durch Vererbung differenziert

werden. Wie unten gezeigt, kann nur die Instanziierung auf Relationen angewandt werden, so dass die Vererbung aus Konsistenzgründen für Relationen nicht möglich ist, da Relationen keine Komponenten sind und daher nicht mit (anderen) Relationen verbunden werden dürfen.

Das Konzept der Metamodellierung in OMICRON basiert auf dieser Instanzierbarkeit von Relationen aus dem Metamodell. So sind direkte Vorgaben von erlaubten relationalen Verbindungen zwischen Komponenten allein durch den Vorgang der Metamodell-Instanziierung möglich, die sonst durch ein zusätzliches Regelmodell für erlaubte Verbindungen mit bestimmten Relationstypen integriert werden müssten.

Die Einbettung der Relationstypkomponenten in das Gesamtmodell wird zum besseren Verständnis in Kapitel 6 ausführlicher beschrieben, nachdem die entsprechenden Konzepte und Selbstbezüge/Axiome des OMICRON Meta-Metamodells eingeführt wurden.

Die Separierung der Meta-Relation im Meta-Metamodell von der Komponente als axiomatische Spezialisierung des Modellelements ermöglicht auch die **semantische Unterdrückung einer Verbindung von Relation und Relation im Metamodell**. Die Meta-Relation erhält nur die Vererbung und Instanziierung vom Modellelement, dem Basiselement des gesamten OMICRON Modells. Gleichzeitig können mit der Definition der Meta-Relation und der separaten Meta-Component mit reflexiven Verbindungen nur noch Komponenten – und zwar ausschließlich – über Relationen verbunden werden (siehe OMICRON-Meta-Metamodell in Kapitel 6).

Als reines Komponenten-Relationen-Modell kann nun die Graphentheorie wieder auf das Konzept angewandt werden, was speziell im Hinblick auf die Implementierung viel Vorteile bietet.

4.3.2 Graphstruktur

Hinsichtlich der **Graphform** stellt ein Komponenten-Relationen-Modell zunächst ein Netz ohne hierarchische Einschränkungen dar. Nichthierarchische Systeme sind bei Instanziierung und Vererbung allerdings wegen möglicher Zyklen zum Scheitern verurteilt.

Ob für die Vererbung ein gerichteter, azyklischer Graph (Directed Acyclic Graph, DAG) oder der stärker eingeschränkte Baum zum Einsatz kommt, entscheidet über Mehrfach- (DAG) oder Einfachvererbung (Baum).

Art / Komplexität	Beispiel	Definition	Beispiel Relation
Linear	Listen, Schlangen, Stapel	Es sind keine Verzweigung zulassen, auf ein Element folgt maximal ein weiteres.	list
baumförmig	Stammbaum einer Person, Verzeichnisstruktur	Es existieren beliebig viele untergeordnete, aber nur ein übergeordnetes Element.	Aggregation Einfachvererbung
gerichtet-azyklisch (Directed Acyclic Graph, DAG)	Familienstammbaum, kausale Abläufe	Es dürfen sowohl beliebig viele Elemente untergeordnet (Baum) als auch übergeordnet sein.	Inheritance Mehrfachvererbung
allgemein gerichtet (allgemeiner Graph)	Nahverkehrsplan	das heißt es existieren keine Einschränkungen in der Struktur. Die Richtung von Relationen ist definiert.	Gerichtete Assoziation Ablauf (Flow)
allgemein ungerichtet	Straßennetz (Landkarte) ohne Einbahnstraßen	Es existieren keinerlei Einschränkungen. Die Richtung von Kanten ist nicht definiert.	Ungerichtete Assoziation

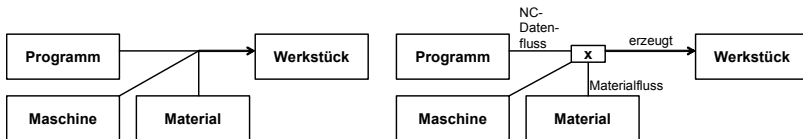
Tabelle 16: Graphtypen in OMIRCON

Die Ausführung und Prüfung von Operationen muss also immer mit Berücksichtigung der maximal möglichen **Komplexität der Graphstruktur** erfolgen. Für die **Konsistenzprüfung** des Modells muss diese maximale Komplexität bekannt sein. Ist beispielsweise eine Mehrfachvererbung durchzuführen, muss das System überprüfen, ob auch nach der Einfügeoperation weiterhin ein azyklischer Graph vorliegt oder durch die neue Verbindung ein

Zyklus entsteht. Für die Definition der unterschiedlichen Graphstrukturen wird auf die Graphentheorie, beispielsweise [AgnarssonGreenlaw 2007], verwiesen.

Hypergraphen, wie sie beispielsweise in SESAM-2 [Reißing 1996] verwendet werden, können Assoziationen mit mehr als zwei Enden bilden, also mehr als zwei Komponenten mit einer Relation verbinden.

Das OMICRON Modell erlaubt nur **binäre Relationen**, da Hypergraphen starken Einschränkungen und einem komplexen Formalismus ohne Definition der Reihenfolge von Relationsenden unterworfen sind. Binäre Relationen sind Relationen mit einem Anfang und einem Ende im Fall der gerichteten (Directed Relation) und zwei Enden im Fall der ungerichteten (Bidirectional Relation). Eine binäre Relation verbindet also immer genau zwei Komponenten oder eine Komponente reflexiv. **Komplexe Relationen** mit mehr als zwei Komponenten werden gegebenenfalls durch die Verwendung einer **Multiplikator Komponente** im Verbund mit binären Relationen ermöglicht. Die Multiplikator Komponente kann im Metamodell neben den Relationstypen auch die Kardinalität definieren. Zudem wird so eine Typisierung von Bestandteilen der komplexen Relationen ermöglicht:



Geringere Kontrolle bei komplexen Relationen (links) im Vergleich zur semantisch verknüpften binären Relationen (rechts).

Abbildung 13: Komplexe Relationen als semantisch verknüpfte, binäre Relationen

Die für Netze aus der theoretischen Informatik verfügbaren Algorithmen sind in der allgemeinen Form nicht für eine nebenläufige Strukturveränderung zur Laufzeit ausgelegt. Daher wird im Folgenden vorgeschlagen, dass über eine entsprechende Sperrung von Teilmodellen während Operationen oder über eine verteilte Sequenzialisierung von Graphoperationen ein konsistenter Zustand sichergestellt wird. Im Index bereits als besucht markierte Elemente (Teilgraph) des Modells dürfen oder können dann nicht verändert werden. Die einfachste Möglichkeit ist die sequentielle Ausführung von Operationen auf dem Gesamtmodell, da die Problematik hier nicht entsteht. Da aber verteilte Umgebungen mit replizierten Modellteilen geplant sind, können nur Mechanismen eingesetzt werden, die eine Abstimmung enthalten oder vollständig transaktionssicher¹⁷ arbeiten.

Eine Zyklenzprüfung (DAG-Kriterium) des Modellgraphen für einen bestimmten Relationstyp kann durch eine markierende Tiefen- oder Breitensuche einfach gelöst werden.

Relationen an sich sind im OMICRON Modell keine Informationsträger über Ihre Verbindungseigenschaft (wie verbundene Komponenten und Kardinalität) hinaus, weil dies dem Konzept der Komponentenbasierung widersprechen würde und so zu einer unendlichen, rekursiven Definition oder nur programmtechnisch zu erfassenden Basiseigenschaften führen würde. Es enthalten also nur Komponenten die Eigenschaft des Informationsträgers und die Möglichkeit zur Aggregation anderer Daten.

Grundlegende, statische beziehungsweise strukturelle Relationstypen wie Vererbung, Instanziierung, Aggregation und Komposition weisen meist eine Baum- oder DAG-Form auf, da diese Hierarchiekonzepte ermöglichen. Eine Flussrelation für Abläufe kann dagegen Zyklen aufweisen, verliert jedoch im Gegenzug die Hierarchieeigenschaft.

¹⁷ Transaktionssicherheit für verteilte, service- beziehungsweise nachrichtenbasierte Systeme ist ein komplexes Themengebiet, das beispielsweise in [Green 2003] und [GreenFurniss 2003] für Web Services erörtert wird.

4.4 Vererbung

„Finally, inheritance is a mechanism by which new classes can be defined as extensions of existing ones.“ [HealyKilgore 1997]

Die Vererbung¹⁸ ist neben der Instanziierung (siehe Kapitel 4.6) das wichtigste Basiskonzept für ein objektorientiertes Modell. In der objektorientierten Programmierung (siehe Kapitel 2.7.2) aber auch in Modellierungssprachen wie der UML [OMG 2007a][OMG 2007b] können Klassifikationen über eine Vererbungsbeziehung mit Hilfe von Typen erfolgen. Im Folgenden wird ihre Übertragung auf Prozessmodelle gezeigt.

Ähnlich wie bei der genetischen Vererbung in der Biologie werden alle strukturellen Informationen mit der Vererbung weitergegeben. In der Objektorientierung erbt das Kind alle Informationen von seinen Eltern und „weiß“ auch, dass es Kind dieser Eltern ist. Vorauszuschicken ist hierbei, dass die Vererbung in einer Form definiert wird, die zur biologischen Vererbung nicht äquivalent ist, da sowohl eine Elternkomponente (Einfachvererbung) als auch mehrere (Mehrfachvererbung) existieren können.

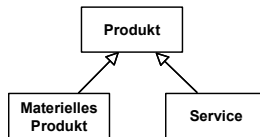


Abbildung 14: Spezialisierende Vererbung im Beispiel in Anlehnung an [Mili et al. 2004]

Die UML verwendet die zur Spezialisierung entgegengesetzte Beziehung der Generalisierung, um jede Art von Begriff („Classifizier“) zur verallgemeinern. (vgl. Fig. 9.22 in [OMG 2007b] und Anhang A).

Semantische Konstrukte wie semantische Netze, Taxonomien, Thesauri etc. behandeln wie auch Ontologien in der Informatik meist nur atomare Begriffe beziehungsweise Komponenten. Die Auswirkungen einer Verwendung von **komplexen Komponenten** wie Prozessen in der Vererbung werfen viele zusätzliche Probleme auf und sind ein Hauptaspekt dieser Arbeit, der im Folgenden vertieft wird.

Von einer Komponente können also im Rahmen der (Sub-)Klassifikation beliebig viele Spezialisierungen, beispielsweise Varianten, erzeugt werden. In der Praxis besteht häufig auch die Notwendigkeit, eine Komponente als Spezialisierung mehrerer übergeordneter Komponenten aufzufassen, um die korrekte Bedeutung und Verwendbarkeit in der Produktion wiederzugeben. Kann beispielsweise ein Bearbeitungszentrum Aufgaben von Fräs- und Bohrmaschinen erfüllen, entsteht eine Hierarchie mit Mehrfachvererbung:

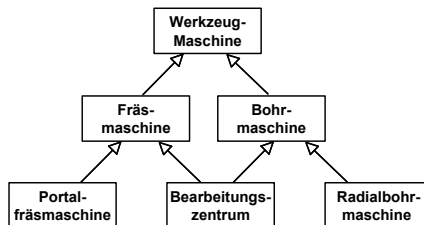


Abbildung 15: Mehrfachvererbung durch Integration von Fähigkeiten

¹⁸ Zum Thema Vererbung in der objektorientierten Programmierung (OOP) wird auf die gängige OOP-Literatur wie [LahresRayman 2006] verwiesen.

Speziell diese Mehrfachvererbung führt bereits bei den nur einfach instanzierbaren (siehe Instanziierung im nächsten Abschnitt) und auf eine Aggregationsebene beschränkten Klassen der objektorientierten Programmierung zur unklaren Ergebnissen. In der Softwareentwicklung aber auch in der Modellierung werden daher häufig technologievereinfachende Vererbungskonzepte genutzt. Unter Vererbung ist hier immer die vollständige Vererbung – in der OOP **Implementation Inheritance** genannt – zu verstehen, die nicht nur Schnittstellen, sondern die **vollständigen Abläufe von Prozessen** bzw. Programmen vererbt.

Bei der **Einfachvererbung** existiert nur eine Vaterkomponente von der die Kindkomponente unmittelbar erbt, wohingegen das Erben von mehreren Komponenten nur transitiv von der Vaterkomponente der Vaterkomponente erfolgt. Die Kindkomponente erbt somit exakt alle Eigenschaften der Vaterkomponente.¹⁹ Eine reine Einfachvererbung ermöglicht einer Kindkomponente nur das Erben von exakt einer Vater-Komponente. Hierdurch entstehen leichter zu handhabende Vererbungsstrukturen in **Baumform**²⁰, da Mehrdeutigkeiten aufgrund der Unterordnung mit einer 1:n-Beziehung semantisch verhindert werden. Aufgrund der Eindeutigkeit der Vater-Kind-Beziehung können Inhalte direkt und ohne Kollisionen übernommen werden, so dass die Kind-Komponente bis auf explizite Änderungen exakt der Vaterkomponente gleicht. Das Erben von mehreren Komponenten ist jedoch ausgeschlossen.

Bei der **Mehrfachvererbung** existieren mehrere Elternkomponenten, da die Komponente von mehreren Komponenten unmittelbar erbt. Eine Überschneidung ist also möglich und wahrscheinlich. Die Graphstruktur bleibt aufgrund der Semantik weiterhin azyklisch und gerichtet, das heißt ein gerichteter azyklischer Graph (DAG). Es kann nun jedoch jede Komponente von mehreren anderen Komponenten erben, so dass eine übergeordnete Komponente häufig über mehrere Wege an ein Kind vererbt. Die Vererbung von Teilelementen ist damit nicht mehr eindeutig beziehungsweise wird redundant.

Für die Zusammenführung mehrerer Klassen, von denen geerbt wird, existieren im Bereich der Programmiersprachen unterschiedliche Konzepte wie:

- Bei Äquivalenz Zusammenlegung der Teile (EIFFEL)
- Manuelle Zusammenlegung von Teilen (EIFFEL, CLOS)
- Auswahl der zu nutzenden Komponenten (C++)
- Auswahl des ersten Vorkommens in der Liste durch Tiefensuche in der Vererbungshierarchie (Perl)

Da für Prozesse meist nicht automatisch entschieden werden kann, welche Teile unnötig sind, setzt OMICRON auf eine **Duplizierung der Prozesse**. Eine Anpassung kann dann manuell erfolgen, beispielsweise durch Verkettung und zusätzliche Verbindungen zwischen den Teilen. Das Basis-Modellkonzept selbst kann jedoch darauf aufbauend prinzipiell jede der Optionen umsetzen – auch parallel unter Nutzung unterschiedlicher Spezialisierungen der Vererbungsbeziehung oder rein algorithmischer Vererbung.

Hierdurch wird auch die Problematik der transitiven Mehrfachvererbung von derselben Komponente gelöst beziehungsweise zur manuellen Anpassung freigegeben werden.²¹ Ist eine

¹⁹ Unterschiedliche Sichtbarkeitsbereiche in der Vererbung wie private, protected etc. existieren zwar bei der OOP und UML-Klassendiagrammen werden im vorliegenden Fall jedoch zunächst von der Betrachtung ausgenommen, da trotzdem alle Bestandteile weitergegeben werden.

²⁰ Jede Komponente kann maximal eine übergeordnete, jedoch beliebig viele untergeordnete Elemente haben. Da die UML eine Generalisierungsbeziehung vorsieht, entsteht eine umgekehrte Baumstruktur mit beliebig vielen Pfeilenden an jeder Komponente, aber nur einem Pfeilbeginn.

²¹ Da die Vererbung meist als zeitunkritische Modellierungs- und nicht als zeitkritische Ausführungsaufgabe auftritt, könnten auch aufwändigere Vergleichsverfahren durchgeführt werden, die eine automatische Zusammenführung

reine Duplikation nicht erwünscht, können kompatible Komponenten fusioniert werden, so dass mehrfach verwendete Felder zu einem verschmolzen und Prozesse synchronisiert und integriert werden können. Dies kann jedoch nicht automatisiert werden, da hierzu Intentions- und Domänenkenntnis notwendig sind.

Die Mehrfachvererbung ist in einem zur Laufzeit änderbaren Modell ein sehr komplexes Thema, dessen Ausschluss das Modell vor allem in der Verarbeitung extrem vereinfachen würde. Im Folgenden werden daher existierende Alternativen zur reinen Mehrfachvererbung kurz erläutert.

Aggregation mit Einfachvererbung ermöglicht den Aufbau einer baumförmigen Vererbungshierarchie. In Spezialisierungen können weitere Prozesse eingebunden werden. Der Typ wird aber immer durch den Vererbungsprozess bestimmt und bleibt hierdurch eindeutig. Problematisch ist dabei jedoch, dass ein Prozess nicht mit anderen zu einem neuen Typ kombiniert werden kann. Das folgende Beispiel beschreibt diese Problematik.

Beispiel 3: Ein Prozess 1&2 „Produziere Grundplatte und Bolzen“ müsste entweder von Prozess 1 „Produziere Grundplatte“ oder von Prozess 2 „Produziere Bolzen“ erben. In beiden Fällen würde der jeweils andere Typ verschwinden und so einen semantischen Nachrichtenaustausch (siehe 9.2 Nachrichtenbasierung – das Semantic-Messaging) blockieren. Erbt Prozess 1&2 von Prozess 1 enthält er alle Abläufe und Ergebnisse von Prozess 1 – zusätzlich jedoch noch einen zweiten, durch Aggregation eingefügten Block aus Prozess 2. Prozess 1&2 bleibt ein Subtyp von Prozess 1 während Prozess 2 keine Berücksichtigung findet. Eine semantische Prüfung und Auswahl kann nur über Prozess 1 erfolgen, so dass die Fertigung des Bolzens durch Prozess 2 nicht auffindbar wäre. Mit Hilfe von Referenzen können zwar beide Prozesstypen 1 und 2 eingebunden werden, der Gesamttyp 1&2 bevorzugen dann aber entweder Prozess 1 oder Prozess 2 oder es entsteht als Aggregat ein neuer Typ, der ebenfalls keine Rückschlüsse auf die Semantik von Prozess 1&2 zulässt.

Auch für Rollenmodelle und überlappende Produktionsschritte scheidet diese Methodik aus demselben Grund aus.

Selbst wenn der vererbende Prozess den überwiegenden Teil ausmacht, kann häufig nicht auf Mehrfachvererbung verzichtet werden: „Produziere Fahrzeug und fertige Prüfbericht aus“ enthält typologisch neben der vollständigen Produktion eines Fahrzeugs auch den Prüfbericht als (weiteres) Ergebnis, das semantisch relevant ist.

Sollen Prozessschritte eingefügt werden, die den Typ des Prozesses nicht bestimmen (beispielsweise die Generierung des Ereignisses „Freigegeben“ am Ende der Montage), so können diese tatsächlich über Referenzierung und Aggregation erstellt werden. Die Aggregation komplementiert dann die Vererbung durch **typneutrales Hinzufügen**.

Viele moderne Programmiersprachen, wie die auf der Common Language Infrastructure (CLI, [ISO 2006]) oder auf dem daraus weiterentwickelten Microsoft .NET basierende Programmiersprachen, ebenso wie Java unterstützen nur eine einfache Vererbung, ergänzt um ein Konzept zur **Schnittstellenvererbung** (Interface Inheritance).

Wie der Name sagt, ist auch hier nur eine Einfachvererbung vorgesehen. Sollen weitere Typen hinzugefügt werden, dürfen diese keine vollständigen Komponenten oder Prozesse (in der OOP-Terminologie Klassen) sein, sondern nur Schnittstellendefinitionen, quasi Signaturen (siehe Kapitel 5.2.2) der Komponente, die noch mit Inhalten gefüllt („implementiert“) werden müssen. Dies hat den Vorteil, dass mehrere Schnittstellen verwendet werden können, jedoch inhaltliche Kollisionen vermieden werden.

Schon die Beschreibung zeigt hier allerdings das Problem auf: Schnittstellenvererbungen sind leere Vererbungen, die nur einen Typ und die Schnittstelle weiterer Klassen beziehungsweise Komponenten zur Verfügung stellen: Jegliche Ablaufbeschreibungen innerhalb von Methoden

durch Projektion der Vererbungshierarchie ermöglichen würden. Dies kann für andere Anwendungsgebiete möglicherweise sinnvoll sein.

sind nicht zugelassen. Die Übertragung auf das Prozesskonzept zeigt: Gerade innere Prozessschritte, die die Semantik und die Kern-Abläufe eines (Sub-)Prozesses ausmachen stehen in dieser Form der Vererbung nur für eine Komponente zur Verfügung: [Montage Teil A] + [Montage Teil B] = [Montage AB] ist per Schnittstellenvererbung nicht realisierbar. Nur eine Schnittstellenvererbung verbunden mit einem aggregierten Hinzufügen würde das gewünschte Verhalten der Mehrfachvererbung annähernd simulieren können.

Dieses Beispiel zeigt, dass erst die Erzeugung eines Subtyps [Montage AB] von beiden gleichberechtigten Typen [Montage A] und [Montage B] durch **Mehrfachvererbung** korrekte semantische Operationen zulässt. Die Mehrfachvererbung ermöglicht hier die dedizierte Fusion von Prozessen bei vollständiger Erhaltung beider Ausgangstypen in der Erbinformation.

Neben der objektorientierten Vererbung existieren viele weitere Ansätze zur Strukturierung von Modellen und Systemen. Die **Objekttaggregation** (Object Aggregation) bildet neue Komponenten aus vorhandenen Objekten indem eine Referenz auf ein Objekt hinzugefügt wird, das nun in der Komponente enthalten ist (Referential Containment). Dieses vermutlich auf der Mengenlehre basierende Konzept wird in OMICRON ergänzend zur Vererbung genutzt, beispielsweise um eine Komponente in einer komplexen Aktivität mehrfach verwenden zu können.

Die **Objektkomposition** (Object Composition) verwendet anstelle der Aggregation mit Referenz einen Einschluss der tatsächlichen Komponente, so dass diese nur einmal verwendet werden kann, was eine starke Begrenzung darstellt. Das OMICRON Konzept erlaubt eine Verwendung der Objektkomposition allerdings mit beliebiger Anzahl von Verknüpfungen. Während aus Aggregaten Teile verwendet werden können, müssen Komposita so bleiben, wie sie sind.

Beispiel 4: Ein Lagerbock ist im fertigen Zustand beispielsweise ein Kompositium aus Grundplatte, angeschweißter Stellplatte und Lager. Der Bausatz für den Lagerbock kann sowohl als Kompositium als auch als Aggregat interpretiert werden. Ein Paket mit Lagern stellt somit nur ein Aggregat dar. Wird aus dem Paket ein Lager entnommen existiert das Aggregat weiter.

Viele weitere, weniger komplexe Ausprägungen von Beziehungen zwischen Komponenten existieren und sind auch in OMICRON als Relationstypkomponenten verfügbar. Die Vererbung stellt jedoch das semantisch bei weitem stärkste, allerdings auch komplexeste Konzept dieser Art dar.

4.5 Referenzierung und Referenzen

Die Literatur zu Objektorientierung geht meist über eine Besonderheit hinweg, die für ein vollständig objektorientiertes Prozessmodell Probleme aufwirft. Geerbte Bestandteile, die nicht überschrieben werden, existieren nur in der jeweiligen Oberklasse O. Dies ist jedoch bei Prozessen nicht ausreichend. Eine geerbte Teilkomponente b muss als eine Art Spiegelung auch in einer Spezialisierung S von vorkommen – als Referenzkomponente.

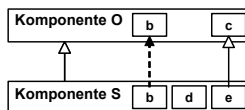


Abbildung 16: Verwendung von Feldern bei der Vererbung

Die Bestandteile sind wie Felder in der OOP zu betrachten. Im Folgenden wird gezeigt, warum die Referenzierung für die Verwendung in Komponenten eingesetzt werden muss.

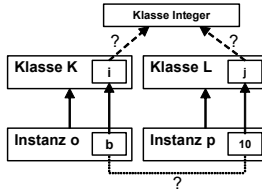
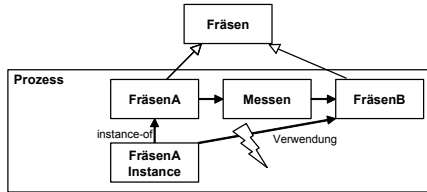


Abbildung 17: Inkompatibilität²² bei Vererbung als Verwendungsrelation – Notwendigkeit der Referenzierung

Ein als Integerwert definiertes Feld „i:integer;“ einer Klasse K wird in der Instanz o von K als Integer instanziiert, das heißt der Typ von b in der Instanz ist Integer und nicht „b Subtyp von Integer“. Mit Hilfe der Vererbung ist also das Feld b nicht hinzufügbare. Eine Aggregation von Integer (nicht von b!) würde zu einer korrekten Instanziierung führen, bedeutet aber, dass die Klasse Integer innerhalb der Klasse K definiert wird. Wird nun ein Integer j in einer anderen Klasse L verwendet, wäre Integer bereits in zwei Klassen verankert und müsste dann (nun vor allem im Fall von Prozessschritten gedacht) auch alle Verbindungen zu anderen (Sub-)Komponenten innerhalb von K und L enthalten.

Parallele Subtypbildung in mehreren Komponenten nur zum Zwecke der „Verwendung“ einer Aktivität in anderen würde direkt zu Typinkompatibilität führen.

Beispiel 5: Feld a:integer kann nicht Feld b:integer zugewiesen werden, weil beide Subtypen von Integer und nicht Integer sind!



FräsenA ist wegen Vererbung nicht typkompatibel mit FräsenB

Abbildung 18: Vererbung als Lösung für Verwendung scheidet aus

Die Instanzbildung von Name-Wert-Paaren erscheint zunächst als mögliche Lösung: Bei jeder Verwendung eines Typs wird eine Instanz der verwendenden Klasse gebildet, so beispielsweise aus [Integer]:[Feld] ein [a]:[Integer], das zu [10]:[a] auf der Instanzebene wird. Name-Wert-([Wert]:[Name]) beziehungsweise Typ-Name-Werte ([Name]:[Typ]) können so jeweils instanziiert werden. Die Typkompatibilität ist gesichert, da [a]:[Integer] mit [b]:[Integer] auf der höheren Instanziierungsebene den Typ gemeinsam hat. Diese Lösung ist jedoch nur scheinbar, da Instanzen von Instanzen desselben Typs nicht kompatibel sind.

Für die OOP kann dieses Konzept aufgrund deren Beschränkung auf Instanzebenen (die Klassen- und die Objektebene) und den Aufbau von Klassen als einstufiges Aggregat von Feldern und Methoden eine semantische Erklärung liefern.

Für die Prozessmodellierung mit verschachtelten Aktivitäten ergibt sich jedoch das Problem, dass mit jeder Verwendung auch wieder eine Instanziierung erfolgen muss. Schon wenn [Aktivität a]

²² Die Objektorientierung folgt in der Vererbung der Definition von Kohyponymie in der Linguistik: Unterklassen oder -begriffe (B, C) einer Oberklasse A beziehungsweise eines Oberbegriffs sind inkompatibel: B is-a A, C is-a A; jedoch: B not is-a C, C not is-a B.

eine [Aktivität a1] nutzt, die ihrerseits [Aktivität a1.2] verwendet wird klar, dass kein gemeinsames Metamodell mehr verwendet werden kann, wenn dieses rekursiv verwendet wird, solange die Instanziierung des Metamodells zum Modell als Axiom angenommen wird.

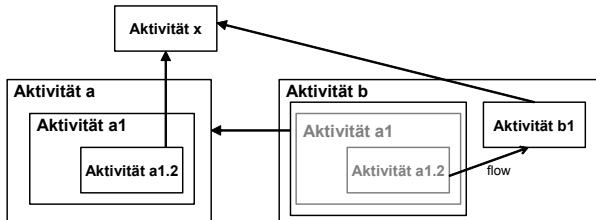


Abbildung 19: Verwendung mit Hilfe der Instanziierung verursacht eine Durchmischung von Instanzebenen

Wird beispielsweise [Aktivität a1.2] auch in [Aktivität b] genutzt, die mit [Aktivität a] auf derselben Instanzebene liegt, kann bei Verwendung von [Aktivität a] in [Aktivität b] (durch Aggregation) eine Aktivität [Aktivität b1] mit einer Instanz von [Aktivität a1.2] eine Flussbeziehung eingehen und gemeinsam mit dieser weiter instanziiert werden – ein Paradoxon wäre die Folge, da [Aktivität b1] eine Instanzebene höher liegt als [Aktivität a1.2] und trotzdem mit dieser per Flussrelation verbunden ist.

Die einzige Lösung für das Komponentenkonzept liegt in einer **Referenzierung** des gewünschten Typs mit Hilfe einer Referenzkomponente. So dass die Instanziierung als Integer quasi „vorgemerkt“ werden kann.

Dieses Konzept entspricht dem in der Softwareentwicklung ebenfalls verwendeten Konzept der **Zeiger (Pointer)**, das hier im Sinne der Objektorientierung fortgeführt wird.

In einer Aktivität beziehungsweise Prozessbeschreibung müssen Referenzen auf die verfügbaren Prozesselemente verwendet werden, da bei Vererbung nicht derselbe Aktivitätstyp zweimal verwendet werden könnte (nur jeweils ein neuer Subtyp): Flussrelationen zwischen zwei Aktivitäten desselben wären dann nur durch Subtypbildung oder Instanziierung möglich.

Eine Referenzkomponente hat dieselbe Funktionalität und interne Struktur wie das Original, jedoch eigene externe Relationen, die sie mit einer vollständig anderen „Außenwelt“ verbinden können. Die Referenzkomponente ist eine Projektion der Originalkomponente, die jedoch ihre externen Relationen²³ als abundante Attribute [Stachowiak 1973] zusätzlich aufweist und die externen Relationen der Originalkomponente als präterierte Attribute [Stachowiak 1973] weitgehend vernachlässigt.

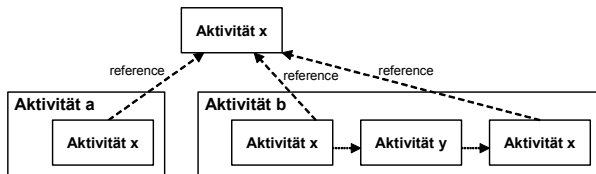


Abbildung 20: Referenzierung als Lösung für die mehrfache Verwendung von Prozessschritten in Aktivitäten

²³ Eine Relation, die Komponente B, mit der Referenzkomponente A beziehungsweise einer ihrer Teilkomponenten verbindet, wobei B nicht zur Menge der von A (rekursiv) aggregierten Teilkomponenten einschließlich ihrer selbst gehört.

So kann dieselbe Komponente beliebig oft als Feld oder Prozessschritt auf einer Ebene und auch in derselben Komponente verwendet werden ohne dass sich eine unerwünschte Seiteneffekte wie die Durchmischung von Instanzebenen oder externen Relationen sowie Typinkompatibilitäten ergeben.

4.6 Instanziierung und Typen

Modellinhärent haben Komponenten einen bestimmten Typ. Dieser wird durch Instanziierung erlangt. Er kann durch Mehrfachvererbung auf der übergeordneten Klassenebene mehrere Typen vereinen und kann nur durch geänderte Vererbung – nicht durch Überschreiben – geändert werden. Dieser Typ kann zu einer eindeutigen Typidentifikation und auch für einen Subtyp-basierten Kompatibilitätsvergleich herangezogen werden. Das heißt, für eine erwartete Komponente kann eine typkompatible andere Komponente, ein kombinierter Typ oder ein Sub-Typ, eingesetzt werden (vgl. Abschnitt zur Polymorphie).

Die Instanziierung muss baumförmig erfolgen, da eine Instanziierung von mehreren Klassen wie bei der Mehrfachvererbung nicht klarstellen würde, welche Eigenschaften die Instanz im Falle von Überschneidungen und welchen (speziellen) Typ im Falle von Vergleichen die Instanz hat. Zudem können alle Fälle dieser Art auch über eine Mehrfachvererbung bereits auf Klassenebene abgedeckt werden, so dass ein semantischer Mehrwert nicht gegeben ist und Mehrdeutigkeiten Vorschub geleistet würde. Wichtig ist vor allem: Eine Instanziierung erzeugt immer eine neue Komponente. Jedes Element im Gesamtmodell darf und muss daher Instanz genau einer anderen Komponente sein.

Zwei über die Instanzbeziehung verbundene Komponenten gehören automatisch zwei hierarchisch untergeordneten Instanzebenen. Daher können sie keine anderen Beziehungen miteinander eingehen. Instanziierung ist damit bereits inhärent irreflexiv.

Die einzige Ausnahme ist die Basiskomponente des Gesamtmodells: ModelElement. Da die Instanziierung keine Zyklen erlaubt, muss sie an einem oder mehreren Punkten beginnen. Die Komponente ModelElement verfügt daher über eine reflexive Instanzbeziehung mit sich selbst (siehe Kapitel 6), die sie nach Instanziierung an die Meta-Komponente und -Relation weitergibt.

4.6.1 Instanziierung über mehrere Ebenen

[AtkinsonKühne 2002] motivieren und definieren zusätzliche Metamodellierungskonzepte für die Instanziierung über mehrere Ebenen so: „*Traditional instantiation is sufficient when dealing with only two levels (e.g., classes and objects) but should ideally be enhanced for a multilevel instantiation hierarchy. [...] The problem with traditional instantiation is that a type may only specify properties of its direct instances but has no bearing on, for example, the instances of its instances. This is why we refer to it as shallow instantiation.*”²⁴

Dies ist jedoch nur teilweise korrekt. Gerade bei einer potentiell unendlichen Instanzierungshierarchie kann eine Ebene gar nicht beliebig viele untergeordnete Ebenen beeinflussen. Vielmehr beschreibt jede Typebene jeweils die Konzepte für die Instanzebene.

Es werden weitere Konstrukte vorgeschlagen, die angeblich ein konsistentes Gesamtmodell ermöglichen: **Powertypes** [Gonzalez-PerezHenderson-Sellers 2006] (siehe Anhang E) zwingen untergeordnete Elemente von der Powertype-Referenz zu erben. Die Klassenebene gibt also jeweils für die Instanzebene vor, dass quasi ein Wurzelement zu erzeugen ist, das auf

²⁴ Hier überwiegt grundsätzlich also wieder das Denkmodell von UML / MOF, was [AtkinsonKühne 2002] teilweise auch selbst reinräumen: „*Interestingly, though, adding the unification of physical classifiers and deep instantiation to the picture removes the need for the first proposal, the distinction between logical and physical classification*”. [VarróPataricza 2002] zeigen für einen auf Deep Instanciation basierenden Metamodellierungsansatz eine Formalisierung mit Beweis, der mit Abwandlungen auf dasOMICRON Konzept übertragen werden könnte.

Instanzebene für alle Instanzen als Vaterknoten Rahmenbedingungen vorgibt. Dies erwies sich jedoch bei korrekter Vererbung und Metamodell-Bildung in OMIRCON als unnötig.

Die dem OMICRON Metamodell zugrundeliegende Erkenntnis, dass die Zahl der Instanzenebenen potentiell beliebig sein kann findet in unterschiedlichen Ansätzen zur Metamodellierung Beachtung. Um das Zwei-Ebenen-Modell der Instanziierung (Klasse und Instanz in der OOP) auf beliebig viele Ebenen zu erweitern, können nicht mehr unterschiedliche Eigenschaften der Klassen und der Instanzebene beibehalten werden, wie dies in UML und ähnlichen Ansätzen aus der Softwareentwicklung geschieht. Da die Instanzebene gleichzeitig die Klassenebene für die nächste Instanzebene ist, müssen die Konzepte vereinheitlicht werden.

Das **Mehrfachinstanziierungs-Konzept** (deep instantiation) wird als eine mögliche Antwort auf dieses Problem der Kontrolle über mehrere Instanziierungen hinweg in [AtkinsonKühne 2002] vorgestellt: *"Deep instantiation is a conservative extension of the traditional two-level, shallow instantiation mechanism. It subsumes shallow instantiation but enhances it to allow information to be carried over more than one instantiation step. The key idea is to assign a potency value to model elements and their fields, indicating how many times they can be instantiated."*

Ein Element soll also über mehrere Instanzenebenen hinweg instanziiert werden können. Mit diesem Potenzierungselement wird eine starke Beliebigkeit gefördert, die [AtkinsonKühne 2002] selbst im eigenen Beispiels zeigen: Die Anzahl Instanziierungen wird willkürlich als Potenzwert manuell festgelegt. Da jedoch in der Produktion und Laufzeitmodellierung von einem nicht einfach änderbaren Metamodell auszugehen ist (es existieren Instanzen), führt ein solcherart beliebiges Vorgehen zu Inkonsistenzen und unüberschaubaren Seiteneffekten. In OMICRON wird dieses willkürliche Konzept der Potenzierung daher durch ein inhärentes objektorientiertes Konzept der Instanzbildung ersetzt, das über alle Ebenen gleichermaßen gilt. Jede Instanzebene wirkt auf die untergeordnete Instanzebene als Klassenebene.

Jede Instanziierung führt für Komponenten automatisch auf die nächsttiefere Instanziierungsebene. Somit sind **Intra-Level-Instanziierungen**, wie sie ebenfalls in [AtkinsonKühne 2002] besprochen werden in OMIRCON nicht möglich. Auch wenn die Ebenen für die Relationen wiederum teilweise durchlässig sind (Instanziierungsrelationen!), ist eine Instanziierung für Komponenten automatisch der Schritt auf die nächsttiefere Ebene.

4.6.2 Prozess-Metamodelle

Die Realisierung von Prozessen (Prozessmodellen) nach einem oder mehreren Prozess-Metamodellen wurde in einigen Methoden bereits praktiziert und zeigt mehrere Vorteile [Rolland 1998]: *„The exploitation of the meta-model helps to define a wide range of process models. It makes the activity of defining process models systematic and versatile. It forces to look for and introduce, in the process meta-model, generic solutions to problems and this makes the derived process models inherit the solution characteristics.“* (vgl. auch Anhang C)

Auch im Hinblick auf für MDA-Ansätze wichtige domänenspezifische Sprachen sind Metamodelle hilfreich. Ein Prozess-Metamodell definiert dabei nach [Rolland 1998] die gemeinsamen, generischen Eigenschaften von Prozessmodellen und repräsentiert diese in einem Konzept-System, so dass es hilft, Prozessmodelle gleicher Eigenschaften generieren.

Die Instanziierung ist neben der Vererbung das grundlegende Konzept für laufzeitbasierte Prozesse in der Produktion. Auf Metamodell-Ebene können die grundlegenden Zusammenhänge wie Rollenmodell, Maschinenklassifikation etc. definiert werden. Die Instanziierung dieser Metamodelle liefert die Typebene der Produktion, die beispielsweise Maschinentypen und deren Spezialisierungen enthält. Selbst die Beziehung zwischen Maschinentypen und Rollen kann so definiert werden. Beispielsweise wird die „Rolle Gabelstaplerfahrer“ dem „Gerät Gabelstapler“ als einzige zugeordnet.

Über diese Typebene können dabei die Regeln und Rahmenbedingungen ebenso wie die eigentliche Klassifikation von Bestandteilen der Produktion geregelt werden. Maschinen und Werkstücke selbst können dann komfortabel als Instanzen der definierten Maschinen- und Produkttypen verwaltet werden, so dass für jedes Element des Produktionssystems die semantischen Bezüge definiert sind.

Auf der Instanzebene können dann Maschinen zu Produktionslinien oder Zellen zusammengefasst werden. Selbst diese Ebene kann wieder als Modell aufgefasst werden, indem für aggregierte Sensoren oder Prüfberichte einer Maschine die Kardinalität verfügbarer Werte für die nächste Instanzebene definiert wird.

Die Notwendigkeit mehrerer Instanzebenen für eine flexible und doch geregelte Modellanimation ist also offensichtlich.

4.6.3 Polymorphie

Die Polymorphie (Vielgestaltigkeit) ist ein mächtiges (Laufzeit-)Konzept der Objektorientierung und ermöglicht die Verwendung eines Subtyps beziehungsweise einer Spezialisierung A' eines Typs A an Stellen die Komponenten vom Typ A fordern. Es wird in OMICRON ebenfalls auf Prozesse übertragen.

Beispiel 6: Wird ein Datum gefordert, kann auch ein Prüfdatum eingesetzt werden, da es sich beim Prüfdatum um eine Spezialisierung von Datum handelt. Die Verwendung eines Lieferdatums statt eines Prüfdatums ist dagegen nicht möglich.

Die Polymorphie ermöglicht eine beliebige Erweiterung von Vererbungshierarchien für deren Instanzen trotzdem Regeln und Verwendung der übergeordneten Elemente weiter gelten können. Da eine Spezialisierung grundsätzlich über die Eigenschaften einer allgemeineren Komponente verfügt, die nur verfeinert oder ergänzt wurden, kann eine Instanz der Spezialisierung auch anstelle der einer Instanz des allgemeineren Typs eingesetzt werden. Die Instanz der Spezialisierung wird dann im Kontext so behandelt als handle es sich um eine Instanz des geforderten allgemeinen Typs.

Beispiel 7: Wird allgemein eine NC-Drehmaschine im Prozess gefordert, kann sowohl eine NC-Drehmaschine mit einer Siemens-Steuerung, als auch mit einer Heidenhain-Steuerung verwendet werden, da beide Instanzen von Spezialisierungen der allgemeinen NC-Drehmaschine darstellen.

Umgekehrt kann ein Prozess allgemein für Drehteile definiert werden und schließt dann alle Typen (Unter-Arten) von Drehteilen mit ein. Diese Zusammenhänge müssen nicht programmiert werden, sondern gehen direkt aus dem jederzeit zur Laufzeit erweiterbaren Gesamtmodell hervor. Wo „spanende Bearbeitung“ vorgesehen ist, kann Fräsen als Spezialisierung eingesetzt werden.

Die Polymorphie ist ein grundlegendes Konzept der Objektorientierung, das sich allerdings erst auf Instanzebene auswirkt. Daher war bisher meist nur die objektorientierte Programmierung betroffen. Die Modellierung nimmt kaum Kenntnis von diesem Phänomen, da Instanzen zum Modellierungszeitpunkt nicht existieren. Auch für die industrielle Praxis wird es aufgrund der Komplexität und hohen Abstraktion bei einer Laufzeitimplementierung und mangels realisierbarer Konzepte bisher nicht eingesetzt.

OMICRON sieht als Laufzeitmodell die Polymorphie sowohl von Elementen als auch von Prozessen vor, um vom oben erwähnten Szenario eines allgemeinen Prozesses für mehrere spezielle Teile- und Maschinentypen zu profitieren.

Wird in OMICRON ein bestimmter Komponententyp an einer Stelle im Modell verwendet, können alle seine Subtypen / Spezialisierungen an dessen Stelle verwendet werden. Dieser Sachverhalt hat komplexe Implikationen auf die Modellkonzeption und stellt eine der großen Stärken objektorientierter Konzepte dar. Im Laufzeitsystem wird Polymorphie ermöglicht, indem anstelle einer geforderten Komponente vom Typ A Instanzen von jeder ihrer Spezialisierungen

A' (Kindern in der Vererbungshierarchie) zugelassen werden. Eine Überprüfung auf im aktuellen Kontext polymorphiefähige Subtypen kann anhand des Modells automatisch durch das Laufzeitsystem erfolgen.

Durch Mehrfachvererbung (vgl. Kapitel 4.4) können auch Subtypen mehrerer Komponenten erzeugt und verwendet werden, beispielsweise ein Bearbeitungszentrum als Spezialisierung mehrerer Typen von Werkzeugmaschinen (siehe Abbildung 15, S. 78).

Bei der Polymorphie kann immer nur die Spezialisierung dem Allgemeineren zugewiesen werden und nicht umgekehrt, da sie asymmetrisch ist. Die Zuweisung von Komponenten desselben Typs ist immer möglich.

Beispiel 8: Ist der Tag (eines Monats) eine Spezialisierung von Nummer, so kann, überall wo allgemein eine Nummer gefordert wird, ein Tag zugewiesen werden. Wo ein Tag (1..31) gefordert ist, muss aber ein solcher und nicht eine beliebige Nummer zugewiesen werden.

Wertebereiche spielen bei der Polymorphie also eine wichtige Rolle. Eine Komponente und auch eine Relation können aufgrund der Polymorphie immer einen **Instanzkontext** und eine tatsächliche **Instanzbeziehung** aufweisen. Die Instanzbeziehung gibt dabei den tatsächlichen Typ der Komponente an, das heißt den Typ, dessen Instanz sie ist (**Realtyp**). Der Instanzkontext gibt an, als welcher Typ die Komponente – gegebenenfalls auch polymorph – verwendet wird (**Kontexttyp**). Wird sie typgleich verwendet, stimmen Instanzkontext und Instanzbeziehung – möglicherweise transitiv über Referenzkomponenten – überein. Liegt polymorphe Verwendung vor, sind sie unterschiedlich: Die Instanzbeziehung bestimmt noch den Realtyp, der Instanzkontext hat einen allgemeineren Kontexttyp.

Beispiel 9: Wird in einem Prozess für eine Aktivität „Fräsen F“ eine Fräsmaschine zur Ausführung gefordert, dann aber ein Bearbeitungszentrum b eingesetzt (siehe Abbildung 25, S. 94), ist der Kontexttyp von b „Fräsmaschine“ in der Aktivität „Fräsen F“, jedoch die den Realtyp bestimmende, eigentliche Instanzbeziehung von b immer noch „Bearbeitungszentrum“.

Bei Polymorphie wird die spezialisierte Komponente [A'] statt [A] verwendet. Findet in der OOP ein Aufruf der Funktion func statt, wurde dieser ursprünglich als [A].func definiert. Es soll jedoch nun [A'].func (der in A' überschriebene Ablauf von A) aufgerufen werden. Hierzu werden in der OOP zur Laufzeit **virtuelle Methodentabellen** genutzt, die die Funktion func der korrekten Komponente A' zur Verfügung stellen. Damit wird die sogenannte dynamische Polymorphie beziehungsweise die „Laufzeitbindung“ ermöglicht, die erst zur Ausführungszeit feststellt, welche Methode aufgerufen werden muss.

Um Komplikation zu vermeiden sind bei OMICRON alle Prozesse in Komponenten **virtuell** definiert. Das heißt im Falle von Polymorphie (Verwendung von A' anstelle von A) wird immer der in A' definierte Ablauf durchgeführt, auch wenn A' polymorph „vorgibt“ A zu sein: Wird bewusst eine Spezialisierung eingesetzt, werden auch deren Abläufe ausgeführt.

Die Verkettung von polymorphen Prozessen wird in Abschnitt 5.2.3 beschrieben.

4.6.4 Konstanten und Defaultwerte

Konstanten sind **unveränderbare Werte eines bestimmten Typs**. Meist handelt es sich in objektorientierten Systemen um Datenfelder, die mit einem bestimmten Wert initialisiert werden und dann unveränderbar sind.

Auch für Abfragen und Verzweigungen wie ($X > 0$) müssen bereits auf Klassenebene Instanzwerte in Form von Konstanten (in diesem Fall die 0 als Instanz von Zahl) verwendet werden können.

Hinsichtlich der Instanzierung stellen Konstanten ein besonderes Konzept dar. Sie sind Instanzen (Werte), die in einer Komponente bereits bei deren Modellierung auf der übergeordneten

Typeebene definiert werden. Die zu verwendende Instanz wird also ähnlich den Powertypes bereits bei der Erstellung des Typs festgelegt.

Für jeden verwendeten Datentyp in einem Prozess kann so bei Bedarf bereits die zu verwendende Instanz festgelegt werden, beispielsweise eine bestimmte Maschine für einen bestimmten Bearbeitungsschritt.

Konstanten werden häufig genutzt, um Einschränkungen zu modellieren, beispielsweise um eine Spezialisierung auf einen bestimmten Wert festzulegen. Dabei ist eine stringente Einbindung der Konstanten-Instanz in den Prozess nicht möglich, weil dann Komponenten mehrerer Instanzebenen bei der Modellierung auf derselben Ebene verwendet werden müssten.

Daher werden in OMICRON Konstanten einfach als spezielle und direkte Instanz der Prozess- oder Referenzkomponente angelegt (ConstantInstance, Subtyp von Instance). Wird ein Prozess instanziiert, muss die Konstante (Instanz) automatisch in jede Prozessinstanz eingebunden werden.

Ähnliches gilt für **Defaultwerte**. Sie finden Verwendung, wenn Teilkomponenten eines Prozesses nicht zwangsläufig mit einem Wert verbunden oder initialisiert werden. Eine InstanceDefault-Relation verbindet dann die Default-Instanz (Defaultwert) mit der Klasse. Erfolgt keine Initialisierung, wird die Default-Komponente eingebunden, sonst der gewählte Wert.

Dies ist vor allem wichtig, wenn im Rahmen von Spezialisierungen Erweiterungen in Prozessen vorgenommen werden. Die allgemeinere Komponente kann die neuen Werte nicht kennen, so dass die Polymorphie gefährdet ist. Existiert ein Defaultwert, kann dieser sicherstellen, dass bei Polymorphie die Spezialisierung wie vorgesehen ausgeführt werden kann, ohne dass zusätzliche Werte interaktiv angegeben werden müssen.

5 Komplexe Komponenten und Abläufe im Metamodell

Die bisher beschriebenen Konzepte und das Meta-Metamodell stellen die Basis für ein objektorientiertes Gesamtmodell dar. Allerdings sind bisher eine Klassifizierung mit Vererbung und eine Instanziierung nur für atomare und einfache Komponenten möglich. Dieses Kapitel erweitert das Konzept nun auf die Verkettung und hierarchische Komposition von Komponenten wie sie für die Prozessmodellierung notwendig ist und führt die Anwendung der Vererbung auch für komplexe, gegebenenfalls hierarchisch aufgebaute Objekte ein.

5.1 Konkatenation

Die Konkatenation stellt ein zentrales Mittel einer komponentenbasierten Prozessentwicklung dar. Sie ermöglicht es, semantisch aufeinander folgende Komponenten zusammenzusetzen und so – wie die Komposition – eine Zusammenstellung umfangreicher Modelle aus Teilkomponenten durchzuführen. Im Gegensatz zur Komposition findet jedoch kein Wechsel der Abstraktionsebene oder der Aggregationsebene statt. Die Konkatenation wirkt orthogonal zur Vererbung, Instanziierung, Aggregation und Komposition, die alle statische Konzepte darstellen. Für die Abläufe (Dynamik) ist diese horizontale Verkettungsrelation notwendig.

Zunächst müssen hierzu die Möglichkeiten einer Verkettung untersucht werden. Hierbei sind beliebige und typkompatible Verkettungen zu unterscheiden. Reine Prozessschritte können meist beliebig verkettet werden, während Ereignis- und Datenflüsse in einem objektorientierten Modell wie OMICRON typabhängig sind.

Große EPK-Diagramme können beispielsweise an Ereignissen zusammengefügt werden. Das „Teilungsereignis“ [Klein et al. 2004] existiert dann in beiden Diagrammen – einmal als Eingangs-, einmal als Ausgangsereignis – und ermöglicht so eine Fortsetzung des Ablaufs. Die Zusammensetzung der beiden Diagramme wird dann ebenfalls als Konkatenation bezeichnet.

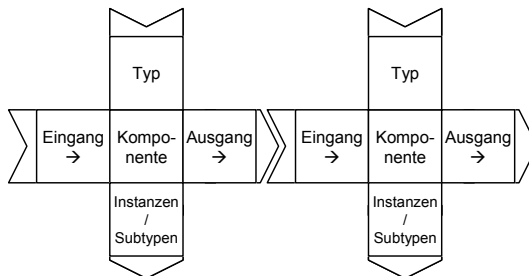


Abbildung 21: Konkatenation – Verkettung von Komponenten anhand kompatibler Ein- und Ausgangsparameter

Allgemein verbindet die Konkatenation zwei oder mehr Komponenten horizontal. Die bisher beschriebenen vertikalen Verbindungen haben eine hierarchische (statische) Abhängigkeit in Form einer typologischen Abhängigkeit (Instanz, Vererbung) oder Aggregation zur Folge. Dagegen stellt die Konkatenation eine gleichberechtigte, horizontale Verbindung zwischen zwei Komponenten her.

5.1.1 Prozess-Konkatenation mit Hilfe der Prozess-Flussrelation

Prozesse sind auf dem Konzept aufgebaut, dass eine Ordnungsrelation existiert, die alle Prozessschritte in eine Reihenfolge einordnet. Die Schrittfolge ist dabei durch eine unidirektionale Flussrelation bestimmt, die aufgrund der Möglichkeit von Zyklen auf der Prozessebene jedoch erst in ihren Zuständen auf der Instanzebene kausal ist.

Die Prozess-Konkatenation verbindet zwei Komponenten mit einer Flussrelation (Flow) oder einer ihrer Spezialisierungen. Damit ist es möglich, definierte Folgen von Prozessschritten zu erzeugen, wie sie für Prozesse notwendig sind. Die Ordnung ist dabei nicht hierarchisch sondern definiert eine kausale Folge von vorgesehenen Aktivierungen. Die Aktivierung bewegt sich dabei immer in Richtung der Flussrelation und baut damit Strukturen auf, wie sie aus Petrinetzen und andere auf gerichteten Ablaufgraphen basierenden Konzepten bereits bekannt sind.

Die **Flussrelation Flow** wird als Instanz der (Meta-)Relation in Form einer Relationstypkomponente definiert. Alle (auch transitiven) Instanzen von Flow, und nur diese, ermöglichen die horizontale Verkettung von Komponenten zu Abläufen und gegebenenfalls eine Animation der Prozessinstanzen zur Laufzeit.

Die Besonderheit ist die **Aktivierungsfähigkeit der Flussrelation**. Wird ein Modell animiert, können Flussrelationen eine Aktivierung aufweisen. Eine aktivierte Flussrelation aktiviert bei der nächsten Taktung (Weiterschaltung) die folgende Komponente, eine aktivierte Komponente die folgenden Flussrelationen.

Im Folgenden wird dazu unter anderem das Token-Konzept eingeführt. Die Animation zur Laufzeit wird dagegen in Kapitel 8 unter Anwendung der hier beschriebenen Grundkonzepte erläutert.

5.1.2 Implizite Konkatenation

Bei der Konkatenation werden kompatible Komponenten wie erläutert mit Hilfe einer unidirektionalen Relation eindeutig verbunden. In einem dezentralen, nachrichtenbasierten System existieren jedoch auch Konzepte, die zu einer impliziten Konkatenation führen.

Während die explizite Verbindung eindeutig ist, müssen implizite Konkatenationen zur Laufzeit erfolgen. Dabei spielen Eingänge und Ausgänge eine wichtige Rolle.

Es zeigt sich aber, dass die Definition und Unterscheidung von Eingängen, Ausgängen und Feldern anhand ihrer expliziten Konkatenation möglich ist. Dabei ist diese Klassifikation immer relativ zum Kontext zu sehen: Eingänge und Ausgänge sind jeweils für eine komplexe Komponente K definiert. Ist diese wieder in einer übergeordneten Komponente U enthalten, kann durch interne Verkettung ein Ein- oder Ausgang von K zur internen Komponente von U werden.

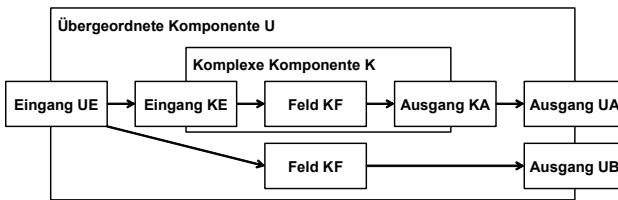


Abbildung 22: Kontextabhängige Definition von Ein- und Ausgängen bei der Konkatenation

Ereignisse (siehe Kapitel 7.2.2) können höchstens innerhalb desselben Prozesses explizit an Listener/Receiver gebunden. Soll ein Prozess durch ein globales oder durch einen anderen Prozess erzeugtes Ereignis im Kontext aktiviert werden, müssen zur Laufzeit existierende Listener durch kompatible Ereignisse angesprochen werden. Eine Typkompatibilität von Ereignis und Listener ist hierbei ausreichend und kann über das Metamodell identifiziert werden.

Die hierauf basierenden Konzepte der Prozessmodellierung sind im Kapitel 7.2 zu finden.

5.2 Komposition und Aggregation

Neben der Verkettung von Komponenten ist auch die Erzeugung komplexer Strukturen mit Hilfe einer hierarchischen Komponentisierung möglich. Diese sieht vor, dass Komponenten wiederum zu größeren Komponenten mit Hilfe der Aggregation zusammengefasst werden. Dabei können die untergeordneten verbunden werden, so dass die Funktionalität und Verbindungseigenschaften des Verbundes im Vergleich zu den einzelnen Komponenten verändert wird: Das Ganze erhält eine von seinen Teilen verschiedene Semantik.

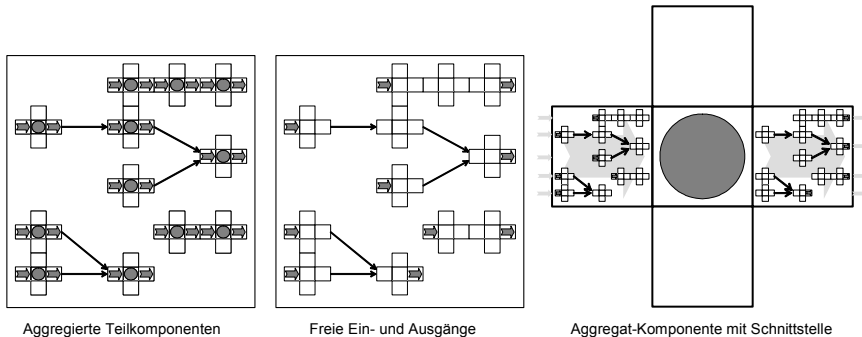


Abbildung 23: Synthese komplexer Komponenten aus verknüpften Teilkomponenten: Ein- und Ausgänge

[Hagen 2005] unterscheidet zwischen atomaren und zusammengesetzten Objekten und Ereignissen, und definiert komplexe Ereignisse als Aggregate von Ereignissen und komplexe Objekte als Aggregate von Objekten. Während dies die Klassifikation und Definition vereinfacht, können bei vollständig objektorientierter Modellierung in OMICRON komplexe Komponenten nicht nur aus Teilkomponenten eines Typs bestehen. So kann es notwendig werden, einem Ereignis weitere Informationen in Form von Daten mitzugeben. Eine **komplexe Komponente** ist somit jedes Aggregat beziehungsweise jede Komposition von Komponenten.

In objektorientierten Programmier- und Modellierungssprachen ist die **Aggregation** eine besondere Art der Assoziation zwischen Objekten: Die Teil-von-Beziehung (part-of). Jede Teilkomponente wird in OMICRON daher mit einer Aggregationsbeziehung an die übergeordnete Komponente gebunden und damit dieser zugeordnet.

Die **Containment**-Beziehung (enthalten sein) wird nicht gesondert modelliert, da im Modell die Auswirkungen gleich sind. Wird ein spezieller Aggregationstyp hierfür benötigt, kann dieser über die Relationstypvererbung zusätzlich geschaffen werden.

In der UML und vielen anderen Sprachen wird die schwache Beziehung der Aggregation durch die Komposition ergänzt. **Aggregate** (Gitterbox) können ohne ihre Teile existieren, durch **Komposition** entstehende **Komposita** (ein Berg Schrauben) dagegen nicht – sie zerfallen ohne ihre Bestandteile.

Die Aggregation wird im Folgenden entweder nach UML-Standard mit einer Raute auf der Aggregatseite der Relation oder als Container dargestellt.

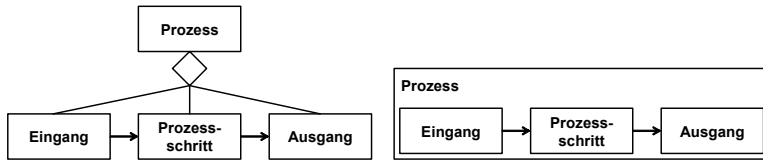


Abbildung 24: Aggregation in UML- und Container-Darstellung

Aggregate können als allgemeine Aggregate auftreten, sind jedoch häufig als typisierte Aggregate anzutreffen. So können beispielsweise in einer prozessualen Beschreibung mehrere Teilschritte im Rahmen eines geordneten Aggregats gekapselt werden. Während in Modellierungssprachen wie der UML nur ein bestimmter Typ aggregiert wird, ermöglicht OMICRON die metamodelkonforme Kombination beliebiger Typen in einem Aggregat.

Aggregate werden mit Hilfe von binären Aggregationsbeziehungen von den Teilkomponenten (aggregierte Komponenten) zum Aggregat (part-of) gebildet. Da alle Relationsenden am Aggregat geordnet sind²⁵, haben die Bestandteile eine implizite Reihenfolge.

Durch die Verallgemeinerung des Aggregationskonzepts können alle Komponenten mit Ausnahme der Relationstypkomponenten durch Aggregation sub-strukturiert oder zusammengefasst werden.

Beispiel 10: Beispielsweise kann ein Ereignis „Grundplatte fertig“, das heißt, eine Komponente vom Ereignis-Subtyp „Grundplatte fertig“, neben diesem Typ sowohl über einen Zeitstempel als auch über eine ID der Grundplatte und einen Prüfbericht verfügen. Daten (ID, Zeitstempel) und Artefakte (Prüfbericht) sind also in einem Ereignis aggregiert.

Die Modellierungsmächtigkeit und damit die Interpretation zur Laufzeit hängen entscheidend davon ab, unterschiedliche Typen konnektieren und zu komplexen Komponenten zusammensetzen zu können.

Oft sind Abläufe und Teilkomponenten nicht von Anfang an klar und definiert. Vielmehr werden in den ersten Phasen der Prozessmodellierung zunächst meist grobe, weitgehend textuell-deklarative Beschreibungen erstellt, die teilweise bereits als Use-Cases oder mit ähnlichen Modellierungs- und Komponentisierungskonzepten vorstrukturiert werden und einen stark schwankenden Detaillierungsgrad aufweisen.

Einzelne Prozessschritte werden dann zu größeren Aktivitäten zusammengefasst (**Komposition**) oder größere Aktivitäten in kleinere zerlegt (**Dekomposition**). Diese beiden Konzepte sind sowohl im Hinblick auf die Flexibilität von Modellen als auch für deren Abstraktions- und Konkretisierungsvermögen ausschlaggebend. Wie im Folgenden zu sehen ist, erhöht sich allerdings auch die Komplexität stark.

Für Petrinetz-basierte Ansätze existieren kaum Möglichkeiten zur mehrstufigen Komponentisierung. Aber auch für objektorientierte Modellierungsverfahren wie die UML ist eine weitere Zerlegung von Use-Cases und Klassen nicht, von Aktivitäten nur mit einem semantischen Bruch möglich.

Ein statisches, unflexibles Typisierungsmodell würde hier keine Detaillierung ermöglichen, während typlose, textbasierte Verfahren jegliche Struktur aufheben würden. Das OMICRON Modell ermöglicht daher unter Beibehaltung des Typisierungsmodells eine Aggregation / Komposition von Komponenten über beliebige viele Aggregationsebenen (vgl. Anforderung A4.3).

²⁵ alle Relationsenden an Komponenten sind in OMICRON linear geordnet, um eine deterministische Verarbeitung zu ermöglichen

Dies ist nur möglich, weil sowohl Aggregate als auch deren aggregierte Komponenten über alle Ebenen vollständig objektorientiert sind. Jede Komponente ist damit typologisch unabhängig von ihrem Aggregat: Der Aggregattyp (beispielsweise das Ereignis „Grundplatte fertig“) ist in der Modellierung nicht abhängig von den aggregierten Typen (wie Zeitstempel, Prüfbericht) sondern unabhängig von diesen definiert. Erst nach seiner Definition wirkt er für die Instanzebene strukturell bindend, das heißt eine Event-Instanz von „Grundplatte fertig“ muss dann immer einen Zeitstempel und einen Prüfbericht enthalten. (vgl. Anforderung A4.1)

Mit Hilfe unterschiedlich stark formalisierter beziehungsweise detaillierter Aggregationsstufen, können in der Modellentwicklung anfangs weitgehend unstrukturierte und artefaktbasierte Beschreibungen durch Untergliederung und Formalisierung in eine Modellform überführt werden, die regelbasiert geprüft und generativ genutzt oder in neue Modelle überführt werden kann, sobald alle informale Beschreibungen durch objektorientierte Komponenten ersetzt sind.

Ob eine Generierung oder Transformation automatisch, semi-automatisch (beispielsweise mit Hilfe eines Wizards) oder modellgestützt manuell erfolgt ist hierbei nur von der Implementierung und Mächtigkeit des definierten Metamodells und des Transformators abhängig. Das OMICRON Metamodell stellt die benötigten Typ-Konzepte auf atomarer und aggregierter Ebene zur Verfügung.

Im OMICRON Metamodell ist eine **direkte Integration** möglich, indem zwar **Teilansichten** definiert werden, das Modell selbst aber vollständig integriert ist. Die Einteilung in Kernfunktionen der Gesamtansicht und Unterfunktionen kann dabei über eine entsprechende Typisierung und die Definition von Sichten geschehen. Mit Hilfe eines Hinterlegungssymbols [Klein et al. 2004] kann dann angezeigt werden, dass sich eine oder mehrere Konkretisierungsebenen unterhalb des Symbols befinden beziehungsweise durch dieses repräsentiert werden.

Eine reflexive Aggregation einer Komponente durch sich selbst ist nicht möglich. Auch die Bildung von Zyklen muss durch das Laufzeitsystem unterdrückt werden: Die Aggregation bildet einen azyklischen, gerichteten Graph (DAG). Eine Baumform wäre jedoch zu beschränkt, da eine Komponente mehreren Aggregaten angehören kann.

5.2.1 Aggregierte Ein- und Ausgangsbereiche von Teilkomponenten

Eine Komponente kann mehrere Ein- und Ausgangskomponenten (Parameter) haben, wie in Kapitel 5.1 beschrieben. Diese können notwendig oder fakultativ sein. Einzelne Werte können bei Bedarf zur weiteren Verarbeitung aus komplexen Datenkomponenten extrahiert oder zu komplexen Datenkomponenten zusammen- oder hinzugefügt werden. Da dies typisiert erfolgt, sind Konstrukte wie die Joinspec²⁶ in UML2 unnötig. Da das OMICRON Modell automatisierbar sein muss, wird für jeden Parameter seine Verwendung im Prozess bereits durch das Modell beschrieben. In OMICRON werden zusätzliche Notationen wie joinspec vermieden, da diese die Semantik aufweichen würden.

Die UML2 nutzt Objektknoten für die Beschreibung Datenflüssen und Pins [Jeckle et al. 2004]. Einen Schritt weiter als Objektknoten und Pins geht das UML2-Konzept der Parametersätze. Ein **Parametersatz** legt fest, dass alle enthaltenen Parameter übergeben werden müssen. Pro Aufruf darf nur je ein Parametersatz verwendet werden, auch wenn mehrere existieren. Es können überlappende Parametersätze erstellt werden, wobei jedoch nur Ein- oder Ausgangparameter sortenrein zusammengefasst werden dürfen. Der Grund liegt auf der Hand und gilt auch für OMICRON: Eingangsparameter müssen einen weiteren Ablauf intern, Ausgangsparameter in der Folgekomponente auslösen können. Hierfür muss ein vollständiger Satz Daten existieren.

²⁶ {joinspec = A or B} fusioniert die Parameter A und B.

Sollen in OMICRON Modellen Parametersätze verwendet werden, können diese einfach in einer neuen Komponente zusammengefasst werden. Erst wenn diese vollständig ist, können Daten und Signale weitergegeben werden. Ein zusätzliches Konzept ist also unnötig.

In OMICRON dürfen Prozessschritte daher nur ausgeführt werden, wenn alle notwendigen Parameter und ein Signal vorliegen (notwendige Bedingung). Die Aktivierung muss neben einem vollständigen Parametersatz über eine aktivierungsfähige Komponente und Relation erfolgen (hinreichende Bedingung). Jeder Teilablauf / Eingang, der einzeln aktiviert werden kann ist daher ein Parametersatz, auch wenn es sich nur um eine atomare Komponente handelt.

Sollen alternative Parameter beziehungsweise Parametersätze für denselben Eingang modelliert werden, muss eine (meist generalisierte) Komponente genutzt werden, die alle Alternativen akzeptiert. Allerdings müssen alle Parametersätze Kinder desselben Typs sein, da sonst eine alternative Weiterleitung nicht möglich ist (Typkontrolle). Alternative Parametersätze unterschiedlichen Typs sind allerdings über unterschiedliche Eingangskomponenten möglich.

Beispiel 11: Ist für eine Aktivität eine Fräsmaschine erforderlich, kann auch das vielseitigere Bearbeitungszentrum eingesetzt werden. Die Objektorientierung in OMICRON akzeptiert das Bearbeitungszentrum als Spezialisierung der Fräsmaschine (und weiterer Maschinentypen).

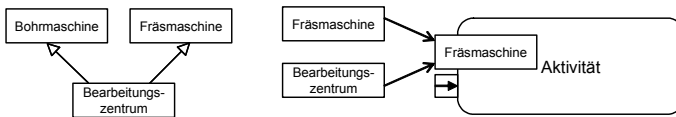


Abbildung 25: Typkontrolle und -kompatibilität für Parameter

Meist ist jedoch, abhängig vom Eingangstyp, eine unterschiedliche Verarbeitung erwünscht oder notwendig, beispielsweise ein teilweise montiertes System statt Einzelteilen in der Fertigung, so dass ohnehin für jeden disjunkten Eingangs-Typ unterschiedliche Abläufe definiert werden.

5.2.2 Komponentensignaturen: Eingang, Ausgang, Typ

Eingänge verfügen bezüglich eines Kontexts K über mindestens eine ausgehende interne Flussrelation und keine eingehende, Ausgänge umgekehrt über mindestens eine eingehende, aber keine ausgehende interne Flussrelation. Ohne Typveränderungen können so anhand der Einbindung die unterschiedlichen Flusstypen identifiziert werden:

Flusstyp	Anzahl eingehende interne Flussrelationen	Anzahl ausgehende interne Flussrelationen
Eingang	0	≥ 1
Ausgang	≥ 1	0
Feld / intern	≥ 1	≥ 1
Undefiniert	0	0

Tabelle 17: Definition der Flusstypen von Komponenten relativ zu einem Kontext K

In funktionaler Sicht besteht in der Softwareentwicklung die Signatur einer Methode aus dem Namen der Methode sowie Eingangs- und Ausgangsparametern:

`<Methodensignatur> := <Rückgabety> <MethodenName> '(' '<Parameter>' (',' '<Parameter>')* ')'`

Das Tripel aus Vor- und Nachbereich sowie Typ einer komplexen Komponente wird daher im Folgenden ebenfalls als **Signatur** bezeichnet, da es einer Signatur in der funktionalen

Programmierung beziehungsweise einer Schnittstelle²⁷ in der objektorientierten Programmierung ähnelt.

Alle Eingangskomponenten bilden den Vorbereich, alle Ausgangskomponenten den Nachbereich. Der Typ der komplexen Komponente ist dabei unabhängig vom Typ der Teilkomponenten, die Signatur jedoch nicht. Die folgende Abbildung zeigt schematisch, wie aus einem Prozess dessen Vor- und Nachbereich extrahiert werden kann.

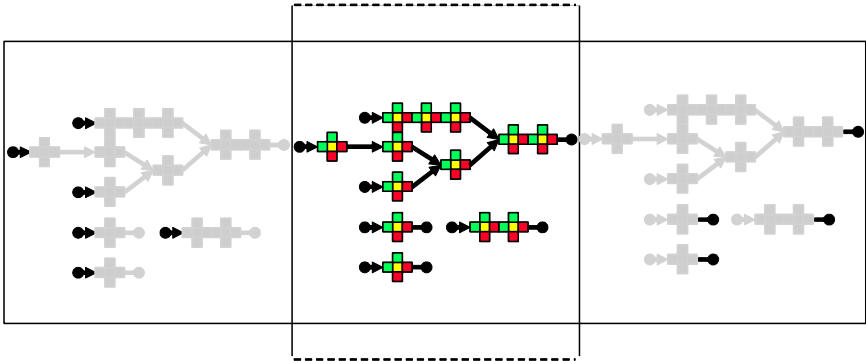


Abbildung 26: Definition Vor- und Nachbereich („Parameter“) einer komplexen Komponente

So haben auch Komponenten Schnittstellen, die aus Eingängen (Eingangssignatur, IN) und Ausgängen (Ausgangssignatur, out) bestehen und diese in statischer (Typ) wie in dynamischer Dimension (Anbindung, Verwendung) beschreiben. Da die Komponentensignatur aus Eingangssignatur, Ausgangssignatur und Typ der Komponente besteht, umfasst sie alle von außen sichtbaren Eigenschaften einer komplexen Komponente.

Anhand der Ein- und Ausgangssignatur, kann das Laufzeitsystem entscheiden, welche Typen relational angebonden werden dürfen und ob solche vorliegen. Zudem kann aus aggregierten Ein- und Ausgangskomponenten einer Teilkomponente die neue Schnittstelle der übergeordneten Aggregat-Komponente abgeleitet werden kann. Auch untergeordnete Aggregationsebenen können so zur Signatur beitragen. Jeder Ein- und Ausgang wird über die Aggregationshierarchie nach oben und „außen“ propagiert.

Dabei sind folgende Schnittstellenelemente zu unterscheiden:

- **Nicht verbundene** sowie **verbundene mit noch freien obligatorischen Verbindungskapazitäten** werden direkt zur Schnittstelle hinzugefügt,
- **verbundene mit noch freien fakultativen Verbindungskapazitäten** können manuell ausgewählt oder bei Verfügbarkeit eines zusätzlichen Wertes ausgewählt werden,
- **verbundene ohne freie Verbindungskapazitäten** werden nicht hinzugefügt.

Bei der Weitergabe beziehungsweise Anbindung eines komplexen Typs im Rahmen der Konkatenation wird zur Laufzeit eine Instanz dieses Typs zusammen mit den aggregierten Komponenten (ebenfalls Instanzen von Typen) weitergegeben und verwendet.

²⁷ Die Ein- und Ausgangskomponentenmengen sind Teil der Signatur und können als Schnittstelle bezeichnet werden. Eine Schnittstelle ist nach [Klein et al. 2004] beziehungsweise DIN 44300 ein „gedachter oder tatsächlicher Übergang an der Grenze zwischen zwei gleichartigen Einheiten mit vereinbarten Regeln für die Übergabe von Daten oder Signalen“.

5.2.3 Polymorphie bei Prozessen

Die **Polymorphie** wirkt sich auch auf Parameter aus. Wird ein Parameter vom Typ A vorgesehen, kann auch jede Spezialisierung vom Typ A' auf der Instanzebene verwendet werden. Aufgrund der Notwendigkeit einer sehr offenen Vererbungssemantik für Prozesse, können zum Teil Spezialisierungen vorkommen, die bestimmte Wertebereiche nicht mehr abdecken. Daher ist eine möglichst exakte Definition der geforderten Parameter notwendig, gegebenenfalls auch ein Ausschluss von oder alternative Abläufe für bestimmte Spezialisierungen.

Während atomare Komponenten problemlos anstelle der Eltern polymorph eingesetzt werden können, bilden bei spezialisierten Prozessen zusätzlich eingefügte Ein- und Ausgabeparameter ein Problem: Durch Polymorphie kann eine Instanz dieser Spezialisierung anstelle der definierten Komponente eingesetzt werden und erwartet dann zusätzliche Parameter.

Das Überschreiben oder Verändern von Feldern und Methodensignaturen wird in der OOP einfach nicht erlaubt – geforderte und zurückgegebene Parameter bleiben erhalten.

Für die objektorientierte Prozessmodellierung ist eine Veränderung der Signatur in der Vererbung jedoch unabdingbar, da sonst immer bereits auf oberster Ebene für jeden Prozess die vollständige Signatur definiert werden müsste, ohne dass Erweiterungen und Veränderungen in den abgeleiteten Spezialisierungen möglich sind. Übertragen auf Prozesse würde das bedeuten, dass nur Prozesse genutzt werden dürfen, die die gleichen Parameter benötigen, das heißt nicht mehr Eingaben fordern und nicht weniger Eingaben verarbeiten oder Ausgaben erzeugen, was zur vollständigen Parametergleichheit führen würde.

Dies würde die praktische Einsetzbarkeit in Frage stellen. In der objektorientierten Programmierung gibt es kein vergleichbares Konzept, da die Methoden inhaltlich prinzipiell beliebig überschrieben werden können und somit auch weitgehend beliebigen Code enthalten dürfen, solange die Signatur gleich bleibt. Anders als bei OMICRON gibt es dort nur eine Aggregationsebene für die Kapselung, das heißt die Kapselung funktioniert nur mit Klassen, nicht jedoch auf Befehls- oder Modulebene. Die Kapselung findet bei OMICRON jedoch auf jeder Ebene statt, so dass das Konzept für alle Schachtelungstiefen gelten muss.

Durch **Prozesspolymorphie** können typkompatible Spezialisierungen A' anstelle eines Teilprozesses A eingesetzt werden, wo eine Referenzkomponente R_A im Gesamtprozess P eigentlich auf A zeigt.

Die Prozessinstanz sieht dann immer noch eine Instanz a des Typs A (vor), jedoch wird eine Instanz a' des Typs A' verwendet. Zwischen a' und A' besteht daher eine Instanzbeziehung, während zwischen a' und der Referenzkomponente R_A eine **Verwendungsbeziehung (Use)** besteht. Auch hier existiert ähnlich wie bei Vererbung, Instanziierung und Referenz eine **SubUse-Beziehung** zwischen allen Teilen von R_A und den zugeordneten Teilen von a'. Da A' eine Spezialisierung ist, können möglicherweise auch Teile auftreten, die nicht verbunden werden.

Auf diese Weise ist immer der tatsächliche Typ von a' durch die Instanzbeziehung zu A' klar, gleichzeitig jedoch auch die Verwendungsstelle unter der Referenzkomponente R_A . So wird wie in der OOP zwischen dem eigentlichen Objekt und seiner Verwendung entschieden. Aus Sicht der Instanz von P läuft der Prozess weitgehend wie gewohnt ab.

Allgemein existieren unterschiedliche Regeln, um auch ohne ein rein objektorientiertes Modell Äquivalenzen zu identifizieren. Nach der semantischen Transformationsregel und der Input-Output-Regel [Keller et al. 1992] sind Funktionen verschieden, wenn die Signatur verschieden ist, jedoch auch nicht zwangsläufig gleich, wenn die Signatur gleich ist.

Problematisch sind hier zwei sich ausschließende Anforderungen: Auf der einen Seite sollten **Prozesse Spezialisierungen ihrer Eingangswerte akzeptieren**, also beispielsweise Wochennummern statt einem beliebigen Zahlenwert, um der Prozess-Spezialisierung Rechnung zu tragen. Auf der anderen Seite soll ein **spezialisierter Prozess anstelle eines allgemeineren**

verwendet werden können und daher vollständig kompatibel sein. Akzeptiert die Spezialisierung nur noch Wochennummern, führen allgemeine Zahlenwerte wie 1000 zu einem Fehler.

Dieser scheinbar unauflösbare Widerspruch ergibt sich, weil angenommen wird, dass jede beliebige Spezialisierung eines Prozesses in jeder Situation genutzt werden kann. In Wirklichkeit dient jedoch eine Spezialisierung häufig der Einschränkung des Anwendungsbereichs. So können Prozesse zur Erstellung eines Lochs die Verwendung eines Bohrers statt einer Fräse bei Spezialisierung auf kleine Löcher vorsehen.

Gerade hier zeigt sich wie wichtig die Laufzeitorientierung für das objektorientierte Modell von OMICRON ist: Erst zur Laufzeit kann anhand der verfügbaren Parameterwerte entschieden werden, ob eine Prozessinstanz einer gewählten Spezialisierung überhaupt im Kontext verwendet werden kann.

Es gilt daher: **Spezialisierungen von Werten können immer verwendet werden, Spezialisierungen von Prozessen nur bei passendem Kontext** (Parameterwerten).

Eingänge (IN) akzeptieren also Instanzen von Spezialisierungen. Enthalten Spezialisierungen weitere Felder werden diese ignoriert. Existierende Felder dürfen daher nur eingeschränkt jedoch nicht erweitert werden, um den Wertebereich der allgemeineren Komponente nicht zu verletzen.

Ausgänge (Out) dürfen ebenso wie Eingänge im Sinne der Objektorientierung spezialisiert werden. Allgemeinere (generalisierte) Ausgänge sind daher unzulässig, weil folgende, konkatenierte Komponenten diese bei Wertebereichüberschreitung nicht verarbeiten könnten. Der Konsument (Consumer/Nachfolger) muss sich auf den Prozess „verlassen“ können. Die durch Spezialisierung möglicherweise vorhandenen zusätzlichen Komponenten werden von der Folgekomponente einfach ignoriert. So kann der folgende Eingang immer die Spezialisierung des Ergebnisses konsumieren.

Bereits bei der im Folgenden beschriebenen Vererbung komplexer Komponenten muss die Konsistenz der Ein- und Ausgänge sichergestellt werden. Nur die Überprüfung auf kompatible Wertebereiche und Typen erfolgt erst bei der Animation im Rahmen der Polymorphie.

5.3 Vererbung komplexer Komponenten

Atomare Komponenten werden durch die Vererbungsrelation zu einer übergeordneten Komponente als Spezialisierung definiert. Da sie keine Substruktur besitzen hat die Vererbung nur typologische Auswirkungen.

Die bisher beschriebene Vererbungsstrategie kann zunächst nur auf solche „leeren“ Komponenten mit reiner Typinformation ohne weitere Inhalte angewandt werden. Sollen nun Inhalte mit vererbt werden, muss jedoch eine Repräsentationsmethode entwickelt werden, die auch Bestandteile in die Vererbung mit einbezieht.

Eine komplexe Komponente aggregiert eine oder mehrere Teilkomponenten. Diese können wiederum Teilkomponenten enthalten und als Teilprozesse oder Prozessschritte auch untereinander verbunden sein. Bei Instanziierung, Vererbung und Referenzierung von komplexen Komponenten erstreckt sich die Wirkung auch auf die Teilkomponenten.

Wird daher eine komplexe Komponente spezialisiert, erbt sie alle Teilkomponenten der Vaterkomponente zunächst ohne Veränderung. Semantisch gleicht sie daher der Vaterkomponente exakt.

Wird eine Vererbungsbeziehung (Inheritance) zwischen A und B hergestellt, werden alle Teilkomponenten von B jeweils mit der entsprechenden Komponente von A durch eine SubInheritance-Beziehung verbunden. Gleiches gilt für Reference und SubReference.

5.3.1 Sub-Komponenten in der Vererbung

Der Prozess der Vererbung beginnt, wenn von einer Komponente A eine Unterklassen-Komponente B durch Vererbung erzeugt werden soll. Zunächst entsteht hierdurch die neue Komponente B, die mit A in einer Vererbungsbeziehung steht und so alle Eigenschaften von A erbt.

Ist A eine atomare Komponente und hat daher keine Teilkomponenten, wird eine reine, leere, spezialisierte Komponente B erzeugt. Verfügt A über Teilkomponenten ist zunächst auch keine weitere Information für die Spezialisierung notwendig.

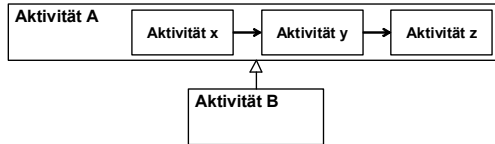


Abbildung 27: Vererbung einer komplexen Komponente

Wie im Folgenden gezeigt, kann es notwendig werden, die von A geerbten Teilkomponenten in der Spezialisierung B anzusprechen. Anders als bei der OOP und auch der objektorientierten Modellierung ist es für Prozesse notwendig, auch unverändert geerbte Teilkomponenten im neuen Kontext zu duplizieren. Mit Hilfe einer besonderen Referenzierung, die eine transitive Teilvererbung anzeigt (SubInheritance) könnten diese Komponenten in B eingebunden – quasi gespiegelt – werden und referenzieren die ursprünglichen Teilkomponenten in A:

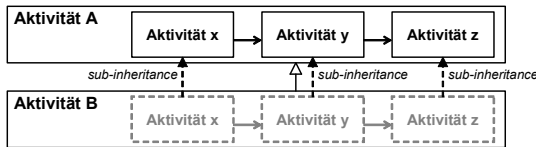


Abbildung 28: Anwendung der SubInheritance zur Referenzierung der geerbten Komponenten

Der Grund hierfür liegt im Kontext für Änderungen und der Referenzierung durch Instanzen und Spezialisierungen. Jede nun an der Spezialisierung vorgenommene Änderung benötigt einen Änderungskontext. Wird beispielsweise eine Teilaktivität y spezialisiert, das heißt durch eine geänderte Variante y' überschrieben, muss diese spezialisierte Teilaktivität y' in den Prozess eingebunden werden. Die geänderte Teilaktivität y' erbt dabei von y, so dass für y' ebenfalls eine Vererbungsbeziehung und eine Berücksichtigung von deren Teilaktivitäten erfolgen muss.

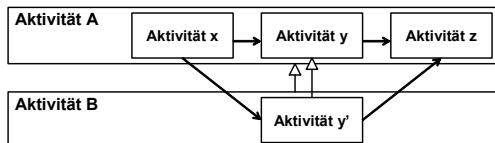


Abbildung 29: Änderungskontext bei Änderung einer Teilkomponente

Um den Umfang des Modells möglichst gering zu halten, wurde mit unterschiedlichen Ansätzen versucht, nur überschriebene Komponenten zu verwenden und alle Komponenten mit SubInheritance-Beziehungen wegzulassen. Dies ist bei der oben gezeigten Änderung durch Vererbung noch möglich, indem algorithmisch für y' alle Relationen von y geprüft werden. Dies

gilt dann für die Flussrelationen ebenso wie für die Aggregation. Die Relationen (x, y') und (y', z) können dabei weggelassen werden, weil sie transitiv klar sind.

Alternativ können aber wie oben beschrieben auch SubInheritance-Beziehungen eingesetzt werden, um in B einen vollständigen Kontext zu erzeugen:

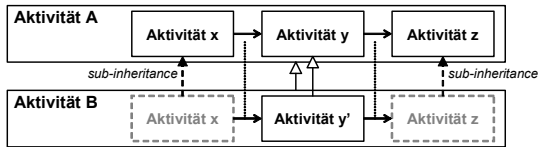


Abbildung 30: Vollständiger Kontext durch SubInheritance, Änderung von y durch Vererbung

Werden innerhalb von B oder y' Komponenten hinzugefügt, zeigt sich, dass dieses Verfahren das einzig mögliche für die komplexe Vererbung ist. Die folgende Abbildung zeigt, dass eine korrekte Integration der Aktivität v nur möglich ist, wenn Aktivität x innerhalb von B referenziert werden kann.

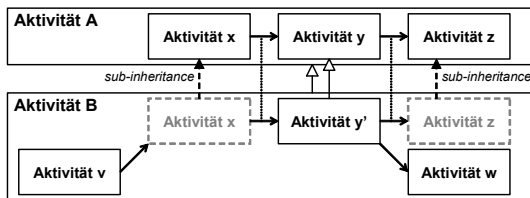


Abbildung 31: Zusätzliche Aktivitäten in der Spezialisierung B erfordern einen vollständigen Kontext

Ein Gesamtmodell muss also die Vererbung mit vollständigem Kontext vorsehen. Die einzige Alternative ist das algorithmische Hinzufügen benötigter Kontextkomponenten. Dieses Vorgehen ist jedoch sehr fehleranfällig, da mit dem Hinzufügen der Aktivität x als Kontext zur neuen Aktivität v gegebenenfalls auch die Relation (x, y') hinzugefügt werden muss. Gerade für geschachtelte Strukturen bietet sich daher die Anwendung der vollständigen Kontextbildung zum Zeitpunkt der Vererbung eher an als die sukzessive Referenzierung.

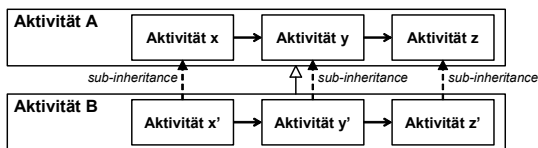


Abbildung 32: Vererbung des vollständigen Kontexts mit Hilfe der SubInheritance

Bei Änderungen in mehrstufigen Vererbungshierarchien ist mit der SubInheritance sofort klar, welchen Kontext die Veränderung hat. Dagegen müsste bei fehlendem Kontext erst die richtige Ausgangskomponente ermittelt werden, was bei Änderungen über mehrere Hierarchien oft nicht mehr gelingt. Wird x' auch geändert, ist für x'' die Änderung sofort wirksam:

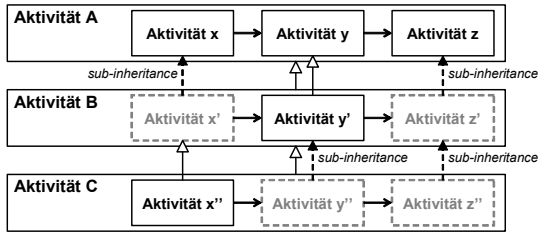


Abbildung 33: Vererbung und Änderungen auf mehreren Ebenen

Bei der Instanziierung wirkt das vollständige Kontextmodell mit SubInheritance ebenfalls konsistenzsichernd. Eine Instanz c einer Aktivität C referenziert mit allen Teilkomponenten deren Instanzvater (Instance-of) oder den Kontext zu der passenden Komponente (beispielsweise Teilaktivität) in C. Wurde y'' nicht verändert, würde es in einer selektiven Duplikation nicht existieren, so dass Instanz y'' im Modell Aktivität y' referenzieren müsste. Diese Beziehung ist nur resistent gegen Modell-Veränderungen in A und B, da Änderungen von y'' in C keinen Einfluss auf Instanzen von B hätten, da Instanz y'' auf Aktivität y' zeigen würde:

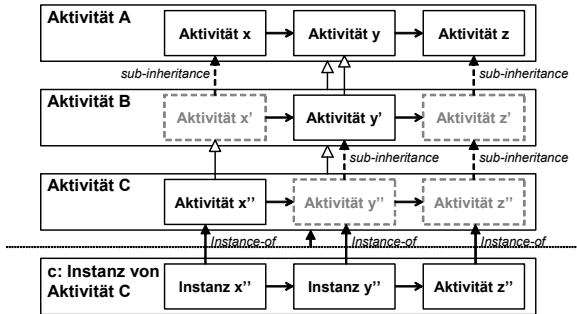


Abbildung 34: Instanziierung mit vollständigem Kontext

Die prinzipiell denkbaren Repräsentations-Ansätze der komplexen Vererbung sind in Anhang F beschrieben. Die bereits beschriebene Vollrepräsentation aller ererbten Teilkomponenten erweist sich auch im Hinblick auf die Verteilung als die beste Option für objektorientierte Prozesse in der Produktion.

Ersetzungen von Teilkomponenten dürfen bei der komplexen Vererbung nur durch typkompatible Komponenten vorgenommen werden, das heißt dass ein Element einer komplexen Komponente nur durch eine ihrer Spezialisierungen ersetzt werden darf. Nur so kann sichergestellt werden, dass die spezialisierte Komponente weiterhin statt der allgemeinen Komponente eingesetzt werden kann. Dies gilt sowohl für Prozesskomponenten als auch für alle Ausgaben, die immer auch speziellere Werte liefern dürfen. Akzeptieren Eingänge abweichende Wertebereiche, sind sie nicht mehr vollständig prozesskompatibel.

Zusätzliche Komponenten können hinzugefügt und in den Ablauf eingebunden werden. Soll eine solche Erweiterung auch bei Eingängen eingesetzt werden, verhindern diese eine Verwendung an Stelle des allgemeineren Prozesses. Um eine Verwendung auch ohne den zusätzlichen Parameter zu ermöglichen, kann ein Defaultwert eingesetzt werden, der anstelle eines übergebenen Parameters verwendet werden kann. Alternativ können nicht zur Verfügung stehende Werte interaktiv abgefragt oder als fakultativ gekennzeichnet werden.

Wurde eine **zu löschende Komponente** bisher nicht referenziert, kann sie durch den Löschvorgang einfach wieder entfernt werden. Bei erfolgter Vererbung ist auf bereits existierende Verbindungen in den Spezialisierungen zu prüfen. Eine spezielle Änderungskomponente ist nicht notwendig, da alle Änderungen dem beschriebenen objektorientierten Konzept folgen.

Die Besonderheiten der Mehrfachvererbung sind im nächsten Abschnitt beschrieben.

5.3.2 Mehrfachvererbung komplexer Komponenten

Die bisher gezeigten Zusammenhänge gelten zunächst nur für die Einfachvererbung. Da aber auch eine Mehrfachvererbung erfolgen kann, müssen die Konzepte auf das parallele Erben von mehreren Komponenten übertragen werden.

Bei einer Mehrfachvererbung von zwei Komponenten A und B muss allerdings entschieden werden, was die beabsichtigte Semantik der Mehrfachvererbung ist. Für Daten ergibt sich eine Vereinigungsmenge aller Elemente von A und B, da alle Teile zugreifbar sein müssen. Unter der Voraussetzung, dass an der Spezialisierung AB keine Veränderungen vorgenommen wurden, müssen auch bei einer Aktivität AB Elemente beider Eltern-Aktivitäten A und B ausführbar sein. Zunächst ist jedoch nicht festgelegt, wie das zu geschehen hat. Beide Aktivitäten stehen gleichberechtigt nebeneinander.

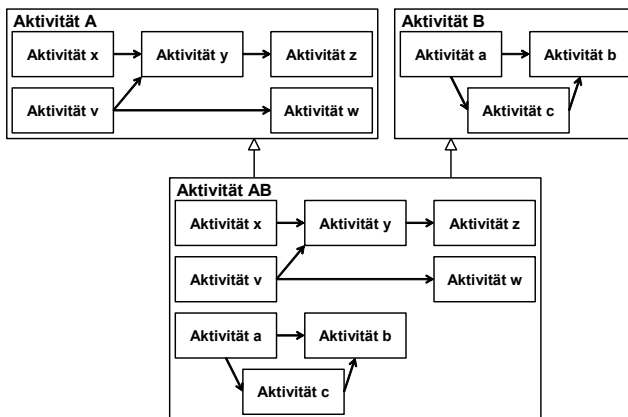


Abbildung 35: Mehrfachvererbung ohne Änderung bedeutet unpriorisierte Aggregation aller Teilkomponenten

AND-Sequenzierung, AND-Parallelisierung etc. müssen durch Verbindungen explizit in der Spezialisierung hinzugefügt werden. Dies kann sich bei der Modellierung schon aus semantischen Abhängigkeiten wie „Stückliste laden“ → „Material aus Lager holen“ ergeben.

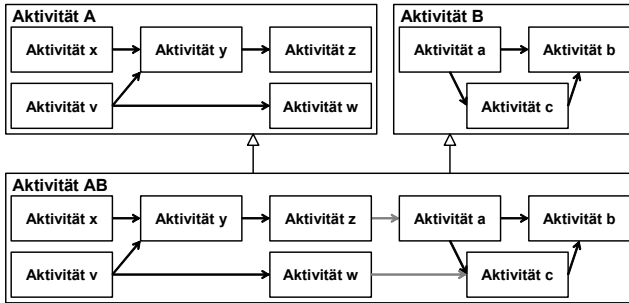


Abbildung 36: AND-Sequenzierung der Aktivität A und Aktivität B in ihrer Spezialisierung AB

Werden Eingangsereignisse und -parameter für Aktivität a sowohl mit Aktivität z als auch extern verfügbar gemacht kann sowohl nur der zu Aktivität B gehörige Teil von Aktivität AB ausgeführt werden als auch die vollständige, sequenzierte Aktivität AB mit den zu Aktivität A gehörigen Teilen. Das OMICRON Modell erlaubt bei der Spezialisierung durch Mehrfachvererbung im Rahmen der Modellsemantik die freie Modellierung der tatsächlichen Fachkonzeptsemantik.

Da die Mehrfachvererbung einen DAG erzeugt, kann eine Mehrfachvererbung über mehrere parallele Pfade erfolgen. Ein Bearbeitungszentrum erbt bestimmte Maschinen-Eigenschaften über Fräsmaschine und Bohrmaschine doppelt (vgl. Abbildung 15). Allgemein erbt eine Aktivität D von Aktivität A die Komponente x doppelt:

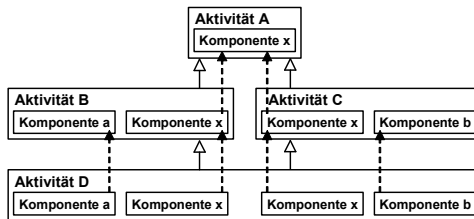


Abbildung 37: Mehrfachvererbung einer Teilkomponente über mehrere Pfade in der Vererbungshierarchie

Bei statischen Komponenten wie Datenstrukturen kann es sinnvoll sein, dieselbe Komponente nicht mehrfach zu übernehmen. Wurde die Teilkomponente x allerdings durch Spezialisierung verändert, ist eine Verschmelzung nicht mehr möglich, da eine Verwendungsgleichheit nicht mehr sichergestellt werden kann.

Da die SubInheritance-Beziehung für aggregierte Teilkomponenten in der Vererbung mit Hilfe der Rekursion aufwärts eine direkte Feststellung der Herkunft einer Komponente ermöglicht, können überschneidende Komponenten gleicher Abstammung algorithmisch festgestellt werden. Dabei dürfen nur über Sub-Inheritance erfolgte Vererbungen berücksichtigt werden, da nicht nur der Typ, sondern auch der Vererbungskontext ausschlaggebend ist. Die Duplikation von Relationen durch die Vollreferenzierung stellt zudem sicher, dass auch einzelne Relationen anhand ihrer „Abstammung“ zugeordnet werden können. Werden Veränderungen an geerbten Abläufen vorgenommen, was häufig gerade die Intention bei der Vererbung darstellt, müssen diese mit Hilfe dieser Relationen ins Gesamtmodell integriert werden.

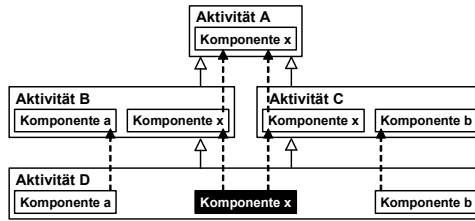


Abbildung 38: Verschmelzung einer mehrfach geerbten Komponente x zu einer einzigen

Für dynamische Komponenten wie Prozessschritte ist die Verschmelzung jedoch meist nicht sinnvoll und wegen abweichender Ablaufsemantik auch für die Modellkonsistenz gefährlich. Da zwei durch Mehrfachvererbung integrierte Teilprozesse mehr und unterschiedliche Artefakte erzeugen, können Schritte nicht einfach zusammengesetzt werden ohne Ablauf, Daten- und Materialflüsse zu stören. Soll ein Teilprozess B tatsächlich nur um zusätzliche Komponenten b eines anderen erweitert werden, bietet sich meist eher die Einfachvererbung und Ergänzung an.

Die folgende Abbildung zeigt die Einfachvererbung und Einbindung benötigter Komponenten wenn eine Typ-, Prozess- und Ergebnisdopplung nicht sinnvoll ist.

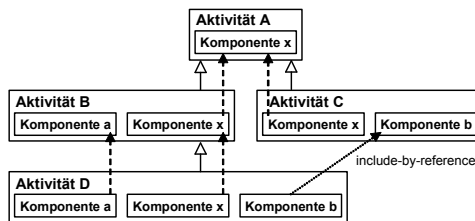


Abbildung 39: Einfachvererbung und Einbindung benötigter Komponenten

Die Verknüpfung und Fusion von Teilprozessen im Rahmen der Mehrfachvererbung muss also dem Modellierer überlassen werden. Eine Mehrfachvererbung ohne weitere Anpassungen führt zu einer parallelen Zusammenstellung aller ererbten Teilkomponenten. So werden Informationsverluste ausgeschlossen und die Verwendung als Subtyp aller Vaterkomponenten auch für komplexe Komponenten automatisiert ermöglicht.

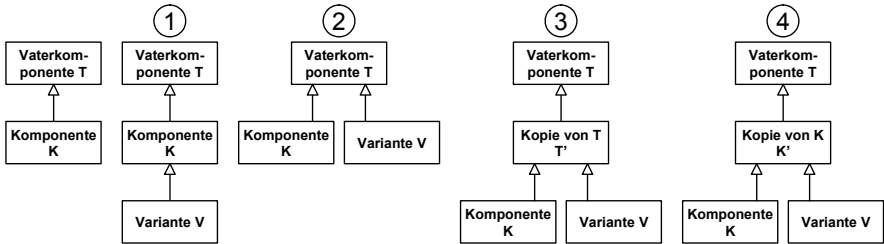
5.3.3 Variantenbildung und Versionen

Die Erzeugung von Varianten kann ebenso wie die Versionen als spezielle Beziehung eingeführt. Allerdings wird hierdurch wieder die Kopienproblematik erzeugt, so dass außerhalb der Vererbungshierarchie eine weitere Selektion besteht.

Die Erzeugung von Varianten stützt sich in OMICRON daher auf die Vererbung. Für eine Variantenbildung zu einer Komponente K mit Vater-Komponente T bestehen folgende Optionen:

- Untergeordnete Variante: Durch Ableiten einer Variante V von einer Komponente K (1)
- Vater-Variante: Durch Einfügen einer Kopie V von K parallel zu dieser in der Vererbungshierarchie (2)
- Vater-Kopie: Ableitung einer Kopie T' von der Vaterkomponente T und Platzierung von K und V als Kinder von T'. Dies ist bei direkter Mehrfachvererbung von K nicht möglich. (3)

- Verlagerung: Durch Einführung einer übergeordneten Kopie K' von Komponente K unter T und Ableitung von K als reine Referenz (vererbungsunschädlich) von dieser Komponente K'. Variante V wird dann ebenfalls von K' parallel abgeleitet. (4)



Untergeordnete Variante (1), Vater-Variante (2), Vater-Kopie (3) und Verlagerung (4)

Abbildung 40: Möglichkeiten der Variantenbildung

Bei der Bildung von Versionen ist eine lineare Folge der Neuerung vorgegeben. Jede Version kann auf einer Ordinalskala eingeordnet werden, so dass immer klar ist, welche Version einer Komponente neuer ist.

Das Konzept, alte Versionen mit neuen spezifisch zu überdecken scheitert an der Anforderung teilweise unterschiedliche Prozess-Versionen parallel zu fahren. Teilweise werden beispielsweise noch alte Versionen eines Bauteils geprüft, während eine neue Version bereits beauftragt wurde. Reduziert um die anfälligeren Möglichkeiten 3 und 4 der Variantenbildung stellt sich daher die Versionierung wie folgt dar:

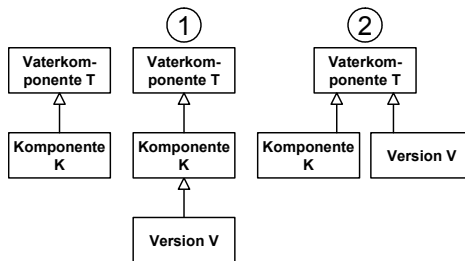


Abbildung 41: Zwei Möglichkeiten der Versionierung mittels einer speziellen Versions-Vererbung

Werden zwei Versionen parallel betrieben (Version 1 läuft aus, Version 2 wird nun eingesetzt), müssen in jedem Fall beide Versionen in der Vererbungshierarchie existieren. Eine Migration kann dabei durch das Verbot neuer Instanzen von Version 1 erfolgen.

5.4 Modelle und Modelltypen

Modelltypen sind ein bekanntes Konzept aus vielen textuellen und allen grafischen Modellierungsmethoden. Ein Modelltyp legt die Grundlage für maschinelles wie menschliches Verständnis der modellierten Zusammenhänge, indem es für alle Modelle des Modelltyps eine verbindliche Syntax und Semantik definiert.

Die Auswahl eines Modelltyps legt die Ausdrucksmittel fest, mit Hilfe derer sich der Modellierer ausdrücken kann, unter anderem:

- Welche Elemente existieren?
- Wie können diese angeordnet, verbunden und in Beziehung gesetzt werden?

- Welche Beschränkungen, Regeln und Freiheitsgrade gibt es?

Beispiel 12: Wird als Modelltyp beispielsweise ein Petrinetz gewählt, stehen Stellen, Transitionen und Kanten zur Verfügung, die nur in einem wechselnden Muster verbunden werden dürfen. Marken werden durch Regeln beschränkt. Handelt es sich um ein Use-Case-Diagramm oder ein Rollenmodell, gelten wieder andere Regeln, die durch deren Metamodell vorgegeben werden.

Die Semantik eines Modelltyps oder Metamodells kann durch unterschiedliche Sprachen ausgedrückt werden, unter anderem mit natürlicher Sprache (informal / semi-informal) oder einem nochmals übergeordneten Meta-Metamodell wie einer Algebra oder dem OMICRON Meta-Metamodell.

Ein Modelltyp – in sichtbarer grafischer Form ein Diagrammtyp – enthält alle zum Modelltyp gehörenden Komponenten(typen). Durch Aggregation werden bei OMICRON Referenzen auf die zu berücksichtigenden – das heißt verwendbaren – Typen eingebunden.

Beispielsweise kann so ein Modelltyp oder ein Prozessmodell erstellt werden, das vorschreibt, welche Elemente in welcher Reihenfolge genutzt werden können. Im Gegensatz zu anderen Modellierungskonzepten (beispielsweise UML2 [OMG 2005a]) sind diese Konzepte in OMICRON unabhängig von der Ebene der Verwendung, das heißt ebenenübergreifend. Ein aus einem Prozess-Metamodell instanziiertes und definiertes Prozessmodell definiert seinerseits die Prozessinstanz (das Projekt).

Ein Modelltyp besteht zunächst aus dieser eigentlichen **Modelldefinition**. Soll eine grafische Visualisierung erfolgen, muss zudem ein **Viewmodell** zur Anzeige der unterschiedlichen Typen erstellt und dem Diagrammtyp zugeordnet werden.

Reicht die Modell-Semantik zur Beschreibung der Regeln nicht aus, kann ein **Regelmodell** hinzugefügt werden, das es erlaubt, umfangreichere Spezifikationen zu erstellen, als dies mit dem Metamodellierungskonzept an sich möglich ist, da nur grundlegende Vorgaben wie Kardinalitäten noch in Verbindung mit den Relationen gespeichert werden können. Ein Modellierungssystem muss anhand des Modelltyps wie auch mit Hilfe des Regelmodells ausgehend vom Ist-Zustand die Korrektheit einer geplanten Änderung wie Hinzufügen, Ändern oder Entfernen von Relationen direkt überprüfen können. Mit Hilfe der vorhandenen Regeln/Definitionen kann so entschieden werden, ob die gewünschte Aktion gültig ist oder verweigert werden muss und welche Modellelemente im Kontext zur Verfügung stehen.

Durch die Verwendung im Metamodell werden Komponententypen per Aggregation an einen Modelltyp gebunden: Jeder im zugehörigen Metamodell verwendete (aggregierte) Komponententyp ist auch Bestandteil des Modelltyps. Komponententypen werden daher häufig nicht im Modelltyp definiert sondern nur referenziert – quasi als Eigenschaften. Durch die auf der Meta-Ebene eingebundenen Komponenten und definierten Relationen zwischen diesen werden auf der Modellebene nur Komponenten und Relationen zugelassen, die diesem Metamodell entsprechen.

Die Definition verwendeter Komponenten kann daher in einem anderen Diagramm(typ) erfolgen, das die notwendige Sicht und die entsprechenden Regeln für die Vererbung zur Verfügung stellt.

5.4.1 Relationen im Diagrammtyp

Um Relationen für Instanzen eines Diagrammtyps zu erlauben, müssen diese im Diagrammtyp definiert sein. Die **Definition „erlaubter“ Relationstypen** erfolgt **durch Verwendung** im Modell- beziehungsweise Diagrammtyp in Form einer relationalen Verbindung zwischen zwei Komponententypen.

Für Instanzen dieser zwei Komponententypen ist damit die Verwendung der definierten Relation wirksam „erlaubt“. Eine Relation zwischen den beiden Instanz-Komponenten im Modell wird als Instanz der definierten Relation im Modelltyp interpretiert. Eine Relation gilt dabei als

verwendet, wenn beide (Referenz-)Komponenten, die sie verbindet im Modelltyp liegen, das heißt von diesem aggregiert werden. Zur direkten Zuordnung referenziert die Relation auf der untergeordneten Ebene die entsprechende Relation im Metamodell (Modelltyp):

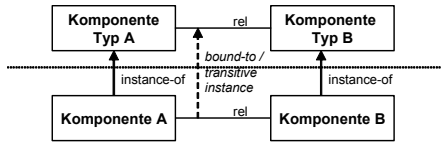


Abbildung 42: Instanziierung von Bestandteilen eines Modelltyps auf Modellebene

Aufgrund der Polymorphie gelten alle Regeln des Modelltyps auch für die definierten Sub-Typen der dort verwendeten Komponenten und Relationen. Dies bedeutet auch, dass die Komponenten auf Modellebene Instanzen von Subtypen der auf der Modelltypebene verwendeten Komponenten sein können. Eine Instanz des Komponententyps C wird im Kontext von Komponententyp A verwendet:

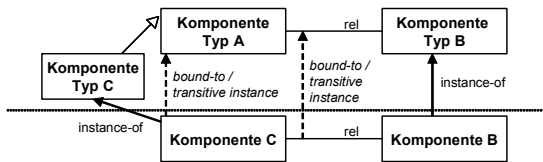


Abbildung 43: Polymorphie bei der Modelltyp-Instanziierung

Relationen eines übergeordneten, instanzgebenden Modells enthalten auch Attribute die für zu bildende Instanzen entscheidend sind. Die Kardinalitäten einer Relation definieren für einen Relationstyp (und dessen Subtypen) die Anzahl von Verbindungen zwischen zwei Typen von Komponenten mithilfe einer Relation und somit die Modellstruktur auf der Instanzebene.

Die Angabe der **Kardinalität** einer Relation an beiden Relationsenden erfolgt ähnlich der Notation und Semantik der UML2:

$$A-[1..5]----[7..n]-B$$

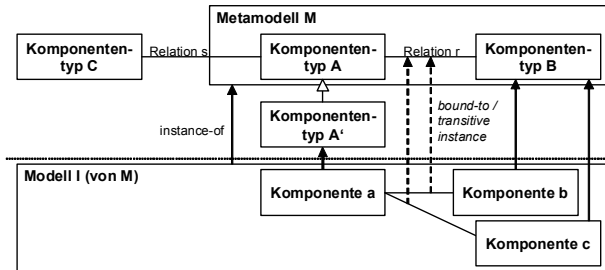
bedeutet, dass Komponenten vom Typ A mit sieben oder mehr Komponenten des Typs B verbunden werden können/müssen. Umgekehrt muss eine Komponente vom Typ B mit mindestens einer und höchstens fünf Komponenten vom Typ A verbunden werden.

Die Auswirkung der definierten Kardinalität wird auf der darunter liegenden Instanzebene, das heißt für Metamodelle auf der Modellebene, sichtbar.

Die Vererbungshierarchie wird modelltypübergreifend gespeichert. Komponenten sind zwar per Aggregation an Modelle gebunden, jedoch im gesamten Modellraum oder in Partitionen (Projekte o.ä.) verfügbar. Durch die Aggregation im Modelltyp ist bekannt, welche Komponententypen und gegebenenfalls welche Relationstypen zwischen diesen in einem Modell genutzt werden können.

Das Meta-Metamodell sieht dagegen die Möglichkeit der Aggregation von Relationen nicht vor. Da Relationen also einem Metamodell nicht explizit zugeordnet werden können, erfolgt die Zuordnung implizit über die Relationsenden beziehungsweise die zwei verbundenen Komponenten einer binären Relation. Liegen beide Komponenten innerhalb des Metamodells, ist die Relation eine interne und damit Bestandteil des Metamodells. Liegt mindestens eine der zwei Komponenten außerhalb des Metamodells, sind also nicht beide Komponenten im Metamodell aggregiert, so handelt es sich um eine externe Relation, die für das Modell auf Instanzebene

ebenso wie bei der Modelltypspezialisierung durch Vererbung nicht oder nur mittelbar berücksichtigt wird.



Instanziierung eines Metamodells M als Modell I. Polymorphe Verwendung der A'-Instanz a anstelle einer Instanz von A. Die Relation s ist extern, da C außerhalb von M liegt.

Abbildung 44: Polymorphe Verwendung und externe Relationen in Modellen

5.4.2 Integration von Diagrammtypen

Sollen mehrere Modelltypen zusammengefasst werden, beispielsweise um mehrere Artefaktspezifikationen und eine Ablaufspezifikation zu integrieren, kann ein übergeordneter Modelltyp geschaffen werden, der die anderen durch Mehrfachvererbung zusammenführt und gegebenenfalls mit zusätzlichen Relationen zwischen Elementen der Modelltypen verbindet.

Sollen nur die Diagramme als Ganzes Verwendung finden, reicht hierzu das Konzept der Aggregation von (Modell-)Komponenten aus. Damit werden selbständige Modelle erzeugt, die jedoch auch die Modellierung von modellübergreifenden Verbindungen zwischen Teilkomponenten der aggregierten Teilmodelle zulassen.

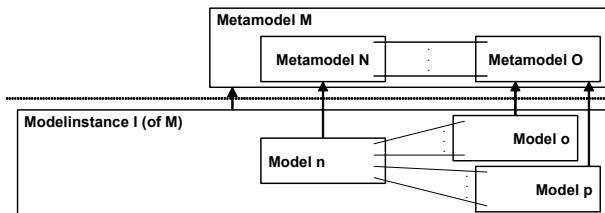


Abbildung 45: Aggregierte Metamodelle erlauben modellübergreifende Relationen

Sollen jedoch auf untergeordneter Ebene alle Komponententypen der Teilmodelle integriert werden, so müssen sich alle Teilmodelle vollständig im neuen Modell auflösen. Durch Mehrfachvererbung und Einbringung der zusätzlichen Relationen oder Komponententypen in das gemeinsame neue Modell entsteht eine neue, teilmodellübergreifende Semantik. Elemente der unterschiedlichen Teilmodelltypen können jetzt gleichberechtigt nebeneinander verwendet werden.

Das Konzept der Entitäten (Entities)²⁸ wird durch die vollständige Objektorientierung des Gesamtmodells und die im Folgenden beschriebene, direkte Integration der Diagrammsemantik in die Komponenten unnötig.

²⁸ Als übergeordnete (Meta-)Aggregate verbinden Entities mehrere Komponenten zu einer virtuellen, fassen also beispielsweise einen Actor im Use-Case Diagramm und eine Klasse im Rollenmodell zusammen.

Somit können Komponenten in unterschiedlichen Diagrammen beziehungsweise Modellen auftauchen – durch Mehrfachvererbung auch mit unterschiedlicher Semantik. Aus einer expliziten Modellierung von Entitäten ist damit ein implizites Konzept des OMICRON Gesamtmodells geworden. Dies erfolgt nicht als zusätzliches Konzept wie beispielsweise Annotationen, modellexternen Hilfen, Dokumentation in anderen Modellierungssystemen.

5.4.3 Implizite Modelltypen

Modell- und Diagrammtypen werden in anderen Ansätzen meist als separates Konzept definiert. Im Rahmen dieser Arbeit werden unterschiedliche Ansätze untersucht, um Modelle und Diagramme als separate Konzepte zu definieren. Während dies auf zwei Ebenen (Modelltyp-Modell) auch mit dem Standardparadigma der Objektorientierung gelingt, müssen bei mehrfacher Instanziierung rekursive Konzepte eingesetzt werden. Der Modelltyp im Metamodell führt zu Instanzen (Modellen), die ihrerseits wieder instanziiert werden müssen und somit ihrerseits auch den Modelltyp für untergeordnete Instanzen bilden. So können Diagramme wieder selbst Diagrammtypen für untergeordnete Diagramminstanzen darstellen. Dieser Vorläufer einer vollständigen Vereinheitlichung erzielte durchaus Teilerfolge und konnte Sichten und Regeln aggregieren. Durch weitere Veränderungen kann jedoch eine vollständige Integration erreicht werden.

In der Objektorientierung wird jede Instanz durch ihren Typ definiert. In OMICRON wurde dieses Konzept von der reinen Klassen-Instanz-Ebene auf alle Instanzebenen erweitert. Aus diesem Grund muss keine separate Definition von Modelltypen erfolgen. Vielmehr definiert jede instanzgebende Komponente (Typ), die Rahmenbedingungen (Metamodell) für ihre Instanzen.

Aufgrund der Verbreitung separater Modell- und Diagrammkonzepte (wie UML Superstructure) und der einfacheren Verständlichkeit der Modelltyp-Semantik in dieser Form wurden oben die entsprechenden Konzepte explizit für Modelltypen beschrieben. Diese sind einem vollständig objektorientierten Gesamtmodell wie OMICRON jedoch inhärent. Die Instanziierung von Modellen aus Metamodellen folgt derselben Semantik wie die Modell-Instanz-Beziehung. Einzig die Erweiterungen wie Views und Regeln bedürfen zusätzlicher Interpretation und Strukturen.

5.5 Sichten

Gerade in grafischen Modellierungskonzepten und bei Systemen mit unterschiedlichen Rollen bei der Modellierung haben sich unterschiedliche Modellsichten bewährt.

Sichten werden häufig eingesetzt, um bestimmte Aspekte desselben Sachverhalts und Modells anzuzeigen oder zu betonen.²⁹ Beispielsweise können dies beliebige thematische Teilsysteme, Schichten, Fehler etc. sein [Moldt 1996] oder eine prozessorientierte Aufteilung in funktionale (Aktivitäten, Teilprozesse), dynamische (Verhalten, Sequenzen), informationale (konsumierte und erzeugte Entitäten, Artefakte) und organisationale (wer, wo) Sicht [Mili et al. 2003].

In fast allen anderen IT-basierten Forschungsfeldern wie beispielsweise bei Geoinformationssystemen (vgl. [Volz 2006]) finden unterschiedliche Schemata und Sichten – auch unter integrierter Anzeige mehrerer Objekttypen – Verwendung. Zu unterscheiden ist hier zwischen **unterschiedlichen Sichten auf dasselbe Modell** und der **Integration unterschiedlicher Modelle in eine neue Sicht**. Oft werden mehrere Sichten auf dasselbe Modell erstellt, um bestimmte Aspekte darzustellen und/oder die Komplexität und den Umfang der Darstellung zu reduzieren. Beispielsweise werden in einer Sicht die durch Vererbung entstehende

²⁹ Beispielsweise EPK aus Business- und UML aus Entwicklerperspektive [ZapfHeinzl 1998]. Bei Experimenten zur Metamodellierung, beispielsweise bei British Airways [Arlow et al. 1997][Emmerich et al. 1996], zeigte sich, dass unterschiedliche Sichten für Verständnis und Überblick notwendig sind.

(statische) Typologie und die statischen Abhängigkeiten dargestellt und in einer zweiten Sicht der reine Ablauf, der durch die Verkettung von Komponenten entsteht.

In OMICRON sind Sichten ein sekundäres Konzept für das primäre Basismodell. In anderen Konzepten wie UML sind jedoch Diagrammtypen das primäre Strukturierungsmerkmal (vgl. [OMG 2005b]), so dass Darstellungen eines Modelltyps und dessen Semantik verknüpft sind, was nur eine Darstellungsweise je Modell- und Komponententyp erlaubt.

Im Folgenden wird hauptsächlich die semantische Integration von Sichten in das OMICRON Modell mit Hilfe der verfügbaren Modellkonzepte beschrieben. Durch die Entkopplung der Darstellung von den Modellen, können bewährte Visualisierungen von Modellsachverhalten übernommen werden. Für die Darstellung selbst werden daher bekannte Ansätze wie UML, EPK und BPMN als Sichten auf das OMICRON Gesamtmodell vorgeschlagen, die nach Bedarf erweitert werden können, um erweiterter Semantik Rechnung zu tragen. Sie sind jedoch nicht Fokus dieser Arbeit. Die Bildung neuer Sichten für existierende Modelle unter Beibehaltung der Modellsemantik ist in OMICRON jederzeit möglich.

Das **Viewmodell** legt auf Metamodellebene fest, wie Modelle angezeigt werden sollen. Hierzu wird auf Typebene jedem ModelElement eine **Elementsicht-Typ (ElementViewType)** und damit einer Komponente eine **Komponentensicht** und einer Relation eine **Relationensicht** zugeordnet, beispielsweise einem Use-Case die Ellipse. Ein Ansichtstyp (**ViewType**) beschreibt grundlegend, ob und in welcher Form bestimmte Relationen und Komponenten dargestellt werden. Ein Modell wird so zu einem Diagramm, ein Modelltyp zu einem Diagrammtyp.

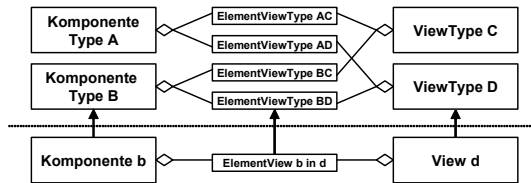


Abbildung 46: Zuordnung von Komponententypen zu Ansichtstypen über individuelle ElementViewTypes

Ein ElementViewType muss Informationen zur grafischen Darstellung enthalten. Er definiert welcher Elementtyp, in welchem Sichttyp (ViewType), wie (Form, Grafik), wo (Koordinatenvorgaben, Ausmaße, Swimlane) und mit welchen Optionen (einzeln ausblendbar, aufgeklappter Container) dargestellt wird. Zudem entscheidet er darüber, ob eine zu einem Diagramm gehörende Komponente oder Relation angezeigt oder ausgeblendet wird.

Die Elementsicht (ElementView) erhält die Informationen für die Darstellung der zugeordneten Komponente von ihrem ElementViewType als dessen Instanz. Über die grundsätzliche Darstellung wird also auf Typebene entschieden, der aktuelle Zustand (aufgeklappt, Position) dagegen auf der Instanzebene. Auch hier greift das Metamodellierungskonzept und ermöglicht die Verwendung einer View- oder ElementView-Komponente ihrerseits als ViewType- oder ElementViewType für die nächste untergeordnete Instanzebene.

Eine Komponentensicht für eine Komponente höherer Ebene kann andere Komponentensichten untergeordneter Elemente aggregieren, so dass die Konsistenz der Ansicht im Kontext gewahrt bleibt: Ein Use-Case-Diagramm aggregiert beispielsweise Sichten für Akteur und Use-Case. Die Komponentensicht enthält dabei die Darstellungsvorschrift (grafisch, textuell) für diesen Typ, die je nach Zustand variieren und an ebenfalls modellierte Bedingungen³⁰ geknüpft sein kann,

³⁰ Regeln für Sichten werden mit einer seiteneffektfreien Regelsprache wie beispielsweise OCL [OMG 2006c] modelliert um objektzustandsabhängige Sichten zu erzeugen. Einen generischen Ansatz für Diagrammeditoren beliebiger grafischer Modelle liefert die vom Autor betreute Arbeit von [Li 2004].

beispielsweise für Container: aufgeklappt. Dabei kann jedes Element mehrere Sichten haben und so auch zu mehreren Views gehören.

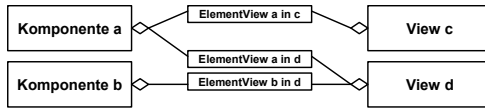


Abbildung 47: Zugehörigkeit zu mehreren Sichten c und d auf dasselbe Modell, das a und b enthält

Views können hierarchisch aufgebaut werden, so dass spezielle Teil-Daten oder Ausschnitte des Modells nur in untergeordneten Views sichtbar sind. So kann die Komplexität von Ansichten reduziert und dabei die Anzeige von bestimmten Relationen angepasst oder in spezielle Diagramme ausgelagert werden. Hinterlegungssymbole [Klein et al. 2004] können dabei anzeigen, dass sich eine oder mehrere Konkretisierungsebenen unterhalb des Symbols befinden. Beispielsweise wäre vom integrierten EPK-Klassen-Diagramm [Dandl 1999] die Referenz auf ein anderes Klassendiagramm möglich, das die gesamte Definition der Komponente enthält.

Die View kann dabei im Gesamtmodell nicht nur instanziiert, sondern auch an jede Spezialisierung vererbt werden, da sie in der Komponente mit Hilfe eines speziellen **View-Aggregationstyps** (ViewAggregation) aggregiert wird.

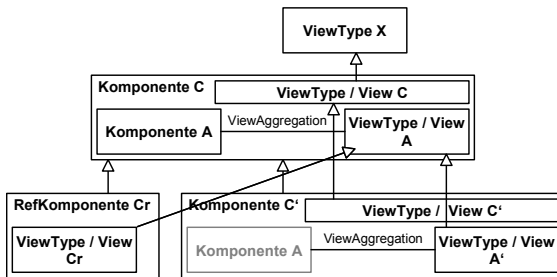


Abbildung 48: Vererbung von Sichten an Spezialisierungen

Sichten werden vererbt aber gewöhnlich verändert, um beispielsweise unterschiedliche Koordinaten aufnehmen zu können. Wie oben ersichtlich, müssen auch in Referenzkomponenten die Sichten überschrieben werden können, beispielsweise um abweichende Koordinaten aufzunehmen.

Wird die Komponentensicht nicht weitergehend überschrieben, bleibt sie so auch für Spezialisierungen verfügbar – wichtig vor allem im Hinblick auf die einheitliche Darstellung. Durch Überschreiben können spezialisierte Typen jedoch auch wieder ausgeblendet oder verändert werden.

Auch automatische, generative Abstraktionen in Form einer High-Level-Ansicht wären mit Technologien wie der automatisierten Abstraktion von Klassendiagrammen [Egyed 2002] möglich. Eine Manipulation generativ erstellter Sichten wirft wegen Seiteneffekten und Abhängigkeiten allerdings viele Probleme auf und schränkt die Flexibilität hinsichtlich des Ausgangsmodells ein, so dass im Rahmen dieser Arbeit nicht weiter auf generative Sichten eingegangen wird.

6 Das OMICRON Meta-Metamodell

Wie die UML benötigt das einfacher strukturierte und allgemeiner gehaltene OMICRON eine Definition grundlegender Konzepte, die hier als **Meta-Metamodell** referenziert werden. Diese beschreiben axiomatisch grundlegende Annahmen und Konstrukte in Form von Basisdefinitionen und einer Meta-Semantik, die den Rahmen für die Erstellung von Metamodellen und Modellen auf niederen Abstraktionsstufen bildet.

Der Begriff **Gesamtmodell** umfasst alle in einem System vorhandenen Modellelemente, unabhängig von ihrer Abstraktionsstufe. Das heißt es sind alle Meta-Ebenen (Instanzebenen), Komponenten und Relationen enthalten, so dass das Gesamtmodell sich bis auf programmierte Details zur Interpretation auch selbst beschreibt. Das Gesamtmodell kann im Speicher gehalten oder dauerhaft abgelegt sein – beispielsweise in einer Datenbank.

Für den **Gesamtmodell-Aufbau** und seine **Abstraktionsschichten** stehen mit der bereits beschriebenen Instanziierung zunächst prinzipiell beliebig viele Meta-Ebenen in beiden Richtungen zur Verfügung – Klassenbildung und Instanziierungen (Instanzen) wären dann ebenso wie Auf- und Ableitung (das heißt Super- und Sub-Klassenbildung) in beliebiger Folge möglich.

Dieses potenziell n-schichtige Modell [Schlegel 2005] mit beliebigem n könnte problemlos beliebig viele Instanz- und Supertyp-Ebenen enthalten und so das Gesamtmodell in der Vererbungs- ebenso wie in Instanziierungsdimension unbegrenzt belassen.

Eine nur referenzierende – beispielsweise XML-basierte – Repräsentation des Modells würde diesen Ansatz erlauben. Allerdings ist eine Zielsetzung dieser Arbeit, dass Metamodell und Implementierung in einander übergehen, um eine möglichst hohe Kohärenz, Systemunabhängigkeit und Flexibilität zu erreichen und die Semantik automatisiert auswerten zu können sowie Laufzeitadaptierbarkeit und (Sub-)Typ-basierte Interpretation zu erlauben.

Bei einer Untersuchung der Implementierungsfähigkeit des Metamodell-Ansatzes stellt sich heraus, dass beliebige Schichten zwar mit Hilfe der allgemeinen Repräsentation graphbasiert verwaltet werden können, jedoch die auch im Programmcode eines Modellierungssystems objektorientierte Verwaltung der Komponententypen nicht mehr möglich ist. Auch für die Modellsemantik muss ein axiomatisch definierter Startpunkt zur Verfügung stehen, der die übergeordneten Konzepte für die Metamodellierung beschreibt, um ein Metamodell überhaupt prüfen und integrieren zu können.

Die Umsetzung in Software mit der Unterstützung von Metamodellen, Typen, Komponenten, Relationen und Sichten (vgl. Anforderung A7.1) kann nur durch die grundlegende, axiomatische Definition dieser Konzepte und deren Umsetzung als Klassen der Modellierungssoftware erfolgen. Die Klassen der Software, die für die Typunterscheidung verantwortlich sind, können zwar eine Instanziierung unterstützen, jedoch aufgrund der Programmiersprachenkonzeption nicht die Typbildung (Metamodellierung) ausgehend von den vorhandenen Klassen.

Die OMG führt für die Meta Object Facility (MOF, [OMG 2006b][OMG 2005d]) beziehungsweise Unified Modeling Language (UML, [OMG 2007b]) eine Restriktion auf vier Abstraktionsebenen ein. Anhang E zeigt ein Beispiel, Probleme und die modellsemantische Herleitung des OMICRON Modells als Weiterentwicklung aus dem Vier-Ebenen-Modell und dessen Erweiterungen [VarróPataricza 2002].

Während Powertypes, Potenzierung und Deep Instantiation verworfen werden, erweist sich Strict Metamodeling als ein wichtiges Konzept. Um eine axiomatische Meta-Metamodell-Schicht zu schaffen, die als Grundlage für die Laufzeitumgebung dienen kann, muss das Konzept jedoch hinsichtlich des Ausgangspunkts invertiert werden. Anstelle eines Starts von der Instanzebene mit beliebigen übergeordneten Metaebenen wird das OMICRON Meta-Metamodell als

Ausgangspunkt definiert. Ausgehend von diesem, unter Beibehaltung aller semantischen Konzepte einschließlich Vererbung und Instanziierung können nun beliebige viele untergeordnete Instanzebenen geschaffen werden, die Metamodelle, Modelle etc. bilden.

Da die objektorientierten Konzepte über alle Instanz-/Abstraktionsebenen vereinheitlicht werden, können die Unterscheidungen zwischen **Metaklassen, Klassen und Objekten** bei OMICRON wie von [AtkinsonKühne 2002] gefordert entfallen. Diese Benennungen werden im Folgenden zum besseren Verständnis als relative Bezeichnungen aber soweit möglich beibehalten.

6.1 Das Multi-Ebenen-Modell

OMICRON ermöglicht daher grundsätzlich eine Instanziierung über beliebig viele Ebenen, so dass beispielsweise eine neue Metamodellierungssprache auf Basis des OMICRON Meta-Metamodells (MMM) eingeführt werden kann, die die weiteren drei Schichten bis zum Projekt ebenfalls unter sich hat. Unter Beibehaltung des Meta-Metamodells ist so trotzdem maximale Flexibilität gewährleistet. Während MOF/UML eine unterschiedliche Semantik je nach Modellebene vorsieht, soll die Laufzeitvererbung und Instanziierung in OMICRON auf allen Ebenen in gleicher Form zur Verfügung stehen, so dass Inkonsistenzen vom System erkannt werden können und eine rein modellbasierte Erweiterung möglich ist. Voraussetzung dafür sind unter anderem Gleichförmigkeit der Instanziierung und Vererbung in allen Instanzebenen, die nur durch die spezifische Semantik dieser Relationen ausgehend von ihren Relationstypkomponenten gewährleistet werden kann.

Ein Metamodellierungskonzept zeigt seine Konsistenz und Vollständigkeit auch darin, dass in der Anwendung **für jede Instanzebene dieselben Regeln** gelten und alle Konstrukte im Modell definiert sind – möglicherweise mit Ausnahme einiger Axiome, die der Definition des Modells selbst dienen. Bei den bekannten Konzepten wie UML, OPEN etc. ist dies nicht so.

Auch können Konstellationen auftreten, die durch das Vier-Ebenen-Modell der OMG nicht ermöglicht werden: Ein Fertigungsprojekt kann häufig aus einem Schritt oder einem Sensor mehrere Instanzwerte erzeugen, die den Fertigungsfortschritt charakterisieren. Auch kann es im Zuge der Generierung notwendig werden, eine weitere Instanzierungsebene hinzuzufügen. Da jede Instanz nur ihren direkt übergeordneten Typ „kennt“, ist die Struktur sehr flexibel und relativ anstelle einer absoluten Ebenenbestimmung.

Die Erstellung eines auf allen Ebenen konsistenten, initialen Gesamtmodells ist daher auch der aufwändigste Teil dieser Arbeit. In anderen Ansätzen wird die oberste Modellschicht meist vollständig axiomatisch unter Zuhilfenahme einer Notationsform wie UML definiert. Die Semantik wird dann einzeln beschrieben und in Software umgesetzt.

Ein Metamodell modelliert die Struktur eines Modells mit Hilfe von Modellelementen. Dabei klassifiziert erst die Existenz eines untergeordneten Modells ein Metamodell als solches, weil die Abstraktionsebene (Metamodellebene) nur relativ zu einer untergeordneten Modellebene identifiziert werden kann: „*Der Begriff Metamodell ist entsprechend immer relativ zu einem anderen Modell zu verstehen und damit kontextabhängig.*“ [Müller 2007]

Das Meta-Metamodell muss als Axiomensystem wiederum Konstrukte nutzen, die auf niedrigeren Ebenen (Instanzen seiner selbst) definiert werden. In Anlehnung an [Thies 2003] werden Axiome in dieser Arbeit als grundlegende Gesetzmäßigkeiten des Modells verstanden. Hierzu zählen die Definition der Basiskomponenten und Relationstypen mit Hilfe der Vererbung, Instanziierung, Attributierung und Aggregation, das heißt das Meta-Metamodell ist in sich selbst definiert und stellt so die es definierenden Konzepte selbst zur Verfügung. Es muss diesen, beispielsweise hinsichtlich erlaubter Relationen, auch gehorchen. Diese Selbstbezüglichkeit ist typisch für ein Axiomensystem und stellt entsprechende Herausforderungen an seine Umsetzung in Software.

6.2 Relationen im Gesamtmodell

Aufgrund der Komplexität und Selbstbezüglichkeit eines axiomatischen Gesamtmodells können Relationen nicht exakt einer „Abstraktionsebene“ zugeordnet werden können.

So wird die Vererbung, auch für die Definition des MMM genutzt. Komponente und Relation werden direkt vom ModelElement per Vererbung abgeleitet. Der Relationstyp „Instantiation“ wird jedoch erst im Metamodell eine Instanzebene tiefer definiert: Die Instanzrelation wird als Instanz der Relationstypkomponente *i* instanziiert, die wiederum Instanz von „Relation“ ist. Trotzdem werden Instanzrelationen und Vererbungsrelationen auf der höchsten Modellebene zur Definition des MMM eingesetzt. So wird die Definition von Relationstypen mit Hilfe der Instanziierung von Relation realisiert. Die verwendeten Instanzrelationen sind wiederum Instanzen des Relationstyps Instantiation.

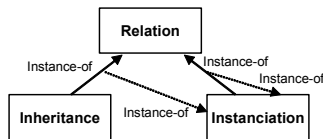


Abbildung 49: Selbstbezügliche Definition der Vererbung im Meta-Metamodell

Dies Instanziierung von Relationen aus Relationstypkomponenten führt zusammen mit der Verwendung dieser Relationstypen für die Definition des MMM zu einem rekursiven Problem. Alle Relationen müssten dann mit Hilfe einer Instanzierungsrelation ausgehend von ihrem Relationstyp erzeugt werden, deren Instanzrelation wiederum Instanz von Instantiation ist:

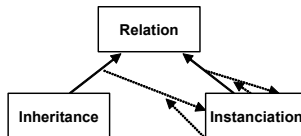


Abbildung 50: Entwicklung einer endlosen Instanzierungsrekursion im Meta-Metamodell

Um dieses Problem und eine hohe Verknüpfungskomplexität zu vermeiden, können Relationen nicht mit Hilfe einer Relation an andere Relationen angebunden werden. Relationstypen verlieren also die Verknüpfungseigenschaft als Komponenten vollständig mit ihrer Instanziierung.

Die Semantik des Meta-Metamodells führt für Relationen einen inhärenten Instanzbegriff ein, der diese immer einem **Modellelement** (Relationstypkomponente oder Relation) zuordnet, deren Instanz sie sind. Der Grund für eine Zuordnung von Relationen an Stelle von Relationstypkomponenten ist in Kapitel 5.4 beschrieben. Jede Relation speichert ihren Instanzvater (*instanceFather*) direkt und verfügt über keine Relationsenden. In einem konsistenten Komponenten-Relationen-Modell können nur Komponenten mit Relationen verknüpft werden.

6.3 Metamodelle

Die grundlegenden Definitionen sind nur durch eine Selbstreferenzierung des Modells möglich, so dass das objektorientierte Modell von OMICRON sich selbst definiert.

Ein **Metamodell** definiert den syntaktischen und semantischen Rahmen für die Erstellung eines Modells, ist aber selbst wiederum eine Modellinstanz des MMM. Ein Metamodell definiert alle erlaubten Typen und Relationen sowie deren Kombinationsmöglichkeiten. Jedes erzeugte Modell ist daher Instanz eines Metamodells. Die verwendeten Relationstypen werden auf der

Metamodellebene als Instanz von „Relation“ definiert und in den Metamodellen als Relationen instanziiert. Dort dienen sie zusammen mit den zugehörigen Komponententypen zur Beschreibung der unterschiedlichen Metamodelle. Auch Modelltypen und Modelle werden bei OMICRON durch Komponenten repräsentiert, um ein konsistentes Konzept für alle Ebenen zu erhalten.

Aus dem Metamodell (Komponente) wird dann mit Hilfe einer Instanziierung ein Modell erzeugt. Die im Metamodell abgeleiteten und verbundenen Komponenten stehen in ihrer Gesamtheit zur Verfügung. Die Zugehörigkeit des Modells zum Metamodell wird durch eine Instanzierungsrelation zum Metamodell, das heißt zum Modelltyp, realisiert. Da die Komponententypen durch das Konzept der Aggregation – quasi als Attribute eines Modelltyps – eingebunden werden, kann derselbe Komponententyp auch in mehreren Metamodellen direkt Verwendung finden oder als Referenz eingebunden werden.

Für das Metamodell und sein instanziiertes Modell ausschlaggebend sind nur Relationen, deren beide Enden/Komponenten mit dem Diagramm/Metamodell assoziiert sind. Es werden immer nur interne Relationen in die Vererbung und Instanziierung mit eingeschlossen, das heißt Relationen zwischen zwei enthaltenen Komponenten. Verbindungen zu Komponenten außerhalb der vererbten oder instanziierten Komponente werden ausgeblendet, so dass nur innerhalb eines Metamodells existierende Relationen auch im Modell verwendet werden können. Transitiv sind diese aber im Gesamtmodell erfasst. Nur die Duplikation des Kontexts wird im Rahmen der Vererbung und Instanziierung unterdrückt.

Auswirkungen von Inkonsistenzen durch Änderungen am Metamodell können so direkt überprüft werden, da die entsprechenden Instanz-Komponenten mit Ihren „Klassen“ verbunden sind. Die Klassen wiederum befinden sich direkt im Metamodell, so dass sowohl eine Auflistung abhängiger Instanzen (abwärts) als auch die Referenzierung der Klasse (aufwärts) bidirektional möglich ist.

Die Beschreibungsmächtigkeit wird dabei soweit möglich an OWL Full orientiert, die mehrere Meta-Ebenen umfassen kann (vgl. [OMG 2006d]), wobei ein Vergleich wegen der Aggregat- und Prozessvererbung nicht direkt möglich ist.

In OMICRON wird dieses Ontologienkonzept hinsichtlich mehrerer Meta-Ebenen und der Verwendung von Instanzen als Klassen umgesetzt. Allerdings ist aufgrund der Verwendung komplexer Komponenten eine einfache Hierarchisierung so nicht umsetzbar. Eine Vermischung von Meta-Ebenen (das heißt Instanzebenen) kann für Komponenten nicht erlaubt werden, da das definierte Modellkonzept auf jeder Instanzebene die Semantik für die nächste Instanzebene definiert. Würden Ebenen vermischt, kann die Semantik für untergeordnete Ebenen nicht mehr erschlossen werden. Eine bedingte Ausnahme bilden, wie beschrieben, nur Relationen und Konstanten. Als Sicht können jedoch aus dem Gesamtmodell bei Bedarf unterschiedliche Instanzebenen zusammen angezeigt werden.

Die gemeinsame Nutzung mehrerer Metamodelle zieht meist eine Kombination von Domänenontologien nach sich [Terrasse et al. 2006]. Da eine nachträgliche Fusion von Ontologien³¹ wie domänenspezifischen Sprachen (DSL, [OMG 2001]) viele Probleme aufwirft, wird die Integration in OMICRON durch ein gemeinsames Meta-Metamodell von Anfang an implizit ermöglicht.

³¹ Die Zusammenführung von Domänenontologien ist möglich aber aufwändig, Methoden [KostisVouros 2004] und Werkzeuge [NoyMusen 2002] hierzu existieren allerdings.

6.4 Basiselemente

Das gesamte Modell muss mit Hilfe einer Software interpretiert und vorgehalten werden. Da das Ziel ist, alle Bestandteile beliebig erweitern zu können, müssen alle Modellelemente durch das Gesamtmodell selbst definiert werden. Dies setzt voraus, dass **Basiselemente** definiert werden, die die Spitze der gesamten Vererbungs- und Instanzierungshierarchie und für die Software einen Ausgangspunkt darstellen

In einem konsistenten, objektorientierten Gesamtmodell ist es notwendig, eine Basiskomponente (Urkomponente) einzuführen, von der alle in der Modellverwaltung und -anwendung unterschiedenen Gesamtmodellbestandteile wie Komponenten und Relationen abgeleitet werden können.

Aus diesem Grund schlagen beispielsweise [AtkinsonKühne 2002] vor ein solches Basiselement einzuführen – allerdings mit der Unterscheidung in physische und logische Metalevels und unter der weitgehenden Beibehaltung beziehungsweise Anpassung der UML Infrastruktur: „*All that is essentially needed in this case is a ModelElement concept. With such a shrunken-down version of the superstructure in place, this advanced unification would—next to simplifying the P1 level and enabling an unbounded number of logical metalevels—have another very important effect.*“ [AtkinsonKühne 2002]

Es wird also notwendig, einen Modellursprung in Form einer Meta-Komponente **ModelElement** (vergleichbar Thing in OWL) im Meta-Metamodell einzuführen, die die Modellbasis nach oben hin bezüglich der Instanzierungsebenen abgrenzt und bereits für die Metamodell-Ebene die Grundsätze definiert. Ausgehend von dieser Urkomponente an der Gesamtmodell-Basis kann nun eine Vererbungs- und Instanzierungshierarchie geschaffen werden, die eine vollständige Modellwelt abbildet und über die Instanzierung (Instanzen) und die Vererbung (Spezialisierungen) ein vollständiges Gesamtmodell zur Verfügung stellt.

Da es sich um eine Komponenten-Relationenmodell handelt, werden diese Elemente bereits im Meta-Metamodell beschrieben und unterschieden: **Component** und **Relation** sind die ergänzenden Meta-Komponenten zu ModelElement.

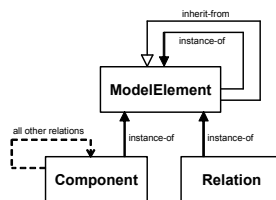


Abbildung 51: Das Meta-Metamodell als oberste Schicht des OMICRON-Gesamtmodelles

Unterschiedliche Ansätze zur Integration der basalen Komponenten- und Relationstypen wurden während der Entwicklung von OMICRON erprobt. Dabei stellte sich heraus, dass das Meta-Metamodell nicht ohne Auswirkung auf die gesamte Semantik mit weiteren Komponenten angereichert werden kann. Die Instanzierung einer allgemeinen Relation und die Ableitung der Relationstypen von dieser ist prinzipiell möglich. Für spezifische Konzepte der Vererbung, der Instanzierung und der Abläufe (Flow) erwies sich jedoch eine strikte Trennung der Basis-Relationstypen als vorteilhaft. So kann auf eine abstrakte allgemeine Relation (Powertype) verzichtet werden und die für die Laufzeitumgebung zur Interpretation relevanten Relations- und Komponententypen direkt als Instanzen des Meta-Relationstyps *Relation* realisiert werden.

So können auch zusätzliche Basis-Relationstypen, beispielsweise für generative Ansätze, durch Instanzierung von *Relation* definiert werden.

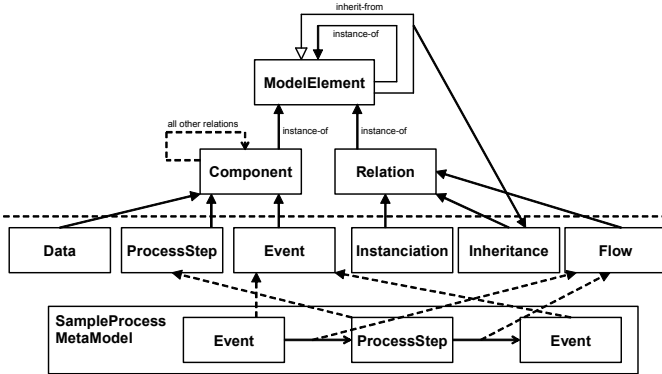


Abbildung 52: Ausschnitt aus dem Meta-Metamodell von OMICRON mit einem beispielhaften Metamodell

Im Folgenden wird gezeigt, dass die Vereinheitlichung der Vererbung- und Instanzierungssemantik über mehrere Ebenen weitgehend möglich ist, sogar unter der Prämisse der Prozessmodellierung mit verschachtelten, komplexen Komponenten und Modellbeziehungswise Diagrammtypen. Die Konzepte ermöglichen so eine Systemimplementierung ohne Unterscheidung von Ebenen und Modelltypen.

6.5 Formalisierung der Zusammenhänge des Meta-Metamodells

Das Meta-Metamodell wurde bis hier entwickelt und eingehend beschrieben. Im Folgenden sollen die grundlegenden Begriffe und Zusammenhänge des OMICRON Modells formal dargestellt werden. Hierbei wird die Mengentheorie wie in der Graphentheorie als Basis verwendet.

Da viele wechselseitige, axiomatische Bezüge bestehen ist nur die Darstellung als Ganzes verständlich, so dass die Formalisierung hier als integriertes Kapitel erfolgt.

Sei e_E das Basiselement des OMICRON Metamodells.

Sei E die Menge aller Modellelemente.

Sei T die Menge der Relationstypen.

Sei K die Menge der Komponenten.

Sei $v \in T$ der basale Relationstyp Vererbung, V die Menge der Vererbungstypen.

Sei $i \in T$ der basale Relationstyp Instanzierung, I die Menge der Instanzrelationstypen.

Sei R die Menge der Relationen, dann ist $E_{KT} = E \setminus R$ die Menge aller Modellelemente ohne die Relationen, das heißt die Menge aller Komponenten und Relationstypen $E_{KT} = K \cup T$.

Sei e_K das Basiselement *Component* aller Komponenten, das von e_E Instanz ist: $\exists (e_K, e_E, i) \in R$

Sei e_R das Basiselement *Relation* aller Relationstypen, das von e_E Instanz ist: $\exists (e_R, e_E, i) \in R$

Die Menge der (binären) typisierten Relationen in OMICRON ist Teilmenge der Verknüpfung aller Komponenten mit allen Komponenten mit allen Relationstypen und der Verknüpfung aller Relationstypen mit allen Relationstypen über Vererbung sowie der Instanzrelationen aller Relationstypen ausgehend von e_R :

Für R gilt daher: $R \subseteq (K \times K \times T) \cup (T \times T \times V) \cup (T \times e_R \times I) \subseteq (E_{KT} \times E_{KT} \times T)$

enthält $\{(e_E, e_E, v), (e_E, e_E, i), (e_K, e_E, v), (e_R, e_E, v)\}$ wegen $M_m \subseteq K$, wobei $(V \cup I) \subseteq T$ (siehe unten).

Die Meta-Komponente ModelElement erbt von sich selbst: $\exists (e_E, e_E, v) \in R$

Und ModelElement ist auch Instanz seiner selbst (Axiom): $\exists (e_E, e_E, i) \in R$

Sei M_m die Menge der Meta-Metamodellelemente (hier $M_m = \{e_E, e_K, e_R\}$), so dürfen alle Meta-Metamodellelemente mit keiner Beziehung einer anderen Komponente als e_E untergeordnet sein (Selbstreferenz fällt wegen $m = k$ nicht unter diese Regel):

$\neg \exists (m, k, t) \in R$ mit $k \in K \setminus \{e_E\}$, $m \in M_m$, $t \in T$, $m \neq k$

Die Menge aller Relationstypkomponenten T wird über die Instanziierung von e_R aus definiert. Alle Relationstypen sind Instanzen von e_R (nicht-transitiv):

$T = (\{e_R\} \cup \{x \mid (x, e_R, i_r) \in R\})$, $i_r \in I$

Alle Relationstypen sind also direkte Instanz von e_R : $\forall t \in T: \exists (t, e_R, i_r) \in R$, $i_r \in I$ (1)

und nur von e_R , da für alle Komponenten und Relationstypen E_{KT} , also alle Modellelemente, die mit Relationen verbunden werden dürfen, gilt, dass sie Instanz genau eines anderen Elements sein dürfen:

$(x, y, i_r) \in R \leftrightarrow \neg \exists (x, z, j_r) \in R$ mit $z \neq y$, $i_r, j_r \in I$

Zu T gehören auch I und $V - (V \cup I) \subseteq T$, da

$\exists (i, e_R, i) ; (i, e_R, i) \rightarrow I \subseteq T$

$\exists (v, e_R, i) ; (v, e_R, i) \rightarrow V \subseteq T$

Instanzen dürfen nur die Relationen haben, die im Metamodell vorgesehen sind (siehe hinten). Für Relationstypen die Vererbung, für Komponenten alles. Wegen (1) gilt:

$((e_R, e_R, v) \in R \wedge \neg \exists (e_R, e_R, x) \in R) \rightarrow \forall t \in T: (\neg \exists (t, s, w) \in R \wedge \neg \exists (t, s, w) \in R)$, $s \in T$, $x \in T$, $w \in T \setminus V$

Komponenten können jedoch ebenso wie Meta-Metamodellelemente alle Beziehungstypen eingehen, weil gilt:

$\forall t \in T (\exists (e_K, e_K, t) \in R \wedge \exists (e_E, e_E, t) \in R)$

Komponenten K sind alle transitiven Instanzen von e_K zusammen mit e_R und e_E :

$K = (\{e_K\} \cup \{x \mid (\exists y \in K \mid (x, y, i_r) \in R)\}) \cup \{e_R, e_E\}$, $i_r \in I$

Alle Komponenten sind Instanz genau einer anderen Komponente (nicht Instanz ihrer selbst):

$\forall k \in K \exists (k, f, i_r) \in R \wedge \neg \exists (k, g, j_r)$ $i_r, j_r \in I$, $f, g \in K$

$\forall k \in K \setminus \{e_E\} \neg \exists (k, k, i_r) \in R$, $i_r \in I$

Die Relationstypen und Komponenten dürfen jeweils nur untereinander Beziehungen haben:

$\forall (x, y, t) \in R: (x, y \in K) \oplus (x, y \in (T \cup \{e_R\}))$ (\oplus ist XOR weil $K \cap T = \emptyset$)

Für Relationstypen ist nur die Vererbung zugelassen.

$\forall (x, y, t) \in R$ mit $x, y \in T$ gilt: $t \in V$

Für K , T , R und E gilt:

$((K \cup T \cup R) = E) \wedge ((K \cap T) = \emptyset) \wedge ((K \cap R) = \emptyset) \wedge ((T \cap R) = \emptyset)$

daraus folgt auch (weil $K \neq \emptyset$, $T \neq \emptyset$, $R \neq \emptyset$) $((K \subset E) \wedge (R \subset E) \wedge (T \subset E))$

Klar ist daraus, ComponentElement e_K und RelationElement e_R haben sonst keine weiteren Beziehungen mit e_E : $\neg \exists s \in R = (e_K, e_E, x)$ mit $x \in T \setminus \{i\}$, $s \neq r$ sowie $\neg \exists s \in R = (e_R, e_E, x)$ mit $x \in T \setminus \{i\}$

Vererbungsrelationstypen V sind alle Relationstypen, die vom Vererbungsrelationstyp v erben.

Hinreichende Bedingung für die Zugehörigkeit eines Elements zur Menge der Vererbungskomponenten: $((x \in T) \wedge (\exists y \in V \mid (x, y, v) \in R)) \Rightarrow x \in V$

Das bedeutet dass die Menge der Vererbungskomponenten definiert werden kann über:

$$\forall x \in V: (\exists y \in V \mid ((x, y, v) \in R) \vee (y = v))$$

Alle mit der Vererbungskomponente per Vererbung des Typs v verbundenen Komponenten sind Teil der Vererbungskomponentenmenge. Hieraus ergibt sich zunächst auch die Beschränkung auf v für die Spezialisierung von Vererbungstypen, diese kann formalisiert aufgehoben und so V polymorph definiert werden, um mit der Objektorientierung auch auf der Metamodellebene konsistent zu bleiben (mit beliebigem $v_r \in V$; (x, y, v) bedeutet dabei „ x erbt von y “):

$$\forall x, v_r \in V: (\exists y \in V \mid ((x, y, v_r) \in R) \vee (y = v_r))$$

$$V = \{v\} \cup \{x \in T \mid (\exists y \in V \mid ((x, y, v_r) \in R) \vee (y = v))\}, v_r \in V. \text{ Damit ist } V \subset T.$$

Die Instanzrelationstypmenge ($I \subset R$) definiert sich gleichermaßen über alle Relationstypkomponenten

$$I = \{i\} \cup \{x \in T \mid (\exists y \in T \mid ((x, y, v_r) \in R) \vee (y = i))\}, v_r \in V$$

Vererbungen und Instanziierungen dürfen sich nicht überschneiden (sind disjunkt, auch keine Mehrfachvererbung): $I \cap V = \emptyset$

Aus obigem kann eine allgemeine Funktion abgeleitet werden, die für einen Relationstyp k und dessen rekursiv Subtypen definiert:

$$TTree(k) = \{k\} \cup \{x \in T \mid (\exists y \in TTree(k) \mid (x, y, v_r) \in R)\}, v_r \in V$$

Auch die Aggregation ist ein direkt von e_R instanzierter Relationstyp:

$$\exists a \in T: (a, e_R, i)$$

$$A = TTree(a)$$

u ist Basis-Relationstyp Use, der eingesetzt wird, wenn in einem Instanz-Kontext polymorph eine Spezialisierung der eigentlich geforderten Komponenten eingesetzt wird. Die polymorphe Komponente ist dann Instanz einer anderen, wird jedoch im Kontext benutzt.

$$\exists (u, e_R, i) \in R$$

$$U = TTree(u)$$

Generell dürfen Instanzen nicht von der Komponente erben, von der sie Instanz sind:

$$\forall (x, y, i_r) \in R: \neg \exists (x, y, v_r) \text{ mit } i_r \in I, v_r \in V \wedge x, y \in K$$

Dies reicht aber nicht aus. Nur Instanzen derselben Instanzebene dürfen Beziehungen eingehen.

Daher wird umfassender definiert, dass alle Relationen außer Instanziierungen ($R \setminus I$) nur Komponenten auf derselben Instanzebene verbinden dürfen.

Übergeordnete Instanz-Komponenten (InstanceFather) einer Komponente k sind dann:

$$I_k(k) = \{k\} \cup \{y \in K \mid (\exists x \in I_k(k) \mid (x, y, i_r) \in R)\}, i_r \in I$$

Dann müssen die Instanzebenen zweier durch eine Relation (außer Instanziierung) verbundener Komponenten gleich sein (**relationale Trennung der Instanzebenen**):

$\forall (k \in K, m \in K, t \in T \setminus I): (k, m, t) \rightarrow |I_k(k)| = |I_k(m)|$

Nur Instanzen des exakt selben Typs dürfen eine Vererbungsbeziehung eingehen:

$\exists (x, y, v_r) \in R \rightarrow \exists z \in K \mid (\exists (x, z, i_r) \in R \wedge \exists (y, z, i_r) \in R), i_r \in I, x \in K, y \in K, v_r \in V$

Die Menge der erbenden Komponenten (Subtypen, transitive Spezialisierungen) einer Komponente p ist:

$P_V(p) = \{p\} \cup \{x \in K \mid (\exists y \in P_V \mid ((x, y, v_r) \in R))\}, v_r \in V, P_V \subset K$

Die Menge der Komponenten von denen eine Komponente p erbt (Supertypen, transitive Generalisierungen, SubtypeOf) ist:

$Q_V(p) = \{p\} \cup \{y \in K \mid (\exists x \in Q_V \mid ((x, y, v_r) \in R))\}, v_r \in V, Q_V(p) \subset K$

Die Menge aller Komponenten $P_V(p)$, die von welchen erben, von denen p erbt (DAG) ist dann:

$P_{VT}(p) = \{Q_V(p)\} \cup \{x \in K \mid (\exists y \in P_{VT}(p) \mid ((x, y, v_r) \in R))\}, v_r \in V, P_V(p) \subset K$

Ist p eine Instanz von q – es gilt also $(p, q, i_r) \in R, p, q \in K, i_r \in I$ – so muss gelten, dass die Vererbungsgraphen von p und q disjunkt sind:

$P_{VT}(p) \cap P_{VT}(q) = \emptyset$

So wird auch eine Überschneidung der Instanzebenen innerhalb der Vererbungshierarchie verhindert. Trotzdem könnten sich noch Instanzebenen überschneiden, daher auch die oben eingeführte „relationale Trennung der Instanzebenen“.

Gleiches lässt sich für die Aggregation verwenden. Die Menge aller zum selben Modell (zur selben übergeordneten Komponente) gehörigen, aggregierten Komponenten ist:

$Q_A(p) = \{p\} \cup \{y \in K \mid (\exists x \in Q_A(p) \mid ((x, y, a_r) \in R))\}, a_r \in A, Q_A(p) \subset K$

$P_A(p) = \{Q_A(p)\} \cup \{x \in K \mid (\exists y \in P_A(p) \mid ((x, y, a_r) \in R))\}, a_r \in A, P_A(p) \subset K$

Die Teilkomponenten einer komplexen Komponente sind definiert als

$P_S(p) = \{p\} \cup \{x \in K \mid (\exists y \in P_S(p) \mid ((x, y, a_r) \in R))\}, a_r \in A, P_S(p) \subseteq P_A(p) \subset K$

($P_S(p) = P_A(p)$, wenn p eine Top-Level-Komponente ist.)

SubInheritance ist eine Spezialisierung der Referenzierung k_{ref} : $\exists (v_{sub}, k_{ref}, v) \in R$

$\exists (v_{sub}, c_R, i) \in R$ klar, da $v_{sub} \in T_{ref} \subset T$

Die Menge aller SubInheritance-Typen ist ein Teil-Ast der Referenztypen $v_{sub} \in T_{ref}$, nicht der Vererbungstypen:

$V_{sub} = P_V(v_{sub}), V_{sub} \subset T_{ref}$

Erbt eine Komponente q von einer Komponente p, dann gilt: für jede Teilkomponente von q existiert genau eine Ausgangskomponente in p (Teilkomponente von p) und umgekehrt. Wichtig ist: Zwar darf es zu einer Subkomponente p_{sub} kein zweites q_{sub} geben, umgekehrt jedoch können bei einer Mehrfachvererbung mehrere Komponenten fusioniert werden (nicht symmetrisch!).

$(\exists (q, p, v_r) \in R) \rightarrow \forall q_{sub} \in (P_S(q) \setminus \{q\}) (\exists p_{sub} \in (P_S(p) \setminus \{p\}) : \exists (q_{sub}, p_{sub}, t)) \wedge (\neg \exists (x, p_{sub}, s), q_{sub} \neq x, s \neq t, x \in (P_S(q) \setminus \{q\}), v_r \in V, t \in (V \cup V_{sub}), s \in (V \cup V_{sub}))$

Instanziierungsregel ist, dass Teilkomponenten der Instanz eine Relation desselben oder eines spezielleren Typs zwischen sich haben dürfen, wie er im Instanzvater definiert ist:

$(\exists (q, p, i_r) \in R \wedge (x_q, y_q, t_q) \in R \wedge x_q, y_q \in (P_S(q) \setminus \{q\})) \rightarrow (\exists (x_p, y_p, t_p) \in R \wedge x_p, y_p \in (P_S(p) \setminus \{p\}) \wedge t_q \in P_V(t_p) \wedge \exists (x_q, x_p, t_x) \in R, \wedge \exists (y_q, y_p, t_y) \in R, t_p, t_q \in T, t_x, t_y \in (I \cup U))$

t_x, t_y sind also entweder Instanzen oder Verwendungen.

In OMICRON existieren auch Referenzkomponenten, die die referenzierte Komponente nur repräsentieren und daher bei Verwendung zur eigentlichen Komponente aufgelöst werden müssen.

Referenz-Relationstypkomponente: $\exists (k_{ref}, e_R, i) \in R$ klar, da $k_{ref} \in T$

$T_{ref} = \{k_{ref}\} \cup \{x \in T \mid (\exists y \in T \mid ((x, y, v_r) \in R) \vee (y = k_{ref}))\}, k_{ref} \in T, T_{ref} \subset T$

Die Menge der Referenzkomponenten ist: $K_{ref} = \{k \in K \mid (k, x, t_r), x \in K, t_r \in T_{ref}\}$

Da eine Referenzkomponente in ihrem internen Aufbau quasi eine Projektion der referenzierten Komponente darstellen, folgt aus einer Referenzbeziehung die innere semantische Äquivalenz (\equiv) der Referenzkomponente zur referenzierten Komponente:

$(k \in K \wedge m \in K \wedge \exists (k, m, r_{ref})) \rightarrow (k \equiv m), r_{ref} \in T_{ref}$

k ist semantisch äquivalent zu m , d.h. Typ, Teilkomponententypen, interne Relationen etc. sind alle semantisch äquivalent und in beiden vorhanden. Wie bei der Vererbung müssen im Modell hierzu Teilkomponenten und interne Relationen erzeugt werden, die in der Referenzkomponente k alle Eigenschaften der Komponente m widerspiegeln.

Referenzierungen arbeiten ähnlich wie die Vererbung (die SubInheritance ist eine Referenzierung). Allerdings bieten Referenzen keine Änderungsmöglichkeiten (q referenziert p):

$(\exists (q, p, r_r) \in R) \rightarrow \forall q_{sub} \in (P_S(q) \setminus \{q\}) (\exists p_{sub} \in (P_S(p) \setminus \{p\}) : \exists (q_{sub}, p_{sub}, t)) \wedge (\neg \exists (q_{sub}, x, s) \wedge x \in (P_S(p) \setminus \{p\}), s \neq t, r_r \in T_{ref}, t \in T_{ref}, s \in T_{ref})$

Referenzkomponenten dürfen nicht erben (sie spiegeln nur die referenzierte):

$\forall k_{ref} \in K_{ref} \neg \exists (k_{ref}, x, v_r), x \in K, v_r \in V$

Eine Referenzkomponente darf die Subkomponenten nur widerspiegeln (genau für jede eine mit T_{ref} referenzierte Sub-Komponente enthalten):

$((\forall q_{sub} \exists p_{sub}) \wedge (\forall p_{sub} \exists q_{sub})) : \exists (q_{sub}, p_{sub}, t_r) \in R \wedge \exists (x_q, q_{sub}, t) \in R \wedge \exists (x_p, p_{sub}, t) \in R \wedge \exists (x_q, x_p, s_r) \in R, x_q, p_{sub} \in (P_S(p) \setminus \{p\}), x_p, q_{sub} \in (P_S(q) \setminus \{q\}), t_r, s_r \in T_{ref}$

Da SubInheritance von Reference erbt, gilt dasselbe für SubInheritance.

7 Prozessmodellierung und Ablaufkonzepte im Modell

Bisher wurden hauptsächlich vertikale, das heißt statische Konzepte beschrieben. Im Folgenden wird daher die Dimension der Dynamik in Prozessmodellen erläutert. Dabei wird speziell auf die Ablaufsemantik der in UML2 stark weiterentwickelten Aktivitäten eingegangen.

7.1 Modellanforderungen und Komponententypen

[Müller 2007] unterscheidet als grundlegende Komponententypen in Prozessen Aktivitätstypen und Resultatstypen, die EPK Ereignisse und Aktivitäten. Jedoch werden immer weitere Typen ergänzt. Im Folgenden werden die für die Anwendung in objektorientierten Prozessen der Produktion identifizierten Komponententypen durch Extraktion typischer Konzepte von Prozess-Metamodellen identifiziert.

Prozessmodelle arbeiten je nach Paradigma mit unterschiedlichen Konzepten. Die folgende Tabelle wurde aus vergleichenden Schriften und speziellen Prozessmodellen erstellt und mit dem im Folgenden vorgestellten Ansatz von OMICRON verglichen, um grundlegende Konzepte der Ablauforientierung zu identifizieren.

Prozessmodell	BPMN [OMG 2006a]	SLANG [Kriow et al. 1997]	[Milli et al. 2004]	[Brambilla et al. 2005]	PROMENA DE [FranchRib o 1999a]	eEPK [Klein et al. 2004] (+VKD)	[ListKorher r 2006]	[Caetano et al. 2005]	OMICRON
Eigenschaft									
Prozess	(Sub) Prozess	Prozess-typ	Prozess	Prozess	○	EPK	Prozess	(Business Objekt)	Aktivität
Aktivität	Aktivität, Aufgabe	Aktivität	Aktivität	Aktivität Typ, Inst.	Aufgabe	Funktion	Aktivität	Aktivität	Aktivität
Aktor	Pool, Lane	○	Aktor	Gruppe, Benutzer	○	Stelle, Person	Prozess-teilnehm.	Aktor	Aktor
Rolle	Pool	○	Rolle	○	Rolle	(Kunde), Rolle	Rolle	Rolle	(Rolle)
Organisation	Pool / Lane	○	Organi-sation	○	○	Orga.-Einheit	Orga. O.-Einheit	Organi-sation	○
Ereignisse	Ereignis	Nachricht	Ereignis	Ausnah-me(typ)	Trigger	Ereignis, Nachricht	Ereignis	○	Ereignis, Ausn.
Kontrollfluss	Kontrollfl., Gateway	Transition Rel., Stelle	Funktion	○	Prece-dence	Konnek-toren	Kontrollfl., -knoten	○	Kontrollfl., -knoten Tokens
Ressourcen	Daten-objekt	Aktive Kopie	Resource	○	○	Soft-/Hardw. Maschine	Resource Informatio nsquelle	Resource	Resource Produktio nsmittel
Modell	○	Modell-Typ	○	○	Modell / M.-Modell	○	○	(Business Objekt)	Modell / M.-Modell
Ziel	○	○	○	(Fall)	○	○	Ziel	Ziel	○
Ergebnis	Daten-objekt	○	○	○	Doku-ment	I/O-Leistung	Ergebnis	○	Output
Datenfluss	Daten-fluss	○	○	○	○	I/O-Daten	Daten-fluss	○	Input, Output
Material-/Erzeugnisfluss	○	○	○	○	○	○	○	○	Material Erzeugn.

○ nicht vorhanden

Tabelle 18: Vergleich von Konzepten der Prozessmodellierung

Materialfluss und Erzeugnisfluss sind Besonderheiten, die speziell für die Produktion relevant sind, jedoch auch in anderen Gebieten Probleme in der Automatisierung der Modelle erzeugen: Während Informationen und immaterielle Artefakte beliebig dupliziert werden können, müssen deren physische Entsprechungen Material und Erzeugnis genau verfolgt und gegebenenfalls

ersetzt werden. Materialien sind häufig Schüttgüter oder Rohmaterial, das als Ganzes häufig wenige vordefinierte Eigenschaften wie Geometrie, Gewicht etc. besitzt. Erst die Verarbeitung liefert Erzeugnisse mit starken typologischen Eigenschaften. Erzeugnisse von Teilschritten können in Folgeschritten weiterverwendet oder als Endprodukte identifiziert werden.

Relationstypkomponenten stellen eine Besonderheit dar und werden in keinem der untersuchten Modelle verwendet.

7.2 Kontrollfluss

Der Kontrollfluss regelt im ablaforientierten Modell die Folge von Komponenten, wie Aktivitäten und Ereignisse. Wichtig ist dabei seine Funktion der Aktivierung, die Ereignisse und Aktionen auslöst sowie neue Prozesse starten oder aktive durch Versiegen auch beenden kann.

7.2.1 Aktivierungseigenschaft des Kontrollflusses

Die **Flussrelation** (Flow) bestimmt, welche Komponenten in welcher Folge aktiviert werden. Wenn eine Flussrelation aktiviert wird, erfolgt im darauffolgenden Zyklus die Aktivierung der verbundenen Komponente. Die reine Aktivierung eines Knotens über eine Aktivierungsrelation hat zur Folge, dass dieser wieder ein neues Token empfangen kann. Dies gilt auch für Ein- und Ausgänge, so dass beispielsweise Listener für Ereignisse mit dem Prozess synchron geschaltet werden können.

Wurde das Aktivierungssignal weitergegeben, wird die Komponente inaktiv. Allerdings kann über eine reflexive Aktivierungskante ein Komponente als selbst(-re-)aktivierend modelliert oder später im Ablauf durch eine andere Komponente reaktiviert werden. Auf diese Weise können auch Parametersätze vollständig und synchron (re-)aktiviert werden

Mit Instanziierung und Befüllen der Eingangsparameter werden diese für weitere Signale geschlossen. Sobald die Daten oder das Ereignis weitertransportiert wurden, ist der Eingang wieder frei, möglicherweise aber noch nicht aktiviert. Erfolgt ein weiteres Signal ohne dass ein Eingang aktiv ist, muss gegebenenfalls eine neue Instanz erzeugt werden, um den nächsten Ablauf aufzunehmen und so doppelte Durchläufe zu vermeiden.

Die Aktivierung erfolgt immer **getaktet**, das heißt in der gesamten Prozessinstanz werden immer alle ausgehenden Aktivierungsrelationen aktiviert und die Komponenten deaktiviert. Im nächsten Takt werden alle mit aktivierten Relationen verbundenen Komponenten aktiviert, sofern alle notwendigen Eingänge aktiviert sind, die Relationen im Gegenzug deaktiviert.

Durch eine beliebige gemeinsame Komponente, die keine Verzögerung verursacht, können also Kontrollflüsse immer synchronisiert werden: Auf alle eingehenden Signale wird gewartet und ein synchrones ausgehendes erzeugt. Dies ist in UML2-Aktivitäten nicht möglich, so dass auch hier die Semantik von UML2 der von OMICRON trotz ähnlicher Konzepte nicht äquivalent ist.

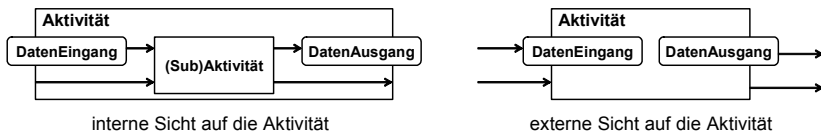


Abbildung 53: Daten-/Objekt- und Kontrollflüsse in und zu einer Aktivitätskomponente

Die UML2 nimmt eine Unterscheidung in **Objektflüsse** (in OMICRON: DataFlow) und **Kontrollflüsse** (in OMICRON: Flow) vor. Sobald mindestens ein Objektknoten an einer Flusskante beteiligt ist gilt die Kante als Objektfluss. Über die **Datenflussrelation** (DataFlow) fließen Marken (Tokens), die allerdings nur zur aktivierten Komponente fließen dürfen. Soll neben den Daten auch ein Aktivierungssignal übertragen werden, wird die **aktivierende**

Datenflussrelation (ActiveDataFlow) als gemeinsame Spezialisierung der Flussrelation und der Datenflussrelation genutzt.

Ein OMICRON Modell sichert bei einem nicht als Prozessschritt aufzufassenden Typ immer die Kompatibilität zum Typ des nächstliegenden Knotens, der mit einer Kante angebunden ist. Jede Flusskante darf nur zwischen kompatiblen Flussdaten (Informationen, Ereignisse, ...) oder zwischen Flussdaten und Aktivitäten existieren. Nach dem objektorientierten Paradigma ist eine Polymorphie nur in dieser Form möglich: Spezielle Ausgangs-Typen (Output) dürfen nur allgemeineren Eingangs-Typen (Input) zugewiesen werden, beispielsweise die Verwendung von „Siemens NC“, wenn eine allgemeine „NC“ gefordert ist.

Eine **Gewichtung von Kanten** (in UML2-Aktivitäten: {weight=n}) kann über die **Kardinalität** der Kante im Modell gegebenenfalls vorgenommen werden. Eine Kardinalität von Objektknoten bietet für die Prozessanimation keinen Mehrwert, da für jedes Objekt eine Instanz angelegt werden kann, mehrere Instanzen jedoch nur passend zu den durch die Kanten vorgegebenen Kardinalitäten. Ungebundene Objekte ohne Kontext dürfen nicht erstellt werden.

7.2.2 Ereignisse, Trigger und Listener

Ereignisse (*Event*) spielen eine wichtige Rolle bei der Definition von dynamischen Prozessen. Während Workflow-Ansätze Abläufe meist statisch definieren [IBM et al. 2003] oder rein planerische Ansätze [Thies 2003] verfolgen bildet ein dynamischer Abruf von flexiblen, jedoch automatisierbaren Abläufen die Grundlage für ein flexibles Prozess-Laufzeitsystem in der Produktion. Ereignisse sind ein häufig in aktivitätenbasierten Prozessdarstellungen verwendetes Konzept, um diese Flexibilisierung durch einen kontextabhängigen Aufruf von Prozessen zu erreichen.

Die strukturierten Abläufe von Prozessen (Aktivitäten) werden mit Hilfe des Komponenten-Relationen-Modells beschrieben und dienen als Grundgerüst für eine planbare Fertigung. Die Aktivitätenfolge ergibt sich dabei aus der Flussrelation, die Aktivitäten in eine Ordnung einbindet.

Häufig treten jedoch Ereignisse auf, die eine Modellierung sehr komplex und redundant machen würden, da alternative Abläufe für jede Situation explizit im Prozess modelliert werden müssten und zudem bei der Modellierung häufig nicht klar ist, welche Situationen im Kontext einer Aktivität zur Laufzeit auftreten können.

Ereignisse ermöglichen hier eine flexible Interaktion zwischen existierenden Prozessinstanzen und auch die bedarfsgerechte Erzeugung zusätzlicher Prozessinstanzen aufgrund der aufgetretenen Ereignisse. OMICRON stellt daher ein hybrides Ablaufmodell zur Verfügung, das konkret definierte ebenso wie ereignisbasierte Kontrollflüsse in Aktivitäten integriert. Der Austausch von Ereignissen ist eine wichtige Voraussetzung für modulare und austauschbare Teilprozesse, da anhand dieser die Steuerung auch ohne vorgegebene Kanten über typisierte Ereignisse und Kontexte erfolgen kann. Ein **Ereignis** ist dabei ein Konstrukt, das sowohl eine eigene Identität als auch einen Ereignis-Typ und gegebenenfalls Daten besitzt.

Auch die UML2 sieht Ereignisse bei der Modellierung von Aktivitäten vor, die genau zeigen, wo ein Ereignis generiert und wo konsumiert wird:



Abbildung 54: Ereignisse in UML und OMICRON: Trigger (Send) und Listener (Receive)

Nachrichtenergebnisse können in der UML [OMG 2007a] Operationen oder Signale sein. In OMICRON sind Operationsereignisse nur implizit definiert: Sobald ein Signal empfangen wird, werden an dieses angebundene Aktivitäten ausgeführt.

Trigger (Auslöser eines Ereignisses) eines Typs erzeugen dabei Ereignisse, die von Listnern aufgefangen und behandelt werden. Durch die Typisierung im OMICRON Modell kann eine umfassende implizite Zuordnung von Ereignistypen erfolgen, die das System flexibel und dezentralisierbar macht. Trigger sind dabei hinreichend aber nicht immer notwendig für die Aktivierung einer Aktivität. Im Gegensatz dazu sind **Vorbedingungen** (Preconditions) zwar notwendig aber nicht hinreichend für die Auslösung, da sie nicht aktivierungsfähig sind.

Listener sind das Gegenstück zu Triggern und sorgen für den Empfang von Ereignissen. Dank des Gesamtmodells können die erzeugten oder aktivierten Prozesse in unterschiedlichen Teilmodellen oder sogar anderen Projekten vorliegen. Ein Listener kann dabei auf Ereignisse seines Typs und kompatibler Subtypen reagieren. Ein vom Laufzeitsystem angebotenes Ereignis wird von einem kompatiblen Listener daher direkt konsumiert.

Eine Flussrelation, die zwei einander folgende Aktivitäten verbindet, kann daher auch als **implizites Ereignis** (beispielsweise „Bolzen nehmen“ löst „Bolzen einsetzen“ aus) interpretiert werden. Dabei ist „Bolzen nehmen“ impliziter Trigger und „Bolzen einsetzen“ impliziter Listener.

Ereignisse können als Typ allgemein, aber auch für bestimmte Bereiche abonniert werden, um zu vermeiden, dass Ereignisse aus anderen Bereichen unerwünschte Auswirkungen haben. Hierzu können Systeme hierarchisiert werden, so dass beispielsweise nur unbehandelte Ereignisse aus einer Zelle in die gesamte Fabrik propagiert werden. Auch kann durch eine entsprechende Implementierung eine Empfangsberechtigung über Rollen geregelt werden.

Wichtig ist, dass ein Ereignis aktivierungsfähig ist und somit den **Prozessfokus**³² auf den Listener und dann auf seine nachfolgend angebotenen Aktivitäten verschieben kann.

Freie Ereignisse werden von **freien Triggern** erzeugt, die nicht mit einem Listener verbunden sind. **Gesteuerte Ereignisse** werden dagegen erzeugt, wenn der Trigger über eine ausgehende Flussrelation verfügt (**gesteuerter Trigger**) und bei Auslösung von einem Trigger direkt zu einem oder mehreren verbundenen Listnern übertragen.

Um Seiteneffekte zu verhindern, wird definiert, dass eine Reaktion auf ein kompatibles Ereignis nur erfolgt, wenn der Listener nicht bereits durch eine eingehende Flussrelation vorbelegt ist, als ein **freier Listener** ist. Ein **gesteuerter Listener** reagiert damit nur auf ihm zugeordnete gesteuerte Ereignisse. Mit Hilfe der expliziten Verbindung von Ereignissen zwischen Trigger (Auslöser) und Listener (Empfänger), kann also ein Ereignisverlauf von anderen Ereignissen abgeschottet werden. Handelt es sich um eine Relation innerhalb einer (aggregierten) Aktivität, ist die Beziehung bereits auf Instanzebene definiert und abgeschottet. Sind lediglich zwei Aktivitäten auf Prozessebene verbunden, ist nur der Typ des Trigger-Prozesses und die entsprechende (Sub-)Aktivität in der Prozessbeschreibung festgelegt. Welche Instanz zum Zuge kommt ist dann offen und kann nur über den Kontext zur Laufzeit bestimmt werden.

Auch Ereignisse können als komplexe aus atomaren Komponenten aufgebaut werden, beispielsweise mit „*super-/subevent relationships*“ [OpdahlBerio 2006]. Sie folgen dabei demselben Aggregations- und Typkonzept wie alle anderen Komponenten in OMICRON.

In UML2-Aktivitätendiagrammen sind `SendSignalAction` und `AcceptEventAction` als Sonderformen der Aktion vorgesehen [OMG 2007a][Jeckle et al. 2004], die mit einem Payload³³

³² Bei parallelisierter Ausführung auch einen der Prozessfoki: Gesamtheit aktivierter Komponenten und Relationen.

³³ zu übermittelnde Daten; im Fall von OMICRON Komponenten beziehungsweise typisierte Objekte

versehen werden können. Diese dienen zum Versand von Ereignissen oder der Aktivierung durch solche. In OMICRON sind diese bereits implizit durch aus- oder eingehende Ereignisse (Trigger und Listener) definiert.

Auch für Ereignisse gilt die **Polymorphie**, so dass ein spezielles Ereignis einen allgemeineren Listener auslösen kann, jedoch nicht umgekehrt.

Beispiel 13: Beispielsweise löst das Ereignis „Bolzen produziert“ auch den Listener „Werkstück produziert“ aus, da Bolzen ein Werkstück und damit „Bolzen produziert“ eine Spezialisierung von „Werkstück produziert“ ist.

Die **Auswahl der korrekten Instanz** erfolgt dabei entweder anhand der direkten Verbindung, anhand der Komponenten-ID der Prozessinstanz bei prozessinternen Ereignissen oder anhand des Ereignistyps. Bei Mehrdeutigkeit kann eine nichtdeterministische oder benutzergesteuerte Auswahl implementiert werden.

Weitere Methoden wie das im Ausblick beschriebene Fuzzy Matching mit FOOP, adaptive Fuzzylogik [AmmerlaanWright 2004], Fuzzylogik auf objektorientierten Modellen [Benedicenti et al. 1998] oder gewichtungs- beziehungsweise grenzwertbasiertes Matching können die Auswahl ergänzen, wenn mehr Flexibilität oder weitere Auswahlkriterien notwendig sind. Grundsätzlich ist jedoch das tybasierte Matching die einzig semantisch sichere Variante, da anhand von Eingangssignaturen, Gewichtungen oder Ähnlichkeiten die Prozessemantik nicht erschlossen werden kann.

7.2.3 Übergeordnete und zeitbezogene Ereignisse

Auch **übergeordnete oder zusätzliche Prozesse** können durch Ereignisse aktiviert werden. Diese zusätzlichen Prozesse werden in [ScheerWerth 2005] im Bereich Geschäftsprozesse als dispositive oder Meta-Prozesse bezeichnet. Da diese jedoch auf derselben Abstraktionsebene wie die aufrufende Aktivität liegen und daher dem Meta-Begriff nicht entsprechen, werden diese in OMICRON als **dispositive oder ergänzende** Aktivitäten definiert.

Für Prozesse ist zunächst nur die Folge beziehungsweise Verknüpfung von Aktivitäten und Ereignissen ausschlaggebend. In der Praxis spielt die Zeit jedoch oft eine entscheidende Rolle. Speziell für die Produktionsplanung mit der Randbedingung einer optimalen Auslastung ist die Dauer von Aktivitäten das Basiskriterium für die Planung, so dass bereits zeitbehafete Netze entwickelt wurden (vgl. [HaggeWagner 2005]).

Zeitbezogene beziehungsweise zeitabhängige Ereignisse sind daher ebenfalls Teile eines vollständigen Prozess-Systems. Die UML enthält das Zeitereignis (TimeEvent) zusätzlich zu Änderungs- (ChangeEvent) und Nachrichtenereignissen. Um dieses Konzept in der Laufzeitumgebung umsetzen zu können, ist ein Zeitereignis-Generator oder zumindest ein Zeitgenerator notwendig. Durch Definition einer zu überschreitenden Schranke und die damit verbundene Auslösung eines Ereignisses sowie getaktete oder kontinuierliche Abfrage entsteht aus einem Zeitgenerator ein Zeitereignisgenerator. In der Produktion kann die Taktung hieran geknüpft werden, so dass der Prozess taktabhängig arbeitet. So können beispielsweise auch Wartezeiten wie Abkühlzeiten etc. berücksichtigt werden. Hierzu ist eine Systemzeit notwendig, die im Prototyp nur lokal gemessen wird, im verteilten System, dann jedoch synchronisiert werden müsste.

7.2.4 Ausnahmen

Wichtig für ein funktionierendes Laufzeitsystem ist eine entsprechende Fehlerbehandlung. In Programmiersprachen wird dies häufig mit dem Konzept der Ausnahmen (Exceptions) gelöst. Hierbei führen bestimmte Fehlertypen zu einem bestimmten Ausnahme-Typ, der mit Hilfe einer Bearbeitungsroutine (Exception Handler) entweder verarbeitet und aufgelöst werden kann oder zu einem Abbruch des Programms – hier der Aktivität – führt. Im Gesamtmodell oder im

spezifischen Prozess können bereits Handler vordefiniert werden, die die Verarbeitung übernehmen, wenn keine sonstige Behandlung vorgesehen ist.

Auch das Konzept der Exception Handler entspricht zunächst dem der allgemeinen Ereignisbehandlung, wird aber in in OMICRON ebenso wie in der UML2 und anderen Sprachen explizit modelliert: Eine Ausnahme *Exception* ist eine Spezialisierung von Ereignis *Event*.

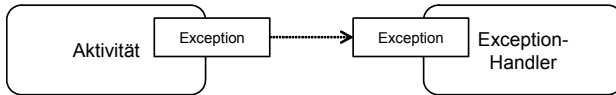


Abbildung 55: Ausnahmen in OMICRON und UML2

Ausnahmen können so als eine spezielle Art von Ereignissen behandelt werden. Im Gegensatz zu Programmiersprachen, darf allerdings ein Laufzeitsystem wie OMICRON und selbst eine Prozessinstanz nicht beendet werden, wenn eine Exception nicht behandelt wird. Vielmehr müssen unterschiedliche Standardreaktionen je nach Schwere des Fehlers definiert werden.

Die Fehlerbehandlung erfolgt in OMICRON für vorgesehene, modellinterne Ausnahmen über das Ereigniskonzept. Eine modellierte Aktivität zur Fehlerbehandlung wird so aktiviert, gegebenenfalls auch erzeugt und erhält alle mit dem Ereignis verbundenen Kontextinformationen. Eine Klassifikation möglicher Ausnahmen und deren Behandlung befindet sich in Anhang K.

7.2.5 Tokens

Petrinetzbasierte Modelle ermöglichen eine gute Darstellung von Abläufen und wurden daher intensiv für die Ablaufmodellierung genutzt. Statt Zustandsautomaten (StateMachine) in UML 1.4 [OMG 2004a] verwendet die UML2 nun ebenfalls das mächtigere Konzept von Marken oder Tokens der Petrinetze (vgl. [OMG 2007a], S.324), das eine einfache Parallelisierung von Abläufen erlaubt.

Auch mit der Objektsemantik von OMICRON (Klassifizierung/Vererbung etc.) ist ein Einsatz dieses Konzepts möglich: Objektorientierten Prozessschritten (Aktivitäten), Ereignissen, Bedingungen, Parametern, Artefakten, Daten, etc. kann eine dem Petrinetz ähnliche Ablaufsemantik zugeordnet werden. Eine Besonderheit ist dabei allerdings, dass durch die Polymorphie selbst **gefärbte Petrinetze**³⁴ nicht die vollständige Semantik abdecken können, da OMICRON nicht mehr mit einfachen Marken, Stellen und Transitionen arbeitet. Vielmehr sind die „gefärbten“ Marken typisiert, identitätsbehaftet und polymorph, so dass nur Marken mit dem korrekten Subtyp zugelassen werden. Dieser wird in OMICRON Modellen durch die typisierte Stelle definiert, die das Token aufnehmen und weiterleiten kann.

Das sehr allgemeine und abstrakte Token-Konzept der UML2-Aktivitäten dient zunächst der Darstellung des Systemzustands: Wo Informationen verfügbar sind oder der Prozessfokus liegt, sind stets Tokens vorhanden, die die entsprechende Komponente oder Relation im Prozess markieren. Das Token-Konzept ist in OMICRON allerdings streng typisiert: Nur Objekte des definierten Typs oder seiner Subtypen sind an der entsprechenden Stelle zulässig. Ein Ereignis, bestimmte Daten oder allgemein der Prozessablauf aktivieren das jeweils nächste Element.

Materialisierung von Tokens: Während des Ablaufs wird für jedes Token eine Instanz des zugehörigen Parameters oder Prozessschritts in der Prozessinstanz erzeugt oder mit dem Tokenobjekt verbunden, so dass das Token in der Prozessinstanz „materialisiert“. Dort bleibt es

³⁴ Eigentlich handelt es sich um ein objektorientiertes Petrinetz als Spezialisierung des gefärbten, da typisierte Marken verwendet werden. Allerdings ist der Begriff des objektorientierten Petrinetzes bereits mit Petrinetzen vorbelegt, die als Systemnetz selbst Teilnetze als Marken bewegen, siehe Grundlagen.

als Instanz oder Use-Objekt – angebunden über die Verwendungsrelation *Use* – verfügbar, bis der Kontext degeneriert, das heißt der Prozessfokus zur nachgeordneten Aktivität übergeht oder gestoppt wird. Die Information wie Parameterwert oder Ereignis wird dann vernichtet.

Die Typisierung und Instanzorientierung von OMICRON verbietet eine implizite Fusion von Tokens zu einem neuen, da diese zu einem Typverlust führt. Eine beliebige explizite Verknüpfung zu einer neuen beziehungsweise geänderten Instanz eines neuen Typs ist möglich.

Startknoten und Endknoten dienen der expliziten Aktivierung und Vernichtung von Prozessinstanzen. Während die WebML nur eine StartActivity und eine EndActivity vorsieht [Brambilla et al. 2006], gibt es in UML2 zwar einen Startknoten aber zwei unterschiedliche Arten von Endknoten:



- a) ein Endknoten für Kontrollflüsse beendet nur diesen Ablauf und vernichtet das aktuelle Token
- b) Endknoten für den gesamten Prozesskontext (finaler Endknoten)

Abbildung 56: Endknoten-Typen

Sowohl Startknoten als auch Endknoten für Kontrollflüsse sind in OMICRON zunächst implizit. Die Aktivierung erfolgt durch den Kontrollfluss und Ereignisse von außen über die Eingangsparameter. Der Kontrollfluss endet in jeder Aktivität ohne Folgeaktivität – hierzu zählen auch Ausgänge. Ein finaler Endknoten zur Beendigung einer gesamten Aktivität muss explizit modelliert werden oder wird implizit erreicht, wenn alle Kontrollflüsse in Endknoten für Kontrollflüssen gemündet sind.

Bei Aktivitäten mit mehreren Aggregationsebenen wird entweder die höchste per Aggregation übergeordnete Aktivität beendet (**absoluter finaler Endknoten**) oder genau die Aktivität, die den Endknoten direkt aggregiert (**relativer finaler Endknoten**). Finale Endknoten sollten wegen der Seiteneffekte allerdings nicht eingesetzt werden. Terminiert die gesamte Prozessinstanz, müssen zur Erhaltung eines konsistenten Zustands auch sämtliche Objekte in der Prozessinstanz zerstört werden, einschließlich derer, die noch den Prozessfokus haben. Wird nur ein Endknoten für Kontrollflüsse erreicht oder die Aktivität durch eine Kante verlassen, bleiben Objekte anderer Kontrollflüsse bestehen.

Während in Modellierungsmethoden, die nicht an ein Laufzeitsystem gekoppelt sind, meist finale Endknoten verwendet werden, ist in OMICRON der Endknoten für Kontrollflüsse der Normalfall. Die Konsistenz eines Gesamtmodells mit vielen Abläufen lässt sich meist nur sichern, wenn alle Abläufe planmäßig bis zum Ende durchgeführt werden.

Beispiel 14: Wird beispielsweise parallel zur Bearbeitung ein Datenblatt ausgefüllt, würde sonst mit dem Ende der Bearbeitung gegebenenfalls kein Datenblatt mehr existieren.

Es muss also sichergestellt werden, dass alle regulären Abläufe zu Ende geführt werden. Unberührt hiervon sind Ausnahmen, die bei irregulären Abläufen eine vollständige Beendigung erfordern können.

Workflow-Werkzeuge animieren einen Prozess meist nur, so dass manuell eine neue „Instanz“ durch Kopieren und Initialisieren von Werten angelegt werden muss, wenn ein neuer Ablauf gestartet wird. Bei OMICRON wird für jeden ausgelösten Ablauf eine komplett neue Instanz erstellt. Die Konsequenz ist, dass für jeden gestarteten Ablauf eine eigenständige Instanz mit eigener Identität existiert.

Ein mehrfaches Eintreten von Ereignissen und Parameterwerten ist so nur möglich, wenn die Instanz absichtlich adressiert wird oder für das entsprechende Token noch ein freier Slot verfügbar ist. Ein Ereignis desselben Typs kann also beispielsweise nicht mehrfach an dieselbe

Instanz gesendet werden, wenn deren Empfänger dafür bereits alle genutzt (belegt) und nicht mehr reaktiviert wurden.

Sollen Ereignisse oder Daten nicht abgewiesen werden oder verloren gehen, müssen die entsprechenden Stellen im Prozess mit einer Kardinalität größer 1 versehen werden (in UML als Objektknoten mit {upperbound=n} definiert). Da die Reihenfolge in Aggregaten auch auf Instanzebene erhalten wird, können Puffer (vergleichbar dem Stereotyp <<centralBuffer>> in UML2) in OMICRON explizit nach dem Stapel- (LIFO-)³⁵ oder Warteschlangen- (FIFO-)³⁶ Verfahren definiert und implementiert werden. OMICRON verwendet standardmäßig den FIFO-Mechanismus, da im dezentralen Konzept die Erhaltung der Reihenfolge Kausalitätsverletzungen vermeiden hilft.

7.2.6 Daten- und Materialflüsse

Da die Objektorientierung im Kern eine daten- und strukturzentrierte Methodik ist, verursachen (Daten-)Felder keine oder geringe Probleme. Einmal definierte Felder behalten über die ganze Hierarchie ihren Typ und Namen bei, so dass keine Veränderungen außer der Sichtbarkeit (public, protected, private, vgl. Anhang I) möglich sind. Daher werden in vielen Ansätzen wie dem Semantischen Objektmodell (SOM) [FerstlSinz 2001] Daten objektorientiert gehandhabt und manipuliert werden, Prozessschritte jedoch von der Objektorientierung ausgeschlossen.

Ebenso wie Aktivitäten und Ereignisse sind Daten- und Materialflusskomponenten anhand ihres Typs zu erkennen. Die Steuerung von Prozessen in Netzwerken erfolgt meist nur über Input/Output, mit den Attributen Menge, Zeit, Qualität [Klein et al. 2004] und Kosten.

Diese Sicht wird in OMICRON erweitert: Input und Output sind als Teil der **Flussdaten** alle dynamischen, flüchtigen Informationen beziehungsweise Informationszuordnungen während des Ablaufs. Diese werden während des Prozessablaufs in Form von Tokens immer weitergereicht. Sie entstehen im Ablauf aus (meist extern aufgerufenen) Aktivitäten und werden gegebenenfalls „verbraucht“, so dass sie als flüchtige Bestandteile des Gesamtmodells im Laufzeitsystem entstehen, fließen und wieder verschwinden.

Beispiel 15: Ereignis 1 aktiviert Aktivität 3 zusammen mit Information X. Dabei wird Information X von Aktivität 1 erzeugt und von Aktivität 3 wieder vernichtet.



Abbildung 57: Ereignisse und Informationen entstehen und werden vernichtet

Das Ergebnis dieses Konzepts ist ein Prozess-System, das immer nur die zur Ausführung benötigten Informationen als Flussdaten in Form von Instanzen eines Prozessmodells enthält. Vergleichbar dem Strom in der Physik existieren und bewegen sich Informationen nur im Ablauf von Aktivitäten und materialisieren in Form von Tokens dort wo sie benötigt werden. Die UML2 sieht ebenso Objekt-Flussdaten innerhalb von Aktivitäten in Form von **Objektknoten** vor. Diese können von Aktivitäten verbraucht oder erzeugt werden.



Abbildung 58: Objektknoten – Flussdaten in der UML2

³⁵ Stapel, Last In First Out Stack (LIFO): das jeweils letzte empfangene Element wird als erstes weitergegeben

³⁶ Warteschlange, First In First Out Queue (FIFO): das erste empfangene Element wird als erstes weitergegeben

Die Notation ermöglicht dabei eine Unterscheidung zwischen externen Objektknoten wie oben und internen Objektknoten, die als Parameter dienen. Diese Darstellung verdeutlicht die Rolle der Objektknoten als Parameter beziehungsweise Signatur in OMICRON:

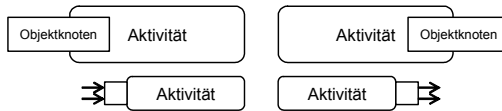


Abbildung 59: Objektknoten in der UML2 Objektknoten- (oben) und Pin-Notation (unten)

Der Objektknoten ist also Teil der Signatur (vgl. Kapitel 5.2.1). Handelt es sich um einen nicht angebundnen Parameter, muss dieser während des Ablaufs angebunden oder interaktiv befüllt werden:



Abbildung 60: Aktivitäten in Pin-Darstellung der UML2 ohne angebundene Quelle / Ziel

Meist entstehen aber in einem System auch Informationen, die auch nach Beendigung ihrer zugehörigen Aktivität(en) verfügbar sein sollen. Hier würde ein reines Flussdatenkonzept zu Datenverlust führen. Daher führt OMICRON das **Artefakt** als zweites Informationskonzept ein. Wie Flussdaten sind Artefakte typisiert und erhalten eine Identität.

Beispiel 16: Eine Teilespezifikation und ein Prüfbericht sind Artefakte, die als Information in den Prozess eingebracht werden (Teilespezifikation) oder aus diesem entstehen (Prüfbericht). Dagegen sind das Ereignis „starte NC-Programm b“ und das Datum „Maschinen-Nummer“ Flussdaten.

Ein Artefakt ist nicht-flüchtig, so dass es auch nach Aufhebung seiner Zuordnung als Parameter (beispielsweise „Dokument 795 ist die Teilespezifikation für das gerade produzierte Teil“) existent bleibt. Jedes Artefakt kann als Referenz- oder Werteartefakt geführt werden:

Ein **Referenzarteфakt** wird mit einem persistenten Objekt in der Datenhaltung, beispielsweise in einer Datenbank oder einem separaten Modellbereich, verknüpft (vgl. Anhang S). So sind Änderungen möglich und der Zustand entspricht immer dem aktuellen Stand der Änderungen in der Aktivität. Auch Änderungen durch andere, parallel existierende Prozessinstanzen werden dann wie bei Datenbankzugriffen direkt sichtbar. Hier kann ein Transaktions- oder Sperr-Mechanismus notwendig werden.

Als **Wertearteфakt** werden die Inhalte des Arteфakts bei Wertübergabe einmalig dupliziert. Änderungen sind daher am lokalen Wertearteфakt möglich, haben jedoch keinen Einfluss auf das Arteфakt selbst, vergleichbar dem Stereotyp <<datastore>> für Aktivitäten in UML2, der nur Kopien von Werten liefert.

Beispiel 17: Wird beispielsweise der Zeitpunkt des Fertigungsbeginns für ein Teil aus einem Arteфakt eingelesen und dieses als Wertearteфakt übergeben, kann während des Ablaufs durch Überschreiben des Beginn-Zeitpunkts des aktuellen Prozessschritts eine Anpassung erfolgen, so dass im Prozess immer ein aktueller Wert zur Verfügung steht. Der Zeitpunkt des Fertigungsbeginns im Ausgangs-Arteфakt bleibt dabei aber unberührt in der Datenbank.

Eine wichtige Rolle spielen hierbei die IDs von Datenobjekten, mit denen diese sowohl in einer Datenbank als auch im Modell eindeutig identifiziert werden können.

Datenströme (Streams) sind für OMICRON nur in ihrer diskretisierten Form verwendbar: Von Positionswerten wird beispielsweise der aktuelle verwendet.

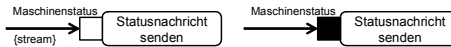


Abbildung 61: Alternative Notationsformen für Streams in UML2

Durch das objektorientierte Gesamtmodell von OMICRON sind Ein- und Ausgabeparameter immer typisiert. Die Unterscheidung in und Interpretation als Flussdaten (Informationen, Ereignisse), Artefakte, Trigger, Ausnahmen, Datenspeicher, Generatoren etc. erfolgt aufgrund dieses Typs.

Neben den Datenflüssen sind **Materialflüsse** für die Modellierung von Prozessen in der Produktion wichtig. Ähnlich wie Datenflüsse können Materialflüsse in unterschiedlichen Prozessschritten Verwendung finden. Dagegen können Materialfluss-Atome (Stücke) nicht dupliziert (aufgeteilt) werden oder in mehreren Schritten oder an mehreren Orten gleichzeitig genutzt werden. Zudem müssen informationstechnisch im Modell realisierte Materialflüsse bei der Prozessanimation immer mit der Realität synchronisiert werden. Hierzu dienen Verfahren wie RFID (Radio Frequency Identification), Barcodes oder Benutzereingaben, die eine Identifikation und Positionsbestimmung zulassen.

Im Modell werden Materialflüsse – abgesehen von der Rückkopplung zur tatsächlichen Umgebung – gleich den Datenflüssen behandelt, jedoch nicht dupliziert.

7.2.7 Verzweigungen und Vereinigungen

Logische Verknüpfungen werden auch zur Aufspaltung und Zusammenführung des Kontrollflusses genutzt. Verbreitet sind dabei AND, OR, XOR als Split und Join Variante sowie N-aus-M-Verknüpfungen [ListKorherr 2006]. OR- und XOR-Split sind dabei mangels Auswahlkriterium nicht automatisierbar – Vereinigungen schon, wenn bei hinreichender Bedingung ausgeführt wird. Bei OR und XOR-Split muss also eine interaktive oder regelbasierte Auswahl erfolgen.

Die logischen Verknüpfungen können nicht nur auf binäre, sondern auch auf Splits und Joins mit mehr als zwei Zweigen angewandt werden:

OR (1..n out of n; ≥ 1),

XOR (1 out of n; =1),

AND (n out of n; =n)

Hier findet eine Auswahl der weiterzuleitenden Signale und nicht der Vergleich mit anschließender Erzeugung eines neuen Aktivierungssignals statt. Anliegende Signale warten gegebenenfalls bis weitere die Bedingung erfüllen. Logische Komponenten berücksichtigen die Operanden in der Reihenfolge der eingehenden Relationen.

UML2 ermöglicht die Kommentarspezifikation an Verbindungsknoten zur informalen Spezifikation von Bedingungen wie $[X>0]$, die WebML [Brambilla et al. 2006] dagegen konkrete Verzweigungen. Die UML2 fordert daher, dass Bedingungen bei Verzweigungen immer vollständig sein müssen, da das Token sonst stecken bleibt beziehungsweise es unklar ist, welcher Pfad bei mehreren gewählt werden soll. Dies ist bei Parallelisierung jedoch naturgemäß unproblematisch. Während Verstöße in der UML daher möglich sind, aktivieren **Vergleichskomponenten** in OMICRON immer einen der beiden Ausgänge als Ergebnis der Auswertung. Ob beide weiterverbunden werden ist Sache des Modellierers, so dass Gates (ein selektiv aktivierter Ausgang) eine Spezialisierung von Vergleichskomponenten (zwei alternierende Ausgänge) sind.

Da Case-Statements (wenn 0, dann x, wenn 1, dann y, wenn 2, dann z, ...) geschachtelte Abfragen darstellen, werden sie ebenfalls mit Vergleichskomponenten modelliert. Bei Parallelisierung und mehreren zutreffenden Bedingungen sind mehrere parallele Abläufe möglich, bei Schachtelung entsteht ein sequentielles Case-Statement, das je Durchgang nur eine Alternative aktivieren kann.

Einer integrierten ebenso wie einer externen Funktionskomponente können beliebige Parameter zugeordnet werden, die an die aggregierende Funktionskomponente „übergeben“ werden. So können Vergleiche durch externe Aufrufe realisiert werden:

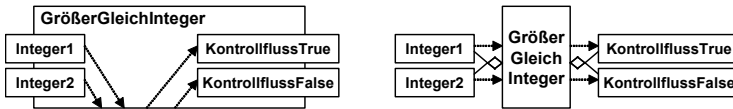


Abbildung 62: Vergleichskomponenten als externe Funktion – in Komponenten und Aggregatdarstellung

Eine Unterscheidung zwischen Parallelisierung und alternativer Entscheidung ist bei einem voll automatisierbaren System wie OMICRON prinzipiell nicht notwendig, da hier zwar auch ein UML-ähnliches Tokenkonzept genutzt wird, jedoch eine Aktivierung der nächsten Schritte nur abhängig von der Bedingung stattfinden kann. Existiert keine Bedingung wird parallelisiert. Existieren Bedingungen werden alle zutreffenden parallelisiert. Der Entscheidungsknoten ist hierbei ein Sonderfall der Verzweigung ohne Aktivierungsverzweigung. Bei einer parallelen Verzweigung mit Nebenläufigkeit erfolgt eine Duplikation von Daten zur Laufzeit. Bei Referenzen wird die Referenz dupliziert, so dass alle Handlungsstränge auf dieselbe Komponente zugreifen. Sonst werden die Werte kopiert und für jeden weiteren Handlungsstrang (Zweig) eine neue Komponente erzeugt.

Besitzt ein **Vereinigungsknoten** (join) für Tokens mehrere Eingänge, wird jeweils das erste in einem Takt ankommende Token weitergegeben. Ausschlaggebend hierfür ist die Reihenfolge der Eingangskanten. Sind in einem folgenden Takt wieder alle Bedingungen erfüllt, wird entweder das nächste am ersten Eingang anliegende oder (wenn hier keines mehr ist) das nächstuntergeordnete weitergegeben. Sollen überzählige Tokens vernichtet werden, muss dies explizit modelliert werden, da so Informationen verlorengehen. Prinzipiell sollte jedes bis zum Ausgang durchlaufende Datenobjekt im Ablauf nur einen definierten Weg wählen und nicht dupliziert werden, um Redundanzen zu vermeiden. Die semantisch korrekte Verarbeitung von Sammlungen wird im Anhang J erläutert.

Durch objektorientierte Eigenschaften wie Polymorphie von Teilprozessen ist eine vollständige Überprüfung auf **Deadlocks**³⁷ nicht immer möglich.

7.3 Aufrufkonzept: Externalisierung von Befehlen

Wie bereits bei den Vergleichsoperatoren gezeigt, ist häufig eine Externalisierung von Aktivitäten notwendig. Interne Datenstrukturen müssen dabei an die externe Funktion übergeben und externe Ergebnisse in das Gesamtmodell zurückgespielt werden.

Die Grundvoraussetzung für die Externalisierung von Befehlen ist eine Separierung der Befehlsausführung von der eigentlichen Modellverwaltung und -animation.

Wie in anderen Prozess-Systemen, beispielsweise dem SPADE Process Enactment Environment (PEE) [Arlow et al. 1997][Emmerich et al. 1996] werden Funktionen und Services extern implementiert, während im Gesamtmodell nur der Aufruf spezifiziert und typisiert wird. Hierzu müssen von der Implementierung eine externe Interaktions- und eine Kommunikationsplattform vergleichbar dem SPADE User Interaction Environment (UIE) und dem SPADE Communication Interface (SCI) zur Verfügung gestellt werden.

³⁷ Deadlocks sind Systemkonstellationen, die dazu führen, dass ein Prozess nicht weiterlaufen kann, weil sich beispielsweise parallele Abläufe gegenseitig behindern. Diese Situation ist unerwünscht, da keine geregelte Beendigung mehr erfolgen kann.

Allerdings werden in SPADE-1 Tools oder ein vorgeschalteter Adapter („Bridge“) für diese einfach als Integerwert angesprochen. In OMICRON ist dagegen durch die Service-Abstraktion für das Modell nicht zu unterscheiden, ob ein Tool direkt oder über einen Gateway-Service aufgerufen wird. Die Unterscheidung zwischen „black-box tools“ und „service-based tools“ in SPADE-1 entfällt also in OMICRON, da mit einer semantischen Beschreibung im Gesamtmodell alle Aufrufe Service-basiert erfolgen.

Das OMICRON Laufzeitsystem übermittelt den Aufruf jeder als **extern** markierten Aktivität an einen Dispatcher, der anhand der Klassenidentität feststellt, ob in der **Methodentabelle** eine externe Funktion für diesen Typ vorliegt. Ist dies nicht der Fall, kann per Breitensuche im Vererbungsbaum aufwärts für jede Komponente ein Interpretier gesucht werden (polymorphe Interpretation). Wird auch so keine Funktion für die Aktivität gefunden, kann das Laufzeitsystem einen Fehler auslösen, Korrekturen ermöglichen oder den Aufruf ignorieren. Auch die Funktion hat die Möglichkeit, per Rückgabewert einen Fehler auszulösen.

Erfolgt ein Funktionsaufruf, müssen die vorhandenen **Aufrufparameter** strukturiert übermittelt und die Funktion ausgeführt werden. Für die Übermittlung bietet sich das XML-Format an. Die **Rückgabeparameter** müssen im selben Format eingelesen und wieder in das Gesamtmodell und die Laufzeitumgebung zurückgespielt werden. Dies gilt für alle externen Systeme.

Die Überprüfbarkeit und Vollständigkeit der Einbindung hängt dabei maßgeblich vom Detaillierungsgrad des Modells ab, das heißt es müssen die Parameter und Aufrufkonditionen sowie die gesamte Klassifikation genau definiert werden, dass eine Übermittlung an den Service möglich ist. Ein Service ist als Black-Box realisiert, so dass die Datenstruktur im Gesamtmodell in der Granularitätsstufe definiert werden muss, die der Service erwartet und verarbeitet.

In SLANG [Arlow et al. 1997] wird zwischen White Transitions und Black Transitions unterschieden: Black Transitions haben im Gegensatz zu White Transitions eine Rückkopplung zur Folge – bei SLANG mit einem User Environment. Diese werden in SLANG mit einem „**Prolog**“ vorbereitet. Nach erfolgtem Durchlauf werden mit einem „**Epilog**“ die Ergebnisse erzeugt oder aufbereitet.

Im System wird die externe Interaktion in einen vom Modell separierten Bereich ausgelagert, so dass das Gesamtmodell von den externen Abläufen separiert und abstrahiert werden kann. Dagegen benötigt die Prozess-Engine zur Animation und Aufzugenerierung vollständigen Zugriff auf das Gesamtmodell. Standard-Prolog und -Epilog eines Aufrufs erfolgen fest programmiert im Laufzeitsystem. Die gezielte Vor- und Nachbereitung kann dagegen flexibel im Prozessmodell definiert werden.

Neben einer lokalen Interpretation durch einen internen oder betriebssystembasierten Aufruf von APIs oder Programmen ist auch eine verteilte Ausführung beispielsweise per Web-Service-Aufruf möglich. Die Methodentabelle muss auch hierfür die Aufrufkonventionen enthalten, so dass nach erfolgter Identifikation des Funktionsaufrufs in der Methodentabelle auch der Aufruftyp und die Parameterübermittlung (Marshaling, Unmarshaling) klar ist. Die Aufrufmethodik ist dabei eine rein technologische Fragestellung. Wichtig für verteilte Aufrufe ist ein eindeutiges Adressierungskonzept (wie URI), das zudem durch ein Auflösungskonzept wie den Domain Name Service (DNS) oder eine Registry wie UDDI ergänzt werden kann.

Für jeden Aktivitätstyp kann also eine spezielle Behandlungsroutine registriert sein, sonst wird eine allgemeinere aufgerufen. Die Handler müssen dabei für den Typ registriert sein. Die Mehrfachvererbung führt dabei gegebenenfalls zu einem Nichtdeterminismus, da nur klar ist, dass der speziellste verfügbare Handler aufgerufen wird. Welcher dies bei zwei auf gleicher Ebene befindlichen ist, bleibt dabei dem Laufzeitsystem überlassen beziehungsweise wird durch die Breitensuche festgelegt, die durch die Relationsreihenfolge deterministisch wird.

7.4 Generierung

Eine Generierung von Artefakten und Benutzungsschnittstellen kann grundsätzlich auf unterschiedliche Arten erfolgen. Die **direkte Interpretation** des Modells erfolgt anfragebasiert aus dem Modell. Für jede Anfrage an das System oder jede atomare Teilaktivität wird also aus dem Modell ein entsprechendes Generat, beispielsweise über einen Web- und Application-Server, auf Anfrage erzeugt.

Ein **interpretierender Interaktions-Anwendungsrumpf** kann als eigenständige Applikation Dialoge und Interaktionselemente basierend auf Beschreibungen aus dem Modell erzeugen. Der Anwendungsrumpf stellt dabei als Thin Client quasi ferngesteuert passende Interaktionselemente und Daten dar und übermittelt die eingegebenen Daten ähnlich wie ein Web-Interface.

Die **Generierung einer vollständigen Anwendung** ist in der Softwareentwicklung bei MDA oder modellbasierten Benutzungsschnittstellen-Generatoren verbreitet, die zum Entwicklungszeitpunkt Anwendungen erzeugen. Für das laufzeitbasierte Prozessmodell in OMICRON ist diese Vorgehensweise jedoch nicht anwendbar.

Von einem komplexen, impementierungslastigen und zum Teil mehrstufigen Generatorkonzept, wie dies modellbasierte User Interface Generator-Ansätze (MB-UIDE, Model-Based User Interface Development Environments, vgl. [Schlegel 2002]) und auch MDA vorsehen muss daher abgesehen werden.

Eine Verwendung von zur Laufzeit einzeln befüllten Target-Templates ist jedoch ebenso möglich. In [Dadam et al. 2005] wird diese Problematik teilweise durch Modellierung von Prozessen aus **Prozessvorlagen** umgangen, so dass passende Anwendungsfunktionen in einem Repository zur Verfügung gestellt und damit proaktiv erzeugt werden können.

Über **Target-Templates** beziehungsweise templatehaltige Transformationsbeschreibungen ist die Erzeugung von meist textbasierten Artefakten aus der Kombination eines Templates als Rahmen mit neuen Elementen möglich. In MDA-Ansätzen der Softwareentwicklung werden hierzu beispielsweise Frameworks wie openArchitecture und Templatesprachen wie XPand eingesetzt. [StahlVölter 2005]

Beispiel 18: Soll beispielsweise eine HTML-Datei erzeugt werden, müssen **Dateiendung HTM(L)** und die **innere Struktur <HTML><HEAD>[Head]</HEAD><BODY>[Body]</BODY></HTML>** vorgegeben werden. Die **dynamischen Bestandteile [Head] und [Body]** werden dann aus dem **Komponentenmodell** erzeugt.

Für die **Transformationsbeschreibung** kann beispielsweise aufbauend auf dem Konzept der XML-basierten **Element Transformation Description (ETD)** [Schlegel et al. 2004][Schlegel 2002] eine Templatesprache aufgebaut werden, die für ein Metamodell und eine Zielumgebung finale Artefakte erzeugen kann. Die Templates orientieren sich dabei an der Struktur der erzeugten Artefakte wie Textdokument, Datei, Klasse etc. ETD können auch in Komponentenform in das System integriert werden, so dass der Generierungsablauf vollständig im Gesamtmodell integriert ist. So kann auch typbasiert die Erzeugung des korrekten Artefakts durch Auswahl der passenden ETD und typsichere Synthese sichergestellt werden.

Ähnlich einer Document Type Definition (DTD, [W3C 2006]) können hier Kardinalitäten (Anzahl möglicher Vorkommen oder Wiederholungen) und Bedingungen in den Generatorkomponenten ergänzt werden. Die Prozesssprache eignet sich nach Modifikation der Laufzeitumgebung auch für die Artefaktgenerierung.

Wird die Transformationsbeschreibung derart in das OMICRON Modell integriert, entstehen **Generatorkomponenten**. Diese dienen sowohl der standardisierten als auch zur klassen- oder objektindividuellen Beschreibung der Ausgaben und Artefakte.

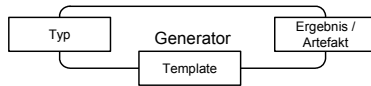


Abbildung 63: Schematischer Aufbau einer Generatorkomponente

Alle Generatorkomponenten sind Instanzen einer allgemeinen Generatorkomponente oder von deren Spezialisierungen und sind daher als solche zu erkennen. Die Basisregel erzeugt eine leere Ausgabe bei beliebiger Eingabe.

Auch hier muss immer das speziellste, noch passende Element verwendet werden. So sollte für die Texteingabe eines Vornamens sinngemäß nicht ein Element „Allgemeine Texteingabe“ sondern das Element „Texteingabe Vorname“ verwendet werden. Ausschlaggebend ist dabei nicht der Generatortyp, sondern der konsumierte Komponententyp (Eingang), da für diesen ein möglichst gutes Ergebnis erzielt werden soll. Der Haupteinfluss der Objektorientierung liegt also hier in der Identifikation einer auf den Typ passenden Generatorkomponente anhand deren Eingangstypen. Bei mehreren Eingangstypen und Kontextabhängigkeiten ist gegebenenfalls eine Gewichtung sinnvoll.

Automatische Generatorkomponenten können aus einem vollständig spezifizierten Eingabekomplex – eventuell in Verbindung mit Parametern – direkt Ausgaben erzeugen, wenn eine manuelle Eingabe, sondern Entscheidung nicht nötig ist. Können alle gewünschten, zu transformierenden Komponenten automatischen Generatorkomponenten zugeordnet werden, kann die Gesamttransformation direkt ausgeführt werden.

Manuelle Generatorkomponenten müssen dagegen in unvollständigen Szenarien eingesetzt, in denen nicht alle Informationen für eine Transformationsentscheidung vorhanden sind und/oder denen ein Entscheidungsspielraum zugrunde liegt, der eine externe Entscheidung durch Menschen notwendig werden lässt, beispielsweise eine unvollständige Mapping Syntax wie bei [Vanderhaegen et al. 2005].

Sind manuelle Generatorkomponenten beteiligt, muss auf deren Bearbeitung gewartet werden, was das Veränderungs- und Latenzproblem verstärkt und das Ad-Hoc-Generierungsverfahren beispielsweise für direkte Abfragen unbrauchbar werden lässt.

Um komplexe Sprachen wie die Interaction Markup Language (IML, [Schlegel 2002][Schlegel et al. 2004]) zu transformieren, funktioniert die elementweise Transformation nicht oder nur eingeschränkt. Daher müssen komplexere Generatorkomponenten genutzt werden, die einen Generierkontext für untergeordnete Generatorkomponenten bieten und so beispielsweise Iterationsdaten in Schleifen oder Dialoginformationen für Elemente von komplexen Dialogen anbieten.

Quellkontext, Generierkontext und Zielkontext bilden den **Kontext eines Generierungsvorgangs**. Der Zustand aller Quellkomponenten bildet dabei den **Quellkontext**. Alle generatorspezifischen Kontextinformationen wie die Anzahl der bisher erfolgten Nutzungen der Komponenten, temporäre Daten und Zähler bilden den **Generierkontext**. Den **Zielkontext** bilden Templates und bereits generierte Artefakte.

Fixierung der Schnittstellenimplementierung: Zwischen den Modellbestandteilen und den verwendeten Generator-Komponenten können statische Generate-Beziehungen erstellt werden, so dass bei jedem Aufruf des Prozessschritts direkt die gewünschte Schnittstellenimplementierung gewählt wird. Dies entschärft Mehrdeutigkeiten und erhöht sowohl die Geschwindigkeit als auch die Ablaufsicherheit. Modelländerungen können dann allerdings zu Inkonsistenzen führen, weil das Modell nicht mehr frei interpretiert werden kann. Bei Änderungen müssen also diese Generatorkanten mit überprüft werden, da nur Typinkonsistenzen direkt auffallen.

Mit einer **Target-Klassifikation** könnte dann eine Generierung oder Umschaltung zwischen generierten Komponenten passend zur Zielumgebung erfolgen. Für jede Plattform wie Maschine, Terminal und mobiles Gerät könnte so eine passende Generierung erfolgen, die nur verfügbar ist, wenn eine passende Target-Klassifikation als Eingang der Generatorkomponente existiert.

Eine Generierung von veränderbaren (meist plattformspezifischen) Zwischenmodellen (PSM) wie in MDA und MDSD hat zwar den Vorteil eines größeren Freiheitsgrades für Prozessentwickler, jedoch auch gravierende Nachteile im Vergleich zur konsistenzsicheren direkten Generierung. In der Praxis überwiegen daher die Vorteile der direkten Generierung. (vgl. [StahlVölter 2005], S. 27).

Die Laufzeitgenerierung schließt im Gegensatz zur entkoppelten Generierung zur Design-Zeit die Veränderung von Zwischenstufen während der endgültigen Generierung aus und legt daher ohnehin eine direkte Generierung nahe. Eine Adaption von Zwischenmodellen wäre nur präskriptiv möglich, um das Generiererergebnis zu beeinflussen. Prinzipiell ist es möglich, unsichtbare Zwischenmodelle zu generieren, solange keine zeitkritischen Anforderungen existieren. Allerdings ist die Laufzeitgenerierung kaum mit der Generierung vollständiger Zwischenmodelle vereinbar, die meist nur in ihrer Gesamtheit die Ausgangsbasis für nachgeordnete Modelle bilden können und Zwischenmodifikationen bei einer Laufzeitgenerierung ohnehin nicht möglich sind.

Artefakt-Regenerierung: Werden generierte Artefakte innerhalb des Gesamtmodells gehalten – das heißt basierend auf demselben Basismodell und verbunden durch Generierungskanten – können Auswirkungen von Änderungen entweder bei direkter Transformation über einfache Transformationsobjekte automatisiert angepasst werden oder bei komplexen Transformationsobjekten semiautomatisch (das heißt teilinteraktiv) regeneriert werden, da ein Generatorsystem die existierenden Relationen im Modell verfolgen kann.

Finale generierte Artefakte haben den „Vorteil“, dass sie keine Seiteneffekte haben oder nach ihrer Erzeugung weitere Bedingungen an das Modell stellen. Treten Änderungen auf, müssen die Artefakte erneut generiert oder geändert werden.

Textbasierten Artefakten kommt ein hoher Stellenwert sowohl in automatisierten als auch in nutzerzentrierten Prozessen zu. Mit den auf SGML [DIN 1991] basierenden, strukturierten Ansätzen haben die textbasierten Artefakte, beispielsweise XML- oder HTML-Dokumente, an Bedeutung gewonnen. Für den Menschen sind maschinell erzeugte, strukturierte Texte eine lesbare und exakte Repräsentation, die beispielsweise über Sprachsynthese (Text-to-speech) in andere Modalitäten exakt umgesetzt werden kann.

7.4.1 Interaktionsgenerierung

Gerade bei flexiblen Prozessen treten häufig Situationen auf, die nicht automatisiert behandelt werden können. Meist sind entweder mehrere Werte beziehungsweise Optionen vorhanden, aus denen einer selektiert werden muss oder es fehlen noch Informationen zur Ausführung eines aktiven Prozesses – beispielsweise, welches der drei im Zulauf befindlichen Teile verwendet werden soll oder welche Priorität hat der Auftrag (unbelegter Eingang „Priorität“).

Um diese Fälle zu behandeln, verfügt OMICRON über ein **implizites Interaktionskonzept**. Sind Informationen mehrdeutig oder fehlen Informationen zur Ausführung, kann aufgrund der Typisierung aller benötigten Informationen und ausgeführten Aktivitäten für jede Information eine Interaktion generiert werden. Für verwendete Eingangstypen muss daher jeweils ein Interaktionselement definiert sein – bei komplexen Typen jeweils für die atomaren und gegebenenfalls Ergänzungen für die Gesamtheit des komplexen Typs. Dabei werden komplexe Typen nach Möglichkeit integriert abgefragt, während atomare Typen optisch oder sequenziell getrennt abgefragt werden.

Eine Methodentabelle zur Externalisierung von Aktivitäten (vgl. 7.3 Aufrufkonzept: Externalisierung von Befehlen, S. 131) muss hierzu auch für Datentypen Einträge enthalten. Auch für die Polymorphie bei Datenfeldern gelten die dort beschriebenen Regeln zur polymorphen Verwendung spezialisierter Komponenten. Die Erzeugung der Interaktion kann dabei jedoch auch durch interne Interpretation mit Modellzugriff gelöst werden.

Für ein **explizites Interaktionskonzept** ist das OMICRON Modell ebenfalls ausgerüstet. Interaktive Aktivitäten sind eine Spezialisierung der allgemeinen Aktivität und können so vom Laufzeitsystem erkannt werden. Auch hier erfolgt der Aufruf über die Methodentabelle.

Zur Strukturierung werden differenzierte Aggregationsebenen verwendet. Dabei unterstützt OMICRON auch Konzepte wie Use-Cases, InteractionCases [Schlegel et al. 2004] und semantischen Gruppen [Schlegel 2002]. Auch eine rollenbasierte Generierung kann durch den Zugriff auf das Gesamtmodell ermöglicht werden.

Die IML [Schlegel et al. 2004] unterscheidet vier Interaktionskonzepte, die in OMICRON ebenfalls anwendbar sind: ENTER für fehlende Informationen, EDIT für mögliche Veränderungen, SELECT für mehrdeutige Informationen und WRITE für reine Ausgaben beziehungsweise Visualisierungen (vgl. Anhang N).

Zur generativen beziehungsweise transformativen Verarbeitung komplexer Interaktionsmodelle (vgl. [Schlegel 2002][Schlegel et al. 2004]) muss das Generatorsystem sowohl erweitert als auch das Modell für den Zugriff durch den Generator geöffnet werden, da der jeweilige Modellkontext mit in die Generierung einbezogen werden muss. Das OMICRON System stellt die Modell- und Generatorkonzepte zur Verfügung, die Generierung selbst kann in Anlehnung an [Schlegel 2002] erfolgen, ist jedoch nicht Gegenstand dieser Arbeit.

Wird für jede benötigte Information eine separate Interaktion erzeugt (**Simple Service User Interfaces**, [BeinhauerSchlegel 2005a]), so stehen in ausreichendem Maße Kontextinformationen zur Verfügung und eine Überschneidung oder Abhängigkeiten sind unwahrscheinlich.

Häufig ist es jedoch wünschenswert, zusammengehörige Informationen in einer komplexeren Interaktion abzufragen (**Complex Service User Interfaces**, [BeinhauerSchlegel 2005a]), wodurch Abhängigkeiten entstehen durch die das System weit komplexer wird. Auf Seiten des Modellierungssystems ist dabei die Festlegung der Gesamtheit der für die Generierung bestimmten Elemente wichtig. Hierbei sind nicht nur die in diesem Takt aktivierten zu berücksichtigen, sondern gegebenenfalls weitere, die nicht durch manipulative (Wertveränderungen etc. im Gegensatz zu beispielsweise Vergleichen) oder interaktive Prozesskomponenten vom derzeitigen Prozessfokus getrennt sind.

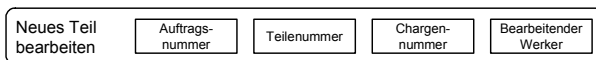


Abbildung 64: Beispiel für einzubeziehende Komponenten bei der Erzeugung einer komplexen Interaktion

Findet zudem noch eine Integration verteilt laufender Prozessinstanzen (**Multi-Service User Interfaces**, [BeinhauerSchlegel 2005a]) statt oder werden sogar alle Interaktionen in eine verteilte Service-Anwendung integriert (**Service Application User interfaces**, [BeinhauerSchlegel 2005b]) steigt die Komplexität weiter an.

Durch die direkte Erzeugung des Dialogs aus dem Modell, ist neben der Dialoggenerierung auch eine **Dialogmitnahme** des Benutzers von einem Peer zum anderen möglich, so dass Prozessinstanzen samt Dialog von einem Peer zum anderen nutzerbasiert wechseln können.

7.4.2 Rollen

Rollen bilden die Basis für eine angepasste, kontextgerechte Prozesssteuerung oder Generierung. Je nach Rolle werden andere Modellbestandteile gezeigt und zugreifbar. Dies dient sowohl bei dynamischer Erzeugung als Parameter für die Anpassung als auch bei Generierung als Steuerungselement für die Erzeugung von der Interaktionslogik. Auch Rechte können auf Basis von Rollen vererbt werden.

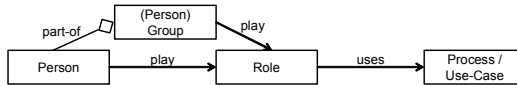


Abbildung 65: Rollenabbildung in Anlehnung an [Caetano et al. 2005]

[ListKorherr 2006] definieren Rollen als Spezialisierung von Menschen, die einer Organisationseinheit angehören und Rollen ebenso wie Software als Spezialisierungen von Prozessteilnehmer. In dieser Arbeit wird eine Rolle dagegen wie in einem Use-Case-Diagramm als Prozessteilnehmer definiert, so dass bei entsprechender Definition und Rechten auch ein System eine Rolle übernehmen kann. Daher entspricht die Rolle in OMICRON eher dem Prozessteilnehmer in [ListKorherr 2006].

Rollen werden auf Prozessebene im Modell definiert. Durch Vererbung erfolgt dabei eine Spezialisierung und Separierung unterschiedlicher Rollen, Durch die Mehrfachvererbung ist es dabei auch möglich, **hybride Rollen** durch Fusion zu bilden, die hinsichtlich Rechten und Aufgaben mehrere Funktionen erfüllen können.

Grundsätzlich kann ein Prozess auch ohne die Beteiligung von Rollen modelliert werden. Allerdings kann dann auch kein dediziertes Rechtekonzept realisiert werden und keine (rollenbasierte) Zuordnung zu Personen im Gesamtmodell sowie eine Anpassung von Sichten erfolgen.

Im Prozess werden die Rollen mit der aus dem UML Use-Case-Diagramm bekannten Use-Beziehung an die zugehörigen Schritte gebunden. Für die Darstellung der Rollen in einem Prozess eignet sich die Swimlane-Darstellung aus BPMN und UML (siehe Anhang G).

Wird ein Prozess instanziiert, dem Rollen beziehungsweise Aktoren zugewiesen sind, so muss vor der Ausführung eines mit einer Rolle assoziierten Prozessschritts die Rolle instanziiert beziehungsweise eine Instanz zugewiesen werden. Diese wird von der Laufzeitumgebung angefordert – beispielsweise durch einen Login. Daher ist eine Modellierung wie im Use-Case-Diagramm am Besten geeignet: Jede Rolle wird in den Prozess nur einmal eingeführt, um mehrfache Zuordnungen zu vermeiden. Für Teilprozesse gilt die Rolle dann rekursiv.

Beispiel 19: Das Werkstück A wird an Arbeitsstation C1 zur Endkontrolle angeliefert. Herr Mayer ist als Kontrolleur an der Koordinatenmessmaschine tätig. Das System weist nun der Rolle „Kontrolleur“ für den Prozessschritt „Endkontrolle“ Herrn Mayer zu (als eine Instanz der Rolle „Kontrolleur“). Dies verhindert, dass die Aufgabe (das heißt ein manueller Prozessschritt) einem Repräsentanten einer hierfür nicht qualifizierten Rolle (beispielsweise „Werker“) zugewiesen wird. Zugleich kann das System auch feststellen, ob qualifiziertes Personal und Maschinen für diesen Prozessschritt zur Verfügung stehen.

Unterschiedliche Rollen benötigen oft komplementäre Sichten und Daten, so dass sowohl die gezielte Interaktionsgenerierung als auch das Berechtigungskonzept auf ein Rollenmodell zugreifen sollte.

Die Verwendung unterschiedlicher Rollen und die mehrfache Verwendung von Rollen oder mehreren Instanzen desselben Rollentyps ermöglichen den Einsatz verschiedener Mitarbeiter und Systeme in derselben Prozessinstanz.

7.5 Ressourcen: Einführung einer Ontologie

Für die Abbildung eines Produktionssystems in OMICRON müssen dessen Bestandteile in das Modell eingefügt werden. Da einzelne Maschinen, Teile und Werkzeuge auf der Instanzebene modelliert werden, müssen zunächst die Konzepte eines Produktionssystems in der Art einer Domänenontologie in das Gesamtmodell eingeführt werden.

Die grundlegenden Bearbeitungsverfahren sind bereits in der Klassifikation der sechs Hauptgruppen von Fertigungsverfahren nach DIN 8580 [DIN 2003] beschrieben und in [WestkämperWarnecke 2006] ausgeführt. Um auch die (Teil-)Produkte eines Produktionssystems zu klassifizieren, wurden weitere Kataloge, Ontologien und Austauschformate wie Proficl@ss⁵⁰, eCl@ss⁵⁰, oder BMEcat³⁸ [IDW 2005] entwickelt, die unter anderem für den Mittelstand eine umfangreichere Klassifikation für Produkte und Dienstleistungen anbieten. Technisch ist auch die direkte Übernahme solcher Klassifikationen in das OMICRON Typsystem möglich. Allerdings ergeben sich bei der Verschmelzung unterschiedlicher Ontologien häufig komplexe Problemstellungen und Inkonsistenzen (vgl. [Gomez-Perez et al. 2004]). Das OMICRON System sollte daher zunächst mit einer einfachen High-Level-Klassifikation initialisiert werden, die um spezifische Elemente ergänzt werden kann:

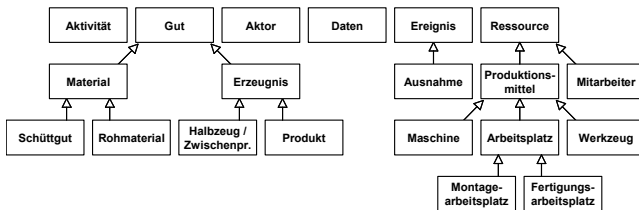


Abbildung 66: Beispielhafte, klassifizierende Ontologie für ein Produktionssystem

Durch Aggregation werden im Ablauf ständig neue, komplexe Typen erzeugt. Diese Sammlungen von Bauelementen und -gruppen zeigen häufig auch ein Standardisierungspotenzial in der Produktion auf: Häufig werden in unterschiedlichen Baureihen oder Produktionslinien dieselben oder sehr ähnliche Elemente und Stücklisten verwendet. Rationalisierungspotenzial besteht dabei vor allem in einer Auslagerung von Zwischenerzeugnissen, die in hohen Stückzahlen benötigt werden. Diese können separat gefertigt und – gegebenenfalls vormontiert – zur Weiterverarbeitung in den Prozess eingebracht werden. Eine weitere Flexibilisierung und Effizienzsteigerung der Abläufe ist die Folge. Zudem können Konzepte des Wissensmanagements kontext- und prozessorientiert in das Gesamtmodell integriert werden [SchlegelKarapidis 2004].

³⁸ <http://www.proficlass.de>, <http://www.eclass.de>, <http://www.bmecat.org>

8 Laufzeitmodellierung und Modellausführung zur Laufzeit

Die Besonderheit von OMICRON ist die vollständige Laufzeitorientierung des Modells. Hieraus ergeben sich neben der Notwendigkeit eines Laufzeitsystems weitere Besonderheiten, die im Folgenden aufgezeigt werden.

8.1 Laufzeitinstanziierung

Prozesse können instanziiert werden: Die Instanz wird dann als Projekt oder Prozessinstanz bezeichnet und ist mit einem konkreten Vorgang in der Fertigung verknüpft.

Für jeden zu durchlaufenden Prozess wird eine Instanz angelegt. Dieses „Projekt“ kann je nach Modellierung die Produktion eines einzigen Produkts oder die Produktion einer ganzen Serie beinhalten, da das Modell prozessorientiert verwaltet wird. Werden beispielsweise in einem Zyklus mehrere Kugeln für ein Kugellager gefertigt, werden sovielen Instanzen der Kugelproduktion angelegt wie für das Kugellager benötigt beziehungsweise (im Falle von Ausschuss mehr-) gefertigt werden.

Für die Erzeugung eines neuen Projekts in einer Produktion kann ein neuer Auftrag ebenso Auslöser sein, wie ein entdecktes Problem, das eine Überprüfung oder einen Wartungsprozess in Gang setzt.

Die Instanziierung eines Prozesses kann sowohl automatisch durch eine Aktivität als auch durch ein **externes Ereignis** oder **manuell** erfolgen. Prinzipiell ist es möglich, Projekte anzulegen, die noch nicht aktiv sind. Meist ist jedoch eine Instanziierung mit einer Aktivierung verbunden. Sollen bestimmte Ereignisse auf eine definierte Weise durch neue Prozessinstanzen abgefangen werden, müssen die Instanzen allerdings ohne Aktivierung erzeugt werden, damit sie auf das Aktivierungssignal warten können, das sie in Gang setzt.

Im Fall einer Verzweigung können wie beschrieben auch parallele Kopien zur Nutzung auf Instanzebene erstellt werden. Dies ist nötig, um Teilprozesse im verteilten System versenden und verteilt bearbeiten zu können. Die Prozessinstanz kann so sukzessive und konform zum Prozessmodell mit Inhalten gefüllt werden. Die Instanz ist also immer nur eine Momentaufnahme, die sich während des Ablaufs im Gegensatz zum Prozess(schema) fortwährend ändert.

In OMICRON wird das Modellsystem in Echtzeit aktualisiert, da sonst kein JIT möglich wäre. Daher können Überlappungen zwischen Projekt und Prozessänderung (Ad-hoc- und Schema-Änderung, [Rinderle et al. 2004]) nur kurzzeitig durch parallele, verteilte Änderung auftreten.

Da Aktivitäten in OMICRON hierarchisch aufgebaut sind, können Teilprozesse losgelöst vom übergeordneten Prozess bearbeitet werden und müssen diesem nur spätestens bei Beendigung des Teilprozesses die Ergebnisse und das Aktivierungssignal wieder zur Verfügung stellen.

8.2 Laufzeitveränderung

Änderungen – meist Anpassungen – von Prozessen kommen ebenfalls zur Laufzeit vor. Meist handelt es sich um Sonderfälle oder weitere Artefakte und Variablen, die in einen existierenden Prozess eingefügt werden müssen oder durch Spezialisierung hinzukommen, seltener um zusätzliche Schritte, die beispielsweise durch neue oder veränderte Verfahren oder rechtliche Bestimmungen hinzukommen.

Flexible und veränderliche Prozesse erfordern Schnittstellen, die sich dynamisch an die Rahmenbedingungen anpassen und auch durch Menschen explizit und direkt angepasst werden können. Hierbei ist sowohl die Anpassung während der Entwicklung als auch die Anpassung im produktiven Betrieb zu berücksichtigen. Während für die Anpassung in der Entwicklung bereits dedizierte, jedoch häufig kaum integrierte und zudem laufzeituntaugliche Modelle geschaffen

wurden (vgl. beispielsweise MDA / MDS, [StahlVölter 2005]), ist dies zur Laufzeit bei existierendem Schema bisher nicht möglich.

Wird ein Prozess A verändert, ist die Erstellung einer parallel in die Vererbungshierarchie einzufügenden Kopie die beste Lösung, wenn alle Spezialisierungen von A im ursprünglichen Zustand erhalten bleiben sollen (Type Duplicate). Sollen alle Subklassen mit verändert werden (Type Cascade), müssen die Änderungen ähnlich wie bei relationalen Datenbanken über die gesamte untergeordnete Objekthierarchie kaskadiert werden. Dann muss für existierende Instanzen nach erfolgter Veränderung gegebenenfalls eine Migration auf den neuen Zustand stattfinden.

Im Folgenden wird von folgenden Prämissen ausgegangen: Zur **Vermeidung von Redundanzen** sollen so wenige Kopien wie möglich angelegt werden, das heißt möglichst wenige Relationen und Komponenten dupliziert werden und daher auch die **Typhierarchie möglichst kompakt** gehalten werden: Zusätzliche Prozesse sollen nur bei gewünschter Koexistenz eingeführt werden.

Die Bildung von Varianten sollte daher nur genutzt werden, wenn tatsächlich vom Nutzer wahrzunehmende, inhaltliche Varianten existieren sollen. Eine bloße Veränderungsverfolgung durch Variantenbildung würde die Modellkomplexität unnötig erhöhen. Wichtig ist daher, dass reine **Änderungen** soweit möglich als solche vorgenommen werden und für parallel existierende **Varianten, Spezialisierungen** oder (den Vorgänger möglicherweise ablösende) neue **Versionen** die **Vererbung** eingesetzt wird.

Überlegungen und Experimente, die dazu dienen sollten, Versionierung und Variantenbildung durch verlinkte Kopien und ähnliche Konstrukte zur ermöglichen, führten immer zu **Kollisionen**, da der über diverse Relationen angebundene Kontext für unterschiedliche Versionen oder Varianten teilweise nicht dupliziert werden kann. Wird nicht die Vererbung für Versionierung und Variantenbildung eingesetzt, existieren Kopien, die ein semantisch unklares Verhältnis zum Original und dessen Kontext haben und gegebenenfalls sogar einen eigenen, abweichenden Kontext besitzen. Deshalb wird in OMICRON nur der Ansatz einer Variantenbildung und Versionierung durch Vererbung oder die direkte Änderung vorgesehen (vgl. Kapitel 5.3.3).

In der Laufzeitumgebung können für die Versionierung von Prozessen weitere Konzepte wie ein **Migrationspfad** und eine **zeitliche Übergangsplanung** implementiert werden, so dass basierend auf dem Gesamtmodell leicht ein Versionsmanagement eingeführt werden kann.

Wenn keine direkten oder transitiven Instanzen vorhanden sind, ist eine Änderung meist direkt in der Aktivitäts-Komponente möglich. Seiteneffekte von Änderungen sind nur zu erwarten, wenn diese direkt Auswirkung über Relationen zu anderen Komponenten haben. Durch den Einsatz der Vollreferenzierung (siehe Kapitel 5.3) können Auswirkungen von Änderungen direkt an den Teilkomponenten im Modellgraph nachvollzogen werden.

Algorithmisch können fast alle Auswirkungen anhand von (transitiver) Verbindung, Typ und Kontext abklärt werden. Bei Prozessen in der Produktion liefert die Praxis teilweise die Notwendigkeit, Teilschritte zu entfernen, beispielsweise wegen neuer Produktvarianten oder Prozessschritte („Leiste unlackiert“, Lackierung weglassen). Das Weglassen ist in der objektorientierten Spezialisierung schwer wiederzugeben und kann meist besser durch eine Generalisierung gelöst werden, so dass der Prozess mit fehlenden Schritten in eine übergeordnete Stufe übernommen wird.

Neben den bereits besprochenen modellinhärenten Änderungsnotwendigkeiten müssen in der Praxis auch die Änderungsrechte beachtet werden, da bei häufigen Änderungen am Prozess Probleme in anderen, ebenfalls dem Prozess folgenden Projekten zu befürchten sind. Die Änderung oder zumindest Zustimmungspflicht zu Änderungen von Prozessen muss deshalb rollenbasiert beispielsweise einem Prozessverantwortlichen zugeordnet werden, der die Änderungen auf Prozess-/Klassenebene durchführen kann. Sonst wären Änderungen mit

Auswirkungen auf alle gleichartigen Prozessinstanzen direkt durch einen Projektleiter oder einen Projektmitarbeiter möglich. Eine fahrlässig erhöhte Änderungsfrequenz ist trotz Unterstützung durch OMICRON weder organisatorisch noch aus Effizienzgründen sinnvoll.

Manche Ansätze ermöglichen eine Prozessänderung nach oder durch Instanzänderung („Prozess-Schemaevolution“, [Dadam et al. 2005]). Hierbei handelt es sich jedoch um objektorientiert inkonsistente Änderungen, die OMICRON nicht erlaubt. Mögliche Veränderungsoperationen bei derartigen inkonsistenten Änderungen, die ein temporäres Abweichen von Instanz und Prozess erlauben werden in [Rinderle 2004] beschrieben.

8.3 Metamodell-Konsistenz: Überprüfung von Änderungen

Für eine in einen Prozess eingefügte Komponente gilt – sofern nicht explizit eine Instanzierungsbeziehung beziehungsweise ein Instanzvater vorgegeben wird: Es wird zunächst die speziellste übergeordnete Komponente im Metamodell ausgewählt, die am engsten mit dem einzufügenden Komponententyp verwandt oder sogar äquivalent ist.

Jede **Komponente** wird zum Erzeugungszeitpunkt mit einer Komponente ihres Metamodells verbunden – als Instanz oder Kontext. Somit kann für Komponente festgestellt werden, welche Relationen möglich sind. Soll nun eine **Relation** hinzugefügt werden, muss diese einem im Metamodell vorhandenen Relationstyp entsprechen oder eine Spezialisierung dieses Typs sein. Es muss also eine Kontext- beziehungsweise Instanzbeziehung zwischen dem **Metamodell-Tripel (Komponente, Relation, Komponente)** im Metamodell und dem neuen Tripel im Modell bestehen.

Trifft für eine neu hinzuzufügende Komponente oder Relation keine Regel im Metamodell zu, muss diese abgelehnt werden. Werden zu einem späteren Zeitpunkt Änderungen vorgenommen oder Modellelemente hinzugefügt, muss eine erneute Prüfung vorgenommen werden.

8.4 Veränderungen innerhalb der Hierarchie: Eine Fallunterscheidung

Objektorientierte Modelle können – wie dies in der objektorientierten Programmierung (OOP) genutzt wird – während der Entwicklung weitgehend problemlos verändert.

Zur Entwicklungszeit bestehen bei der OOP nur Klassen und keinerlei Instanzen. Zudem ist sichergestellt, dass auch keine Daten aus einem laufenden System existieren. Falls doch, wird im Regelfall eine Migration durchgeführt, um Laufzeitdaten der neuen Struktur anzupassen.

In einer Laufzeitumgebung ist diese einfache Vorgehensweise nicht mehr möglich. Ansätze für objektorientierte Veränderungen zur Laufzeit werden unter anderem für Betriebssysteme beschrieben: CHEOPS [Schubert 1996] arbeitet wie Smalltalk [Mittendorfer 1997] und andere objektorientierte Programminterpretierer auf der Basis von Klassenobjekten, die durch neues Laden oder Verändern auch den statischen Aufbau der zugehörigen Objekte ändern können. Allerdings ist dies nur möglich, wenn alle Verbindungen vorher getrennt werden, was im Rahmen der Problemstellung für laufende Prozesse nicht möglich ist.

Es kann der Versuch unternommen werden,

- nur Instanzen zu ändern. Problem: Eine Ad-hoc Änderung [Dadam et al. 2005], führt zur Inkonsistenz von Instanz und Prozess
- den Prozess zu ändern (Schema-Evolution) [Dadam et al. 2005]. Problem: Angleichung von Prozessinstanzen komplex und nicht immer durchführbar.

[Rinderle 2004] differenziert Prozessinstanzen in „unbiased“ und „biased“. „Biased“ steht hierbei für geänderte Instanzen, die gleichzeitig auf Prozessebene Veränderungen erfahren haben. In OMICRON wird dagegen die Veränderungen von Instanzen ausgeschlossen, da das Modell nicht

„bottom-up“ von der Instanz zum Prozess, sondern objektorientiert „top-down“ die Instanzbildung vornimmt und so die Klasse das bestimmende Element ist.

Die **Ableitung einer neuen Klasse** kann genauso wie zur Entwurfszeit gehandhabt werden, da zur abgeleiteten Klasse keine Instanzen existieren. Existierende Informationen der übergeordneten Klasse werden an die neue Klasse vererbt.

Bei Veränderungen zur Laufzeit ist eine Fallunterscheidung zu treffen, von der abhängt, welche Änderungen zulässig sind und wie diese im Modell durchgeführt werden können. Die Veränderung einer existierenden Klasse muss also fallabhängig erfolgen. Referenzen werden dabei vollständig überprüft. Alle Verbindungen wie Instantiation, SubInheritance etc. aller Referenzkomponenten sowie der Ausgangskomponente zusammen müssen dabei den folgenden Fallprüfungen unterzogen werden. Wird die Komponente $k \in K$ verändert, müssen $\forall m : (m, k, t) \in R, t \in (T_{ref} \cup V)$ die folgenden Überprüfungen durchgeführt werden. Gleiches gilt für die k (auch transitiv) aggregierenden Komponenten:

- **Ohne existierende Spezialisierungen, Instanzen und Verwendungen:** Existieren keine Spezialisierungen, Instanzen und/oder Verwendungen, müssen keine direkten Seiteneffekte betrachtet werden.
- **Mit existierenden Spezialisierungen:** Existieren bereits Spezialisierungen, müssen diese nach den genannten Kriterien (Instanzen, Verwendungen, Spezialisierungen) überprüft werden, da Änderungen der Vaterklasse auf die Kind-Klasse durchschlagen.

Wichtig ist, dass zu einer gesicherten Überprüfung und Veränderung die Unterklassen durch Sperren vor parallelen Veränderungen geschützt werden, die zu Modellinkonsistenzen führen könnten. Dies kann nur über ein temporäres, vererbendes Sperren³⁹ geschehen, bei dem der gesamte Teilbaum gesperrt wird.

- **Mit existierenden Instanzen**

- Die zu verändernde Teil-Komponente wurde bereits instanziiert und mit einem Wert belegt. Mögliche Vorgehensweisen:
 - automatische Rückabwicklung oder Neubelegung der Folgeschritte und -werte
 - Manuelle Neubelegung des Werts
 - Ableitung einer neuen Komponenten beziehungsweise Versionierung
 - Verbot der Veränderung
- Die zu verändernde Teil-Komponente wurde noch nicht beziehungsweise nur provisorisch instanziiert, das heißt es fand keine Ausführung und keine oder nur initiale Wertebelegung statt: Die Änderungen kann durchgeführt werden. Nutzer müssen gegebenenfalls benachrichtigt werden, dass nun eine geänderte Teil-Aktivität genutzt wird. Auch das im folgenden Kapitel beschriebene Hinzufügen von Schritten direkt vor deren Ausführung (just in time, Laufzeitentwicklung) gehört dazu.

³⁹ Bei entsprechenden Veränderungen sollen häufig nicht nur die betroffene, sondern alle Unterklassen, möglicherweise sogar alle im Aggregat enthaltenen Komponenten, gesperrt werden (siehe oben). Um in Echtzeit eine Sperrung durchführen zu können, ist ein vererbendes Locking möglich: Hierbei überprüft eine Unterklasse die Oberklassen auf deren Locking-Status, der so nicht einzeln eingetragen werden muss, sondern nur aus der übergeordneten Hierarchie abgefragt wird. Ein ähnliches Verfahren ist auch für Aggregate und andere baum- oder DAG-förmige Relationstypen beziehungsweise Hierarchien (azyklische Teilgraphen eines bestimmten Relationstyps) denkbar.

- Die zugehörige Prozessinstanz wurde bereits wieder beendet: Existieren Ergebnisse (Artefakte), sind diese dann ein Problem, wenn die Erstellung nachvollzogen werden muss (beispielsweise wegen der Prozessdokumentation) oder der Typ des Artefakts oder seiner Bestandteile sich ändert. Sonst ist eine Änderung unproblematisch.
- Mit **existierenden Instanzen von Spezialisierungen**: Für jede Spezialisierung: Projektion der Veränderungen auf die Spezialisierung und dieselbe Überprüfung für jede Instanz der Spezialisierung wie bei „mit existierenden Instanzen“. Für jede Spezialisierung muss also geprüft werden, ob für jede ihrer Instanzen die in die Spezialisierung eingebrachten Veränderungen durchführbar sind.
- Alle Fälle enthalten die Überprüfung von **existierenden Verwendungen** (horizontale Verbindungen wie Input, Output)
 - Die Verwendung nicht betroffener Schnittstellen verursacht keine Seiteneffekte, daher kann die Änderung durchgeführt werden.
 - Bei der Verwendung betroffener Schnittstellen muss geklärt werden, ob die Verwendung auch weiterhin erfolgen kann. Gegebenenfalls könnte dann eine Anpassung der verwendenden Komponente erfolgen, was eine Kaskade von Änderungen wie die oben beschriebenen für diese Komponente nach sich ziehen kann.
 - Innerhalb einer Instanz kann ein bisher nicht verwendetes Ergebnis prinzipiell sogar noch rückwirkend geändert oder gelöscht werden, da kein kausaler Zusammenhang zum jetzigen Zustand besteht.

Tritt auch nur bei einer einzigen Teilüberprüfung ein Problem auf, kann die Änderung vor Lösung des Problems (beispielsweise Beendigung der behindernden Instanz) nicht durchgeführt werden, da der Systemzustand konsistent bleiben muss. Zu beachten ist immer, ob die Änderung auf eine der existierenden Komponenten (meist auf einer – auch transitiven – Instanzebene) eine **kausale Wirkung** hat.

Häufig ist die polymorphe Verwendung von Teilprozessen eine adäquate Lösung für eine temporäre Änderung eines Teilprozesses. Die Teilprozess-Spezialisierung ist ohne weiteres möglich, die Verwendung ebenfalls, ohne dass der Gesamtprozess geändert werden muss.

Historyfunktionen, in Form der **Bewahrung von abgelaufenen Prozessen** wurden ursprünglich auch in die Überlegungen mit einbezogen. Allerdings existiert ein Spannungsfeld (Trade-off) zwischen Archivierung und flexibler Adaption von Prozessen. Abgelaufene Instanzen sind nur zu Änderungen kompatibel, wenn diese Erweiterungen nur fakultative Abläufe betreffen, die keine Wirkung auf die abgelaufenen Instanzen hatten. Jegliche Änderung ist nur noch durch Spezialisierung möglich, wenn die abgelaufenen Instanzen weiterhin über ihr aktuelles Typmodell verfügen müssen.

Zudem wächst das Modell dann sehr schnell an. Das vorgestellte Modell erlaubt daher nur die Erzeugung eines Snapshots bei Ablauf, der den Zustand des Metamodells mit abbildet. Abgelaufene Instanzen werden konform zum Konzept eines Laufzeitmodells aus dem Modell entfernt, um wieder Prozessanpassungen zu ermöglichen. Würden einmal genutzt Prozesse allerdings nur noch durch Spezialisierung und Variantenbildung verändert, könnten abgelaufene Instanzen zur Archivierung und Qualitätssicherung verfügbar gehalten werden.

Allgemein sind die Gefahren von Änderungen in der (Vererbungs-/Instanzierungs-)Hierarchie nicht zu unterschätzen. Gerade wenn Instanzen bereits abgelaufen und archiviert sind, können Änderungen auf Typebene Inkonsistenzen mit den Artefakten dieser ehemaligen Instanzen erzeugen. Auch können erst während der Durchführung von Änderungen, also der Angleichung

von Spezialisierungen und Instanzen, Probleme auftreten, die eine Transaktionssicherheit mit der Möglichkeit der Rückabwicklung erfordern würden. Für den laufenden Betrieb erscheint die Variantenbildung und Spezialisierung von geänderten Abläufen daher wesentlich zweckdienlicher. Das Konzept der Just in Time-Modellierung trägt diesem Umstand durch eine einfache Erzeugung von Abläufen und Varianten Rechnung.

8.5 Laufzeitmodellierung Just in time: Rapid Prototyping für Prozesse

Im Supply Chain Management (SCM) existiert zwar eine fertigungssynchrone oder just-in-time (JIT) Anlieferung [Schweitzer 1994], in der Produktion die „Just-in-time Production“ (JIT Production [Majima 1995]), jedoch wird in der Fertigungstechnik eine fertigungssynchrone Prozessplanung und -modellierung kaum diskutiert. Während existierende Varianten-Prozesse teilweise als Katalog abgerufen werden können, wird die zeitnahe Erstellung neuer Prozesse im Kontext von Aufträgen bisher nicht von geeigneten Hilfsmitteln unterstützt.

Gerade bei interaktiven und komplexen Prozessen, die in der Produktion eng mit den Gegebenheiten der „Realwelt“ verzahnt sind, ist es jedoch wichtig, Benutzer und andere Stakeholder bereits in die Modellierung mit einzubeziehen und die Abläufe anhand der tatsächlichen Schritte und nicht nur aus der Vorstellung oder abstrakten Modellbildung nachzuvollziehen. Gerade im Hinblick auf eine schnelle Rekonfiguration der Fabrik (Rapid Reconfiguration of the Factory, [I*PROMS 2006b]) für neue Produkte und Varianten kommt diesem Aspekt eine hohe Bedeutung zu.

Die Entwicklung von Prozessen begleitet teilweise bereits die erste Ausführung. Auch aus Zeit- und Aufwandsgründen sowie fehlendem Kontextwissen kann häufig kein Prozessdesign vor dem eigentlichen Ablauf finalisiert werden. Daher ist es notwendig Prozesse zumindest partiell erst während der erstmaligen Ausführung schrittweise zu definieren.

Objektorientierte Systeme bieten diese Möglichkeit nach erfolgter Instanziierung gewöhnlich nicht, da dies nur möglich ist, wenn der Erzeugungsvorgang des Prozesses auf Klassenebene die Existenz einer aktiven Instanz nicht ausschließt. Existierende Ansätze passen beispielsweise BPEL-Workflows nach Prozessänderungen (Instanz, Schema) an, jedoch ohne Metamodell-Compliance [ReichertRinderle 2006]: „*Adaptations of single process instances become necessary when exceptional situations occur or the structure of a process dynamically evolves.*“ [Rinderle 2004].

So wird nur die Instanz zu einer beliebigen Zeit verändert – ohne Auswirkungen auf den Prozess, jedoch mit Inkonsistenzen zwischen Prozess- und Instanzebene. Die Wirkung in OMICRON erfolgt jedoch auch bei der Laufzeitmodellierung von der Prozess- auf die Instanzebene. Die hier vorgestellte Laufzeitmodellierung darf daher nicht mit Ad-hoc-Änderungen [Rinderle 2004][Dadam et al. 2005] verwechselt werden, die zu inkonsistenten Zuständen führen und keine objektorientierte Metamodellierung beinhalten.

In der Art eines Laufzeit Forward Engineering⁴⁰, des Service Blueprinting ([Shostack 1982]) im Service Engineering [BullingerScheer 2003] oder des Prototypings können Prozesse so direkt während des Ablaufs – statt separiert in der Entwicklung – modelliert werden. Entsprechend Anforderung A6 ermöglicht OMICRON daher das Hinzufügen weiterer Prozessschritte zu einer bereits existierenden und gegebenenfalls sogar in Ausführung befindlichen Aktivität.

Durch das laufzeitbasierte Objektmodell, das in OMICRON Verwendung findet, müssen Prozesse nicht mehr erst komplett modelliert und dann instanziiert werden, wie dies beispielsweise bei Workflowsystemen oder in der Softwareentwicklung (MDA etc.) der Fall ist.

⁴⁰ der Entwicklung von im Konkretisierungsverlauf nachgeordneten Artefakten aus vorangehenden [StahlVölter 2005]

Die Umsetzung einer **Laufzeitmodellierung** oder sogar Just-in-time-Modellierung kann direkt durch die Modellierung nachfolgender Schritte erfolgen. Dann kann die Ausführung weiterer Aktionen im Prozess erst beginnen, wenn der gesamte nächste Schritt modelliert ist.

Alternativ kann auf der Instanzebene ein **Sentinel** („Wächterkomponente“, wie sie aus der Informatik bekannt ist) verwendet werden, der bereits den Fokus erhält, und dann mit der Aktion detailliert wird. Ein Sentinel stellt dabei sicher, dass keine Folgekante aktiviert wird, solange der Sentinel nicht durch eine korrekt und vollständig modellierte Komponente ersetzt wurde und ein neuer, nachfolgender Sentinel existiert. Die Verwendung mehrerer Sentinels lässt auch eine **Parallelisierung** des Ablaufs bei der Laufzeitmodellierung zu.

9 Modell-Verteilung: Ein verteiltes semantisches System

Im Projekt INT-MANUS wurde im Hinblick auf dynamische Unternehmens- und Fertigungsstrukturen ein Konzept für ein dezentrales, peer-basiertes System zur Produktionsüberwachung und -steuerung [SchlegelMüller 2005], ähnlich anderer Integrationsansätze für Peer-to-Peer und Semantic Web Technologien [StaabStuckenschmidt 2006] entwickelt.

Es stellt eine als Production Service Bus (PSB) bezeichnete Infrastruktur zur Verfügung. Diese basiert auf einem nachrichtenbasierten Semantic Messaging System, das jedoch mittels existierender SOA-Konzepte die Übertragungsqualität sichert. Direkte Aufrufe übertragen XML-basierte Nachrichtenblöcke, die sowohl semantische Protokollinformationen als auch Daten und Fragmente des OMICRON Modells enthalten.

Eine Interpretation der Modellinformationen durch den empfangenden Peer ergibt dabei, ob Nachrichten weitergeleitet oder intern verarbeitet werden. Der Anschluss proprietärer Systeme an die Plattform erfolgt mit spezialisierten Peers.

So entsteht eine automatisierte, semantische Nachrichtenübermittlung und Interpretation innerhalb des gesamten Produktionssystems.

Durch die Übertragung von Begrifflichkeiten und Konzepten des Web 2.0 Paradigmas auf die Produktion entsteht so das Produktion 2.0 Paradigma [Schlegel et al. 2007] einer dezentralen, vernetzten und auf intelligenten Modellen und Prozessen basierenden Produktion für die OMICRON das grundlegende Prozess- und Modellierungsframework liefert.

Anders als im Web 2.0 müssen in der Produktion jedoch Sicherheitsaspekte wie Zugriffsrechte, gelenkte Informationsflüsse und eine weitgehend „echtzeitfähige“ Infrastruktur berücksichtigt werden. Um Sicherheitskonzepte, wie sie für die SOA-basierte Open Process Control Unified Architecture (OPC UA) definiert werden [LeitnerMahnke 2006], einsetzen zu können, muss das Modell entsprechend vorbereitet und angepasst werden.

Als vorteilhaft in der Praxis erweist sich für die Dezentralisierung, dass die Typebene eines Produktionssystems wesentlich weniger umfangreich als die Instanzebene ist: In einem mittleren Betrieb stehen 280.000 Betriebsmittelzuordnungen nur 750 Einzelbetriebsmittel und 150 Betriebsmittelgruppen gegenüber [Scheer 1976][Kurbel 2005]. So kann eine kompakte Replikation der für die Interpretation wichtigen Typebene erfolgen, während Instanzen nur soweit für die Abläufe notwendig dupliziert werden müssen.

9.1 Dezentralisierung durch Peer-to-Peer-Konzepte

Dezentralisierung und Agentenkonzepte in der Produktion (vgl. [HarrisonColombo 2006]) haben in den vergangenen Jahren eine große Bedeutung aufgrund ihrer Vorteile gegenüber zentralen Ansätzen erlangt [MarikMcFarlane 2005], gerade auch als Peer-to-Peer-Multiagentensysteme⁴¹ auf Basis aktueller Technologien wie Semantic Web Services mit verteilten Ontologien [LastraDelamer 2005].

OMICRON wurde aus diesen Gründen für den Einsatz in einem Peer-to-Peer-System optimiert, wie es im INT-MANUS Projekt zur Produktionsüberwachung und -steuerung zum Einsatz kommt. Hierzu werden die folgenden Konzepte zur Verteilung in das Laufzeitsystem eingeführt.

⁴¹ Eine Betrachtung hinsichtlich der Performanz von Multiagentensystemen in industriellen Umgebungen bietet [Lian et al. 2006]

9.1.1 Caching des Modells

Ein **Typ-Cache**⁴² in Form einer virtuellen Typ-Tabelle dient der Replikation von Vererbungsinformationen. Er kann in einem verteilten, objektorientierten Komponenten-System eingesetzt werden, wenn der Zugriff auf die Klassenstrukturen nicht in schneller Weise abgewickelt oder der passende Modellausschnitt, beispielsweise aus Platzgründen nicht vorgehalten werden kann. Auch die Ausfallsicherheit kann so erhöht werden. Vererbungsinformationen können damit lokal in Tabellenform zwischengepuffert werden. Mit dieser Vererbungsinformation können allerdings nur Typ-/Subtyp-Prüfungen durchgeführt werden, wie dies mit `IsTransitiveInheritingFrom` (vgl. Kapitel 11.3) im Prototyp erfolgt.

Eine vollständige Interpretation ist aufgrund der fehlenden restlichen Modellinformationen nur über eine nachträgliche **Anforderung** oder durch einen **Cache** der gesamten Vererbungs-Struktur möglich: In einem verteilten System sind nicht alle Modellinformationen enthalten, daher müssen Teile des Modells eventuell zwischengespeichert und synchronisiert werden. Die Replikation kann auf unterschiedliche Weise erfolgen:

- Teilreplikativ: nur die benötigten Teile werden repliziert
- Clusterreplikativ: mehrere Peers replizieren und speichern zusammen das Gesamtmodell, gegebenenfalls wie bei RAID-Speichersystemen mit Teil-Ausfallschutz durch Parität oder Duplikate
- Vollreplikativ: manche oder alle Knoten replizieren das vollständige Modell vor

Bei guter Ausfall- und Netzwerksicherheit kann zur Vereinfachung auch ein Modellserverkonzept (vgl. INT-MANUS Prototyp) genutzt werden, der als Master für die Modelldaten dient. Die Synchronisation kann dabei mit Hilfe eines Update-Broadcasts, über das Semantic-Messaging-System realisiert (erzwungen) werden. Nachteilig wirkt sich jedoch aus, dass die Modelldaten bei Serverausfall teilweise nicht mehr verfügbar sind.

Hierzu sind unterschiedliche Cache-Mechanismen zu unterscheiden, die in Anhang B beschrieben werden.

Die **Verteilbarkeit** des Gesamtmodells ermöglicht eine einfache **Auslagerung** von Produktiv-Komponenten in ein eigenständiges Repository. So kann beispielsweise eine Maschinen- oder Werkzeugontologie in einem oder mehreren dedizierten Modellrepositories vorgehalten und editiert sowie von anderen Peers repliziert werden.

9.2 Nachrichtenbasierung – das Semantic-Messaging-Konzept

„It is convenient to think of message generation and processing as just additional types of behaviors defined on the sending and receiving classes.“ [HealyKilgore 1997]

Für die reine Agentenkommunikation wurden bereits allgemeine, implementierungsnahen Ansätze wie der von [Talib et al. 2005] und die Agent Unified Modeling Language (AUML, [EhrlerCranefield 2004]) vorgestellt. Die Definition der Agentenkonzepte ist daher nicht Teil dieser Arbeit. Vielmehr wurde das Konzept so entwickelt, dass es eine beliebig strukturierte, ontologie- und nachrichtenbasierte Kommunikation in der dezentralen Plattform unterstützt.

In INT-MANUS wird ein komplexes Nachrichtenformat spezifiziert, um Nachrichten aller Art gestützt auf eine gemeinsame Ontologie versenden, routen und auswerten zu können. Dieses Semantic-Messaging-System ermöglicht die Klassifikation von Nachrichten („Fehler“, „Daten“, „Modellaktualisierung“) und Inhalten („Positionssensorwert“), was eine direkte Interpretation, Verarbeitung und Weiterleitung basierend auf dem Nachrichteninhalt ermöglicht.

⁴² lokaler Zwischenpuffer für Typinformationen aus dem OMICRON Gesamtmodell

Die Übermittlung von Informationen und Modellbestandteilen wie Sensorwerten und Prozessinstanzen auf Basis semantischer Nachrichten ermöglicht die (auch parallele) Verwendung unterschiedlicher Übertragungsmethoden. Der XML-basierte Inhalt der Nachrichten kann von jedem Peer interpretiert werden, ohne dass das Kommunikationsmedium einen Einfluss ausübt. Ebenso wie die verwendeten Web-Services mit SOAP können auch direkte Verbindungen zur reinen Textübertragung eingesetzt werden.

Ausfallsicherheit und Integrationsfähigkeit im Hinblick auf virtuelle Unternehmen und Betreiber sowie wechselnde Produktions- und Unternehmensstrukturen wird zum einen durch die Dezentralisierung, Peer- und Nachrichtenbasierung erreicht. Zum anderen ermöglicht das Modell eine Definition von Hierarchien und zugeordneten Sichtbarkeiten.

Ausfallsicherheit: Das System kann im laufenden Betrieb zerteilt werden – Verfügbarkeit aller notwendigen Daten in allen Teilsystem durch Replikation vorausgesetzt. Ein SuperNode-Konzept handelt automatisch für jedes konnetzte Teilsystem einen neuen SuperNode aus, der von diesem Zeitpunkt an das Teilsystem verwaltet und bei einer Wiedervereinigung für die Integration sorgt [SchlegelThiel 2007a][SchlegelThiel 2007b]. Ähnliche Konzepte für semantische Super-Peers [Chirita et al. 2006] existieren derzeit außerhalb der Produktion.

Integrationsfähigkeit wird durch die Vereinigung mit bisher separaten Systemen erreicht – ein gemeinsames Metamodell und das SuperNode-Konzept liefern hierfür die Voraussetzungen. Ein Super-Node erzeugt und verteilt die Peer-Infrastruktur, muss aber nicht zwangsläufig auch Modellteile vorhalten. Ein Modellserver muss also keine Supernode sein – auch nicht umgekehrt.

Production Service Bus (PSB): Sprachen wie ISDL [Quartel et al. 2004][Quartel et al. 2006] und BPEL [IBM et al. 2003] definieren verteilte Abläufe über ihre Endpunkte und Zusammenhänge. Eine starre Verkettung von Diensten ist für die Anforderungen eines Laufzeitsystems für objektorientierte Prozesse jedoch zu unflexibel. In Kapitel 2.3.4 ist die Problematik der Prozessverteilung bei Web Services beschrieben: Eine statische Definition und zentralisierte Technologie verhindern eine flexible Ausführung von Prozessen zur Laufzeit. Daher wurden neben reinen Web-Service-Konzepten auch Semantic Web-Service Konzepte eingesetzt.

Ansätze mit Semantic Web Services versuchen meist durch eine ontologiebasierte Verknüpfung atomare Dienste aufzurufen [LastraDelamer 2005]. Es werden dann wieder dezentrale Funktionen zentral von einem MES verwaltet: „*A Manufacturing Execution System (MES) is assigned with the functionality of orchestrating several simultaneous process workflows. A SWS-enabled MES will typically receive production sequences from a previous scheduling stage.*“ [LastraDelamer 2005]. So wird wieder ein zentrales Konzept wiewenigleich mit flexibleren dezentralen Funktionen erzeugt.

Durch das semantische Nachrichtensystem (Semantic-Messaging-System, SMS) soll diese Einschränkung der Dynamik und Typbasierung hinfällig werden, indem Nachrichten zur Laufzeit typbasiert interpretiert und weitergeleitet werden. Die Verteilung erfolgt dabei transparent für den Prozess und die ausführenden „Services“ (Peers), die die Aktivitäten ausführen und hierzu Informationen aus Modellfragmenten erhalten und weitergeben.

Ziel ist es, alle lokal verfügbaren Funktionalitäten auch lokal zu nutzen. Steht keine passende Komponente für eine Aktivität oder Information zur Verfügung, wird eine Nachricht mit allen erforderlichen Informationen und dem Kontrollflussstatus generiert. Diese wird über das SMS weitergeleitet und gelangt schließlich zum passenden Peer, der die Aktion ausführen und so die Prozessinstanz weiterbearbeiten beziehungsweise das Ereignis verarbeiten kann. Die im INT-MANUS Projekt entwickelte Smart Connected Control Platform (SCCP) [SchlegelMüller 2005] stellt eine Basisschicht hierfür zur Verfügung. Über eine Web-Service-Infrastruktur wird sichergestellt, dass Nachrichten beziehungsweise Aufrufe korrekt übertragen werden. Die nachrichtenbasierte Kommunikation verhindert dabei Deadlocks trotz Web-Service-Technologie

und ermöglicht zudem das beliebige Weiterreichen von Aktionen, Daten und Ereignissen. Trotzdem wird durch einen Benachrichtigungsschritt (Notification) die korrekte Weiterverarbeitung signalisiert.

Ergänzend zum Enterprise Service Bus (ESB, vgl. [BeinhauerSchlegel 2006]) für die servicebasierte Kommunikation im Unternehmen entsteht ein Production Service Bus (PSB, [SchlegelThiel 2007b]), der für das OMICRON System eine semantikbasierte Nachrichteninfrastruktur zur verteilten Modellierung und Ausführung von Prozessen bietet. Mit definierter Nachrichtenschnittstelle und OMICRON als typologischem Modell können so komplexe Aufrufe erfolgen und Prozesse dezentral ausgeführt werden.

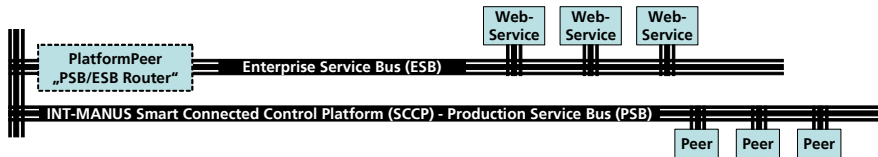


Abbildung 67: Production Service Bus (PSB), basierend auf semantischen Nachrichten

Neue Kommunikationsprimitive und Message-Typen sind gestützt auf das semantische Modell jederzeit möglich, da alle Nachrichten im Modell klassifiziert werden. Wenn keine eigene Behandlung für einen Nachrichtentyp vorliegt erfolgt wie für Prozessschritte die Suche nach einer möglichst speziellen Bearbeitungsroutine für die Nachricht. Liegt für „Stromausfall“ keine Behandlung vor, wird beispielsweise „Kritischer Fehler“ als der speziellste übergeordnete Typ herangezogen, so dass für jeden Nachrichtentyp eine semantische Interpretation und Behandlung erfolgen kann.

Die Modellinterpretation schließt neben dem Nachrichtentyp auch Sender, Empfänger etc. mit ein. So enthält ein Modellbereich für eine Zelle, Produktionslinie oder Fabrik die Instanzen der Maschinentypen, Werkzeugtypen, Hallen etc. Aus dieser Einordnung in den Kontext lassen sich gegebenenfalls auch regelbasiert weitere Schlussfolgerungen ziehen. Das Gesamtmodell kann zudem auch für angrenzende Systeme wie die Logistik oder Enterprise Resource Planning (ERP) geöffnet werden („wie viele Instanzen des Typs Schraube M5 sind noch im Lager verfügbar?“).

Die Anbindung angrenzender, proprietärer beziehungsweise auf einer anderen Technik basierender Systeme muss für eine semantische Integration über einen „Router“ erfolgen, der in das PSB-System eingehende Daten nach Typ, Bedeutung und Herkunft klassifiziert und ausgehende Daten in das spezifische Format und Aufrufe des angrenzenden Systems umwandelt.

Jeder **Peer ist eine Instanz des Typs Peer**; diesem sind unter anderem Maschinen als Instanzen von Maschinentypen, und gegebenenfalls Mitarbeiter zugeordnet. So können mehrere Maschinen einem Peer (Cell Peer), jedoch auch mehrere Peers einer Maschine (je Sensor ein Peer) zugeordnet werden.

Aufgrund des dezentralen Aufbaus muss sichergestellt werden, dass für genau einen Prozessschritt bestimmte Nachrichten nur diese Instanz erreichen und nicht in anderen (beispielsweise parallelisierten) Teil-Instanzen des Prozesses genutzt werden. Absender und späterer Empfänger sind ebenso wie der versendende und gegebenenfalls empfangende Prozess anhand ihrer ID eindeutig identifizierte Entitäten. Rückgabewerte beziehungsweise Ergebnisse können so gezielt an einen dedizierten Empfänger versandt werden.

Zudem werden Nachrichten, die nicht als Broadcast markiert sind, nur an jeweils einen möglichst passenden Empfänger weitergeleitet, der diese Nachricht noch nicht erhalten hat. Um festzustellen welche Empfänger die Nachricht bereits erhalten haben, wird im Nachrichtenkopf eine Liste von bereits erfolgten Empfängen geführt. Über das semantische Modell lassen sich

dann auch typologische Fragen beantworten: Hat bereits eine Fräsmaschine diese Nachricht erhalten?

9.3 Hinzufügen neuer Elemente zum System

Häufig müssen zur Laufzeit neue Modelle oder Modellbestandteile hinzugefügt oder entfernt werden. Neben der direkten Modellierung im Gesamtmodell ist zur Laufzeit das Hinzufügen und Entfernen größerer Modelle ein wahrscheinliches Szenario. Wird beispielsweise eine neue Maschine, ein neues Interaktionsgerät oder ein neuer Prozess in das System integriert, müssen eine Reihe von Schritten durchlaufen werden.

1. ähnlich wie beim Einstecken neuer Geräte an einem PC muss sowohl das Gerät als auch der Gerätetyp vom System erkannt oder durch Broadcast im System angemeldet werden. Wichtig ist hierbei, dass eine eindeutige Adresse direkt mitgebracht oder zugewiesen wird, über die das Gerät erreicht werden kann. Meist handelt es sich hierbei um den zugehörigen Agenten, bei Hardware auch um eine direkte Adresse und ID. Ist dies mangels Software nicht möglich, kann die Identifikation und Parametrisierung auch über Schritt 3 erfolgen.
2. Für das Gerät muss dann vom System ein beschreibendes Teilmodell ähnlich wie ein Treiber erfragt werden
3. Dieses kann auf unterschiedliche Weise akquiriert werden:
 - a. Erfolgreiche Anfrage an und Versendung durch die neue Komponente beziehungsweise den zugehörigen Agenten
 - b. Erfolgreiche Anfrage an ein dem System zur Verfügung stehendes Modell-Repository, das ein entsprechendes Modellfragment in der Art einer Treiberdatenbank zur Verfügung stellen kann. Hierbei handelt es sich nicht um das aktive Modell sondern um ein zusätzliches Repository.
 - c. Manuelle Eingabe, gegebenenfalls durch einen modellbasierten oder extern implementierten Wizard.
4. Integration des Modellfragments in das aktive Gesamtmodell in folgender Reihenfolge, sowohl auf Klassenebene (beispielsweise Hinzufügen neuer Typen) als auch auf Instanzebene (beispielsweise Erzeugen einer neuen Maschine):
 - a. Überprüfung auf bereits existierende Bestandteile anhand von eindeutigen IDs
 - b. Gegebenenfalls Aktivierung bereits existierender aber deaktivierter Elemente
 - c. Aktualisierung oder Verwerfen bereits existierender Bestandteile
 - d. Einfügen neuer Bestandteile
5. Gegebenenfalls Start einer Aktivierungs- oder Konfigurationsroutine, falls ein entsprechender Prozess durch das generierte Hinzufügen-Ereignis gestartet wird.

Das Entfernen oder Verändern von Instanzen und Klassifikationen aus dem Laufzeitsystem, beispielsweise wegen Stilllegung einer Maschine oder Zerstörung des Werkzeugs, kann gegebenenfalls erfolgen, um das Modell kompakt zu halten, ist im Prototyp jedoch nicht vorgesehen. Instanzen können dabei deaktiviert oder entfernt werden.

Die **Desaktivierung** kann hierbei von einer vorübergehenden Inaktivität bis hin zur endgültigen Desaktivierung unterschiedliche Eigenschaften haben. Jedoch entscheidet dies nicht über die Entfernung aus dem Modell, da beispielsweise nach einer erfolgten Löschung auch alle mit diesen verbundenen Produktdaten und Projekte unbrauchbar würden.

Bei der **Entfernung** von Modellbestandteilen oder kompletten Teilmodellen aus dem Gesamtmodell müssen alle **Seiteneffekte** beachtet werden.

Mit dem Entfernen von Komponenten werden auch immer alle Relationen entfernt, die mit diesen verbunden sind. Löschungen am Ende der Vererbungshierarchie (sog. Blätter im Vererbungsbaum beziehungsweise -DAG) müssen nur Instanzen und eventuelle andere problematische Relationen beachten. Sollen Elemente mitten aus dem Vererbungsbaum entfernt werden, muss entweder eine Ersatzkomponente installiert oder der gesamte Teilbaum entfernt werden.

Die Entfernung von Typen aus dem Gesamtmodell ist nur sinnvoll, wenn diese explizit nie mehr angeboten werden sollen. Sonst ist eine Desaktivierung mit Sicherstellung der ID meist vorzuziehen.

9.3.1 Nicht-integrierte Geräte

Sollen Elemente angebunden, die selbst nicht über eine PlatformPeer-Funktionalität verfügen, wie beispielsweise Sensoren, Aktoren etc., müssen diese über einen PlatformPeer kontrolliert werden. [Jammes et al. 2005] schlagen hierzu Residential Device Controller (RDC) auf der Basis von Web-Services vor, die der Plattform untergeordnete Gerätefunktionalitäten zur Verfügung stellen. Secure Device Control Gateways (SDCG, [Xu et al. 2005]), akzeptieren aus Sicherheitsgründen als Gateway zudem nur korrekte, verschlüsselte Kommandos, was den Zugriff sicherer aber komplexer macht.

Wie bei allen Datenein- und -ausgängen für externe Daten, müssen diese mit Hilfe von Wrappern in ein OMICRON-konformes Format beziehungsweise eine entsprechende Repräsentation im Komponentenmodell umgewandelt werden. Ein Beispiel hierfür sind Sensordaten, wie sie im INT-MANUS System von den NC-Steuerungen über das SMART System herstellerunabhängig geliefert werden. Ein kontinuierlicher Strom von Sensorwerten wird dabei klassifiziert.

Wichtige Informationen können so vollständig als Instanzen in das Modell integriert werden. Meist ist jedoch nur die Zuordnung des Sensors – aggregiert von der Maschine oder NC-Steuerung – im Gesamtmodell enthalten und die Massendaten extern gespeichert, um einer Überfüllung des Modells vorzubeugen. Diese Sensordaten werden dabei in ein klassifiziertes Nachrichtenformat übergeführt und über die nachrichtenbasierte Plattform in eine Datenbank transferiert (vgl. [Aravind 2007]).

Die Anbindung kann auch erfolgen, indem **virtuelle PlatformPeers** angelegt werden, die auf demselben Rechner laufen, jedoch als Agenten für ein spezielles Geräte zuständig sind. So steht für jedes Gerät ein eigener Peer zur Verfügung, statt mehrere Geräte durch einen einzigen Peer verwalten zu lassen.

9.3.2 Hierarchien im dezentralen Peer-Netzwerk

Das hierarchische Strukturmodell basiert auf der Zusammenfassung von Peers mit Hilfe der Aggregation und integriert sich in das Gesamtmodell.

Über Vererbung werden Typen von Produktionsentitäten erzeugt. Diese sind zunächst meist physisch-organisational wie Maschinen, Bereiche (Zellen, Produktionslinien), Produktionsstätten, Produktionsverbände, Unternehmen, Verbundunternehmen in virtuellen Unternehmen und Lieferketten (Zulieferer, Betreiber, Kunden) definiert, können aber auch zu anderen Hierarchien zusammengefasst werden, die dynamisch virtuelle Gruppen bilden.

Über die Aggregation können jeder Instanz dieser Produktionsentitäts-Typen, das heißt jeder Produktionsentität, sowohl untergeordnete Entitäten zugeordnet werden (Maschine zu Zelle) als auch diese wiederum einer höheren Entität (Zelle zu Fabrik) hinzugefügt werden. Mit einer Modellierung als Komposition kann das Hinzufügen zu mehreren Entitäten verhindert werden.

Die Hierarchie wird dabei im Metamodell als Spezialisierung von ProductionControlEntity in Anlehnung an die Produktionssteuerungsebenen nach Pritschow [Gehnen 1999] umgesetzt:

- (Planungsfunktionen)
- Leitsteuerungsfunktionen
- Zellensteuerungsfunktionen
- Maschinensteuerungsfunktionen
- Einzelfunktionen
- Physikalische Ebene (Antriebe, Aktoren, Sensoren)

Diese werden mittels Aggregationsbeziehung hierarchisiert, jedoch organisational flexibelisiert:

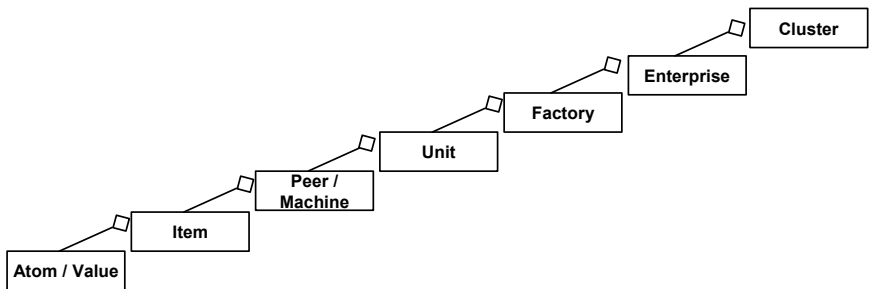


Abbildung 68: Beispielhafte Aggregationshierarchie einer Produktionsinfrastruktur

Die einzelnen Peers lassen sich anhand dieser Aggregationsstruktur ebenfalls zu neuen **High-Level-PlatformPeers** zusammenfassen, die beispielsweise eine **Zelle (Cell)** in der Produktion als Einheit verwalten. Auf Instanzebene werden dann Zellen mit realen Maschinen verknüpft.

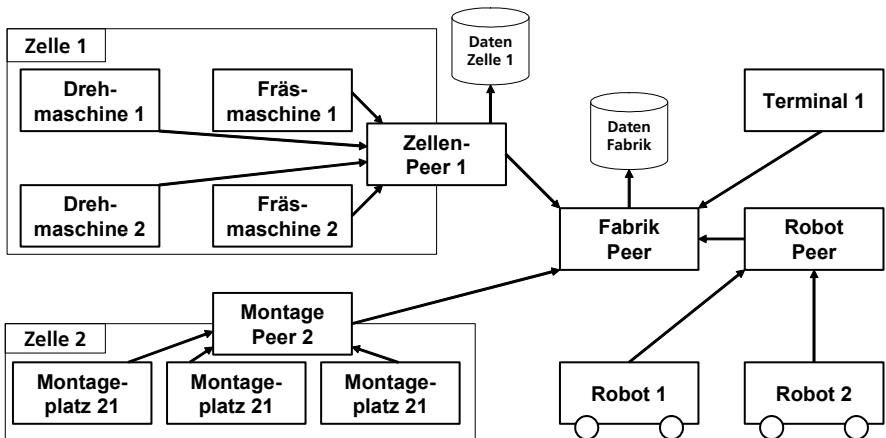


Abbildung 69: Beispielhafte organisationale Strukturierung auf Instanzebene

Die basale Vererbungshierarchie für diese hierarchischen Elemente umfasst erste Spezialisierungen dieser Elemente. Für diese gelten dieselben Aggregationsregeln aufgrund der Objektorientierung:

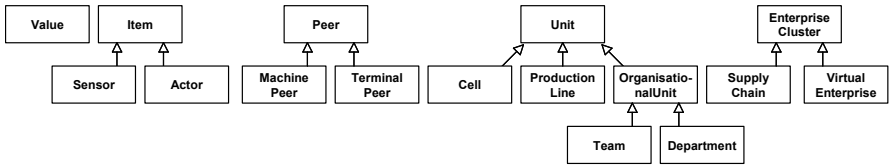


Abbildung 70: Prototypische Klassifikation von Struktur- beziehungsweise Organisationseinheiten

10 Szenarien in der Produktion

Im Folgenden werden drei Szenarien zur Verdeutlichung der Einsatzmöglichkeiten und Wechselwirkungen von OMICRON vorgestellt. Ein Einsatzszenario zeigt die Einbindung des OMICRON-Systems in eine bestehende IT- und Fertigungslandschaft mit existierenden Systemtypen und einer prototypischen Prozessausführung.

In der Folge wird ein Szenario aus der Praxis eines führenden Fertigungsunternehmens aus dem Bereich der Variantenfertigung geschildert und im Hinblick auf den Einsatz von OMICRON analysiert. Der Abschnitt zeigt ein Anwendungsszenario in der Variantenfertigung mit einer weitgehenden Massenproduktions-Infrastruktur für mehrere Marken und Unternehmen.

Teilszenarien beleuchten im dritten Teil bestimmte Modellaspekte, vor allem die Variantenbildung, genauer.

10.1 Einsatzszenario: Verteilte Ausführung in existierenden Systemen

In einem produzierenden Unternehmen besteht für existierende und auch neue Fertigungssysteme meist eine Systemlandschaft, die große Teile des Informationsflusses abdeckt.

Abbildung 71 zeigt beispielhaft die Einbindung von OMICRON in eine solche Systemlandschaft. Zentrales Element ist dabei OMICRON als Prozesseigner, da die Prozessausführung das Zentralstück von OMICRON darstellt. Mit Hilfe der beschriebenen Verteilungsarchitektur können weitere Systeme wie MES und ERP durch spezifische PlattformPeers integriert werden. Dieser regelt die Kommunikation zwischen den Teilsystemen und OMICRON.

Die (onto)logische Zusammenfassung von Maschinen und Montageplätzen zu größeren Strukturen wie Zellen oder Produktionslinien geschieht dabei ebenso wie die Prozessmodellierung und Ausführung in OMICRON. Neben der Definition von Strukturen ist auch die Klassifikation aller Elemente von NC-Programmen über Maschinentypen und Steuerungstypen in der Ontologie enthalten.

Im abgebildeten Szenario wird ein im ERP-System erzeugter Auftrag 548397 durch den „ERP Bridge“ PlatformPeer kommuniziert.

Der zuständige Peer erzeugt aus dem Auftrag ein Projekt, das wie im Prozess definiert die Auftragsnummer als Information erhält. Der Prozess läuft weiter bis zur Produktionsanweisung für das Teil T136.

Nachdem eine passende und verfügbare Drehmaschine für die Aufgabe Drehen durch die Zelle A identifiziert wurde, wird ebenfalls anhand der Ontologie das korrekte NC-Programm für Teil T136 und Maschine A.1 identifiziert und über Netzwerk aus der Datenbank übermittelt.

Wie das Programm für die NC-Maschine, können auch SPS-Programme für Steuerungen oder Montageanweisungen für das Montagepersonal identifiziert und zur Verfügung gestellt werden.

Die Optimalform einer vollständigen Kontrolle über die Produktion durch OMICRON kann allerdings in der Praxis nur in entsprechend konzipierten Neusystemen erreicht werden. Daher besteht die Möglichkeit, beispielsweise Stücklisten zwischen einem geeigneten PPS und OMICRON zu synchronisieren.

Weitere Teilsysteme wie die Maschinendaten-Erfassung (MDE) oder die Werkzeugverwaltung können über das Peer-to-Peer-System über definierte Schnittstellen angebunden werden.

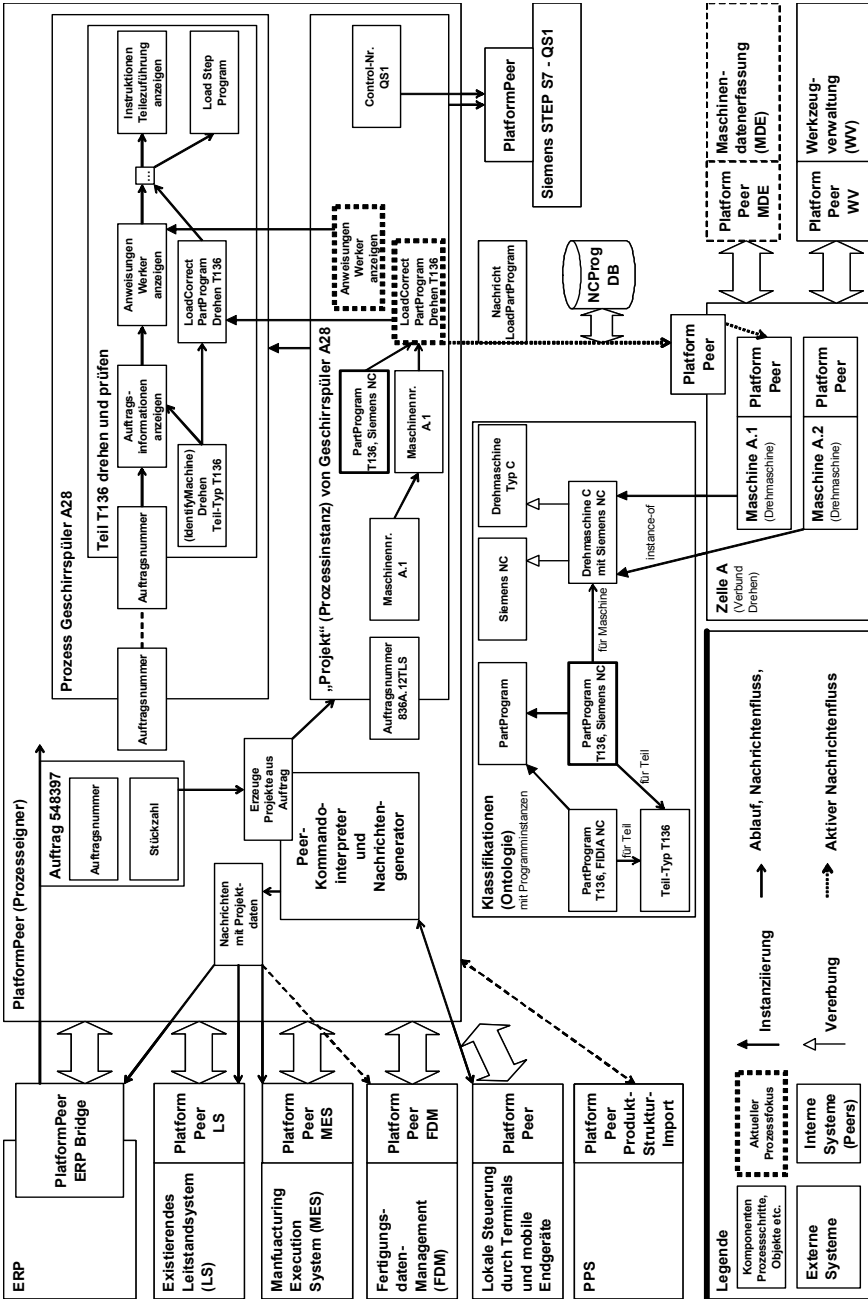
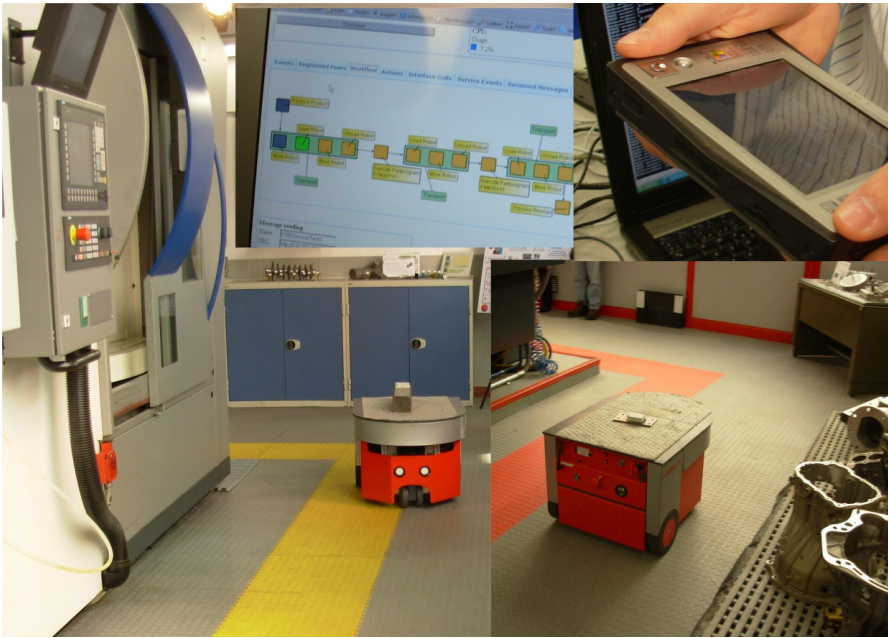


Abbildung 71: Einsatzszenario für OMICRON in einer bestehenden Fertigungsinfrastruktur



Quelle: Centro Ricerche Fiat S.C.p.A. (CRF), Turin

Abbildung 72: Bilder vom Test der Peer-Plattform von INT-MANUS, November 2007

Das obige Bild zeigt die Erprobung der prozessorientierten Peer-Plattform zusammen mit dem Roboter-System, NC-Maschine und mobilen Endgeräten zur Produktionsüberwachung im Fiat-Forschungszentrum in Turin im Rahmen des europäischen Forschungsprojekts INT-MANUS.

10.2 Variantenfertigung: Flexibilisierung und Effizienzsteigerung

Die BSH Bosch und Siemens Hausgeräte GmbH produziert vier Millionen Geschirrspüler pro Jahr, von denen jeder zu einer von etwa 1000 Varianten gehört. Eine Pflege des Variantenbestands, mit Entfernung oder Tausch alter mit neuen Varianten, ermöglicht dabei eine nur leicht steigende Tendenz der Variantenvielfalt beizubehalten.

Eine Problematik der vielen Varianten ergibt sich aus unterschiedlichen Teilen (Variantenteilen). Varianten mit vielen Teilen, die nur für eine Variante und Baugruppe benötigt werden, erhöhen den Aufwand und das Risiko in Logistik und Beschaffung. OMICRON hilft dabei, diese typologisch zu erkennen und Alternativen zu finden und hat seine Stärken auch bei vielen Varianten mit gemeinsamen Teilen (Gleichteilen). Hier wird der Aufwand für Prozess- und Variantendefinition reduziert, sowie die Verwendung von Gleichteilen gefördert.

Bei Losgrößen von 50, 80 oder 100 Geschirrspülern (selten 200 bis 300) entstehen für die Losfolge Intervalle von durchschnittlich 40 Minuten. Während im Prüfbereich noch das vorangegangene Los überprüft wird, läuft in derselben Linie bereits die Fertigung des nächsten an. Parallel laufende Prozessinstanzen in OMICRON ermöglichen hier eine simultane Bearbeitung. Werkstattführungsmannschaft und Gruppeneinsteller überwachen speziell den Loswechsel und sind am Neuanlauf von Varianten beteiligt. Die Rollenbasierung von OMICRON ermöglicht dabei eigene Prozessschritte und Informationen für diese speziellen Gruppen.

Eine Variantenstückliste zeigt für jedes Los die zu verwendenden Teile. Diese können in OMICRON, zusammen mit notwendigen Eingaben und Bedingungen aus der Prozessschnittstelle, das heißt den Eingängen der Prozesskomponente, ermittelt werden. Auch Fertigungshinweise werden mit jedem Los angezeigt. Diese können mit Hilfe spezieller Dokumentationsrelationen im OMICRON Modell modelliert und extrahiert werden.

Die **Trennung** von Hoch- und Niedrigpreissegment kann unter anderem nach Produktionslinien flexibel in OMICRON modelliert werden. Auch Häufigkeiten (Rennerlinien und Kleinlose) und Produktklassen wie 60- und 45-Zentimeter-Modelle könnten durch Klassifikation und entsprechende Optimierung abgebildet und bevorzugt oder ausschließlich auf bestimmten Linien gefertigt werden. Hier zeigt sich, dass für die Produktionsoptimierung das reine objektorientierte, semantische Modell durch weitere Werkzeuge und Werte ergänzt werden muss.

Eine Verfolgung der Abläufe erfolgt per IT-basiert. Allerdings ist, besonders für Einzelerzeugnisse, bisher keine volle Verfolgung auf Instanzebene möglich. Dies ändert sich mit einem OMICRON-System, da die Instanzen vollständig mitlaufen und somit immer die aktuellen Ablaufinformationen tragen.

Bisherige Ansätze streben keine Einzelverfolgung sondern Optimierung der wichtigsten Schritte an. OMICRON kann dagegen nicht nur statistische Informationen zur Verfügung stellen, sondern erlaubt die Instanzverfolgung, so dass, die besonders bei neuen Modellreihen oder stark abweichenden Varianten wichtige Einzelverfolgung und Modellierung möglich ist.

Änderungen von Prozessen werden dabei bisher in Projekttagen und über Änderungsmitteilungen kommuniziert und gegebenenfalls in kurzen Treffen vor Schichtbeginn („Lernstatt“, Erfahrungsaustausch) mit weiteren Themen wie Zuständigkeiten, Planung und Probleme angesprochen.

Zeitdruck bestimmt gerade auch den **Neuanlauf**. Hier kann OMICRON den kritischen Freigabeprozess (kritischer Pfad) unterstützen und koordinieren helfen, an dem Stakeholder wie Marketing, Produktionsplanung, Fertigungsplanung, Materialwirtschaft / Einkauf, Qualitätsmanagement und weitere Spezialisten (Liefereraussagen, ...) beteiligt sind. Dabei können separierte OMICRON-Systeme für die Prozessunterstützung in Entwicklung/Freigabe und Fertigung oder ein integriertes System verwendet werden. Bei Integration können die Prozesse koordiniert werden, indem Entwicklung, Freigabe, Auftrag, Fertigung, etc. in OMICRON verschaltet werden und die Prozesse miteinander kommunizieren.

Der Erstellung der Stückliste folgt die maschinelle Planung und der Beschaffungsprozess. Neben der Prozessunterstützung sind sowohl die direkte Modellierung der Prozesse während eines prototypischen Durchlaufs (Just-in-Time-Modellierung) als auch eine Verbesserung der definierten Prozesse während des Arbeitsablaufes mit OMICRON möglich.

Bis zu fünf Vorläufermuster werden vor Prüfung und Freigabe erzeugt, ein weiteres kurz vor dem Produktionsstart eines neuen Modells (Start of Production, SOP) als Nullserie. Dabei entsprechen Prozess, Werkzeug und Maschinen bereits der geplanten Serie.

Bei bestehenden Prozesssystemen stellt häufig die zu geringe Berücksichtigung von menschlicher Interaktion und Flexibilität einen Kritikpunkt dar. Eine regelmäßige Neuanlaufbesprechung für neue Varianten/Modelle berücksichtigt hier den **Menschen**, der auch in Prozesssystemen eine ungemein wichtige Rolle spielt. Mit Interaktionen und der polymorphen / semantischen Auswahl von Teilprozessvarianten in OMICRON soll diesem Umstand Rechnung getragen werden. Die Maxime lautet dabei Teilschritte zu automatisieren, die aus technischen oder organisatorischen Gründen in einer bestimmten Art ausgestaltet werden müssen – bei Erhaltung größtmöglicher Flexibilität.

Planungshorizonte sind in der Praxis sieben Tage fix mit 6 Monaten Gesamtübersicht. Dies ist auch mit der flexiblen OMICRON Konfiguration möglich. Dabei kann die Planung und Übersicht

in Echtzeit durchgeführt werden – Veränderungen und Seiteneffekte werden direkt sichtbar und berechenbar. Die Abweichung kann dabei größer oder kleiner als die statistik- und erfahrungsbasierte sein, hängt aber nicht mehr so stark von der Einschätzung der Planungskräfte ab und ist konsistenzgesichert.

Bei einer Materialreichweite von fünf Tagen – bei speziellen Teilen auch von weniger als einem halben Tag (Just in time) – wird die Logistik zum begrenzenden Faktor für größere und spezielle Änderungen im Ablauf und schränkt so die Planungsflexibilität ein. Daher müssen diese in Planung und Modell berücksichtigt werden. Bei Losausfall und Vorziehen anderer Lose kann so ein Logistikproblem entstehen. Mit OMICRON kann eine dynamische Laufzeitplanung erfolgen, für die jedoch genaue Logistikinformationen essentiell sind. Ein vollständiges Vorgespräch zum Schichtwechsel ist mit einer dynamisch angepassten Fertigung nicht mehr möglich, da keine stabile Planung für die gesamte Schicht besteht.

Kernstück einer Flexibilisierung ist die vollständige Abbildung, beziehungsweise Ankopplung der Logistik an das Modell, um die Materialverfügbarkeit sicherzustellen. Just-in-Sequence ist dagegen bei kurzfristiger Änderung nur noch fabriekintern möglich. Hier würde sich eine Logistik-Leitstandsoftware anbieten, die alle anstehenden und im Prozesslauf zu erwartenden Eingänge der OMICRON-Prozesse überwacht und so Vorhersagen und Beschaffungsaufträge für die Logistik ableiten kann.

Hier zeigt sich, dass Zeitabschätzungen eine große Rolle spielen. Das OMICRON Modell eignet sich auch für die direkte Überlagerung jeder Komponente mit Zeitinformationen. So könnten alle existierenden Prozessschrittinstanzen zur Klasse ihre Zeit zurückmelden. Im einfachsten Fall in einem angenäherten Update der Form:

```
DurchschnittNeu := Durchschnitt  
+ ((AktuelleDauer - Durchschnitt) / Anzahl der bisherigen Messwerte)
```

Dann müssen Instanzen nicht zwangsläufig gespeichert werden, aber jedes Schema (Klasse) enthält pro Teilschritt die Zeitabschätzung, so dass Fertigungszeiten für Prozesspfade vorhergesagt werden können – wichtig wiederum für Planung und Logistik. Ist keine Information vorhanden, beispielsweise wegen eines Neuanlaufs, können aus bekannten und abgeleiteten Prozessschritten bereits geschätzte Zeiten errechnet werden, die sich im Laufe der Zeit asymptotisch annähern.

Der **Prozess an sich wird beherrscht**, so dass die Fehler meist Einzelfehler sind, die sich teilweise durch Vorrichtungsoptimierung, selten jedoch durch Schulung etc. beheben lassen. Beispielsweise werden Schrauben fallen gelassen, vergessen oder nicht voll angezogen.

Etwa 30 unterschiedliche Prozesse, wie die Montagefolge, existieren pro Montagelinie. Dies sind die Grundprozesse in OMICRON, die teilweise in Varianten für eine Vollintegration vorliegen müssen.

Ein 100 Prozent konformer Prozessablauf ist meist Theorie, so dass ein möglichst hoher Annäherungsgrad und eine Systemabdeckung von Abweichungen wichtig sind. Optimal wäre für OMICRON, wenn diese konform über Polymorphie abgedeckt werden könnten, was sich in der Praxis bei unvorhergesehenen und auch nicht registrierten Abweichungen als schwierig erweist.

Als eine Herausforderung erweisen sich für ein weltweit operierendes Unternehmen die unterschiedlichen Gegebenheiten der Länder in Verbindung mit der Prozessvereinheitlichung und Qualitätssicherung. Hierbei müssen einerseits exakte Abläufe und Standards definiert und eingehalten werden, andererseits Anpassungen auf spezifische Gegebenheiten wie das Verhältnis von Personal- und Investitionsmittelkosten vorgenommen werden. Hier kann eine länderspezifische Prozessvariantenbildung mit spezifischer Umsetzung von Teilschritten eine hohe Prozessqualität bei gleichzeitiger Lokalisierung ermöglichen.

Da die optimierte Fertigung mittlerweile kein „Stück verliert“, also keinen Anlaufausschuss produziert, wäre auch eine Fertigung mit **Losgröße 1** bei Anpassung und entsprechender Beherrschbarkeit der Produktionslogistik prinzipiell möglich. Es müssten alle Teile nahe der Linie und im Modell verfügbar sein.

Eine Bestückung mit Körben und Anleitungen, sowie die eigentliche Verpackung, müsste nachgelagert über ein Identifikationssystem (Barcode oder RFID) erfolgen. Ein Abgleich mit dem Modell muss sichergestellt werden, um bei ständig wechselnden Varianten und Modellen potenzielle Fehler zu erkennen und zu eliminieren. Der Vorteil von OMICRON ist dabei die vollständige Prozessintegration, die jedoch anfangs durch die Modellbildung und Erfassung aller Teiletypen einen Mehraufwand erfordert.

10.3 Detailszenario

Im Folgenden werden kompakte Teilaspekte des Szenarios vorgestellt, die helfen, das Vorgehen und die Leistungsfähigkeit des Ansatzes zu beurteilen.

Hinzufügen einer Maschine

Existiert bereits der Typ der Maschine, kann die Maschine als weitere Instanz dieses Typs direkt dem Modell hinzugefügt werden.

Existiert dieser noch nicht, muss durch Vererbung beziehungsweise Mehrfachvererbung der neue Maschinentyp klassifiziert werden.

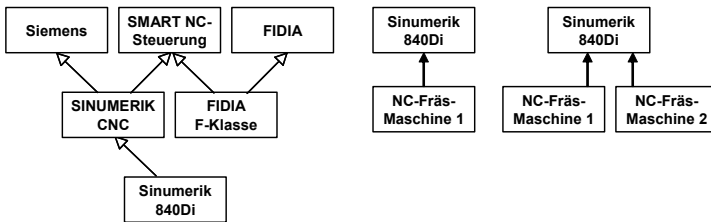


Abbildung 73: Hinzufügen einer NC-Steuerung bei existierender Definition und Instanz des Steuerungstyps

Sondermodell

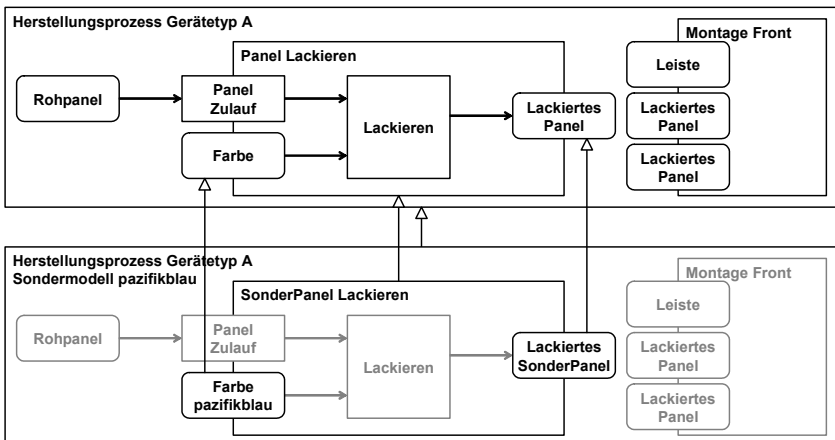


Abbildung 74: Spezifizierung eines Sondermodells – vereinfachte Darstellung

Als Spezialisierung der Modellreihe wird ein Sondermodell durch Ableitung definiert. Der Prozess wird ebenfalls durch Spezialisierung erzeugt. Teilschritte im Prozess wie Lackierungen werden danach in ihrem Schritttyp und ihrem Ergebnis angepasst.

Das Bedienpanel des Sondermodells erhält im Szenario eine spezielle pazifikblaue Farbe und liefert damit auch einen spezialisierten Paneltyp.

Neueinführung von Varianten und Hinzufügen einer Variante

Für ein existierendes Modell G1a, das bisher nur in einer Variante existiert, soll eine weitere Variante G1e gefertigt werden. Es handelt sich dabei um ein Energiesparmodell mit geänderter Steuerelektronik. Zudem soll eine weitere, mit dem ursprünglichen Modell technisch gleiche Designvariante G1d eingeführt werden.

Von G1a werden daher ein Prozess G1e und ein Prozess G1d abgeleitet. Für G1d wird nur an der Beschriftung des Bedienpanels eine Änderung vorgenommen, so dass ein spezialisiertes Bedienpanel im weiteren – sonst genau geerbten – Prozess Verwendung findet. G1e wird nach der Ableitung so modifiziert, dass nur das spezielle Steuergerät eingesetzt werden kann. Existieren von diesem speziellen Steuergerät unterschiedliche Varianten (beispielsweise von unterschiedlichen Zulieferern) können alle verbaut werden, wenn keine weiteren Einschränkungen getroffen werden.

Abnahme: Automatische Erweiterung aller betroffenen Prozesse

Nach Qualitätsdefiziten bei der Verkabelung wurde eine spezielle Überprüfung gefordert. Die Messung soll dabei als letzter Schritt der Fertigung durchgeführt werden. Mit der Sonderprüfung und parallelen Anbringung eines Prüfsiegels wird der Prozess ergänzt.

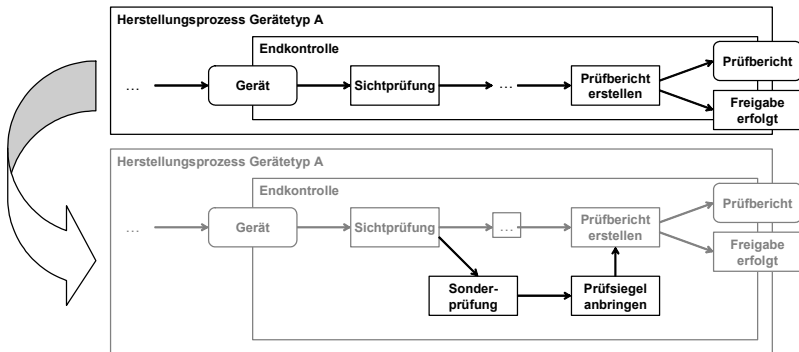


Abbildung 75: Erweiterung eines bestehenden Grundprozesses

Von diesem erbede, spezialisierte Prozesse werden nun vom allgemeinen Prozess „Herstellungsprozess Gerätetyp A“ ausgehend angepasst. Die Anpassung kann für jede Spezialisierung S allerdings nur erfolgen, wenn entweder gerade keine Variante S gefertigt wird (keine Instanzen) oder diese noch nicht die Sichtprüfung durchlaufen haben. Haben S-Geräte die Sichtprüfung durchlaufen, jedoch nicht den Teilprozess Endkontrolle beendet, wird auf die Beendigung gewartet. Alle Varianten des Gerätetyps werden nach Durchführung dieses Änderungsprozesses nach dem neuen Prozess mit Sonderprüfung gefertigt.

Diese Besonderheit entsteht bei der Änderung von laufenden Prozessen. Die – häufiger vorkommende – Einführung einer neuen Variante hat diese Auswirkungen aufgrund der Objektorientierung nicht. Ein Variantenprozess kann im laufenden Betrieb eingefügt werden und direkt Geräte produzieren.

Die Position einer Instanz im Prozess wird mit einem gefüllten Kreis und Strichlinie angezeigt. Blockierende Instanzen sind invertiert dargestellt. Grundlage bildet die in der obigen Abbildung beschriebene Prozessweiterung.

Wird von G1 eine neue Modellversion eingeführt, die die alte ersetzen soll, kann so auch G1 direkt angepasst und überschrieben werden. Auswirkungen auf Varianten werden damit direkt geprüft und nachgeführt. Alternativ kann eine parallele Version gebildet werden, so dass das Folgemodell bei Bedarf parallel gefertigt werden können.

Ein häufiger Fall sind dabei Cross-Promotions, die eine Zugabe von Geschenken wie bestimmten Reinigern vor der Verpackung erfordern. Hierzu wird in gleicher Form eine Prozessvariante geschaffen, die diesen zusätzlichen Schritt abbildet und kontrolliert. Sogar während eines bereits angelaufenen Loses kann diese Anpassung mit OMICRON erfolgen.

Die jeweilige Prozessvariante definiert und regelt auch die spezifische Verpackung, unterschiedliche Körbe bei gleichem Innenraum, andere Materialien wie Edelstahl, Einbau- und Stand-Alone-Varianten, sowie länderspezifische Stecker, Spannungswandler und Kabelbäume.

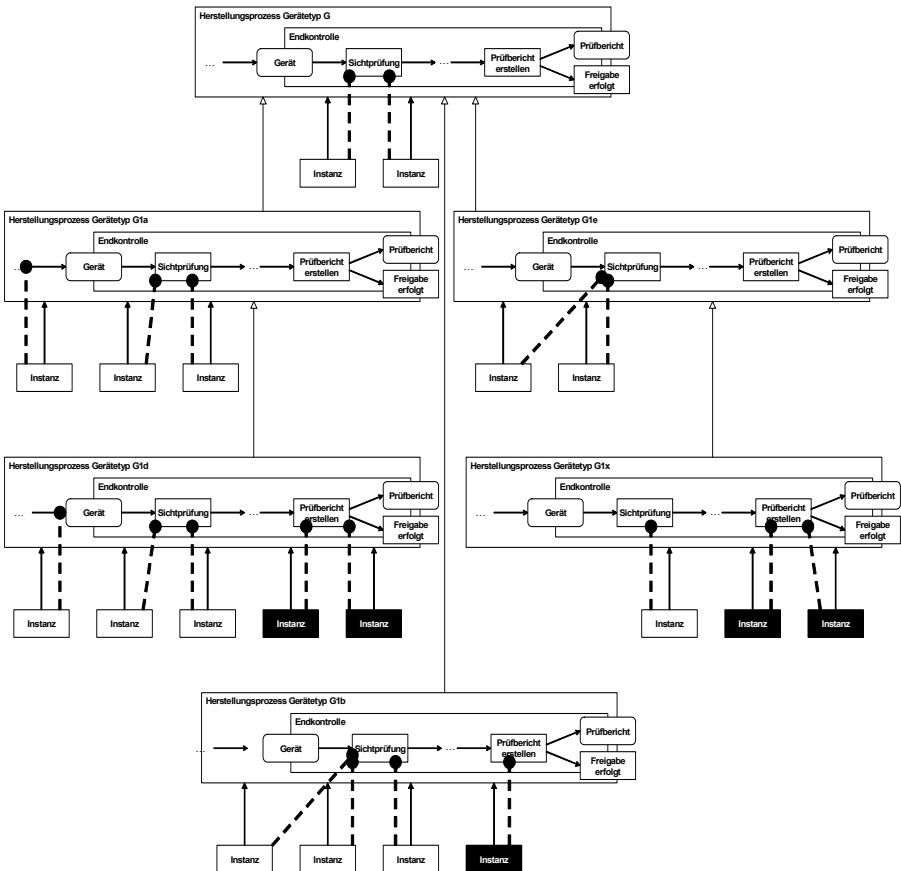


Abbildung 76: Blockierende Instanzen bei Änderungen mit existierenden Varianten

11 Implementierung

Die Implementierung soll die Umsetzbarkeit des Konzepts und dessen Systemeigenschaften zeigen.

Da die Arbeit stark methodisch und konzeptuell geprägt ist, wird die Implementierung vergleichsweise kurz dargestellt. Der Prototyp soll zeigen, dass die Umsetzung des Modellierungskonzepts möglich ist und die Integration von Objektorientierung und Prozessmodellierung auch in der Realisierung die gewünschten Eigenschaften zeigt. Auch die Grenzen und Fallstricke einer Umsetzung werden anhand der Implementierung erläutert.

Die **Implementierung der Modellverwaltung** für den Prototyp erfolgte in C# mit dem Microsoft Visual Studio 2005. Die Typsicherheit der .NET-Umgebung ermöglichte eine einfache Implementierung trotz hoher Typkomplexität.

Kern der Architektur des Laufzeitsystems ist die Modellverwaltung und das in Klassen/Objekten abgebildete Meta-Metamodell. Anhang P gibt einen Überblick über die Architektur und Klassenhierarchie des Komponenten-Relationenmodells einerseits sowie einige ergänzende Klassen zur Verwaltung und Laufzeitumgebung.

Das Laufzeitsystem setzt dabei das Meta-Metamodell in eine Modellverwaltung um, die die Modellzusammenhänge mit programmiersprachlichen Konstrukten abbildet und so für Abfragen und Modellmanipulationen zugänglich macht.

11.1 Komponenten-Server

Basierend auf den vorangegangenen Überlegungen kann ein System konzipiert werden, das die Modellverwaltung übernimmt. Jeder Peer verfügt über einen Modellservers, dessen Basis durch die Klasse *ComponentServerBase* implementiert wird.

Beim Start wird zunächst die Umgebung initialisiert. Hierzu wird ein **KeyServer** erzeugt, der für den Peer eindeutige Schlüssel für Modellelemente (Referenzquellensignatur, *ModelElementKey*) erzeugt. Das Meta-Metamodell wird bereits basierend auf dem unten beschriebenen Modellsystem erstellt. Da es axiomatische Voraussetzung für alle darauf basierenden Komponenten und Relationen ist, wird es durch den Initialisierungscode direkt erzeugt. Auch die weiteren in dieser Arbeit beschriebenen Grundelemente wie Instanziierung-, Aggregations-, Flow- und Vererbungs-Relationstyp sowie Basis-Komponententypen existieren bereits nach der Initialisierungsroutine. Dies stellt sicher, dass auch die Behandlungsroutinen anhand plattformweit eindeutiger Modellelemente korrekt funktionieren. Auch die programmiersprachliche Integration von Komponenteneigenschaften ist vorgesehen (vgl. Anhang L).

Nach erfolgter Initialisierung können weitere (Teil-)Modelle geladen werden. Hierzu wird die Klasse *ModelImporter* verwendet, die ein im XML-Format gespeichertes OMICON-Teilmodell einlesen und in das Gesamtmodell integrieren kann.

Initial müssen bereits grundlegende Behandlungsroutinen für die wichtigsten Relations- und Komponententypen definiert werden, da auf ihnen die Navigation und Auswertung des Gesamtmodells basiert. Existieren dedizierte Behandlungsroutinen für spezialisierte Relationen, können diese allgemeinere Behandlungen für diesen Typ ergänzen oder ersetzen.

11.2 Modellsystem

Als Basis für alle (speicherbaren) Modellbestandteile wurde das **ModelElement** eingeführt. Von diesem erben unter anderem **Component** und **Relation**, die beiden anderen Basiselemente. Bereits auf dieser Ebene wird mit dem *ModelElementKey* ein Identifier-Konzept eingeführt, das sowohl seinen Erzeugung- und Verwaltungsort (*cServer*) als auch eine eindeutige ID für jedes auf diesem erzeugte Modellelement zur Verfügung stellt. Für ein verteiltes System ist die

Generierung einer solchen ID notwendig, um Kollisionen bei der Verwendung auf unterschiedlichen Servern zu vermeiden. Das ID-System in OMICRON stützt sich auf die Voraussetzung, dass ein Peer im Netzwerk anhand seiner URI, IP-Adresse, oder MAC-Adresse des Netzwerkadapters eindeutig identifiziert werden kann, so dass nur die auf dem jeweiligen Peer erzeugten Modellbestandteile zusätzlich indiziert werden müssen.

Wird ein neues *ModelElement* erzeugt, wird seine ID anhand der eindeutigen Kennung des erzeugenden Peers und einer auf diesem Peer eindeutigen ID, beispielsweise einem inkrementellen Index, generiert.

Für die Animation und besondere Behandlung von Prozesskomponenten wird eine Spezialisierung *ProcessComponent* von *Component* eingeführt. Dies ermöglicht eine Trennung der Ablaufintelligenz für Aktivitäten von der allgemeinen Komponentenfunktionalität.

In der Implementierung wird die **Relationsreihenfolge** über eine Liste wiedergegeben und sofern nötig überwacht, um eine gezielte, deterministische Behandlung sicherzustellen. Speziell bei der Speicherung wird daher auf die richtige Reihenfolge geachtet.

Während im Meta-Metamodell Relationstyp-Komponenten ein von den Komponenten seapriertes Konzept mit ähnlichen Eigenschaften darstellen, implementiert die Laufzeitumgebung *RelationTypeComponent* als Spezialisierung von *Component*. Sie kann damit in der Implementierung alle Komponentenfunktionen für die Vererbung nutzen und verfügt über die entsprechenden Methoden zur Identifikation von Subtypen. Allerdings muss durch Überschreiben sichergestellt werden, dass nur Vererbungs- und Instanzrelationen benutzt werden dürfen. Eine vollständig separate Implementierung von *RelationTypeComponent* und *Component* führt zu vielen Redundanzen, wäre aber prinzipiell möglich.

Bereits in den Konstruktoren der Modellelemente sind Konsistenzsicherungsmechanismen enthalten. So kann eine Komponente nur erzeugt werden, wenn der ihm zugeordnete beziehungsweise übergebene Metamodell-Kontext korrekt ist (*IsComponentAllowed*). Auch Relationen können erst erzeugt werden, wenn deren beide Komponenten bereits existieren und die Verbindung durch im Metamodell vorgesehen ist. Bei Mehrdeutigkeit liefert *RelationAllowed* eine Liste von möglichen Relationen im Metamodell.

Beim Import eines neuen Modells zeigt sich jedoch, dass dieses Konzept problematisch ist. Teilmodelle enthalten häufig komplexe Beziehungen zwischen den Komponenten, die eine korrekte Festlegung der Import-/Exportreihenfolge erschweren. So muss beispielsweise zuerst das Metamodell importiert werden, bevor ein Modell erzeugt werden kann, das auf diesem basiert. Vererbungsbeziehungen müssen eine Komponente bereits als Subtyp einer anderen identifizieren, bevor diese in ein Modell eingegliedert und dabei überprüft werden kann. In der Weiterentwicklung der Laufzeitumgebung wird also ein differenziertes Modellaustauschkonzept zu entwickeln sein, das entweder Modellebenen und Modelle separiert oder die bisherige Steuerung der Komponenten- und Relationenreihenfolge beim Export weiter verbessert.

11.3 Grundfunktionen

Für das Funktionieren der objektorientierten Laufzeitumgebung sind einige Grundfunktionen maßgeblich.

BuildSubComponentHash (vgl. Anhang Q) erzeugt eine im Vergleich zur Modelldarstellung als Graph oder Liste effizientere Hashtable beliebiger untergeordneter Elemente. Diese dient der Verwendung in nachfolgenden Operationen für diese oder die untergeordneten Komponenten. Die Funktion kann ausgehend von einer beliebigen Komponente *k* für jeden im Modell existierenden Relationstyp eine Hashtable erzeugen, die alle untergeordneten Elemente enthält. Wird in *SubTypeRelationHash* die *Aggregation* als Typ vorgegeben, handelt es sich um Teilkomponenten von *k*, bei *Inheritance* um Spezialisierungen, bei *Instanciation* um Instanzen. Wird die Suchrichtung mit *reverseDirection=true* umgekehrt, handelt es sich um übergeordnete

Komponenten (Aggregation), Generalisierungen (Vererbung) oder die übergeordneten „Instanz-Väter“ beziehungsweise Klassen (Instanziierung).

```
public Hashtable BuildSubComponentHash(Hashtable hashtableUsed, Hashtable  
SubTypeRelationHash, bool doRecursion, bool reverseDirection) {...}
```

Zur Effizienzsteigerung wird unter anderem für die Vererbungsabfrage zwischen zwei Komponenten *k* und *l* nicht ein vollständiger Hash oder eine Liste aufgebaut und abgefragt, sondern mit *IsTransitiveInheritingFrom* direkt eine rekursive Tiefensuche in inverser Vererbungsrichtung ausgehend von *k* durchgeführt und bei Auffinden abgebrochen. So wird die maximale Laufzeit nur erreicht, wenn die Abfrage negativ ausfällt. Die Funktionalität wird beispielsweise mit *IsTransitiveInheritingInstanceOf* ergänzt durch Einbeziehung von Instanzebenen oder nur als unmittelbare Vererbung abgefragt: *isDirectChildOf*.

Wie das Klassendiagramm in Anhang P zeigt, sind für Komponenten unter anderem Methoden für folgende Bereiche vorhanden

- Vererbung und Instanziierung (auch komplexer) Komponenten: *CreateChildByInheritance* / *MakeChildByInheritance*, *CreateInstance*,
- Duplikation (*Duplicate*) und Entfernung von Komponenten,
- Einfügen (*AddRelation*) und Entfernen (*RemoveRelationFromComponent*) von Relationen und
- Verarbeitung von Abläufen und Ereignissen.

Im Code wird noch eine Unterscheidung zwischen allgemeinen und Diagrammkomponenten getroffen, die semantisch durch das Modell zwar nicht mehr vorgegeben ist, jedoch die Umsetzung erleichtert und für den Benutzer eine explizite Differenzierung zwischen Diagrammen (*DiagramComponent*, Spezialisierung von *Component*) und Bestandteilen von diesen (*Component*) ermöglicht. Auch die Zuordnung von Sichten wird so in der Implementierung vereinfacht und in ihrer Komplexität reduziert.

Das externe Regelsystem in *RuleComponent* wurde für den Prototyp nicht umgesetzt, da es mehr für die Modellentwicklung als für die Laufzeitumgebung wichtig ist und ausreichend Stoff für weitere Arbeiten bietet.

11.4 Optimierung und Verteilung

Aufgrund des vollständig objektorientierten Modells können auch für den Relationstyp *Inheritance* Subtypen gebildet werden. Unter Umständen können also hunderte von Vererbungsrelationstypen existieren. Da die Vererbung jedoch auch für die Identifikation der Zugehörigkeit von Relationen und Komponenten zu allen anderen Relations- und Komponententypen verwendet wird, muss in jeder Operation ein Vererbungs-Hash aufgebaut werden.

Um bei dieser hochfrequentierten Operation einen Effizienzgewinn zu erzielen, wird für die Vererbung – prototypisch für häufig einzusetzende Basisrelationen – ein dauerhafter Vererbungs-Hash erzeugt, der von jedem Peer in dessen Instanz der Klasse *ComponentServerBase* angelegt und verwaltet wird. Die Abfrageeffizienz für *IsInheritance()* erhöht sich dabei für *n* existierende Vererbungstypen durch den Hash von $O(n)$ auf $O(1)$, was bei vielen Spezialisierungen der Vererbung im Modell einen auffälligen Effizienzgewinn bringt.

Die vorgesehene **Verteilung von Änderungen** wird bereits berücksichtigt: Alle Veränderungsoperationen, Neueinfügungen und Löschungen aller Modellelemente können durch das jeweilige Element bemerkt und in die Änderungs-Queue geschrieben werden. Die *ModelElementKeys* enthalten dabei auch den zuständigen Server. Ausgehend von dieser Queue können alle Änderungen über das Netz propagiert werden. Ist die Zustimmung oder Änderung

durch einen anderen Server notwendig, weil das Element auf diesem erzeugt wurde, müssen alle vorangehenden internen Modelländerungen versandt und der Server nachfolgend mit dem entsprechenden Request angesprochen werden. Im Zuge der Weiterentwicklung soll allerdings die Zustimmung des Quellservers zu Änderungen durch ein Transaktionskonzept mit lokalen Änderungen ersetzt werden, um eine Abhängigkeit von einzelnen Peers zu vermeiden. Dies ist vor allem für die Ausfallsicherheit wichtig, die ein Warten auf einen ausgefallenen Peer nicht toleriert. Zudem sollen Änderungsbereiche eingeführt werden, die eine unabhängige Verarbeitung unterschiedlicher Prozesse und Prozessinstanzen ermöglichen. Da Teilkomponenten von Prozessinstanzen immer auf dem ausführenden Peer entstehen, ist die Verteilung hier systemimmanent und problematisch.

11.5 Externalisierung von Aufrufen

Neue Komponententypen wie spezielle Templates, Rollen etc. können wie neue Sichten oder Basiskomponenten eine andere oder zusätzliche Interpretation benötigen. Auch hier greift wie beschrieben die Vererbung, so dass neue Typen zunächst von existierenden Typen auch die Behandlung erben. Plug-Ins oder verarbeitungsfähige Handler können hier aber auch zusätzliche Funktionen durch Ergänzung oder vollständiges Überschreiben anbieten.

Wie bei der OOP müssen hier Konstrukte wie „super“ möglich sein, die die Verarbeitung des allgemeineren Vater-Typs an geeigneter Stelle aufrufen. So müssen nur zusätzliche Behandlungen implementiert werden, was eine redundante Implementierung vermeidet.

Es wird immer die typnächste Behandlungsroutine aufgerufen, so dass spezialisierte Interpretationen allgemeine überschreiben. Falls keine spezielle Behandlung vorliegt, wird die nächst-allgemeinere genutzt.

Die Verarbeitung von erweiterten Typen wie Spezialisierungen der Basistypen wird noch im Laufzeitsystem gekapselt, das im Prototyp die Verarbeitung intern löst. Durch das Konzept der Handler ist jedoch später auch für die Modellverwaltung eine Auslagerung der Behandlungsroutinen und das Hinzufügen neuer möglich. Die Informationen würden dann nicht mehr intern verarbeitet, sondern an den externen Handler weitergegeben.

Während die Interpretation von Relationen extern erfolgen kann, muss die Überprüfung der Graphstruktur ebenso wie andere das Gesamtmodell betreffende Konsistenzprüfungen modellintern durchgeführt werden.

Die **Ausführung von Prozessschritten auf der Instanzebene** ist dagegen weitgehend externalisierbar. Anhand der Methodentabelle kann jedem Typ und seinen Spezialisierungen eine Behandlungsroutine zugeordnet werden. Dieser Routine werden alle der Komponente zur Verfügung stehenden Parameter und Kontextinformationen im XML-Format übergeben.

Nach erfolgter Ausführung werden gegebenenfalls Ergebniswerte ebenfalls im XML-Format zurückgeliefert. Ob die Ausführung dabei lokal oder verteilt erfolgt, ist für das Modellsystem transparent, sobald es das Messaging-System für die Übermittlung nutzen kann.

Im OMICRON Prototyp werden alle externen Aufrufe lokal bearbeitet, da der Fokus der Arbeit auf dem Modell und seiner Animation liegt.

12 Evaluation

Die qualitative Bewertung anhand der Anforderungen soll den Erreichungsgrad aufzeigen. An diese schließt sich eine quantitative Evaluation der Laufzeitumgebung in Bezug auf die Leistungsfähigkeit des Systems mit Hilfe automatisierter Testfälle an.

12.1 Überprüfung der Anforderungserfüllung

Die vorliegende Arbeit stellt ein generalisiertes Grundkonzept für objektorientierte Prozesse mittels Integration von Ablauf- und Objektorientierung vor. Eine Optimierung für die Produktionstechnik soll dabei eine Integration mit allen Prozessen und relevanten Daten ermöglichen. Die Umsetzung der in Kapitel 3.4 beschriebenen Anforderungen wird im Folgenden überprüft und der Erreichungsgrad visualisiert.

A1 Integration von Ablauforientierung (Dynamik) und Typ-/Objektorientierung (Statik)

Alle Bestandteile von Abläufen – sowohl Prozessschritte, Ein- und Ausgänge als auch alle Relationen – sind in OMICRON typisiert und in der Modellstruktur mit Hilfe der Vererbung und Instanziierung integriert. Eine Kapselung von Abläufen in typisierten Komponenten ermöglicht die Schaffung einer beliebigen Aggregations- und Vererbungshierarchie, ebenso wie die semantikbasierte Konkatenation zu komplexen Abläufen mit Hilfe typisierter Ein- und Ausgänge.

A2 Integratives Gesamtmodell

Das OMICRON Modell ermöglicht eine typologische Abdeckung der gesamten Produktionsabläufe, Artefakte und Bestandteile eines Produktionssystems im selben Gesamtmodell. Interaktionen, Daten und Abläufe können ebenso wie die Klassifikation externer Artefakte automatisiert ausgewertet werden. Zudem kann die Struktur und Typologie dem Geschäftsprozess angepasst werden, so dass eine integrierende Schnittstelle beispielsweise für ERP entwickelt werden kann. Allerdings ist eine Erprobung der Auswertung und Kompatibilität erst mit einem vollständig modellierten Produktionssystem und dessen Schnittstellen möglich. Eine Anpassung und Beschränkung auf bestimmte Teilbereiche, Unternehmen und Produktionstypen ist dabei immer notwendig, um das Modell erstellen zu können.

A3 Hierarchische Modularisierung und Verwendung von Komponenten

Das Aggregations- und Schnittstellenkonzept von OMICRON ermöglicht die vollständige Modularisierung und Hierarchisierung des Modells. Spezialisierungen der Aggregationsfunktion ermöglichen eine – auch überschneidende – Partitionierung nach Aspekten, Projekten, Produkttypen etc., die typisiert und in die Vererbungshierarchie eingebunden sind. Alle Teilmodelle sind somit vollständig typisiert und in einfacher Weise wiederverwendbar. Ein einmal vorhandener (Teil-)Prozess kann beliebig wiederverwendet und spezialisiert werden, so dass eine einfache Variantenbildung und Spezialisierung – beispielsweise für Sondermodelle – möglich ist. Die Selbstbeschreibungsfähigkeit der Komponenten ermöglicht eine automatische Auswertung des Modells, beispielsweise zur Auffindung kompatibler Teilprozesse, die dann polymorph verwendet werden können. Einheitliche Konzepte auf allen Modellebenen stellen sicher, dass die Laufzeitumgebung das vollständige Modell interpretieren kann und eine einfache Erweiterung möglich ist.

A4 Konsistentes Typ- und Spezialisierungskonzept

Ein konsistentes Typkonzept ist die Voraussetzung für eine semantisch korrekte Variantenbildung und Spezialisierung. Hier sorgt die Typunabhängigkeit der Aggregate von den aggregierten Komponenten dafür, dass das „Ganze mehr als die Summe der Teile“ ist: Ein Verbund vieler Teile unterschiedlichen Typs wird nicht als Verbundtyp dieser Teile („Teilesammlung“) sondern mit einem eigenständigen Typ klassifiziert, der die Bedeutung der Teile als Ganzes wiedergibt („Werkzeugmaschine“).

Für Spezialisierungen besteht dagegen eine Typabhängigkeit von übergeordneten Komponenten, die erst Polymorphie und semantische Beziehungen ermöglicht. Hierzu sind alle Modellelemente typisiert und ermöglichen die Bildung und Verwendung von semantischen Typ-Varianten und Spezialisierungen. Durch die Mehrfachvererbung können komplexe Prozesse und Komponenten erstellt werden, die die Semantik ihrer Teilschritte beziehungsweise Teilkomponenten enthalten: „Teil A erodieren“ und „Teil B fräsen“ können so zu einem Vorgang „Teil A und B fertigen“ zusammengefasst werden, der semantisch immer noch die Fertigungsverfahren für Teil A und B erhält, jedoch für die Modellierung hiervon abstrahiert.

A5 Laufzeitkonzept

Das Animationskonzept für Prozessmodelle zur Laufzeit orientiert sich an der Semantik der Aktivitäten in UML2, wurde jedoch modifiziert und durch eigene Konzepte ergänzt. Dies stellt sicher, dass jeder in OMICRON modellierte Prozess typpgesteuert animiert werden kann. Die Taktung alterniert zwischen der Aktivierung von Flussrelationen und von Komponenten. Die praktische Effizienz des Typ- und Laufzeitmodells hinsichtlich der internen Antwortzeiten (A5.3) zeigt Abschnitt 12.2.

Soweit keine unauflösbaren Abhängigkeiten wie existierende Instanzen bestehen, ermöglicht OMICRON auch eine dynamische Modelladaptation zur Systemlaufzeit. Zudem ist eine polymorphe Verwendung von Spezialisierungen möglich, so dass in einem allgemeinen Fertigungsprozess immer kompatible Teilprozess-Varianten eingesetzt werden können.

A6 Laufzeitentwicklung von objektorientierten Prozessen

Einen Schritt weiter geht die Laufzeitentwicklung. Sie ermöglicht unvollständige Instanzen während der Prozessmodellierung bis zum letzten modellierten Schritt zu animieren. So können „just-in-time“ immer die nächsten Schritte im Kontext modelliert werden, während der Fertigungsprozess prototypisch durchlaufen wird.

Für eine typbasierte Auswahl und Überprüfung der verwendeten Prozessschritte steht bei der Modellierung das gesamte OMICRON Modell mit allen im Unternehmen vorhandenen, für den Modellierer sichtbaren Komponenten zur Verfügung. Änderungen an Teilprozessen werden dabei automatisch über die Vererbungshierarchie an die spezialisierten Komponenten weitergegeben (Kaskadierung). Die Existenz von Instanzen schränkt die Laufzeitentwicklung gegenüber der separaten Prozessmodellierung, bei der keine Instanzen existieren können, wie gezeigt auf eine Fallunterscheidung ein. Dies ist eine Auswirkung des objektorientierten Paradigmas auf die Laufzeitmodellierung und beschränkt daher alle (zukünftigen) Ansätze dieser Art.

A7 Flexibles Modell

Das Meta-Meta-Modell ermöglicht eine freie Umsetzung des OMICRON Modells in Software. Nur ein minimales Basismodell fließt in die Architektur ein und wird durch Modellerweiterungen ergänzt. Dabei ist OMICRON darstellungsunabhängig, da beliebige, differenzierte Sichten an die Teilmodelle gebunden werden können. Die Modellsemantik bleibt hiervon unberührt und ermöglicht die Modellierung teilmodellübergreifender Beziehungen. Im Gegensatz zu festgelegten Notationen und Modelltypen wie BPMN/BPML können unterschiedliche Metamodelle erstellt werden, die zudem eine Integration unterschiedlicher Modellierungstechniken in einem übergeordneten Metamodell erlauben. Die Ausdrucksmächtigkeit des Modells liegt klar über der von EPK und vergleichbaren Modellen. Kapitel 7 zeigt zudem, dass die Konzepte der neuen UML2 Aktivitäten – teilweise mit Anpassungen an die Anforderungen – direkt umgesetzt werden können.

A8 Integriertes Interaktionskonzept

OMICRON ermöglicht auch die Ausführung von Prozessen, für deren Start oder Ablauf noch Informationen fehlen, beispielsweise wenn eine Teilnummer nicht im Model automatisch weitergeleitet wird, sondern für diese Teilart abgelesen und eingegeben werden muss. Das Laufzeitsystem kann hier direkt erkennen, dass implizit eine Interaktion notwendig ist. Häufig

sollen auch bestimmte Interaktionen wie beispielsweise Freigaben explizit in den Prozess integriert werden. OMICRON sieht hier ein ähnliches Konzept wie für die implizite Interaktion vor, jedoch sind bei der expliziten Interaktion die Interaktionsroutinen und deren Ein- und Ausgangstypen bereits definiert. Beide nutzen jedoch dasselbe Grundkonzept, was die Integration erleichtern soll.

Das Konzept wurde ausgearbeitet und im Modell integriert. Da die Interpretation externalisiert ist, werden allerdings erst mit der Implementierung eines plattformspezifischen, externen Interpreters und Front-Ends die Interaktionen zur Verfügung stehen, deren Ergebnisse in das Modell zurückgespielt werden sollen.

A9 Unterstützung von Verteilung und Generierung

Die vorgesehene Verteilung erfolgt über das Semantic-Messaging-Konzept, das einen Austausch von klassifizierten Modellfragmenten ermöglicht. Modellbestandteile können hierzu in ein XML-Format exportiert werden, das zwischen Peers ausgetauscht werden kann. OMICRON sieht vor, dass das Zielsystem partitioniert werden kann, so dass jede Partition nur Teile des Gesamtmodells enthält. Jede Partition und jeder Peer müssen dabei nur das Meta-Metamodell und die verwendeten Metamodelle enthalten. Prozessinstanzen können daher beispielsweise ausgetauscht werden und nur auf den derzeit ausführenden Peers existieren. Diese Verteilbarkeit eignet sich gerade auch für service-orientierte Architekturen (SOA). Das unter A8 beschriebene Interaktionskonzept ermöglicht basierend auf dieser Verteilbarkeit auch die dezentrale Erzeugung von Benutzungsschnittstellen, da jeder Peer die vorhandenen Modellbestandteile interpretieren und aus dem Modell- und Ablaufkontext Interaktionen erzeugen kann.

Nr.	Anforderungsgruppe	Bewertung
A1	Integration von Ablauforientierung (Dynamik) und Typ-/ Objektorientierung (Statik)	●
A2	Integratives Gesamtmodell	◐
A3	Hierarchische Modularisierung und Verwendung von Komponenten	●
A4	Konsistentes Typ- und Spezialisierungskonzept	●
A5	Laufzeitkonzept	●
A6	Laufzeitentwicklung von objektorientierten Prozessen	◐
A7	Flexibles Modell	●
A8	Integriertes Interaktionskonzept	◐
A9	Unterstützung von Verteilung und Generierung	●

Legende: ● voll erfüllt, ◐ weitgehend erfüllt, ◑ teilweise erfüllt, ◒ kaum erfüllt, ○ nicht erfüllt

Tabelle 19: Anforderungserfüllung anhand der Anforderungsgruppen

Die geforderte Modellunterstützung von transformativen und generativen Verfahren wird durch die nun folgende Einordnung des Verfahrens beschrieben.

12.2 Klassifikation möglicher Transformationsansätze

Wie in Anforderung A9 beschrieben soll das Konzept eine möglichst große Flexibilität bei der Transformation und Generierung von Interaktionen, Artefakten und Programmcode ermöglichen.

OMICRON stellt zunächst ein Modellierungskonzept dar, das unterschiedliche Ansätze ermöglicht. Anhand des in [CzarneckiHelsen 2003] vorgestellten Klassifikationsschemas für Transformationen in der modellgetriebenen Entwicklung wird der OMICRON-Ansatz eingeordnet, indem die Klassifizierung anhand der Eignung von OMICRON für unterschiedliche Transformationsansätze erfolgt.

Kriterium	Ausprägung	Wert
Variablen	Variablen intern	++
	Variablen extern	O
Patterns	Patterns textuell	++
	grafische Muster	O
Logic	statisch / relational	++
	imperative Konzepte	+
	Direct Manipulation	-
Model-to-Model	relational	O
	Graph-Transformation	++
	strukturgetrieben	+
	hybrid	O
	Visitor-basiert	O
Model-to-code	Template-basiert	++
	Code-to-Code	-
Typisierung Variablen bzw. Patterns	untypisiert	-
	syntaktisch typisiert	+
	semantisch typisiert	++
Transformationsmodelle	syntaktische Separation	O
	Bidirektionalität	-
	Regelparametrisierung	O
	Zwischenmodelle	+
Rule Application Scoping	Quellmodell	++
	Ziel	+/-
Generierung in Modell	selbes	+
	neues	++
	beides	O

Kriterium	Ausprägung	Wert
Modellveränderung	destruktiv	-
	Erweiterung	++
Anwendungsstrategie für Regeln	deterministisch	++
	nichtdeterm.: ein Ziel	+
	nichtdeterm.: nebenläufig	--
Rule Scheduling	frei	--
	implizit	+
	explizit	++
Regelauswahl	explizit	++
	nichtdeterministisch	+
	interaktiv	O
Regelorganisation	Modularität	++
	Wiederverwendung	++
	Strukturierung	++
Tracability Links	intern	++
	extern	+
Richtung bzw. Synchronisation	unidirektional	++
	gemischt	O
	bidirektional	-
Generierzeitpunkt	Entwicklung	O
	Laufzeit vorher	+
	Laufzeit im Prozess	++
Zielformat	Code	O
	Artefakte	++
	externe Modelle	+
	interne Modelle	++

Legende Eignung: ++ = optimal, + = gut, O = durchschnittlich, - = schlecht, -- = untauglich bzw. nicht möglich

Tabelle 20: Tauglichkeit des Modellierungsansatzes für generative Verfahren nach [CzarneckiHelsen 2003]

Eine Erläuterung der Kriterien, basierend auf [CzarneckiHelsen 2003] enthält Anhang O.

12.3 Effizienz des Ansatzes

Die qualitative Leistungsfähigkeit des Systems wurde im Hinblick auf die Anforderungserfüllung und speziell relevante Aspekte wie die Erweiterungsfähigkeit, Flexibilität und Generizität somit nachgewiesen.

Für den Einsatz in der Praxis ist zudem ausschlaggebend, ob auch die **Effizienz des Ansatzes** für die Ausführung zur Laufzeit nachgewiesen werden kann (vgl. Anforderung A5.3). Hierzu wird die prototypische, systemtechnische Umsetzung des Laufzeitmodells untersucht.

Das Einfügen von Komponenten in das Modell stellt einen limitierenden Faktor dar. Da das Einfügen neuer Komponenten in eine Instanz die Hauptaufgabe des Laufzeitsystems, neben der Weiterschaltung im Ablauf, darstellt, wurde die **Einfügeoperation** als Basis für die folgenden Messungen eingesetzt. Um ein komplexeres System zu simulieren, wurde die Ausführung im Debug-Modus der Entwicklungsumgebung gestartet. Als System diente ein auf Windows XP laufender Intel Core2Duo E6300 mit SATA-RAID-System und 2 Gigabyte Hauptspeicher.

Das mit dem Basismodell initialisierte Laufzeitsystem wurde durch den in Anhang R gezeigten Code angesteuert, um ein umfangreiches Modell zu erzeugen.

Die zufällige Generierung eines umfangreichen, algorithmisch verschachtelten Komponenten-Relationen-Modells mit Vererbung und Instanziierung wurde basierend auf dem Meta-Meta-Modell erzeugt.

Die hierbei erfolgte Messung der Aufbauzeit (vgl. Code in Anhang R) lieferte eine weitgehende Resistenz gegenüber der Verschachtelungstiefe. Die Wahrscheinlichkeit r für eine Subtypbildung und die Erhöhung der untergeordneten Komponenten mit Hilfe von i zeigten keine Auswirkungen. Allerdings zeigt sich, dass mit zunehmender Modellgröße nicht nur eine lineare Steigerung auftritt.

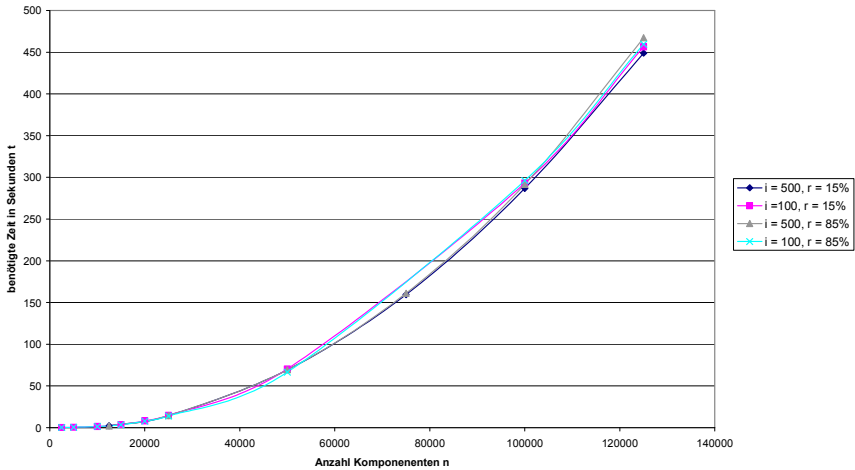


Abbildung 77: Effizienz der Einfügeoperation im Komponentenmodell des Laufzeitsystems

Der Export eines vollständigen Modells als XML-Daten wurde mit variiertem i_2 und konstantem $i=100, r=85\%$ durchgeführt. Auch hier zeigt sich mit wachsender Modellgröße ein überproportionaler Rechenzeitbedarf.

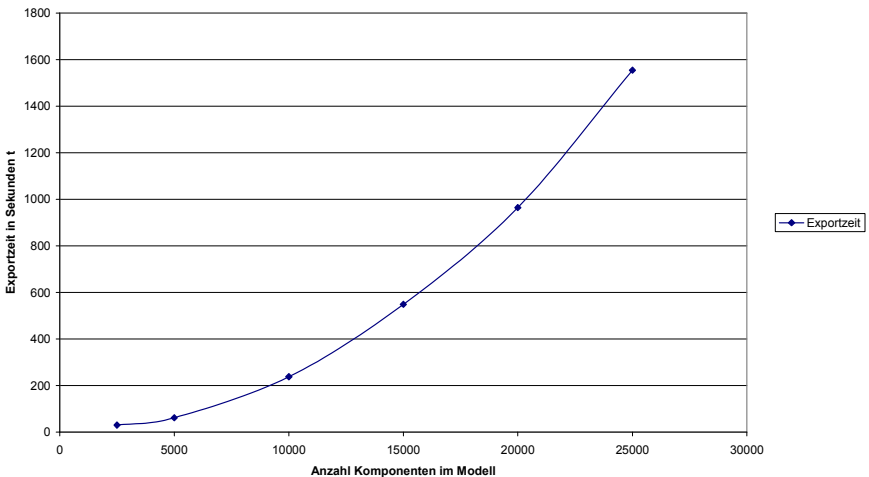


Abbildung 78: Rechenaufwand für den Modellexport – Zeit in Sekunden, abhängig von der Modellgröße

Die im Zuge dieser Durchläufe ermittelten Dateigrößen verlaufen wie zu erwarten linear. Da nur einfache RumpfkompONENTEN ohne weitere Inhalte erzeugt werden, ist der Dateiumfang für die Komponenten- und Relationenzahl auch insgesamt vergleichsweise gering.

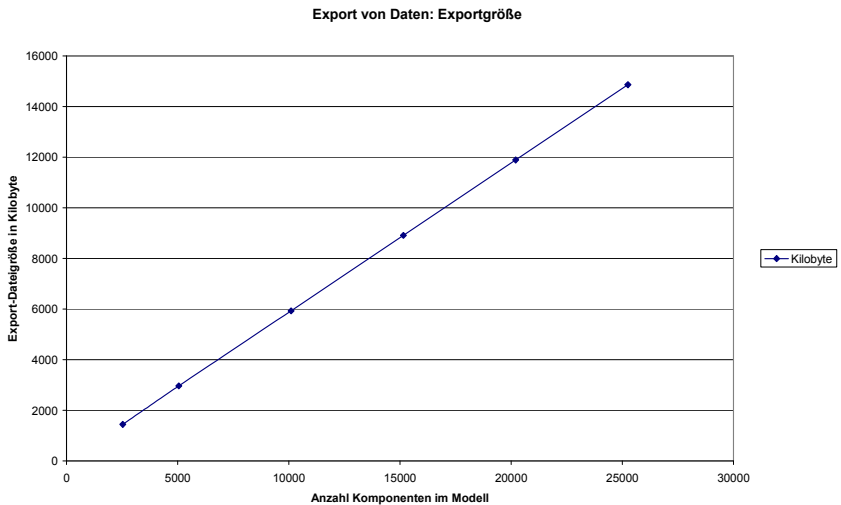


Abbildung 79: Exportdateigröße in Abhängigkeit von der Anzahl Komponenten im Modell

Auffällig ist, dass die Prozessorauslastung beim Export bei beiden Kernen fast gleich hoch ist, was auf nahelegt, dass auf einem Einzelprozessorsystem nur mit etwa 55% der Exportleistung zu rechnen ist. Für den Prototyp wurde kein Multithreading eingeführt, so dass die Modellmanipulationen nur auf einem Kern ausgeführt werden und die Zweikern-Architektur hier keinen Vorteil bietet. Der mittlere Datendurchsatz ist so gering, dass das RAID-System kaum belastet wird und daher keinen nennenswerten Beitrag zur Performanz leistet.

Ein zusätzlicher Vergleich der Leistungen bei der (rekursiven) Überprüfung der transitiven Instanziierungs- und Vererbungsbeziehung zwischen zwei Komponenten zeigt das Potenzial einer Performanzverbesserung der Laufzeitumgebung (siehe Anhang R). Während die transitive Instanziierung für alle übergeordneten Komponenten einzeln algorithmisch geprüft wird, kann bei der Vererbung der InheritanceHash direkt eingesetzt werden, so dass in $O(1)$ über die Relationszugehörigkeit zur Vererbung entschieden werden kann. Bei Verwendung desselben Messaufbaus im Code wird bereits bei einer hohen Schachtelungstiefe ($i2:i = 10:5000$) ein Geschwindigkeitsvorteil um den Faktor 29 möglich, bei geringerer Schachtelungstiefe ($i2:i = 1000:50$) schon über 1200. Die Tabellen sind ebenfalls im Anhang R zu finden.

Die Werte lassen eine hohe Effizienz von typologischen Fragestellungen (Vererbung) erkennen, die zeigt, dass Modell und System auf diese Fragestellungen hin ausgerichtet und optimiert wurden.

Veränderungsoperationen sind um ein vielfaches langsamer als Typabfragen, da entsprechende Überprüfungen hinsichtlich Metamodell-Konsistenz stattfinden. Auch hier ist jedoch sowohl die Modellierung, als auch die Prozessanimation bereits auf durchschnittlichen Systemen so schnell, dass die Verteilung und Persistenz den limitierenden Faktor darstellt.

12.4 Evaluation der Formalisierung und Automatisierbarkeit

[Müller 2007] S. 34 ff. liefert **Kriterien für die Evaluation von Prozessmodellierungsmethoden** hinsichtlich der Formalisierung und der daraus hergeleiteten Automatisierbarkeit.

Kriterium	Erfüllung	EG
Die Korrektheit des Modellierungskonzepts muss die eindeutige Identifikation unzulässiger Modelle gewährleisten.	Dies wird in OMICRON durch die axiomatische Definition des Meta-Metamodells, die Metamodellierung sowie die Instanziierungs- und Vererbungssemantik gewährleistet. Der einzige Problempunkt hierbei ist die Verwendung von objektorientierten Spezialisierungen anstelle des ursprünglich vorgesehenen Komponententyps. Hier musste das Modell „aufgeweicht“ werden, da eine Beschränkung auf den entscheidbaren, von der Superklasse vorgegebenen Rahmen die Flexibilität bei der Spezialisierung zu stark eingeschränkt hätte.	●
Eine hinreichende Definition der Modellierungskonstrukte sichert die Vollständigkeit , ohne die keine Ableitung von objektiven Modellierungsregeln möglich ist.	Hierzu wurden alle Basis-Modellbestandteile im Rahmen dieser Arbeit definiert und die für diese geltenden Regeln beschrieben. Beliebige weitere Modellbestandteile werden auf Basis dieser Elemente und Regeln erzeugt, so dass ihre Definition und Verarbeitung ebenfalls jederzeit herzuleiten ist, jedoch bei Bedarf spezialisiert werden kann.	●
Einheitlichkeit und Redundanzfreiheit werden durch eine klare, redundanzfreie Definition gewährleistet, die alle Konzepte mit denselben Ausdrucksmitteln behandelt.	Die Vereinheitlichung von Diagramm und Modellkomponenten ermöglicht die homogene Anwendung der Definitionen auf alle Komponenten im System und geht damit über herkömmliche Modellierungssysteme hinaus, die für jede Abstraktionsebene unterschiedliche Regeln anwenden.	●
Die Wiederverwendbarkeit dient der Komplexitätsreduktion durch Kapselung und der Steigerung der Modellqualität.	Die Wiederverwendbarkeit mit zusätzlicher Adaption durch Vererbung ist ein grundlegendes Konzept von OMICRON und wurde bereits oben detailliert beschrieben.	●
Um die Wartbarkeit zu gewährleisten müssen Änderungen möglichst lokal eingegrenzt und die Komplexität durch Partitionierung reduziert werden.	Das OMICRON Modell wurde auf eine dynamische Adaption zur Laufzeit ausgelegt, so dass Veränderungen von vorneherein auf das notwendige Maß beschränkt werden – beispielsweise Prozessänderungen bei existierenden Prozessinstanzen. Wo Änderungen möglich sind, werden diese möglichst frei von Seiteneffekten und Inkonsistenzen vorgenommen, beispielsweise durch Subklassenbildung, wobei Änderungen über die Vererbungshierarchie propagiert werden und so eine einfache Wartung von komplexen Prozesshierarchien ermöglichen.	●
Die Erweiterbarkeit von Modellen, Klassifikationen, Ereignissen, Datentypen und Abläufen im Rahmen der Modellsemantik darf selbst zur Laufzeit nicht eingeschränkt werden.	Sie ist durch das generalisierte Meta-Metamodell umfangreicher als in vergleichbaren Ansätzen. So können neue Modelltypen und Sichten entwickelt und bestehende Elemente spezialisiert und zusammengefasst werden. Durch das Konzept des Meta-Metamodells können beliebige Modelltypen (Metamodelle) erzeugt werden, die so wesentlich mehr Spielraum zur Verfügung stellen als die Verwendung eines vordefinierten Modelltyps.	●
Eine gute Ausdrucksmächtigkeit setzt die Abbildbarkeit aller relevanten Elemente und Abläufe einer Domäne voraus.	Dies wird durch die Flexibilität des OMICRON Modells gewährleistet. Dabei kann die Genauigkeit sogar auf den Einsatzzweck abgestimmt und durch Detaillierung sogar im Laufe der Modellierung vergrößert werden. Die Entwicklung domänenspezifischer Sprachen (DSL) ermöglicht dabei auch eine Spezialisierung von Modellen für den Anwendungsbereich.	●
Wichtig für die praktische Anwendbarkeit in der Produktion ist die Operationalisierbarkeit .	Das Modellierungssystem soll nicht nur der Kommunikation dienen, sondern direkt für die Ausführung der Prozesse und Speicherung von Daten und Zuständen eingesetzt werden können. Ein Ablaufmodell in OMICRON kann deshalb zur Laufzeit ausgeführt werden und erzeugt alle definierten Aktionen, Ereignisse, externen Aufrufe und Datenflüsse selbständig im Rahmen der Modellvorgaben. Die Ausführung setzt allerdings jeweils ein konkretes, unternehmens- und fallspezifisches Modell voraus.	●

Legende: Erfüllungsgrad (EG) – ● voll erfüllt, ● weitgehend erfüllt, ● teilweise erfüllt, ● kaum erfüllt, ○ nicht erfüllt

Tabelle 21: Evaluation der Formalisierung und Automatisierbarkeit nach den Kriterien von [Müller 2007]

12.5 Diskussion der Ergebnisse

Wie bei allen integrativen Modellierungs- und Steuerungssystemen besteht die Gefahr des „Second System Effect“⁴³, der bei allen holistischen Lösungen auftritt, wenn diese die Aufgaben mehrerer Systeme übernehmen sollen. Häufig führt dies zu parallelen, inkonsistenten Altsystemen, die trotz einer möglichen Integration nicht abgeschaltet werden. Dieser Effekt wird durch die servicebasierte Anbindung und das Peer- und Messaging-System bereits abgeschwächt, da eine einfache Einbindung von proprietären Systemen möglich ist.

Auch soll und kann das Konzept nicht die Funktionalität etablierter Fertigungssoftware verdrängen. Jedoch muss für eine Integration von gekapselten Fremdsystemen ein Peer- oder Message-Gateway zur Verfügung gestellt werden, das ergänzende Systeme integriert. Dies führt zu zusätzlichem Integrationsaufwand und begünstigt teilweise wiederum den Second System Effect durch Neuentwicklung statt Integration existierender Lösungen.

Die Arbeit verfolgte einen Top-Down-Ansatz zur Entwicklung eines grundlegenden Meta-Metamodells, das alle geforderten Eigenschaften mitbringt und für die praktische Anwendung über umfangreiche Konzepte zur Anpassung und Erweiterung mit geringem technischen Aufwand verfügt.

Dies bringt die Nachteile einer generischen, holistischen Lösung mit sich, unter anderem die Notwendigkeit einer umfangreichen Modellanpassung und -erweiterung. Allerdings stoßen von engbegrenzten Teilproblemen der Praxis ausgehend erzeugte Modellierungsverfahren bei Erweiterungen und Integrationen schnell an die Grenzen, was die eingangs beschriebene Systemheterogenität in der Produktion weiter verschärft. Hier kann OMICRON einen stringenten Ansatz liefern.

Die Objektorientierung in OMICRON wurde daher vom Klassen-Objekt-Modell der Objektorientierung aus in Richtung eines generischen Meta-Metamodells weiterentwickelt. Daher erzielt es dieselben Vorteile in der Produktion wie die in neuerer Zeit in der Metamodellierungs-Community vorgeschlagenen Ansätze zur Weiterentwicklung der objektorientierten Modellierung im Softwarebereich: *„[...]providing a single unified concept enables unbounded logical metalevels without implying changes to the superstructure whenever another level is needed;“* [AtkinsonKühne 2002]

Für aktuelle Ansätze von Mass Customization und Build-to-Order hat OMICRON große Vorteile hinsichtlich Varianten und Prozessflexibilität. Gleichzeitig schränkt das Typsystem den „Wildwuchs“ von Teilen ein und fördert die Teile- und Prozessstandardisierung aufgrund der Typologie. Statt flachen Teile-Katalogen mit Duplikaten [Anderson 2003] kann eine hierarchische Teileklassifizierung nach Art und Verwendung erfolgen, die die Teilesemantik mit berücksichtigt und auch im Sinne der Qualitätssicherung und Zertifizierung weitreichende Vorteile bringt.

Das Ziel, schnelle Rekonfiguration und Adaptierbarkeit an spezielle Produkte zu ermöglichen, führt auch zur Verwirklichung der Anforderung A4.5: Produkte und auch die sie generierenden Prozesse können verschmolzen, Varianten von Prozessen und Produkten mit minimalem Aufwand in das System integriert werden. Die Mehrfachvererbung ermöglicht die Produktion von kombinierten Produkten durch eine Prozessverschmelzung mit erhöhter Fertigungstiefe. So können, basierend auf existierenden Prozessen und Produkten, neue, komplexere Produkte und Sondermodelle erstellt werden.

⁴³ Second System Effect [Brooks 1975]. Häufiger Komplexitätseffekt einer neuen Lösung oder Versionen für bewährte kleine Teilsysteme, die ausfunkt und „Lösungen für alles“ durch Feature-Überladung produziert, die letztlich die praktische Anwendbarkeit gefährden.

Selbst Änderungen an einem Teilprodukt beziehungsweise -prozess können direkt auf das kombinierte Produkt übertragen werden. Dies ermöglicht auch eine schnelle Übertragung von neuen (beispielsweise rechtlichen oder sicherheitstechnischen) Veränderungen auf die existierenden Abläufe und Produkte.

Wie bei der Einführung von ontologiebasierten oder ERP-Systemen ist für die Einsetzbarkeit des Systems eine exakte Modellierung der Zusammenhänge und Besonderheiten des Anwenderunternehmens für den Erfolg ausschlaggebend.

Eine Erweiterung, Integration und umfangreiche, spezifische Grundmodellbildung für die produzierenden Unternehmen sind daher erforderliche weitere Schritte für eine erfolgreiche Anwendung in der betrieblichen Praxis.

13 Zusammenfassung

In dieser Arbeit wurde für dynamische, flexible Prozesse in der Produktion als grundlegendes Defizit die fehlende Integration von Abläufen und objektorientierten Modellen identifiziert. Mit dieser gehen fehlende Aussagekraft, Modellintegration, Animationsfähigkeit, Dezentralisierbarkeit und Genierungsseignung einher.

Als Kern des Ansatzes wurde daher die Integration von objektorientierten und ablaufzentrierten Modellen angestrebt. Hierzu wurden Kernkonzepte der Objektorientierung identifiziert und auf ein hierfür geschaffenes, allgemeines Komponenten-Relationen-Modell angewandt. Speziell für komplexe Komponenten, wie sie in praktischen Prozessen auftreten, waren die Aussagen und Implikationen der Objektorientierung zu überprüfen und teilweise weiterzuentwickeln. Die objektorientierten Kernkonzepte Vererbung und Instanziierung sind nun auch für umfangreiche und tief geschachtelte Prozesse in dynamischen Umgebungen anwendbar. So können die Vorteile der Objektorientierung wie Typsicherheit und flexibler Einsatz von Prozessvarianten durch Polymorphie und Variantenbildung im laufenden Betrieb zur Verfügung gestellt werden.

Speziell die Laufzeitorientierung, welche Änderungen, Erweiterungen und generative Konzepte mit bereits bestehenden Prozessinstanzen ermöglichen soll, führt zu unterschiedlichen Restriktionen und zusätzlichen Mechanismen, wie der Abklärung der Durchführbarkeit und der Seiteneffekte von Änderungen, die ebenfalls beschrieben wurden.

Es wurde zudem ein formal beschriebenes Meta-Metamodell erstellt, das die Erweiterung und Ergänzung von Metamodellen ermöglicht und unter anderem die Konzepte der Instanziierung, Aggregation und Vererbung auf eine beliebige Anzahl von Instanzebenen verallgemeinert.

Mit dem beschriebenen Gesamtmodell und Laufzeitkonzept ist es nun möglich, in einer bereits laufenden Fertigungsumgebung Prozesse zu erstellen, zu verändern und auszuführen. Dabei werden alle Aspekte des Systems vollständig objektorientiert beschrieben und damit einer semantischen Auswertung zugänglich gemacht. Von standardisierten Prozessschritten, über schnelle Variantenbildung bis hin zu einer Standardisierung von Teilen und Betriebsmitteln, kann so eine vollständige Integration in das Gesamtmodell und Ablaufkonzept erfolgen.

Eine schnelle Rekonfiguration der Fertigung wird damit ebenso unterstützt wie modellbasiert individualisierte Produkte, für die Prozesse zur Laufzeit angepasst und variiert werden können.

Durch das flexible Gesamtmodell können neue Modelltypen, Modelle und Klassifikationen jederzeit hinzugefügt werden, um neuen Anforderungen an die Fertigung Rechnung zu tragen. Für die Weiterentwicklung hin zur Verteilbarkeit war das Ziel, eine dezentrale Fertigungsinfrastruktur zu ermöglichen, wie sie im Rahmen von Betreibermodellen und virtuellen Unternehmen, aber auch im Hinblick auf Daten- und Ausfallsicherheit, notwendig wird.

Durch die Evaluation konnte gezeigt werden, dass das Konzept die gestellten Anforderungen erfüllt und viele neue Möglichkeiten eröffnet. Wie für viele, sich stark von existierenden Grundkonzepten unterscheidende Ansätze, ist allerdings sowohl eine umfangreiche Umsetzung in Bezug auf die Modellbildung notwendig, als auch weiterer Forschungsbedarf für die Integration mit existierenden Ansätzen und Systemen gegeben.

14 Ausblick

Die vorliegende Arbeit zeigt, dass eine semantisch korrekte Integration von Prozessmodellierung und Objektorientierung möglich ist und für Prozesse in der Produktion erhebliche Vorteile für die Variantenbildung und Flexibilisierung im Hinblick auf eine kunden- und auftragsindividuelle Produktion (Mass Customization) mit sich bringt. Die Konzentration auf die grundlegende und umfangreiche Modellbildung als Ausgangsvoraussetzung für eine Weiterentwicklung des Themengebiets wirft naturgemäß viele neue Fragestellungen auf.

Für die User Interface Generierung, basierend auf den im Modell verfügbaren Informationen, wurden mit der impliziten und expliziten Interaktion bereits Voraussetzungen geschaffen. Allerdings sind generative Ansätze aus den 1990er Jahren mittlerweile teilweise in den Hintergrund getreten oder in anderen Domänen wie Web-Engineering und Content-Management aufgegangen. Mit User Interfaces für Service-orientierte oder verteilte Architekturen ist, wie bereits in den Grundlagen dargelegt, ein neues Forschungsfeld entstanden, das der durch Internet-Technologien ausgelösten Verteilung auch im Produktionsumfeld Rechnung tragen soll und viele neue Fragen aufwirft.

Die Anpassung existierender Prozessinstanzen an den geänderten Prozess – unter anderem die Migration von „non-compliant“ Instanzen mit Hilfe eines *Compliance Graphs* – beschreibt [Rinderle 2004] ausführlich, zusammen mit existierenden Konzepten, allerdings nicht für ein objektorientiertes Metamodell. Die Übertragung dieser Konzepte auf das OMICRON Modell ist sicherlich ein spannendes Thema für die Zukunft.

Im INT-MANUS Forschungsprojekt wurden die Ergebnisse der Arbeit in das dezentrale Produktionsüberwachungs- und -steuerungssystem integriert, um flexible Prozesse abzubilden. Sowohl für die dezentrale Verteilung und Nachrichtenbasierung eines solchen semantischen Produktions-Workflowsystems als auch für die dezentrale Verwendung objektorientierter Modelle in Prozessen besteht weiterer Forschungsbedarf. Semantic Messaging zur dezentralen Kommunikation und Modellausführung ist hier ebenso ein beherrschendes Thema, wie Semantic Web Services in der Produktion und eine umfangreiche Beschreibung und Steuerung der Produktion mit semantischen und objektorientierten Mitteln unter dem Stichwort „Produktion 2.0“ [Schlegel et al. 2007].

Die Erstellung domänenspezifischer semantischer Modelle beziehungsweise Ontologien für produzierende Unternehmen stellt neben weiteren Implementierungsarbeiten eine Grundvoraussetzung für die Weiterentwicklung des Ansatzes dar. Hierzu gehören auch weitere Import- und Exportmöglichkeiten [Grose et al. 2002] wie XMI [OMG 2005c].

Die Planung und Optimierung der Abläufe im verteilten Produktionssystem ist Teil des großen Gebiets von Advanced Planning Systemen (APS) und der Produktionsplanung mit PPS, ERP oder MES allgemein [Kurbel 2005]. Jedoch sind mit einem dezentralen und gleichzeitig hierarchisch sowie nachrichtenbasiert aufgebauten Prozesssystem für die Produktion völlig neue Voraussetzungen und Rahmenbedingungen für die Planung geschaffen, die in die Richtung der situierten Planung und lokalen Optimierung deuten, so dass umfangreicher Forschungsbedarf auf diesem speziellen Gebiet der Produktionsplanung besteht. Die Verteilung von Arbeitsschritten muss daher für dezentrale, auf objektorientierten Prozessen basierende Systeme angepasst oder sogar neu erforscht werden, da sich mangels zentral verfügbarer Informationen im Rahmen der Virtualisierung auch zentrale Ansätze nicht mehr verwirklichen lassen.

Die Optimierung der Aufgaben-Vermittlung an andere Peers im dezentralen System muss anhand von Transportzeiten, Maschinenlaufzeiten und Produktionskosten lokal optimiert werden, wobei Eskalation, Corrective Actions und Recommender Systeme weitere Themen sind.

Ein hardwareseitiger Austausch von Technologiemodulen [SpathKoch 2007] erfordert auch modell- und softwareseitig flexible und intelligente Module, die bei Rekonfiguration und

Upgrade die physischen Strukturen abbilden und vollständig nutzen können. Entsprechende semantische Modelle können in Zukunft basierend auf OMICRON sowohl im Prozess als auch für die Mechatronikintegration entwickelt werden.

Mit der Dezentralisierung von Produktionssystemen und zunehmender Strukturkomplexität durch Virtualisierung und Betreibermodelle fällt eine bei zentralen Systemen nur in geringerem Maße zu beobachtende Skaleninvarianz beziehungsweise Selbstähnlichkeit von Produktionssystemen auf. Teilbereiche verhalten sich ähnlich wie ganze Unternehmen, Intelligenz wird zunehmend auf Maschinen verlagert, so dass möglicherweise rein objektorientierte Ansätze in Zukunft ebenfalls nicht ausreichen. „Fraktale“ objektorientierte Prozesse könnten Selbstähnlichkeit – ähnlich den Fraktalen – als Kriterium einführen und auf Fuzzy-Logik basierende, ergänzende Ähnlichkeitsfunktionen weiterentwickeln.

Fraktale, objektorientierte Prozesse (FOOP) seien hier in Anlehnung an Fraktale als Prozesse definiert, deren Teilbereiche über verwandte Komponenten beschrieben werden. Durch wiederholte Verwendung und Parametrisierung entsteht so eine selbstähnliche Struktur, die, trotz zunächst eher chaotisch wirkenden Beziehungen und Typen, ein hohes Maß an Kohärenz und Wiederverwendung beziehungsweise Wiederverwendbarkeit entwickelt. Eine rekursive Anwendbarkeit muss hier vom Komponentenmodell her ermöglicht werden. Zudem erscheint es notwendig, eine Ähnlichkeitsrelation oder einen berechenbaren Ähnlichkeitsfaktor einzuführen. Dieser kann beispielsweise über den gewichteten Vergleich von Teilkomponenten mit Hilfe von Teilmengen, Typgleichheit, Subtypen, Signaturähnlichkeit, bisherigen Verwendungskontexten etc. sowie über transitive Beziehungen erfolgen. So könnte eine unscharfe Typisierung und weitere Flexibilisierung von Abläufen und Elementen der Produktion erreicht werden, beispielsweise, wenn Komponenten aus verschiedenen Bereichen ähnliche Funktionen erfüllen.

Mit fraktalen und unscharfen Ansätzen lassen sich zudem komplexe Systeme, wie sie durch die dezentrale Produktion und dynamische und virtuelle Unternehmensstrukturen entstehen, möglicherweise in der Zukunft besser beherrschen und vorhersagen.

Einen Schritt weiter gehen selbstorganisierende Systeme wie Autonomic Computing [Kephart/Chess 2003] oder Organic Computing⁴⁴. Die Self-X-Eigenschaften wie Selbstkonfiguration, Selbstheilung, Selbstanpassung und Selbstoptimierung aus der Informatik können auch auf Produktionsprozesse angewandt werden und stellen trotz Rückschlägen in der künstlichen Intelligenz weiterhin einen aussichtsreichen Ansatz für die Ausführung objektorientierter Prozesse dar. Diese sind jedoch für die Produktion bisher kaum realisiert worden.

Für Fertigungssysteme haben sich – unter anderem bedingt durch globale Marktentwicklungen – in den vergangenen Jahren die Paradigmen stark geändert. Steigende Flexibilität bei stabilen Preisen führt zu neuen Ansätzen, die häufig nur mit extensiver Nutzung von Informationstechnologie zu realisieren sind.

Die Wirkung neuer Konzepte auf die Technik und Organisation von Produktionssystemen als auch auf das sozio-technische Systems Produktion als Ganzes wird erst die zukünftige Entwicklung zeigen. Diese wirkt weit über den technischen Bereich hinaus und beeinflusst bereits seit der industriellen Revolution alle Gesellschaftsbereiche fundamental.

⁴⁴ <http://www.organic-computing.de/spp>

Literaturverzeichnis

- [AbadiCardelli 1996] Abadi, M.; Cardelli, L.: A theory of objects. In: Monographs in computer science, New York: Springer-Verlag, 1996
- [Active Endpoints et al. 2007a] Active Endpoints, Adobe, BEA, IBM, Oracle, SAP: Web Services Human Task (WS-HumanTask), Version 1.0, 2007, [online] http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel4people/WS-HumanTask_v1.pdf, 23.11.2007
- [Active Endpoints et al. 2007b] Active Endpoints, Adobe, BEA, IBM, Oracle, SAP: WS-BPEL Extension for People (BPEL4People), Version 1.0, 2007, [online] http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel4people/BPEL4People_v1.pdf, 23.11.2007
- [AgnarssonGreenlaw 2007] Agnarsson, G.; Greenlaw, R.: Graph Theory: Modeling, Applications, and Algorithms. Pearson Prentice Hall / Pearson Education, 2007
- [AlbertFuchs 2007] Albert, C.; Fuchs, C.: Durchblick im Begriffsdschungel der Business-Software. 2007, [online] <http://www.logistik-inside.de/fm/2248/Durchblick%20Business%20Software.pdf>, 28.3.2007
- [AmbriolaJaccheri 1991] Ambriola, V.; Jaccheri, M. L.: Definition and Enactment of Oikos software entities. In: Proceedings of the First European Workshop on Software Process Modeling, Milan, Italy, 1991
- [AmmerlaanWright 2004] Ammerlaan, J.; Wright, D.: Adaptive cooperative fuzzy logic controller. In: Proceedings of the 27th Australasian Conference on Computer Science, Volume 26, Darlinghurst: Australian Computer Society, 2004, pp. 255-263
- [Anderson 2003] Anderson, D.M.: Build-to-Order & Mass Customization – The Ultimate Supply Chain Management and Lean Manufacturing Strategy for Low-Cost On-Demand Production without Forecasts or Inventory. Cambria: CIM Press, 2003
- [Aravind 2007] Aravind (Srinivasan): Database Handling and Analysis of Sensor Data. Master Thesis, Hochschule für Technik, Stuttgart, 2007
- [Arlow et al. 1997] Arlow, J.; Bandinelli, S.; Emmerich, W.; Lavazza, L.: Fine grained Process Modelling: An Experiment at British Airways. In: Software Process - Improvement and Practice, Volume 3, Number 2, 1997, pp. 105-131
- [Armenise et. al. 1993] Armenise, P.; Bandinelli, S.; Ghezzi, C. Morzenti, A.: A survey and assessment of software process representation formalisms. In: International Journal of Software Engineering and Knowledge Engineering, Volume 3, Number 3, 1993
- [Armstrong 2006] Armstrong, D.J.: The quarks of object-oriented development. In: Communications of the ACM, Volume 49, Issue 2, 2006, pp. 123-128
- [AtkinsonKühne 2001] Atkinson, C.; Kühne, T.: Processes and Products in a Multi-Level Metamodeling Architecture. In: International Journal of Software Engineering and Knowledge Engineering, Volume 11, Number 6, 2001, pp. 761-783
- [AtkinsonKühne 2002] Atkinson, C.; Kühne, T.: Rearchitecting the UML infrastructure. In: ACM Transactions on Modeling and Computer Simulation (TOMACS) 12(4), New York: ACM Press, 2002, pp. 290-321
- [Aurich et al. 2006] Aurich, J. C.; Drews, O.; Fuchs, C.; Wagenknecht, C.: Produktionssysteme für den Mittelstand – Gestaltung prozessorientierter Produktionssysteme unter Flexibilitätsgesichtspunkten. In: wt Werkstattstechnik online Jahrgang 96 (2006) Heft 5, 2006, pp. 302-307

- [Bandinelli et al. 1994] Bandinelli, S.; Fuggetta, A.; Ghezzi, C.; Lavazza, L.: SPADE: An Environment for Software Process Analysis, Design, and Enactment. In: Finkelstein, A.; Kramer, J.; Nuseibeh, B. (Eds.): Software Process Modelling and Technology. Research Studies Press, Taunton, England, 1994, pp. 223-247
- [Bandinelli et. al. 1993] Bandinelli, S. Fuggetta, A. Grigoli, S.: Process Modelling in the large with SLANG, Proceedings of of the 2nd Int. Conf. on Software Process, Berlin, Germany, 1993, pp. 75-93
- [Bastide 1995] Bastide, R.: Approaches in unifying Petri Nets and the Object-Oriented Approach. In: Agha, G.; de Cindio, F. (Hrsg.): Proceedings of the 1st Workshop on Object-Oriented Programming and Models of Concurrency. Turin/Italy, 1995
- [Battiston et al. 1988] Battiston, E.; De Cindio, F.; Mauri, G.: OBJSA Nets: A class of high-level nets having objects as domains. In: Rozenberg (Hrsg.): Advances in Petri Nets 1988. LNCS 340. Berlin: Springer, 1988, pp. 20-43.
- [Beck 2000] Beck, K.: Extreme Programming Explained: Embrace Change. Addison-Wesley, 2000.
- [BeckerRosemann 1996] Becker, J.; Rosemann, M. (Hrsg.): Workflowmanagement – State-of-the-Art aus Sicht von Theorie und Praxis. In: Proceedings zum Workshop vom 10. April 1996, Arbeitsberichte des Instituts für Wirtschaftsinformatik, Arbeitsbericht Nr. 47, 1996
- [BeinhauerSchlegel 2005a] Beinhauer, W.; Schlegel, T.: User Interface for Service Oriented Architectures. In: Intelligent Production Machines and Systems, Amsterdam: Elsevier, 2005, pp. 129-134
- [BeinhauerSchlegel 2005b] Beinhauer, W.; Schlegel, T.: User interfaces for service oriented architectures, CD-ROM. Proceedings of HCI International 2005, Erlbaum: 2005
- [BeinhauerSchlegel 2006] Beinhauer, W.; Schlegel, T.: Service orientation in production control. In: Intelligent Production Machines and Systems, Amsterdam: Elsevier, 2006, pp. 350-355
- [Belkhatir et al. 1994] Belkhatir, N.; Estublier, J.; Melo, W.: ADELE-TEMPO: An Environment to Support Process Modelling and Enaction. In: Finkelstein, A.; Kramer, J.; Nuseibeh, B. (Eds.): Software Process Modelling and Technology. Research Studies Press, Taunton, England, 1994, pp. 187-222
- [BelkhatirMelo 1994] Belkhatir, N.; Melo, W. L.: Supporting Software Development Processes in Adele2. In: The Computer Journal, Volume 37, N°7, 1994, pp. 621-628
- [Benali et. al. 1989] Benali, K.; Boudjlida, N.; Charoy, F.; Derniame, J. C.; Godart, C.; Griffiths, Ph.; Gruhn, V.; Jamart, Ph.; Oldfield, D.; Oquendo, F.: Presentation of the ALF project. In: Proceedings of the International Conference on System Development Environments and Factories, 1989
- [Benedicenti et al. 1998] Benedicenti, L.; Succi, G.; Vernazza, T.; Valerio, A.: Object oriented process modeling with fuzzy logic, Symposium on Applied Computing, Proceedings of the 1998 ACM symposium on Applied Computing, Atlanta, Georgia, United States, 1998, pp. 267-271
- [BenyoucefRinderle 2005] Benyoucef, M.; Rinderle, S.: A Model-Driven Approach for the Rapid Development of E-Negotiation Systems. In: Proceedings of the Workshop on Enterprise Modelling and Information Systems Architectures (EMISA'05), Klagenfurt, Austria, October 2005
- [Bergholz 2005] Bergholz, M.A.: Objektorientierte Fabrikplanung. Dissertation, Fakultät Maschinenwesen, RWTH Aachen, 2005
- [Bergin 1997] Bergin, J.: What IS Object-Oriented Programming--Really? 1997, [online] <http://csis.pace.edu/~bergin/papers/oop.html>, 12.03.2007

- [Billington 2003] Billington, J.; Christensen, S.; van Hee, K. E.; Kindler, E.; Kummer, O.; Petrucci, L.; Post, R.; Stehno, C.; Weber, M.: The Petri Net Markup Language: Concepts, Technology and Tools. In: van der Aalst, W.; Best, E. (Eds.): Proceedings of the 24th International Conference Applications and Theory of PetriNets (ICATPN 2003), Lecture Notes in Computer Science, Volume 2679, Springer Verlag, 2003, pp. 483-505
- [Boehm 1988] Boehm, B.W.: A Spiral Model of Software Development and Enhancement. In: IEEE Computer, Volume 21, Issue 5, May 1988, 1998, pp. 61-72
- [Boldyreff et al. 2002] Boldyreff, C.; Nutter, D.; Rank, S.: Active artefact management for distributed software engineering. In: Proceedings of the Computer Software and Applications Conference (COMPSAC 2002) , 2002, pp. 1081-1086
- [BoltonDavies 2000] Bolton, C.; Davies, J.: Activity graphs and processes. In: Proceedings of Integrated Formal Methods (IFM 2000), LNCS 1945, Springer Verlag, 2000, pp. 77-96
- [Börger et al. 2000] Börger, E.; Cavarra, A.; Riccobene, E.: An ASM Semantics for UML Activity Diagrams. In: Proceedings of the International Conference on Algebraic Methodology and Software Technology (AMAST 2000), LNCS 1826, Springer Verlag, 2000, pp. 293-308
- [Brambilla 2006] Brambilla, M.: Generation of WebML web application models from business process specifications. In: Proceedings of the 6th International Conference on Web Engineering, Session 4: modeling and tools I, New York: ACM Press, 2006, pp. 85-86
- [Brambilla et al. 2006] Brambilla, M.; Ceri, S.; Fraternali, P.; Manolescu, I.: Process modeling in Web applications. In: ACM Transactions on Software Engineering and Methodology (TOSEM) archive, Volume 15 , Issue 4 (October 2006), New York: ACM Press, 2006, pp. 360-409
- [Brambilla et. al. 2005] Brambilla, M.; Ceri, S.; Comai, S.; Tziviskou, C.: Exception handling in workflow-driven Web applications. In: Proceedings of the 14th international conference on World Wide Web, SESSION: Web application design table of contents, New York: ACM Press, 2005, pp. 170-179
- [BretonBézivin 2001] Breton, E.; Bézivin, J.: Using Meta-Model Technologies to Organize Functionalities for Active System Schemes In: Workshop of Ontologies in Agent Systems (OAS'2001), Montreal, Canada, 2001
- [Breu et al. 2005] Breu, R.; Matzner, M.; Nickl, F.; Wieger, O. (Hrsg.): Software Engineering: Objektorientierte Techniken, Methoden und Prozesse in der Praxis. München, Wien: Oldenbourg, 2005
- [Brooks 1975] Brooks, F.: The Mythical Man-Month: Essays on Software Engineering. Addison-Wesley, 1975
- [Bruno 1994] Bruno, G.: Model-based Software Engineering. London: Chapman & Hall, 1994
- [Bruynooghe et al. 1994] Bruynooghe, R.F.; Greenwood, R.M.; Robertson, I.; Sa, J.; Warboys, B. C.: PADM: Towards a Total Process Modelling System In: Finkelstein, A.; Kramer, J.; Nuseibeh, B. (Eds.): Software Process Modelling and Technology. Taunton, England: Research Studies Press, 1994, pp. 293-334
- [BullingerScheer 2003] Bullinger, H.-J.; Scheer, A.-W. (Hrsg.): Service Engineering. Berlin, Heidelberg, New York: Springer, 2003
- [BullingerSchweizer 2005] Bullinger, H.-J.; Schweizer, W.: Mass customization in process industries: »from mass production to mass customization«. In: Proceedings of the Interdisciplinary world congress on mass customization and personalization (MCPC 2005), special MCP seminar, Hong Kong: Hong Kong University of Science and Technology, 2005, CD-ROM

- [BurkhardLaures 2003] Burkhard, B.; Laures, G.: SOA - Wertstiftendes Architektur-Paradigma. In: Objektspektrum 06/2003, SIGS DATACOM, 2003
- [Burkhardt 1994] Burkhardt, R.: Modellierung dynamischer Aspekte mit dem Objekt-Prozeß-Modell. Dissertation, Fakultät für Informatik und Automatisierung der Technischen Universität Ilmenau, Ilmenau 1994.
- [Caetano et al. 2005] Caetano, A.; Silva, A.R.; Tribolet, J.: Using roles and business objects to model and understand business processes. In: Proceedings of the 2005 ACM symposium on Applied computing, New York: ACM Press, 2005, pp. 1308-1313
- [Cass et al. 2000] Cass, A.G.; Staudt-Lerner, B.; Sutton, S.M.; McCall, E.K.; Wise, A.; Osterweil, L.J.: Little-JIL/Juliette: a process definition language and interpreter. In: Proceedings of the 22nd International Conference on Software Engineering. New York: ACM Press, 2000, pp. 754-757
- [Ceri et al. 2000] Ceri, S.; Fraternali, P.; Bongio, A.: Web Modeling Language (WebML): A Modeling Language for Designing Web Sites. In: Computer Networks Volume 33, Numbers 1-6 (June 2000), 2000, pp. 137-157
- [Chaugule 2001] Chaugule, A.: A User-Oriented Enterprise Process Modeling Language. Masters Thesis, Oklahoma State University, Stillwater, OK, 2001
- [Chen 1993] Chen, M.: CASE data interchange format (CDIF) standards: introduction and evaluation. In: Proceedings of the Twenty-Sixth Hawaii International Conference on System Sciences, Volume 3, 1993, pp. 31-40
- [Chen 1997] Chen, J.J.: CSPL: An Ada95-Like, Unix-Based Process Environment. In: IEEE Transactions on Software Engineering, Volume 23, Number 3, March 1997, 1997, pp. 171-184.
- [Chirita et al. 2006] Chirita, P.-A.; Idreos, S.; Koubarakis, M.; Nejdl, W.: Designing Semantic Publish/Subscribe Networks Using Super-Peers. In: [StaabStuckenschmidt 2006], 2006, pp. 159-179
- [Clark et al. 2001] Clark, T.; Evans, A.; Kent, S.: The Metamodelling Language Calculus: Foundation semantics for UML. In: Hussmann, H. (Ed.): Proceedings of Fundamental Approaches to Software Engineering (FASE 2001), Genova, Italy, Volume 2029, LNCS, Springer Verlag, 2001, pp. 17-31
- [Coad 1991] Coad, P.; Yourdon, E.: Object-Oriented Analysis. 2nd Edition, Yourdon Press, Englewood Cliffs N.J.: Prentice-Hall, 1991
- [Colombo et al. 2001] Colombo, A.W.; Neubert, R.; Schoop, R.: A solution to holonic control systems. In: Proceedings of the 8th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'01), Special Session on Multi-Agent-based Factory Automation, Industrial Applications of Intelligent Production Systems, Volume 2, 2001, pp. 489-498
- [Conradi et al. 1994] Conradi, R.; Hagaseth, M.; Larsen, J.O.; Nguyễn, N. M.; Munch, B.P.; Westby, P.H.; Zhu, W.; Jaccheri, M.L.; Liu, C.: Object-Oriented and Cooperative Process Modelling in EPOS. In: Finkelstein, A.; Kramer, J.; Nuseibeh, B.A. (Eds.): Software Process Modelling and Technology (PROMOTER), Advanced Software Development Series, Research Studies Press Ltd. (John Wiley), 1994, pp. 33-70
- [Constantinescu et al. 2006] Constantinescu, C.; Hummel, V.; Westkämper, E.: Collaborative factory planning, steps towards grid manufacturing. In: Production Engineering. Knowledge – Vision – Framework Programmes. Wrocław: Oficyna Wydawnicza Politechniki Wrocławskiej, 2006, pp. 75-82
- [CorstenGössinger 2001] Corsten, H.; Gössinger, R.: Advanced Planning Systems - Anspruch und Wirklichkeit, in: PPS Management, 6. Jg. (2001), Heft 2, 2001, pp. 32-39
- [Curtis et al. 1992] Curtis, B.; Kellner, M. and Over, J. Process Modeling. In: Communications of the ACM - Special issue on analysis and modeling in software development, Volume 35, Issue 9 (September 1992), 1992, pp. 75-90

- [CzarneckiEisenecker 2000] Czarnecki, K.; Eisenecker, U.: Generative Programmierung, Addison-Wesley, 2000
- [CzarneckiHelsen 2003] Czarnecki, K.; Helsen, S.: Classification of Model Transformation Approaches. In: OOPSLA'03 Workshop on Generative Techniques in the Context of Model-Driven Architecture, 2003
- [Dadam et al. 2005] Dadam, P.; Reichert, M.; Rinderle, S.; Atkinson, C.: Auf dem Weg zu prozessorientierten Informationssystemen der nächsten Generation – Herausforderungen und Lösungskonzepte. In: Spath, D.; Haasis, K.; Klumpp, D. (Hrsg.): Aktuelle Trends in der Softwareforschung - Tagungsband zum dIT-Forschungstag, Karlsruhe, Juni 2005, pp. 47-67
- [Dahr et al. 1994] Dahr, M.; Lautenbach, K.; Marx, T.; Ridder, H.: NET CASE: Towards a Petri Net Based Technique for the Development of Expert-Database Systems. Research Report 2-94, University of Koblenz, 1994
- [Dalal et. al. 2004] Dalal, N. P.; Kamath, M.; Kolarik, W.J.; Sivaraman, E.: Toward an integrated framework for modeling enterprise processes. In: Communications of the ACM, Volume 47, Issue 3, New York: ACM Press, 2004, pp. 83 - 87
- [Dami et al. 1998] Dami, S.; Estublier, J. Amieur, M.: APEL: A Graphical Yet Executable Formalism for Process Modeling. In: Automated Software Engineering: An International Journal, Volume 5, Number 1, January 1998, 1998, pp. 61-96
- [Dandl 1999] Dandl, J.: Objektorientierte Prozessmodellierung mit der UML und EPL. In: Arbeitspapiere WI Nr. 12/1999, Universität Mainz, Lehrstuhl für Allgemeine BWL und Wirtschaftsinformatik, 1999
- [Davenport 1993] Davenport, T.H.: Process Innovation – Reengineering Work through Information Technology, Boston: Harvard Business School Press, 1993
- [Davenport 1998] Davenport, T.H.: Putting the enterprise into the enterprise system, Harvard Business Review, Volume 76, Number 4, July-August 1998, 1998, p.121-131
- [DeitersGruhn 1990] Deiters, W.; Gruhn, V.: Managing Software Processes in the Environment MELMAC. In: Proceedings of the 4th ACM SIGSOFT Symposium on Software Development Environment, 1990, pp. 193-205
- [DeitersGruhn 1994] Deiters, W.; Gruhn, V.: The FUNSOFT Net Approach to Software Process Management. In: International Journal of Software Engineering and Knowledge Engineering, 4(2), 1994, pp. 229-256
- [DeMarco 1979] DeMarco, T.: Structured Analysis and System Specification. Englewood Cliffs, N.J.: Prentice-Hall, 1979
- [DeselOberweis 1996] Desel, J.; Oberweis, A.: Petri-Netze in der Angewandten Informatik: Einführung, Grundlagen und Perspektiven. In: Wirtschaftsinformatik, 38. Jahrgang, Heft 4, 1996, pp. 359-367
- [DIN 1991] DIN: DIN EN 28879 – Informationsverarbeitung; Textverarbeitung und -kommunikation; Genormte Verallgemeinerte Auszeichnungssprache (SGML) (ISO 8879:1986 + A1:1988); EN 28879:1990, Berlin: Beuth Verlag, 1991
- [DIN 2003] DIN: DIN 8580 – Fertigungsverfahren - Begriffe, Einteilung, Berlin: Beuth Verlag, 2003
- [Dostal et al. 2005] Dostal, W.; Jeckle, M.; Melzer, I.; Zengler, B.: Service-orientierte Architekturen mit Web Services, Spektrum Akademischer Verlag, 2005
- [Egyed 2002] Egyed, A.: Automated Abstraction of Class Diagrams. In: ACM Transactions on Software Engineering and Methodology, volume 11, Number 4, October 2002, 2002, pp. 449-491

- [EhrlerCranefield 2004] Ehrler, L.; Cranefield, S.: Executing Agent UML Diagrams. In: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, Volume 2. New York: IEEE Computer Society, 2004, pp. 906-913
- [Emmerich et al. 1996] Emmerich, W.; Bandinelli, S.; Lavazza, L.; Arlow, J.: Fine grained process modelling: an experiment at British Airways. In: Proceedings of the 4th International Conference on the Software Process, Brighton, United Kingdom. IEEE Computer Society Press, 1996, pp. 2-12.
- [Emmerich et. al. 1991] Emmerich, W.; Junkermann, G.; Schafer W.: MERLIN: knowledge-based process modelling. In: Proceedings of the First European Workshop on Software Process Modeling, Milan, Italy, 1991
- [Engelke et al. 1983] Engelke, H.; Grotrian, J.; Scheuing, C.; Schmackpfeffer, A.; Solf, B.: Structured modeling of manufacturing processes. In: Proceedings of the 16th annual symposium on Simulation, IEEE Computer Society Press, 1983, pp. 55-68
- [Engels et al. 1994] Engels, G.; Groenewegen, L.: SOCCA: Specifications of Coordinated and Cooperative Activities. In: Finkelstein, A.; Kramer, J.; Nuseibeh, B. (Hrsg.): Software Process Modelling and Technology, Research Studies Press, 1994, pp. 71-102
- [English 1993] English, S.L.: Coloured Petri Nets for object-oriented modelling. Dissertation, University of Brighton, Brighton 1993
- [Epicentric 2001] Epicentric Inc.: WSUI Executive White Paper, June 2001, 2001, [online] <http://xml.coverpages.org/WSUI-wp20010627.pdf>, 18.03.2007 nur noch über coverpages.org erreichbar, WSUI.org ist nicht mehr verfügbar.
- [Erl 2005] Erl, T.: Service-Oriented Architecture – Concepts, Technology, and Design. Prentice Hall, 2005
- [Esser 1996] Esser, R.: An Object Oriented Approach to Embedded System Design. Dissertation, Institut für Technische Informatik und Kommunikationsnetze, ETH Zürich, 1997
- [Feldmann et al. 2007a] Feldmann, K.; Wolf, W.; Weber, M.: Design of a formal model for the specification of agent platforms based on Plug&Produce-able production systems. In: Production Engineering, Berlin, Heidelberg: Springer Verlag, Volume 1, Number 3, November 2007, pp. 321-328
- [Feldmann et al. 2007b] Feldmann, K.; Weber, M.; Wolf, W.: Design of a theoretical holistic system model as base of construction kits for building Plug&Produce-able modular production systems. In: Production Engineering, Berlin, Heidelberg: Springer Verlag, Volume 1, Number 3, November 2007, pp. 329-336
- [FerstlSinz 2001] Ferstl, O.K.; Sinz E.J.: Grundlagen der Wirtschaftsinformatik. 4., überarb. und erw. Aufl.; München: Oldenbourg-Verlag, 2001
- [Finkelstein et al. 1994] Finkelstein, A.; Kramer, J.; Nuseibeh, B. (Eds): Software process modelling and technology. New York: Wiley, 1994
- [Firesmith et al. 1998] Firesmith, D.; Henderson-Sellers, B.; Graham, I.: OPEN Modeling Language (OML) Reference Manual. Sigs Reference Library Cambridge, New York: Cambridge University Press, 1998
- [FiresmithHenderson-Sellers 2002] Firesmith, D.G.; Henderson-Sellers, B.: The OPEN Process Framework. London: Addison-Wesley, 2002
- [FleischhackLichtblau 1993] Fleischhack, H.; Lichtblau, U.: MOBY - A Tool for High Level Petri Nets with Objects. In: Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, "Systems Engineering in the Service of Humans", LeTouquet/France 1993, Volume 4, New York: IEEE Press, 1993, pp. 644-649

- [FranchRibo 1999a] Franch, X.; Ribó, J.M.: Using UML for Modelling the Static Part of a Software Process. In: France, R.; Rumpe, B. (Eds.): UML'99 - The Unified Modeling Language. Beyond the Standard, Berlin, Heidelberg, New York: Springer-Verlag, LNCS Volume 1723, 1999, pp. 292-307
- [FranchRibo 1999b] Franch, X.; Ribó, J.M.: PROMENADE: A Modular Approach to Software Process Modeling and Enaction. Research Report LSI-99-13-R, Dept. LSI (UPC), 1999
- [Frankel 2003] Frankel, D.S.: Model Driven Architecture – Applying MDA to Enterprise Computing, OMG Press, Wiley Publishing, 2003
- [Frenkler 2005] Frenkler, C.: Modellierung und Analyse transaktionaler Geschäftsprozesse. Diplomarbeit, Humboldt-Universität zu Berlin, July 2005, [online] <http://www2.informatik.hu-berlin.de/top/download/publications/Frenkler05.pdf>, 27.03.2007
- [Fröming et al. 2005] Fröming, J.; Korf, R.; Fürstenau, D.: Knowledge Modelling and Description Language KMDL® v2.0, Arbeitsbericht WI 23/2005, Universität Potsdam, 2005
- [Gadatsch 2005] Gadatsch, A.: Grundkurs Geschäftsprozess-Management. Wiesbaden: Vieweg & Sohn Verlag / GWV Fachverlage, 4. erweiterte Auflage, 2005
- [Gehnen 1999] Gehnen, G.: Integriertes Netzwerk zur Fertigungssteuerung und -automatisierung, Dissertation, Heinz-Nixdorf-Institut, Universität Paderborn, HNI-Verlagsschriftenreihe, 1999
- [Gehrke et al. 1998] Gehrke, T.; Goltz, U.; Wehrheim, H.: The Dynamic Models of UML: Towards a Semantics and its Application in the Development Process, 11/1998, Hildesheimer Informatik Berichte, 1998
- [Genrich 1987] Genrich, H. J.: Predicate/transition nets. In: Brauer, W.; Reisig, W.; Rozenberg, G.: Advances in Petri nets 1986, part I on Petri nets: central models and their properties, London: Springer Verlag, pp. 207-247, 1987
- [Gomez-Perez et al. 2004] Gomez-Perez, A.; Fernandez-Lopez, M.; Corcho-Garcia, O.: Ontological Engineering, London: Springer, 2004
- [Gonzalez-PerezHenderson-Sellers 2006] Gonzalez-Perez, C.; Henderson-Sellers, B.: A powertype-based metamodeling framework. In: Software & System Modeling (2006) 5(1), Springer Verlag, pp. 72-90
- [GötzLiddle 2003] Götz, A.; Liddle, J.: "Process Driven Architecture" und "Business Process Management". In: Objektspektrum 06/2003, SIGS DATACOM, 2003
- [Grau 2004] Grau, B.C.: A possible simplification of the semantic web architecture. In: Proceedings of the 13th international conference on World Wide Web, SESSION: Semantic web foundations table of contents, New York: ACM Press, 2004, pp. 704-713
- [Green 2003] Green, A.: Transacting Business with Web Services, Part I. In: SOA Web Services Journal, Volume 3, issue 9, September 2003, 2003, www.wsj2.com
- [GreenfieldShort 2004] Greenfield, J.; Short, K.: Software Factories: Assembling Applications with Patterns, Models Frameworks and Tools. Indianapolis: Wiley & sons, 2004
- [GreenFurniss 2003] Green, A.; Furniss, P.: Transacting Business with Web Services, Part 2, SOA Web Services Journal (WSJ) , Volume 3, issue 11, September 2003, 2003, www.wsj2.com
- [Gronau et al. 2003] Gronau, N.; Palmer, U.; Schulte, K.; Winkler, T.: Modellierung von wissensintensiven Geschäftsprozessen mit der Beschreibungssprache K-Modeler. In: Reimer, U.; Abecker, A.; Staab, S.; Stumme, G. (Hrsg.): Professionelles Wissensmanagement - Erfahrungen und Visionen, Proceedings der GI, Bonn, 2003, pp. 315-322

- [Grose et al. 2002] Grose, T.J.; Doney, C.D.; Brodsky, S.A.: *Maturing XMI – Java Programming with XMI, XML, and UML*. OMG Press / John Wiley & Sons, 2004
- [Gruber 1993] Gruber, T. R.: A translation approach to portable ontologies. In: *Knowledge Acquisition*, Band 5, Nummer 2, 1993, pp. 199-220
- [GruhnUrbainczyk 1998] Gruhn, V.; Urbainczyk, J.: Software process modeling and enactment: an experience report related to problem tracking in an industrial project. In: *Proceedings of the 20th international conference on Software engineering*, Washington: IEEE Computer Society Press, 1998, pp. 13-21
- [GrundyHosking 1998] Grundy, J.C.; Hosking, J.G.: Serendipity: Integrated Environment Support for Process Modelling, Enactment and Work Coordination. In: *Automated Software Engineering: An International Journal*, Volume 5, Number 1, January 1998, 1998, pp. 27-60
- [Grüninger 2003] Grüninger, M.: Applications of PSL to Semantic Web Services. In: *Proceedings of the first International Workshop on Semantic Web and Databases (SWDB'03)*. Very Large Databases Conference, Berlin, 2003
- [Guelfi et al. 2003] Guelfi, N.; Ries, B.; Sterges, P.: MEDAL: a case tool extension for model-driven software engineering. In: *Proceedings of IEEE International Conference on Software: Science, Technology and Engineering (SwSTE '03)*, 2003, pp. 33-42
- [Hagen 2005] Hagen, M.: *Definition einer Sprache zur Beschreibung von Prozessmustern zur Unterstützung agiler Softwareentwicklungsprozesse*, Dissertation, Universität Leipzig, 2005
- [HaggeWagner 2005] Hagge, N.; Wagner, B.: A New Function Block Modeling Language Based on Petri Nets for Automatic Code Generation. In: *IEEE Transactions on Industrial Informatics*, Volume 1, Number 4, November 2005, 2005
- [Hahn et al. 1997] Hahn, H.; Hahn, J.; Kim, J.: A Cognitive Engineering Study on the Development of an Object-Oriented Process Modeling Formalism. In: *Proceedings of the Thirtieth Annual Hawaii International Conference on System Sciences*, Piscataway/NJ: IEEE Press, 1997
- [HammerChampy 1993] Hammer, M.; Champy, J.: *Reengineering the Corporation – A Manifesto for Business Revolution*, London: Nicholas Brealey, 1993
- [HärderReuter 1983] Härder, T.; Reuter, A.: Principles of transaction-oriented database recovery. In: *ACM Computing Surveys (CSUR)*, Volume 15, Issue 4 (December 1983), New York: ACM Press, 1983, pp. -317
- [Harel 1987] Harel, D.: Statecharts a visual formalism for complex systems. In: *Science of Computer Programming 8/1987*, North Holland: Elsevier Science Publishers, 1987, pp. 231-274
- [HarrisonColombo 2006] Harrison, R.; Colombo, A. W.: Collaborative automation: from rigid coupling toward dynamic reconfigurable production systems. In: Piztek, P. (Ed.): *Proceedings of the 16th IFAC World Congress 2005*, 2006
- [Haumer 2005] Haumer, P.: IBM Rational Method Composer: Part 1: Key concepts, 2005, [online] <http://www-128.ibm.com/developerworks/rational/library/dec05/haumer/>, 3.4.2007
- [HealyKilgore 1997] Healy, K.J.; Kilgore, R.A.: SilkTM: a Java-Based Process Simulation Language. In: *Proceedings of the 1997 Winter Simulation Conference*, 1997, pp. 475-481
- [Heimann et al. 1996] Heimann, P.; Joeris, G.; Krapp, C.-A.; Westfechtel, B.: DYNAMITE: Dynamic Task Nets for Software Process Management. In: *Proceedings of the 18th International Conference on Software Engineering*, IEEE Computer Society Press, 1996, pp. 331-341
- [Henderson-SellersFiresmith 1999] Henderson-Sellers, B.; Firesmith, D.G.: Comparing OPEN and UML: the two third-generation OO development approaches. In: *Information and Software Technology 41.3*, 1999, pp. 139-156

- [Henderson-SellersGonzalez-Perez 2005a] Henderson-Sellers, B.; Gonzalez-Perez, C.: The Rationale of Powertype-based Metamodelling to Underpin Software Development Methodologies. In: Proceedings of the 2nd Asia-Pacific Conference on Conceptual Modelling (APCCM2005), Volume 43, Darlinghurst: Australian Computer Society, 2005, pp. 7-16
- [Henderson-SellersGonzalez-Perez 2005b] Henderson-Sellers, Gonzalez-Perez: Connecting Powertypes and Stereotypes. In: Journal of Object Technology, Volume 4, Number 7, September - October 2005, 2005, pp. 83-96
- [Hendrickx et al. 2006] Hendrickx, W.; Gorissen, D.; Dhaene, T.: Grid enabled sequential design and adaptive metamodelling. In: Proceedings of the 37th conference on Winter simulation, SESSION: Modeling methodology a: metamodelling. Monterey: Winter Simulation Conference, 2006, pp. 872-881
- [Holt et al. 1983] Holt, A.; Ramsey, R. and Grimes, J. Coordinating System Technology as the Basis for a Programming Environment. Electrical Communication, Volume 57, Number 4 (1983), 1983, pp. 307-314
- [Holvoet/Verbaeten 1995] Holvoet, T.; Verbaeten, P.: PN-TOX: a Paradigm and Development Environment for Object Concurrency Specifications. In: Agha, G.; de Cindio, F. (Hrsg.): Proceedings of the 1st Workshop on Object-Oriented Programming and Models of Concurrency (ICATPN'95), 1995
- [Höbß 2005] Höbß, O.: Ein System für das Wiederverwendungs-Management von Software-Komponenten, Dissertation, IPA-IAO Forschung und Praxis, Heimshelm: Jost-Jetter Verlag, 2005
- [Howerton 2007] Howerton, J. T.: Service-Oriented Architecture and Web 2.0, IEEE IT Pro, May/June 2007, Piscataway/NJ: IEEE Press, 2007, pp. 62-64
- [HuffLesser 1988] Huff, K.E.; Lesser, V.: A Plan-Based Intelligent Assistant that Supports the Software Development Process. In: Proceedings of the 3rd ACM Symposium on Practical Software Development Environments. New York: ACM Press 1988, pp. 97-106
- [HumphreyFeiler 1992] Humphrey, W.S. and Feiler, P.H.: Software Process Development and Enactment: Concepts and Definitions, Technical Report CMU/SEI-92-TR-004, Pittsburgh: Software Engineering Institute, Carnegie Mellon University, 1992
- [HumphreyKellner 1994] Humphrey, W. S.; Kellner, M. I.: Software process modeling, Proceedings of the 11th international conference on Software engineering, 1989, pp. 331-342
- [Hupe et al. 1998] Hupe, P.; Matthes, F.; Wegner, H.: Ein bruchloser Übergang von der Prozeßmodellierung zu kooperativen Software-Architekturen. Technical report, November 1998, Institut für Softwaresysteme, Technische Universität Hamburg-Harburg, 1998
- [I*PROMS 2006a] I*PROMS: Innovative Production Machines and Systems: Identification of research knowledge gaps based on roadmapping exercises, Deliverable D2.8, 2006, [online]
http://www.iproms.org/filestore2/download/699/iproms_D2.8_research_knowledge_gaps_edited_27_2_06_0.pdf, 3.6.2007
- [I*PROMS 2006b] I*PROMS: Innovative Production Machines and Systems: Updated research roadmap covering all POM research areas, Version 1.4, 2006, [online]
http://www.iproms.org/filestore2/download/991/iproms_D7.13_POM_updated_Roadmap_10_11_06_0.pdf, 3.6.2007
- [IBM 2002] IBM: (WSXL) Web Service Experience Language Version 2. 2002, [online]
<http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-wsxl/ws-wsxl2.pdf>, 3.6.2007

- [IBM et al. 2003] IBM: Business Process Execution Language for Web Services, Version 1.1, 2003, [Online]
<http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel/ws-bpel.pdf>, 18.3.2007
- [IBMSAP 2005] IBM, SAP: WS-BPEL Extension for People – BPEL4People. Joint White Paper by IBM and SAP, July 2005, 2005, [online]
http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel4people/BPEL4People_white_paper.pdf
- [IDW 2005] IDW: eCI@ss - The leading classification system, White Paper, Juni 2005, Institut der deutschen Wirtschaft (IDW) , 2005, [online]
[http://www.eclass.de/user/documents/eng_white_paper_v1_1_\[june_2005\].pdf](http://www.eclass.de/user/documents/eng_white_paper_v1_1_[june_2005].pdf)
- [IEC 2005] IEC 61499-1, Function blocks - Part 1: Architecture, International Electrotechnical Commission (IEC), 2005
- [IETF 1996] IETF: Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types - RFC 2046, November 1996, Draft Standard, 1996, [online]
<http://tools.ietf.org/html/rfc2046>
- [Inoue et al. 1986] Inoue, K.; Seki, H.; Taniguchi, K. et al.: Compiling and Optimizing Methods for the functional Language ASL/F, Science of Computer Programming, 7, 11, 1986, pp. 297-312
- [ISO 1998] ISO 2382-15 : 1998: Information technology - Vocabulary - Part 15: Programming languages; Revision of first edition ISO 2382-15 : 1985, 1998
- [ISO 2006] ISO: ISO/IEC 23271:2006 – Information technology -- Common Language Infrastructure (CLI) Partitions I to VI. ISO – International Standards Organisation, 2006
- [Jablonski 1994] Jablonski, S.: MOBILE: A Modular Workflow Model and Architecture. In: Proceedings of 4th International Working Conference on Dynamic Modelling and Information Systems, Nordwijkerhout, 1994
- [JablonskiBussler 1996] Jablonski, S.; Bussler, C.: Workflow Management – Modeling Concepts, Architecture and Implementation. London: International Thomson Computer Press, 1996.
- [Jaccheri et al. 1992] Jaccheri, L.; Larsen, J.-O.; Conradi, R.: Software Process Modeling and Evolution in EPOS. In: Proceedings of Fourth International Conference on Software Engineering and Knowledge Engineering (SEKE), Italien, 1992, pp. 574-589
- [Jaccheri et al. 1998] Jaccheri, M.L.; Picco, G.P.; Lago, P.; Eliciting Software Process Models with the E3 Language. ACM Transactions on Software Engineering and Methodology Volume 7, Number 4, 1998, pp. 368-410
- [Jackson 2000] Jackson, M.: An Analysis of Flexible and Reconfigurable Production Systems: An Approach to a Holistic Method for the Development of Flexibility and Reconfigurability, Linköping Studies in Science and Technology, Dissertation Number 640, Production Systems Department of Mechanical Engineering, Linköpings Universitet, Linköping/Schweden, 2000
- [Jacobson 1992] Jacobson, I.; Christerson, M.; Jonsson, P.; Övergaard, G.: Object-Oriented Software Engineering, Reading, Mass.: Addison-Wesley, 1992
- [Jäger et al. 1999] Jäger, D.; Schleicher, A.; Westfechtel, B.: Using UML for Software Process Modeling. In: Proceedings of the Joint 7th European Software Engineering and Foundation of Software Engineering (ESEC / SIGSOFT FSE 1999), 1999, pp. 91-108
- [Jammes et al. 2005] Jammes, F.; Mensch, A.; Smit, H.: Service-Oriented Device Communications Using the Devices Profile for Web Services. In: Proceedings of MPAC '05, 2005, Article Number 16

- [JammesSmit 2005] Jammes, F.; Smit, H.: Service-oriented paradigms in industrial automation. In: Industrial Informatics, IEEE Transactions on Industrial Informatics, Volume 1, Issue 1, 2005, pp. 62-70
- [JanneckNaegele 1999] Janneck, J. W.; Naedele, M.: Modeling Hierarchical and Recursive Structures Using Parametric Petri Nets. In: Tentner, A. (Hrsg.): Proceedings of the High Performance Computing Symposium - HPC 99, Advanced Simulation Technologies Conference, 1999, pp. 445-452
- [Janssen et al. 2003] Janssen, D.; Schlegel, T.; Wissen, M.; Ziegler, J.: MetaChart – Using Creativity Methods in a CSCW environment. In: Proceedings of HCI International 2003, Mahwah: Lawrence Erlbaum Associates, 2003
- [Jeckle et al. 2004] Jeckle, M.; Rupp, C.; Hahn, J.; Zengler, B.; Queins, S.: UML 2 glasklar. München, Wien: Carl Hanser Verlag, 2004
- [Jensen 1992] Jensen, K.: Colored Petri Nets: Basic Concepts, Analysis Methods and Practical Use. , Volume 1, Berlin: Springer-Verlag, 1992
- [Jensen 1994] Jensen, K.: An Introduction to the Theoretical Aspects of Coloured Petri Nets. In: de Bakker, J.W.; de Roever, W.-P. , Rozenberg, G. (Eds.): A Decade of Concurrency, Lecture Notes in Computer Science Volume 803, Springer-Verlag, 1994, pp. 230-272
- [Jensen 1997] Jensen, K.: A Brief Introduction to Coloured Petri Nets. In: E. Brinksma (Ed.): Tools and Algorithms for the Construction and Analysis of Systems. Proceeding of the TACAS'97 Workshop, Enschede, Lecture Notes in Computer Science Volume 1217, Springer-Verlag, 1997, pp. 203-208
- [Jensen 1998] Jensen, K.: An Introduction to the Practical Use of Coloured Petri Nets. In: Reisig, W.; Rozenberg, G. (Eds.): Lectures on Petri Nets II: Applications, Lecture Notes in Computer Science Volume 1492, Springer-Verlag, 1998, pp. 237-292.
- [Junkermann 1995] Junkermann, G.: ESCAPE – Eine graphische Sprache zur Spezifikation von Software-Prozessen. Dissertation, Universität Dortmund, Fachbereich Informatik, Lehrstuhl Softwaretechnik, September 1995, 1995
- [Kaiser et al. 1988] Kaiser, G. E.; Barghouti, N. S.; Feiler, P. H.; Schwanke, R. W.: Database Support for Knowledge-Based Engineering Environments, IEEE Expert, 3(2), 1988, pp. 18-32
- [Kaiser et al. 1990] Kaiser, G. E.; Barghouti, N. S.; Sokolsky, M. H.: Preliminary experience with process modeling in the MARVEL softwaredevelopment environment kernel. In: Proceedings of the Twenty-Third Annual Hawaii International Conference on System Sciences, volume II, 1990, pp. 131-140
- [Katayama 1989] Katayama, T.: A hierarchical and functional software process description and its enactment. In Proceedings of the 11th International Conference on Software Engineering, 1989, pp. 343-352
- [Kay 2003] "Dr. Alan Kay on the Meaning of "Object-Oriented Programming". [online] http://www.purl.org/stefan_ram/pub/doc_kay_oop_en, 2003
- [Keller et al. 1992] Keller, G.; Nüttgens, M.; Scheer, A.-W.: Semantische Prozeßmodellierung auf der Grundlage „Ereignisgesteuerter Prozeßketten“ (EPK). In: Scheer, A.-W. (Hrsg.): Veröffentlichungen des Instituts für Wirtschaftsinformatik, Heft 89, Saarbrücken, 1992
- [Keller et al. 1994] Keller, R.K.; Shen, X.; Bochmann, G.v.: Macronet - A Simple, yet Expressive and Flexible Formalism for Business Modelling. In: Proceedings of the Workshop on Computer-Supported Cooperative Work, Petri Nets and Related Formalisms within the 15th International Conference on Application and Theory of Petri Nets. Zaragoza/Spain, 1994, pp. 51-55
- [KempaMann 2005] Kempa, M.; Mann, Z.A.: Model Driven Architecture. In: Informatik Spektrum; Berlin, Heidelberg: Springer Verlag, Heft 4, August 2005, 2005, pp. 298-302

- [KephartChess 2003] Kephart, J.O.; Chess, D.M.: The Vision of Autonomic Computing. In: IEEE Computer, January 2003, Volume 36, Number 1, 2003, pp. 41-50
- [Klein et al. 2004] Klein, R.; Kupsch, F.; Scheer, A.-W.: Modellierung inter-organisationaler Prozesse mit Ereignisgesteuerten Prozessketten. In: Scheer, A.-W. (Hrsg.): Veröffentlichungen des Instituts für Wirtschaftsinformatik. Heft 178, November 2004, 2004
- [Kletti 2006] Kletti, J.: MES Manufacturing Execution System – Moderne Informationstechnologie zur Prozessfähigkeit und Werschöpfung. Berlin, Heidelberg, New York: Springer, 2006
- [Knott et al. 2003] Knott, R.P.; Merunka, V.; Polak, J.: The role of object-oriented process modeling in requirements engineering phase of information systems development. In: Proceedings of EFITA 2003, 2003
- [Koestler 1967] Koestler, A.: The Ghost in the Machine. London: Arkana Books, 1967
- [Kopp et al. 2006] Kopp, O.; Unger, T.; Leymann, F.: Nautilus Event-driven Process Chains: Syntax, Semantics, and their mapping to BPEL. In: Nüttgens, M, Rump, F.J.; Mendling J. (Hrsg.): EPK 2006 - 5. Workshop der Gesellschaft für Informatik e.V. Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten, 2006
- [Koren 2005] Koren, Y.: Reconfigurable Manufacturing and Beyond. In: The summary of keynote Speech of CIRP05 3rd International Conference on Reconfigurable Manufacturing, Ann Arbor, Michigan, USA, May 10-12, 2005
- [KostisVouros 2004] Kotis, K.; Vouros, G.: The HCONE Approach to Ontology Merging. In: Proceedings of the European Semantic Web Symposium (ESWS'04), 2004, pp. 137-151
- [KrallmannScholz-Reiter 1990] Krallmann, H.; Scholz-Reiter, B.: CIM-KSA – Eine rechnergestützte Methode für die Planung von CIM-Informations- und Kommunikationssystemen. In: Reuter, A. (Hrsg.): Informatik auf dem Weg zum Anwender - Proceedings der 20. GI-Jahrestagung, Berlin, Heidelberg: Springer, pp. 57-66, 1990
- [Kristensen et al. 1998] Kristensen, L.M.; Christensen, S.; Jensen, K.: The Practitioner's Guide to Coloured Petri Nets. In: International Journal on Software Tools for Technology Transfer, 2 (1998), Springer Verlag, 1998, pp. 98-132
- [Kruschinski 1998] Kruschinski, V.: Layoutgestaltung grafischer Benutzungsoberflächen: Generierung aus OOA-Modellen. Heidelberg, Berlin: Spektrum, Akademischer Verlag, 1999
- [KuhnHellingrath 2002] Kuhn, A.; Hellingrath, B.: Supply Chain Management. – Optimierte Zusammenarbeit in der Wertschöpfungskette; Berlin: Springer Verlag, 2002
- [Kummer 2002] Kummer, O.: Referenznetze. Logos-Verlag, Berlin, 2002
- [Kurbel 2005] Kurbel, K.: Produktionsplanung und -steuerung im Enterprise Resource Planning und Supply Chain Management. 6. Auflage, München, Wien: Oldenbourg Verlag, 2005
- [LahresRayman 2006] Lahres, B.; Rayman, G.: Praxisbuch Objektorientierung – von den Grundlagen zur Umsetzung, Galileo Computing, 2006
- [Lakos 1994] Lakos, C.: Object Petri Nets - Definition and Relationship to Coloured Petri Nets, Technical Report TR94-3, Department of Electrical Engineering and Computer Science, University of Tasmania, 1994
- [Lakos 1995] Lakos, C.A.: From coloured Petri Nets to Object Petri Nets. In: Proceeding of the 16th International Conference on Application and Theory of Petri Nets, Turin, June 1995, 1995, pp. 278-297

- [LakosKeen 1991] Lakos, C.A.; Keen, C.D.: Simulation with Object-Oriented Petri Nets. In: Proceedings of the Australian Software Engineering Conference. Sydney 1991
- [LastraDelamer 2006] Martinez Lastra, J.L.; Delamer, I.M.: Semantic Web Services in Factory Automation: Fundamental Insights and Research Roadmap. IEEE Transactions on Industrial Informatics, Volume 2, Number 1, February 2006, 2006, pp. 1-11
- [Laubscher 1997] Laubscher, H.-P.: Ein objektorientiertes Modell zur Abbildung von Produktionsverbänden in Planungssystemen. In: IPA-IAO Forschung und Praxis, Dissertation Universität Stuttgart, Berlin, Heidelberg, New York: Springer Verlag, 1997
- [Laufer 2006] Laufer, P.: Grundlagen für die Anpassung der Petrinetz-Semantik an WS-BPEL 2.0. Studienarbeit, Humboldt-Universität Berlin, 2006, [online] http://www2.informatik.hu-berlin.de/top/download/publications/Laufer2006_sa.pdf, 27.03.2007
- [Lee et al. 1996] Lee, J.; Yost, G. et al.: The PIF Process Interchange Format and Framework (Version 1.0, December 22, 1994), 1994, [online] <http://ccs.mit.edu/pif8.html>
- [LeitnerMahnke 2006] Leitner, S.H.; Mahnke, W.: OPC UA – Service-oriented Architecture for Industrial Applications. In: Softwaretechnik-Trends 26:4, November 2006, 2006, pp. 28-33
- [Leymann 2003] Leymann, F.: Choreographie: Geschäftsprozesse mit Web-Services. In: Objektspektrum 06/2003, SIGS DATACOM, 2003
- [Li 2004] Li, Q.: Entwurf und Realisierung eines generischen Modellvisualisierers und -editors für die Serviceentwicklung, Diplomarbeit Nummer: 2150, Fakultät Informatik, Universität Stuttgart, 2004
- [Li 2006] Li, X.: Entwurf, Implementierung und Erprobung einer grafischen Informations- und Interaktionsschnittstelle für Leitstandssysteme. Diplomarbeit, Fakultät für Informatik der Universität Stuttgart / Fraunhofer IAO, 2006
- [Li et al. 2006] Li, X.; Schlegel, T.; Rotard, M.; Ertl, T.: A Model-Based Graphical User-Interface for Process Control Systems in Manufacturing. In: Proceedings of the Intelligent Production Machines and Systems - 2nd I*PROMS Virtual International Conference 2006, 2006
- [Lian et al. 2006] Lian, F.-L.; Yook, J.K.; Tilbury, D.M.; Moyne, J.: Network architecture and communication modules for guaranteeing acceptable control and communication performance for networked multi-agent systems. In: IEEE Transactions on Industrial Informatics, Volume 2, Issue 1, 2006, pp. 12-24
- [ListKorherr 2006] List, B.; Korherr, B.: An evaluation of conceptual business process modelling languages. In: Proceedings of the 2006 ACM symposium on Applied computing, New York: ACM Press, 2006, pp. 1532-1539
- [LiuHorowitz 1989] Liu, L.-C.; Horowitz, E.: A formal model for Software Project Management. IEEE Transactions on Software Engineering, October 1989, pp. 1280-1293
- [LoosAllweyer 1998] Loos, P.; Allweyer, T.: Process Orientation and Object-Orientation - An Approach for Integrating UMLand Event-Driven Process Chains (EPC), Publication of the Institut für Wirtschaftsinformatik, Paper 144, Saarbrücken, March 1998
- [LoosFettke 2001] Loos, P.; Fettke, P.: Towards an Integration of Business Process Modeling and Object-Oriented Software Development. In: Proceedings of the Fifth International Symposium on Economic Informatics Bucharest, Bucharest, 2001
- [LudewigLichter 2007] Ludewig, J.; Licher, H.: Software Engineering – Grundlagen, Menschen, Prozesse, Techniken, Heidelberg: dpunkt.verlag, 2007

- [Luyten et al. 2005] Luyten, K.; Coninx, K.; Abrams, M.: Integrating UIML, Task and Dialogs with Layout Patterns for Multi-Device User Interface Designs. In: Proceedings of 11th International Conference on Human-Computer Interaction (HCI International 2005), Mahwah, New Jersey: Lawrence Erlbaum, 2005
- [Ly et al. 2006] Ly, L.T.; Rinderle, S.; Dadam, P.: Semantic Correctness In Adaptive Process Management Systems. In: Proceedings of the International Conference on Business Process Management, BPM 2006, Wien, Österreich, September 2006, LNCS 4102, 2006, pp. 193-208
- [Lyons 1995] Lyons, J.: Linguistic semantics: an introduction, 1. Auflage, Cambridge: Cambridge University Press, 1995
- [Majima 1995] Majima, I.: JIT. Kostensenkung durch Just-in-Time-Production. Ullstein Management, Ullstein Taschenbuch, 1995
- [MarikMcFarlane 2005] Marik, V.; McFarlane, D.: Industrial adoption of agent-based technologies. In: IEEE Intelligent Systems, Volume 20, Number 1 (January 2005), Piscataway/NJ: IEEE Press, 2005, pp. 27-35
- [Martens 2002] Martens, M.: Agentenorientierte Modellierung von Entscheidungsprozessen mit Petrinetzen. Diplomarbeit, Fachbereich Informatik, Universität Hamburg, 2002
- [Martin 1992] Martin, J.; Odell, J.: Object-Oriented Analysis and Design. Englewood Cliffs, N.J.: Prentice-Hall, 1992
- [Mayer et al. 1995] Mayer, R.; Menzel, C.; Painter, M.; Perakath, B.; de Witte, P.; Blinn, T.; Perakath, B.: Information Integration For Concurrent Engineering (IICE) - IDEF3 Process Description Capture Method Report. Technical Report September 1995, http://www.idef.com/pdf/idef3_fn.pdf, 15.5.2007
- [MeierHomuth 2006] Meier, H.; Homuth, M.: Holistic Ramp-Up Management in SME-Networks. In: Proceedings of the 39th International Seminar on Manufacturing Systems, Ljubljana (Slowenien), 2006, pp. 123-128
- [MeierHomuth 2007] Meier, H.; Homuth, M.: Wohin steuert der Workflow? – Strategisch-operative Zielvorgabe durch Prognose von technischen Anlaufkurven. In: wt Werkstattstechnik online Jahrgang 97 (2007) Heft 5, pp. 354-357
- [Mending et al. 2004] Mending, J.; Neumann, G.; Nüttgens, M.: A Comparison of XML Interchange Formats for Business Process Modelling. EMISA 2004 – Information Systems in E-Business and E-Government, 2004, pp. 129-140
- [MendingNüttgens 2003] Mending, J.; Nüttgens, M.: Konzeption eines XML-basierten Austauschformates für Ereignisgesteuerte Prozessketten (EPK). In: Gesellschaft für Informatik (GI) e.V. (Hrsg.): Informationssystem Architekturen, Wirtschaftsinformatik Rundbrief der GI Fachgruppe WI-MobilS, Jahrgang 10, Heft 2, 2003, pp. 89-103
- [MendingNüttgens 2005] Mending, J.; Nüttgens, M.: EPC Markup Language (EPML) - An XML-Based Interchange Format for Event-Driven Process Chains (EPC). Technical Report JM-2005-03-10. Vienna University of Economics and Business Administration, Version 2005-03-10, [online] <http://wi.wu-wien.ac.at/home/mending/publications/TR05-EPML.pdf>, 19.5.2007
- [Mili et al. 2003] Mili, H.; Jaoude, G.B.; Tremblay, G.; Lefebvre, E.; Petrenko, A.: Business Process Modeling Languages: Sorting Through the Alphabet Soup, Technical Report LATECE, November 2003, 2003, [online] <http://www.ischool.berkeley.edu/~glushko/IS243Readings/BusinessProcessModelingLanguages.pdf>, 7.3.2007
- [Mili et al. 2004] Mili, H.; Jaoude, G.B.; Lefebvre, É.; Tremblay, G.: Going beyond MDA: Business Process Modeling for Software Reuse, Proceedings of the Workshop OOPSLA on Legacy Transformation: Capturing Business Knowledge from Legacy Systems, OOPSLA'2004, Vancouver, 2004

- [Mittendorfer 1997] Mittendorfer, J.: Smalltalk : Sprache, Klassen und Programmierwerkzeuge, 2. Auflage, Bonn u.a.: Addison-Wesley, 1997
- [Moldt 1996] Moldt, D.: Höhere Petrinetze als Grundlage für Systemspezifikation. Dissertation, Universität Hamburg, 1996
- [MoldtRodenhagen 2000] Moldt, D.; Rodenhagen, J.: Ereignisgesteuerte Prozessketten und Petrinetze zur Modellierung von Workflows. In: Visuelle Verhaltensmodellierung verteilter und nebenläufiger Software-Systeme, 8.Workshop des Arbeitskreises GROOM der GI Fachgruppe 2.1.9 Objektorientierte Software-Entwicklung, Universität Münster, 2000
- [MontangeroAmbriola 1994] Montangero, C.; Ambriola, V.: OIKOS: Constructing Process-Centered Environment. In: [Finkelstein et al. 1994], 1994, pp. 131-152
- [Moreno et al. 2006] N. Moreno, P. Fraternali, A.Vallecillo: A UML 2.0 Profile for WebML Modeling. In: Proceedings of the sixth international conference on Web engineering. Model-Driven Web Engineering Workshop, New York: ACM Press, 2006, Artikel Nr. 4
- [Müller 2007] Müller, S.: Modellbasierte IT-Unterstützung von wissensintensiven Prozessen – dargestellt am Beispiel medizinischer Forschungsprozesse. Dissertation, Technische Fakultät der Universität Erlangen-Nürnberg, Erlangen, 2007
- [Murata 1989] Murata, T.: Petri nets: Properties, analysis, and applications. Proceedings of the IEEE 77, 4 (Apr. 1989), 1989, pp. 541-580
- [Neugebauer et al. 2007] Neugebauer, R.; Weidlich, D.; Riegel, J.: Approach to describing virtual reality competence components for services in production networks. In: Production Management, Production Engineering Research and Development, 1/2007, 2007, pp. 291-296
- [Neumann 2002] Neumann, O.: Integration von Werkzeugen in heterogene, prozessgesteuerte Software-Entwicklungsumgebungen – Tool Integration Concept, Dissertation, Universitäts-Gesamthochschule Paderborn, Paderborn, 2002, [online] http://deposit.ddb.de/cgi-bin/dokserv?idn=968554695&dok_var=d1&dok_ext=pdf&filename=968554695.pdf, 27.03.2007
- [Neumann et al. 1996] Neumann, O.; Sachweh, S.; Schäfer, W.: A High-Level Object-Oriented Specification Language for Configuration Management and Tool Integration. In: Carlo Montangero (Ed.): Software Process Technology, 5th European Workshop, EWSPT '96, Lecture Notes in Computer Science, Volume 1149, Springer, 1996, pp. 137-143.
- [NoyMusen 2002] Noy, F. N.; Musen, A. M.: Evaluating Ontology-Mapping Tools: Requirements and Experience. In: Proceedings of OntoWeb-SIG3 Workshop at the 13th International Conference on Knowledge Engineering and Knowledge Management, Siguenza, 2002, pp. 1-14
- [OASIS 2001] OASIS: ebXML Business Process Specification Schema V 1.01, 11 May 2001, 2001, [online] <http://www.ebxml.org/specs/ebBPSS.pdf>, 26.3.2007
- [OASIS 2004] OASIS / Abrahms, M.; Helms, J. (Eds.): User Interface Markup Language (UIML) Specification, Working Draft 3.1, 11 March 2004, wd-UIML-UIMLspecification-3.1, 2004, [online] <http://www.oasis-open.org/committees/download.php/5937/uiml-core-3.1-draft-01-20040311.pdf>, 23.11.2007
- [OASIS 2006a] OASIS: ebXML Business Process Specification Schema Technical Specification v2.0.4, OASIS Standard, 21 December 2006, 2006, [online] <http://docs.oasis-open.org/ebxml-bp/2.0.4/OS/spec/ebxmlbp-v2.0.4-Spec-os-en.pdf>, 26.3.2007
- [OASIS 2006b] OASIS / Thompson, R. (Ed.): Web Services for Remote Portlets Specification v2.0, Committee Draft, 11th October 2006, wsrp-2.0-spec-cd-04, 2006, [online] <http://docs.oasis-open.org/wsrp/v2/wsrp-2.0-spec-cd-04.html>, 23.11.2007

- [OASIS 2007] OASIS: Web Services Business Process Execution Language Version 2.0, Committee Specification, 31 January 2007, 2007, [Online] <http://docs.oasis-open.org/wsbpel/2.0/CS01/wsbpel-v2.0-CS01.pdf>, 18.3.2007
- [OMG 2000] OMG: The Unified Process Model (UPM). document ad/2000-05-05, 2000
- [OMG 2001] OMG: Model Driven Architecture (MDA). 2001, [online] <http://www.omg.org/docs/ormsc/01-07-01.pdf>, 23.11.2007
- [OMG 2003] OMG, Miller, J.; Mukerji, J. (Hrsg.): MDA Guide Version 1.0.1. omg/2003-06-01, 2003
- [OMG 2004a] Unified Modeling Language Specification Version 1.4.2. formal/04-07-02, 2004, [online] <http://www.omg.org/docs/formal/04-07-02.pdf>, 2.4.2007
- [OMG 2004b] OMG: Common Object Request Broker Architecture: Core Specification. Version 3.0.3, 2004, [online] <http://www.omg.org/docs/formal/04-03-12.pdf>, 23.11.2007
- [OMG 2005a] OMG: Unified Modeling Language: Infrastructure Version 2.0. 2005, [online] <http://www.omg.org/docs/formal/05-07-05.pdf>, 23.11.2007
- [OMG 2005b] OMG: Unified Modeling Language: Superstructure Version 2.0. 2005, [online] <http://www.omg.org/docs/formal/05-07-04.pdf>, 23.11.2007
- [OMG 2005c] OMG: MOF 2.0/XMI Mapping Specification, Version 2.1. formal/05-09-01, 2005, [online] <http://www.omg.org/cgi-bin/apps/doc?formal/05-09-01.pdf>, 28.3.2007
- [OMG 2005d] OMG: MOF Query / Views / Transformations (MOF QVT final adopted specification), ptc/05-11-01, 2005, [online] <http://www.omg.org/cgi-bin/apps/doc?ptc/05-11-01.pdf>, 28.3.2007
- [OMG 2006a] OMG: Business Process Modeling Notation Specification, OMG Final Adopted Specification, February 2006, dtc/06-02-01, 2006, [online] <http://www.bpmn.org/Documents/OMG%20Final%20Adopted%20BPMN%201-0%20Spec%2006-02-01.pdf>, 28.03.2007
- [OMG 2006b] OMG: MOF Core Specification, version 2.0, formal/2006-01-01, 2006, [online] <http://www.omg.org/cgi-bin/apps/doc?formal/06-01-01.pdf>, 28.3.2007
- [OMG 2006c] OMG: Object Constraint Language Specification, version 2.0, formal/2006-05-01, 2006, [online] <http://www.omg.org/cgi-bin/apps/doc?formal/06-05-01.pdf>, 28.3.2007
- [OMG 2006d] OMG: Ontology Definition Metamodel (ODM) Final Adopted Specification, ptc/06-10-11, 2006, [online] <http://www.omg.org/cgi-bin/apps/doc?ptc/06-10-11.pdf>, 28.3.2007
- [OMG 2006e] OMG: Software Process Engineering Metamodel (SPEM) Version 1.1. formal/05-01-06, 2006, [online] <http://www.omg.org/cgi-bin/apps/doc?formal/05-01-06.pdf>, 28.3.2007
- [OMG 2007a] OMG: Unified Modeling Language (UML) Superstructure Version 2.1.1. formal/07-02-05, 2007, [online] <http://www.omg.org/cgi-bin/apps/doc?formal/07-02-05.pdf>, 28.3.2007
- [OMG 2007b] OMG: Unified Modeling Language (UML) Infrastructure Version 2.1.1. formal/07-02-06, 2007, [online] <http://www.omg.org/cgi-bin/apps/doc?formal/07-02-06.pdf>, 28.3.2007
- [OMG 2007c] OMG: Business Process Definition MetaModel (BPDM), Final submission, In Response to: Business Process Definition Metamodel RFP (OMG Document bmi/2007-03-01), OMG Document bmi/2007-03-01, 2007, [online] <http://www.omg.org/cgi-bin/apps/doc?bmi/07-03-01.pdf>, 23.11.2007

- [OpdahlBerio 2006] Opdahl, A.L.; Berio, G.: Interoperable Language and Model Management Using the UEML Approach. In: Proceedings of the 2006 international workshop on Global integrated model management, SESSION: Megamodeling and interoperability table of contents, New York: ACM Press, 2006, pp. 35-42
- [Otterbein 1994] Otterbein, T.: Eine objektorientierte Architektur für Leitstände zur Feinplanung. In: IPA-IAO Forschung und Praxis, Dissertation Universität Stuttgart, Berlin, Heidelberg, New York: Springer Verlag, 1994
- [Ould, 1995] Ould, M.A.: Business Processes: Modelling and Analysis for Re-engineering and Improvement, John Wiley & Sons, 1995
- [PanHorroks 2003] Pan, J. Z.; Horrocks, I.: RDFS(FA): A DL-ised sub-language of RDFS. In: Proceedings of the 2003 Description Logic Workshop (DL 2003), 2003
- [PanHorroks 2007] Pan, J. Z.; Horrocks, I.: RDFS(FA): Connecting RDF(S) and OWL DL. In: IEEE Transactions on Knowledge and Data Engineering, Piscataway/NJ: IEEE Press, Volume 19, Issue 2, 2007, pp. 192-206
- [Parnas 1972] Parnas, D.L.: On the Criteria To Be Used in Decomposing Systems into Modules, Communications of the ACM, Volume 15, Number 12, December 1972, 1972, pp. 1053-1058
- [Patel-Schneider et al. 2004] Patel-Schneider, P.F.; Hayes, P.; Horrocks, I.: OWL Web Ontology Language – Semantics and Abstract Syntax, 2004, [online] <http://www.w3.org/TR/2004/REC-owl-semantics-20040210/>, 23.11.2007
- [Peltz 2003] Peltz, C.: Web Services orchestration and choreography, IEEE Computer, Volume 36, Number 10, Piscataway/NJ: IEEE Press, 2003, pp. 46-52
- [PesticVan der Aalst 2006] Pestic, M.; van der Aalst, W.M.P.: Analyzing the resource perspective of workflow management systems: using a meta model and constraints.; Beta Working Paper (Interim Report Number 157), 28 Seiten, 2006, [online] http://fp.tm.tue.nl/beta/publications/working%20papers/beta_wp157.pdf, 23.11.2007
- [Petkov et al. 2005] Petkov, S.; Oren, E.; Haller, A.: Aspects in Workflow Management, DERI Technical Report 2005-04-10, April 2005, [online] <http://www.deri.ie/fileadmin/documents/DERI-TR-2005-04-10.pdf>, 23.11.2007
- [Petri 1962] Petri, C. A. Kommunikation mit Automaten. Dissertation, Schriften des IIM Nr. 2, Institut für Instrumentelle Mathematik, Universität Bonn, 1962
- [Philippi 1999a] Philippi, S.: Synthese von Petri-Netzen und objektorientierten Konzepten, Dissertation, Universität Koblenz-Landau, 1999
- [Philippi 1999b] Philippi, S.: OOPr/T-Modelle - ein Pr/T-Netz basierter Ansatz zur objektorientierten Modellierung. In: Proceedings of AWPn1999, 1999
- [Pinheiro Da Silva 2001] Pinheiro da Silva, P.: A proposal for a LOTOS-based semantics for UML. Technical Report UMCS-01-06-1, Department of Computer Science, University of Manchester, 2001
- [Poernomo 2006] Poernomo, I.: The meta-object facility typed. In: Proceedings of the 2006 ACM symposium on Applied computing table of contents, SESSION: Software verification (SV), New York: ACM Press, 2006, pp. 1845-1849
- [Quartel et al. 2004] Quartel, D.; Dijkman, R.; van Sinderen, M.: Methodological support for service-oriented design with ISDL. In: Proceedings of the 2nd international conference on Service oriented computing, SESSION: Service design and modeling table of contents. New York: ACM Press, 2004, pp. 1-10
- [Quartel et. al. 2006] Quartel, D.; Dijkman, R.; van Sinderen, M.: Methodological support for service-oriented design with ISDL. In: Proceedings of the 2nd international conference on Service oriented computing, Session Service design and modeling, New York: ACM Press, 2004, pp. 1-10

- [Rath 2003] Rath, H.-H.: The Topic Maps Handbook. Whitepaper Empolis GmbH, www.empolis.com, 2003, [online] http://www.sts.tu-harburg.de/~r.f.moeller/lectures/anatomie-i-und-k-system/empolisticpicmapswhitepaper_eng.pdf, 5.4.2007
- [ReichertRinderle 2006] Reichert, M.U.; Rinderle, S.B.: On Design Principles for Realizing Adaptive Service Flows with BPEL. In: Proceedings EMISA 2006 - Methoden, Konzepte und Technologien für die Entwicklung von dienstebasierten Informationssystemen, Köllen Verlag. GI Lecture Notes in Informatics LNI P-95, 2006, pp. 133-146
- [ReinhartLulay 1998] Reinhart, G.; Lulay, W. E.: Coordinating Order Processing in Decentralized Production Units Using Hierarchical Simulation Models and Web-Technologies. In: Medeiros, D.J.; Watson, E.F.; Carson, J.S.; Manivannan, M.S. (Eds.): Proceedings of the 1998 Winter Simulation Conference, 1998, pp. 1655-1661
- [Reisig 1986] Reisig, W.: Petrinetze – Eine Einführung. 2. Auflage, Berlin, Heidelberg: Springer-Verlag, 1986
- [Reißing 1996] Reißing, R.: Konzeption und Realisierung einer Basismaschine für SESAM-2. Diplomarbeit Nr. 1345, Universität Stuttgart, Fakultät Informatik, 1996
- [Rinderle 2004] Rinderle, S.: Schema Evolution in Process Management Systems. Dissertation, Universität Ulm, Fakultät für Informatik, 2004
- [Rinderle et al. 2004] Rinderle, S.; Reichert, M.; Dadam, P.: Disjoint and Overlapping Process Changes - Challenges, Solutions, Applications. Proceedings of 12th Int'l Conf. Cooperative Information Systems (CoopIS'04), Agia Napa, Zypern, LNCS 3290, Oktober 2004, 2004, pp. 101-121
- [Rinderle et al. 2006] Rinderle, S.B. and Bassil, S. and Reichert, M.U.: A Framework for Semantic Recovery Strategies in Case of Process Activity Failures. In: Proceedings of the Eighth International Conference on Enterprise Information Systems (ICEIS'06): Databases and Information Systems Integration, Setúbal: INSTICC Press, 2006, pp. 136-143
- [Rölke 1999] Rölke, H.: Modellierung und Implementation eines Multi-Agenten-Systems auf der Basis von Referenznetzen. Diplomarbeit, Universität Hamburg, Fachbereich Informatik, 1999
- [Rolland 1993] Rolland, C.: Modeling the Requirements Engineering Process. In: 3rd European-Japanese Seminar on Information Modelling and Knowledge Bases, Budapest, Hungary, June 1993, 1993
- [Rolland 1998] Rolland, C.: A Comprehensive View of Process Engineering (CREWS-98-18). In: Proceedings of the 10th International Conference CAiSE'98, Lecture Notes in Computer Science 1413, Berlin, Heidelberg: Springer. Pisa, Italy, June 1998, 1998
- [Rolland et al. 1999] Rolland, C.; Prakash, N.; Benjamin, A.: A Multi-Model View of Process Modelling. In: Requirements Engineering. Volume 4, Number 4., London: Springer Verlag, 1999
- [Rubin 1992] Rubin, K.S.; Goldberg, A.: Object behaviour analysis. In: Communications of the ACM 35.9, 1992, pp. 48-62
- [Rumbaugh 1991] Rumbaugh, J.; Blaha, M.; Premerlani, W.; Eddy, F.; Lorensen, W.: Object-Oriented Modeling and Design. Englewood Cliffs N.J.: Prentice-Hall, 1991
- [Santos et. al. 2006] Santos, I.J.G.; Fluegge, M.; Tizzo, N.P.; Madeira, E.R.M.: Challenges and techniques on the road to dynamically compose web services. In: Proceedings of the 6th international conference on Web engineering, SESSON: Session 2: web services I, New York: ACM Press, 2006, pp. 40 – 47
- [Scheer 1976] Scheer, A.-W.: Produktionsplanung auf der Grundlage einer Datenbank des Fertigungsbereichs, München: Oldenbourg, 1976

- [Scheer 1996] Scheer, A. W.; Jost, W.: Geschäftsprozessmodellierung innerhalb einer Unternehmensarchitektur. In: Vossen, G.; Becker, J. (Hrsg.): Geschäftsprozessmodellierung und Workflow-Management – Modelle, Methoden, Werkzeuge. 1. Aufl.; Bonn, 1996
- [ScheerWerth 2005] Scheer, A.-W.; Werth, D.: Geschäftsprozessmanagement und Geschäftsregeln. In: Forschungsberichte des Instituts für Wirtschaftsinformatik, Heft 183 (Februar 2005), 2005
- [Schlegel 2002] Schlegel, T.: Entwurf und Erprobung eines software-gestützten Verfahrens zur Anwendung software-ergonomischer Methoden in den frühen Phasen der Anwendungsentwicklung, Diplomarbeit DIP-1985, Fakultät Informatik, Universität Stuttgart, 2002
- [Schlegel 2005] Schlegel, T.: Integrating interaction and service workflow models by object-orientation. In: Proceedings of HCI International 2005, CD-ROM, Erlbaum, 2005
- [Schlegel et al. 2004] Schlegel, T.; Burst, A.; Ertl, T.: A flow centric interaction model for requirements specification and user interface generation. In: Proceedings of the 7th international conference on WWCS, bridging diversity at work, Kuala Lumpur: Damai Sciences, 2004
- [SchlegelKarapidis 2004] Schlegel, T.; Karapidis, A.: Wissen und Kreativität in der modellbasierten Dienstleistungsentwicklung – Fragestellungen und Ergebnisse aus dem Vorhaben LIKE. In: Gronau, N.: Wissensmanagement - Wandel, Wertschöpfung, Wachstum : Tagungsband zur KnowTech 2004, München, 18. - 19. Oktober 2004 im Rahmen der Systems 2004, Berlin: GITO-Verlag, 2004, pp. 307-315
- [Schlegel et al. 2008] Schlegel, T.; Beinbauer, W.; Meo, F.: Object-Oriented in Planning and Control of Decentralized Production Systems. In: VIMation Journal on Human-Machine Interaction and Models in Production, Accepted Paper, Issue 1, 2008
- [Schlegel et al. 2007] Schlegel, T.; Thiel, S.; Lotterbach, S.: Produktion 2.0 – Semantic Web Services in der Produktion. In: ZWF Zeitschrift für wirtschaftlichen Fabrikbetrieb, Heft 3/2007, pp. 169-171
- [SchlegelMüller 2005] Schlegel, T.; Müller, K.: Integrating human personnel, robots, and machines in manufacturing plants using ubiquitous augmented reality and smart agents. In: Intelligent Production Machines and Systems, 2005, pp. 189-194
- [SchlegelThiel 2007a] Schlegel, T.; Thiel, S.: Semantic Models and Processes for Information Gathering and Interactive Workflow Execution in Decentralized Production Environments. In: Intelligent Production Machines and Systems – 3rd I*PROMS Virtual International Conference, 2007
- [SchlegelThiel 2007b] Schlegel, T.; Thiel, S.: Information Gathering and Interactive Workflow Execution based on Semantic Models in a Decentralized Production Environment. In: Proceedings of WAMS, 1st Workshop on Advanced Manufacturing Systems, Magdeburg, 2007
- [Schlenoff et al. 1999] Schlenoff, C.; Gruninger, M.; Tissot, F.; Valois, J.; Lubell, J.; Lee, J.: The Process Specification Language (PSL): Overview and Version 1.0 Specification. In: NIST Internal Report (NISTIR) 6459, National Institute of Standards and Technology, Gaithersburg, MD: NIST, 1999
- [Schöf et al. 1995] Schöf, S.; Sonnenschein, M.; Wieting, R.: Efficient Simulation of THOR Nets. In: Proceedings of the 16th International Conference on Application and Theory of Petri Nets, Torino/Italy, 1995, LNCS 935. Berlin: Springer, 1995, pp. 412-431
- [Schöning 1995] Schöning, U.: Logik für Informatiker, 4. Auflage, Heidelberg, Berlin, Oxford: Spektrum Akad. Verlag, 1995

- [Schubert 1996] Schubert, F.: Dynamic Adaptability in Operating Systems by Means of an Object-Oriented Architecture. In: Proceedings of ECOOP'96, Workshop Adaptability in Object-Oriented Software Development, Johannes Kepler Universität Linz, 1996
- [Schuh et al. 2007a] Schuh, G.; Wesch, C.; Koch, S.; Gulden, A.; Gottschalk, S.: Objektorientierte Fabrikplanung – Teil 1: Modellierung modularer Produktionsstrukturen. In: wt Werkstattstechnik online Jahrgang 97, Heft 3, 2007, pp. 166-169
- [Schuh et al. 2007b] Schuh, G.; Wesch, C.; Gulden, A.; Gottschalk, S.: Objektorientierte Fabrikplanung – Teil 2: Architekturen und Vorgehensweise. In: wt Werkstattstechnik online Jahrgang 97, Heft 3, 2007, pp. 170-174
- [Schweitzer 1994] Schweitzer, M. (Hrsg.): Industriebetriebslehre – Das Wirtschaften in Industrieunternehmen, 2. Auflage, München: Vahlen, 1994
- [Schwichtenberg 2003] Schwichtenberg, H.: Universal Component Trading – Dynamisch erweiterbares Trading mit heterogenen Softwarekomponenten. Dissertation, Universität Duisburg-Essen, 2003
- [Shams-AlieeWarboys 1995] Shams-Aliee, F.; Warboys, B.C.: Applying Object-Oriented Modelling to Support Process Technology. In: Proceedings of IDPT 1995, Austin, Texas, 1995
- [Shaofei et al. 2007] Shaofei, W.; Siying, W.; Jinlong, Z.: A service-based Workflow Model for Virtual Enterprise. In: Proceedings of Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD 2007), 2007, pp. 32-36
- [ShenNorrie 1999] Shen, W.; Norrie, D.H.: Agent-Based Systems for Intelligent Manufacturing: A State-of-the-Art Survey. In: International Journal on Knowledge and Information Systems, 1(2), 1999, pp. 129-156
- [Shostack 1982] Shostack, L.: How to Design a Service. In: European Journal of Marketing, Volume 16, Number 1, 1984, pp. 49-63
- [Simon et al. 2006] Simon, C.; Freiheit, J.; Olbrich, S.: Using BPEL processes defined by Event-driven Process Chains. In: Nüttgens, M.; Rump, F.J.; Mendling J. (Hrsg.): EPK 2006 - 5. Workshop der Gesellschaft für Informatik e.V. Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten, 2006, pp. 85-104
- [Snell 2001] Snell, J.: The Web services insider, Part 4: Introducing the Web Services Flow Language, 2001, [online] <http://www-128.ibm.com/developerworks/webservices/library/ws-ref4/>, 21.4.2007
- [Spath 2007] Spath, D.: Wettbewerbsfaktor Digitale Produktion. In: Innovationscluster „Digitale Produktion“: Verknüpfung von realer und digitaler Welt, Nutzenpotenziale erkennen und ausschöpfen, Forum, 20. Juni 2007, Institutszentrum Stuttgart der Fraunhofer-Gesellschaft Stuttgart: Fraunhofer IAO, 2007
- [Spath et al. 2002] Spath, D.; Landwehr, R.; Gönninger, C.: Webserver für Feldgeräte: Vom Prozess direkt ins Web. In: SPS-Magazin, HMI-Special, 2002, pp. 51-52
- [Spath et al. 2006] Spath, D.; Weisbecker, A.; Höß, O. (Hrsg.): Serviceorientierte Architekturen (SOA), Tagungsband des Stuttgarter Softwaretechnik Forums 2006, 8. November 2006. Stuttgart: Fraunhofer IRB Verlag, 2006
- [SpathDemuß 2001] Spath, D.; Demuß, L.: Betreibermodelle für den Maschinen- und Anlagenbau: Chancen und Risiken einer komplexen Kunden-Lieferanten-Beziehung. In: ZWF Zeitschrift für wirtschaftlichen Fabrikbetrieb, Jahrgang 96, Heft 1/2, München: Carl Hanser Verlag, 2001, S. 35-39
- [SpathKoch 2005] Spath, D.; Koch, S.: Das "Mechatronische Betriebssystem". Ein innovativer Befähiger neuer Geschäftsmodelle. In: wt Werkstattstechnik online 95, Heft 7/8, 2005, pp. 577-580

- [SpathKoch 2007] Spath, D.; Koch, S.: Neue Modularisierung für Systemdienstleistungen – Stand der Modularisierung und neue Entwicklungen. In: wt Werkstattstechnik online Jahrgang 97, Heft 7/8, 2007, pp. 498-503
- [SpathSchuster 2004] Spath, D.; Schuster, E.: Neue Geschäftsmodelle durch Betreibermodelle. In: wt Werkstattstechnik online, Jahrgang 94, Heft 7/8, 2004, pp. 319-321
- [Sprinkle 2004] Sprinkle, J.: Model-integrated computing. In: IEEE Potentials, Volume 23, Issue 1, IEEE Press, 2004, pp. 28 - 30
- [StaabStuckenschmidt 2006] Staab, S.; Stuckenschmidt, H. (Hrsg.): Semantic Web and Peer-to-Peer. Berlin, Heidelberg, New York: Springer, 2006
- [Stachowiak 1973] Stachowiak, H.: Allgemeine Modelltheorie. Wien: Springer, 1973
- [StahlVölter 2005] Stahl, T.; Völter, M.: Modellgetriebene Softwareentwicklung. Heidelberg: dpunkt.verlag, 2005
- [Stormer 2003] Stormer, H.: Ein flexibles und sicheres agentenbasiertes Workflow Management System, Dissertation, Wirtschaftswissenschaftliche Fakultät, Universität Zürich, 2003
- [Störrle 2006] Störrle, H.: A Comparison of (e)EPCs and UML 2 Activity Diagrams. In: Nüttgens, M.; Rump, F.J.; Mendling J. (Hrsg.): EPK 2006 - 5. Workshop der Gesellschaft für Informatik e.V. Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten, 2006, pp. 177-188
- [Streekmann et al. 2006] Streekmann, N.; Steffens, U.; Möbus, C.; Garbe, H.: Model-Driven Integration of Business Information Systems. In: Softwaretechnik-Trends 26:4, November 2006, 2006, pp. 9-14
- [Sutton et al. 1995] Sutton, S. M.; Heimbigner, D.; Osterweil, L. J.: APPL/A: a language for software process programming, ACM Transactions on Software Engineering and Methodology (TOSEM), Volume 4, Number 3, 1995, pp. 221-286
- [SuttonOsterweil 1997] Sutton, S.M.; Osterweil, L.J.: The design of a next-generation process language. In: Proceedings of the 6th European conference held jointly with the 5th LNCS Volume 1301, ACM SIGSOFT international symposium on Foundations of software engineering, New York: Springer-Verlag, 1997, pp. 142-158
- [Swenson 1993] Swenson, K. D.: A Visual Language to Describe Collaborative Work. In: Proceedings of the 1993 Symposium on Visual Languages, Bergen, Norway. IEEE Computer Science Press, 1993, pp. 298-303
- [Swenson et al. 2004] Swenson, K.D.; Pradhan, S.; Gilger, M.D.: Wf-XML 2.0 - XML Based Protocol for Run-Time Integration of Process Engines, Workflow Management Coalition, 2004, [online] <http://www.wfmc.org/standards/docs/WfXML20-200410c.pdf>, 23.11.2007
- [Talib et al. 2005] Talib, M.A.; Yang, Z.; Ilyas, Q.M.: A framework towards Web services composition modeling and execution. In: Proceedings of the IEEE IEEE05 international workshop on Business services networks table of contents; Piscataway/NJ: IEEE Press, 2005
- [Terrasse et al. 2006] Terrasse, M.-N.; Savonnet, M.; Becker, G.; Leclercq, E.; Grison, T.: Do We Need Metamodels AND Ontologies for Engineering Platforms? In: Proceedings of the 2006 international workshop on Global integrated model management GaMMa'06, International Conference on Software Engineering, 2006, pp. 21-28
- [Thatte 2001] Thatte, S.: XLANG - Web Services for Business Process Design. 2001 [online] <http://xml.coverpages.org/XLANG-C-200106.html>
- [Thies 2003] Thies, P.: Eine Methode zur deklarativen Modellierung von Koordinationsanforderungen in kooperativen Arbeitsprozessen. Dissertation Universität Stuttgart, Institut für Arbeitswissenschaft und Technologiemanagement, Heimsheim: Jost-Jetter-Verlag, 2003

- [Thomas 2003] Thomas, D.: Unified or Universal Modeling Language? In: Journal of Object Technology, Volume 2, Number 1, Zürich: ETH Zürich, [online] http://www.jot.fm/issues/issue_2003_01/column1.pdf, 2003, pp. 7-12
- [ThomasFellmann 2006] Thomas, O.; Fellmann, M.: Semantische Integration von Ontologien und Ereignisgesteuerten Prozessketten. In: Nüttgens, M, Rump, F.J.; Mendling J. (Hrsg.): EPK 2006 - 5. Workshop der Gesellschaft für Informatik e.V. Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten, 2006, pp. 7-23
- [Thramboulidis 2005] Thramboulidis, K.: Model-Integrated Mechatronics-Toward a New Paradigm in the Development of Manufacturing Systems. In: IEEE Transactions on Industrial Informatics, Volume 1, Number 1, February 2005, pp. 54-61
- [TöpferSommerlatte 1991] Töpfer, A.; Sommerlatte, T.: Technologiemarketing. Verlag Moderne Industrie, Landsberg, 1991
- [Ulieru et al. 2001] Ulieru, M.; Walker, S.S.; Brennan, R.W.: The Holonic Enterprise as a Collaborative Information Ecosystem. In: Proceedings of the 5th International Conference on Autonomous Agents, 2001
- [Ulieru et al. 2002] Ulieru, M.; Walker, S.S.; Brennan, R.W.: The Holonic Enterprise - A Model for Internet-Enabled Global Supply Chain and Workflow Management. In: International Journal of Integrated Manufacturing Systems, Number 13, Issue 8, 2002
- [Valk 1998] Valk, R.: Petri Nets as Token Objects: An Introduction to Elementary Object Nets. In: Desel, J.; Silva, M. (Eds.): Proceedings of the 19th International Conference on Application and Theory of Petri Nets, Lecture Notes In Computer Science Number 1420, 1998, pp. 1-25
- [van Brussel et al. 1998] van Brussel, H.; Wyns, J.; Valckenaers, P.; Bongaerts, L.; Peeters, P.: Reference architecture for holonic manufacturing systems: PROSA. In: Computers in Industry, Volume 37, Number 3, Elsevier Science Publishers B. V., 1998, pp. 255-274
- [Van LeeuwenNorrie 1997] van Leeuwen, E.H.; Norrie, D.H. (1997). Intelligent manufacturing: holons and holarchies. In: Manufacturing Engineer, 76(2), pp. 86-88
- [van Ommering 2005] van Ommering, R. C.: Software Reuse in Product Populations. In: IEEE Trans. Software Eng. 31(7), 2005, pp. 537-550
- [Vanderhaegen et al. 2005] Vanderhaegen, D.; Zang, S.; Scheer, A.-W.: Interorganisationales Geschäftsprozessmanagement durch Modelltransformation. In: Scheer, A.-W. (Hrsg.): Veröffentlichungen des Instituts für Wirtschaftsinformatik, Heft 182, Februar 2005, Saarbrücken, 2005
- [VarróPataricza 2002] Varró, D.; Pataricza, A.: A Unifying Semantic Framework for Multilevel Metamodeling, 2002, [online] <http://citeseer.ist.psu.edu/680376.html>, 22.03.2007
- [VarróPataricza 2003] Varró, D.; Pataricza, A.: UML Action Semantics for Model Transformation Systems. In: Periodica Polytechnica, Volume 47, Number 3, 2003, pp. 167-186
- [Verkoulen 1993] Verkoulen, P.A..C.: Integrated Information Systems Design. Dissertation, Technische Universität Eindhoven, Eindhoven 1993
- [Volz 2006] Volz, S.: Modellierung und Nutzung von Relationen zwischen Mehrfachrepräsentationen in Geo-Informationssystemen. Dissertation Universität Stuttgart, 2006
- [Vyatkin et al. 2005] Vyatkin, V.V.; Christensen, J. H.; Martinez Lastra, J. L.: OOONEIDA: An Open, Object-Oriented Knowledge Economy for Intelligent Industrial Automation, IEEE Transactions on Industrial Informatics, Volume 1, Number 1, February 2005, 2005, pp. 4-17

- [W3C 2002a] W3C: Web Service Choreography Interface (WSCI) 1.0. W3C Note 8 August 2002, 2002, [online] <http://www.w3.org/TR/2002/NOTE-wsci-20020808>, 23.11.2007
- [W3C 2002b] W3C: Web Services Conversation Language (WSCL) 1.0, W3C Note 14 March 2002, 2002, [online] <http://www.w3.org/TR/2002/NOTE-wsci10-20020314/>, 23.11.2007
- [W3C 2003] W3C: Scalable Vector Graphics (SVG) 1.1 Specification, W3C Recommendation, 2003, [online] <http://www.w3.org/TR/2003/REC-SVG11-20030114/>, 23.11.2007
- [W3C 2004a] W3C, Klyne, G.; Carroll, J. (Eds.): Resource Description Framework (RDF): Concepts and Abstract Syntax, W3C Recommendation 10 February 2004, 2004, [online] <http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/>, 21.5.2007
- [W3C 2004b] W3C, Brickley, D.; Guha, R.V. (Eds.): RDF Vocabulary Description Language 1.0: RDF Schema, W3C Recommendation 10 February 2004, 2004, [online] <http://www.w3.org/TR/2004/REC-rdf-schema-20040210/>, 18.3.2007
- [W3C 2004c] W3C, Hayes, P. (Ed.): RDF Semantics, W3C Recommendation 10 February 2004, [online] <http://www.w3.org/TR/2004/REC-rdf-mt-20040210/>, 18.3.2007
- [W3C 2004d] W3C: OWL Web Ontology Language Reference, W3C Recommendation 10 February 2004, [online] <http://www.w3.org/TR/2004/REC-owl-ref-20040210/>, 18.3.2007
- [W3C 2004e] W3C / Horrocks, I.; Patel-Schneider, P.F.; Boley, H.; Said, T.; Grosz, B.; Dean, M.: SWRL: A Semantic Web Rule Language — Combining OWL and RuleML. W3C Member Submission, 2004, [online] <http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/>, 23.11.2007
- [W3C 2004f] W3C / Martin, D. (Ed.): OWL-S: Semantic Markup for Web Services, W3C Member Submission 22 November 2004, 2004, [online] <http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/>, 23.11.2007
- [W3C 2004g] W3C: Web Services Choreography Description Language Version 1.0. W3C Working Draft, 17 December 2004, 2004, [online] <http://www.w3.org/TR/2004/WD-ws-cdl-10-20041217/>, 23.11.2007
- [W3C 2006] W3C: Extensible Markup Language (XML) 1.0 (Fourth Edition). W3C Recommendation 16 August 2006, edited in place 29 September 2006, 2006, [online] <http://www.w3.org/TR/2006/REC-xml-20060816>, 23.11.2007
- [Wagner 2007] Wagner, M.: Entwicklung eines ontologiebasierten Moduls zur Produktionsüberwachung und -steuerung. Diplomarbeit, Institut für Visualisierung und interaktive Systeme, Universität Stuttgart / Fraunhofer Institut für Arbeitswirtschaft und Organisation (IAO), 2007
- [Wamatu 2006] Wamatu, W.M.: Design, Implementation and Test of an Object-oriented Process Modeling Graphical Editor. Bachelor Project Report, Hochschule für Technik Stuttgart / Fraunhofer Institut für Arbeitswirtschaft und Organisation (IAO), 2006
- [Wamatu 2007] Wamatu, W.M.: Design, Implementation and Test of an Object-oriented Process Modeling Graphical Editor. Bachelor Thesis, Hochschule für Technik Stuttgart / Fraunhofer Institut für Arbeitswirtschaft und Organisation (IAO), 2007
- [Wang et al. 2005] Wang, D.; Bastani, F. B.; Yen, I.-L.: Automated Aspect-Oriented Decomposition of Process-Control Systems for Ultra-High Dependability Assurance. In: IEEE Transactions on Software Engineering, Volume 31, Number 9, September 2005, 2005, pp. 713-732

- [WarmerKleppe 2003] Warmer, J.; Kleppe, A.: The Object Constraint Language Second Edition, Getting Your Models Ready for MDA. Pearson Education, 2003
- [Warnecke 1984] Warnecke, H.-J.: Der Produktionsbetrieb. Berlin Heidelberg, New York: Springer Verlag, 1984
- [Weber 2002] Weber, M.: Allgemeine Konzepte zur software-technischen Unterstützung verschiedener Petrinetz-Typen. Dissertation, Humboldt-Universität zu Berlin, 2002
- [Wegner 1987] Wegner, P.: Dimensions of Object-Based Language Design. In: Proceedings of the Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA) 1987 und ACM Special Interest Group on Programming Languages (SIGPLAN) Notices, Volume 22, New York: ACM Press, 1987, pp. 168-182,
- [Weisbecker 2002] Weisbecker, A.: Software-Management für komponentenbasierte Software-Entwicklung. Habilitationsschrift Universität Stuttgart, Jost-Jetter Verlag, 2002
- [WeltyGuarino 2001] Welty, C.; Guarino, N.: Supporting ontological analysis of taxonomic relationships. In: Data & Knowledge Engineering 39 (2001), Amsterdam: Elsevier Science, pp. 51-74
- [Westkämper et al. 2005] Westkämper, E.; Gottwald, B.; Fisser, F.: Migration of the digital and virtual factory to reality. CIRP Journal of Manufacturing Systems, No. 34(5), 2005, pp. 391-396
- [Westkämper 2007] Westkämper, E.: Digitale Produktion. In: Quelle Bullinger, H.-J., Encarnação, J.L., Unbescheiden, M., Nouak, A., Hahn, V.: Technologieführer: Grundlagen - Anwendungen - Trends, Berlin: Springer, 2007, pp. 434-439
- [WestkämperWarnecke 2006] Westkämper, E.; Warnecke, H.-J.: Einführung in die Fertigungstechnik. Stuttgart: Teubner, 7.Auflage, 2006
- [WestkämperSchreiber 2006] Westkämper, E.; Schreiber, S.: Verbesserte Ressourcenausnutzung in der Montage – Planung und Optimierung von Montagesystemen im wandlungsfähigen Unternehmen. In: wt Werkstattstechnik online Jahrgang 96, Heft 3, 2006, pp. 115-121
- [WettkloSchultze 2003] Wettklo, M.; Schultze, A.: ERP-Strategien im Collaborative Business – ERP in der Sackgasse? White Paper, Detecon, 2003, [online] http://download.messe-muenchen.de/media_pub/mediacenter/gb3portale.messe-muenchen.de/systems-world.de/systems-world/business/unternehmensfuehrung/deteconerp.pdf, 23.11.2007
- [WfMC 1999] Workflow Management Coalition: Terminology & Glossary. Document Number WFMC-TC-1011, Issue 3.0, February 1999, 1999, [online] http://www.wfmc.org/standards/docs/TC-1011_term_glossary_v3.pdf, 26.04.2007
- [WfMC 2005] Workflow Management Coalition (WfMC): Process Definition Interface - XML Process Definition Language, Workflow Management Coalition Workflow Standard, Document Number WFMC-TC-1025, Version 2.00, 2005, [online] http://www.wfmc.org/standards/docs/TC-1025_xpdl_2_2005-10-03.pdf, 23.11.2007
- [Wolf et al. 1999] Wolf, T.; Decker, S.; Abecker, A.: Unterstützung des Wissensmanagements durch Informations- und Kommunikationstechnologie. In: Scheer, A.-W.; Nüttgens, M. (Hrsg.): Electronic Business Engineering / 4. Internationale Tagung Wirtschaftsinformatik 1999, Heidelberg: Physica-Verlag, 1999, pp. 746-766

- [WurmusWagner 2000] Wurmus, H.; Wagner, B.: IEC 61499 konforme Beschreibung verteilter Steuerungen mit Petri-Netzen. In: Fachtagung 2000 'Verteilte Automatisierung', 22.-23.3.2000, Magdeburg, Magdeburg: Institut für Automation und Kommunikation e.V., 2000
- [Xu et al. 2005] Xu, Y.; Song, R.; Korba, L.; Wang, L.; Shen, W.; Lang, S.: Distributed device networks with security constraints. In: IEEE Transactions on Industrial Informatics, November 2005, 2005, Volume 1, Issue 4, pp. 217-225
- [Zäh et al. 2006] Zäh, M. F.; Rimpau, C.; Wiedemann, M.; Prash, M.: Geschäftsprozessmodellierung für die kundenindividuelle Auftragsabwicklung. In: wt Werkstattstechnik online Jahrgang 96, Heft 9, 2006, pp. 676-682
- [Zamli 2001] Zamli, K.Z.: Process Modeling Languages: A Literature Review. In: Malaysian Journal of Computer Science, Volume 14, Number 2, December 2001, 2001, pp. 26-37
- [ZamliIsa 2004] Zamli, K.Z.; Isa, N.A.T.: A Survey and Analysis of Process Modeling Languages. In: Malaysian Journal of Computer Science, Volume 17, Number 2, December 2004, 2004, pp. 68-89
- [ZamliLee 2001] Zamli, K.Z.; Lee, P.A.: Taxonomy of Process Modeling Languages. In: Proceedings of ACS/IEEE, International Conference on Computer Systems and Applications, AICCSA 2001, Beirut, Libanon. IEEE, Computer Science Press, 2001
- [ZapfHeinzl 1998] Zapf, M.; Heinzl, A.: Ansätze zur Integration von objektorientierten Konzepten und Petri-Netzen, Arbeitspapier 1/1998. In: Heinzl, A. (Hrsg.): Arbeitspapiere Wirtschaftsinformatik, Universität Bayreuth, 1998
- [ZapfHeinzl 2000] Zapf, M.; Heinzl, A.: Ansätze zur Integration von Petri-Netzen und objektorientierten Konzepten, Wirtschaftsinformatik 42 (2000), Heft 1/2000, 2000

ANHANG A: Grundbegriffe und ihre Herkunft

Die **Ontologie** ist eine Disziplin der Philosophie, die sich mit der Art und Struktur von Objekten, Eigenschaften und anderen Aspekten der Realität beschäftigt. Eine Ontologie wird auch in der aktuellen Anwendung häufig eher als konzeptuelles Modell verwendet: „*A conceptual model is an actual implementation of an ontology that has to satisfy the engineering trade-offs of a running application, while the design of an ontology is independent of run-time considerations.*“ [WeltyGuarino 2001]

Der sehr allgemeine Begriff der Ontologie stammt daher ursprünglich ebenfalls aus der Philosophie. Die Ontologie soll der Klassifikation und Beschreibung von Dingen und deren Beziehungen in einer Diskurswelt dienen, um so ein gemeinsames Verständnis dieser zu erzielen, eine also „*explizite formale Spezifikation einer gemeinsamen Konzeptualisierung (Begriffsbildung)*“ [Gruber 1993]

In der (Computer-)Linguistik (vgl. beispielsweise [Lyons 1995]) wird die **Vererbungsbeziehung** – beispielsweise in einem semantischen Netz – so beschrieben: Ein **Hyponym** U (Unterbegriff eines Begriffs) ist seinem **Hyperonym** O (Oberbegriff) mit einer semantischen Relation (der Hyponymie) untergeordnet. Die Relation Hyponymie ist

- **transitiv**, das heißt ein U untergeordneter Begriff S ist auch O transitiv untergeordnet ($O \rightarrow U \rightarrow S$)
- **nicht reflexiv**, das heißt U kann nicht Unterbegriff seiner selbst sein, kann also in der objektorientierten Vererbung nicht von sich selbst erben und
- **asymmetrisch**, das heißt O kann nicht Unterbegriff von U sein, weil bereits U Unterbegriff von O ist. Für den Vererbungs-Graph ergibt sich hieraus eine azyklische Form. (Ein Kind kann in der Analogie nicht sein eigener Großvater sein.)

Ein Hyperonym kann immer mehrere Hyponyme, ein Hyponym allerdings auch mehrere Hyperonyme haben. Hieraus ergibt sich, dass nicht nur eine **Spezialisierung mehrerer Komponenten** möglich ist (ein Vater hat mehrere Kinder), sondern auch die **Spezialisierung aus mehreren Komponenten** (ein Kind hat mehrere Väter), die der biologischen Vererbung nicht mehr entspricht und meist als **Mehrfachvererbung** bezeichnet wird.

In der Terminologie der Objektorientierung entspricht die Hyponymie der Spezialisierung.

Die UML2 [OMG 2005b] unterscheidet **Aktivitäten** und **Aktionen**. Aktionen sind dabei zu Prozessschritten in anderen Prozessmodellierungsmethoden äquivalent, während Aktivitäten (früher Activity Diagrams [OMG 2004a]) einem vollständigen Prozess(diagramm) entsprechen.

Der in dieser Arbeit beschriebene Ansatz unterscheidet aufgrund gleicher Semantik von atomaren und aggregierten Prozessschritten nicht zwischen beiden Formen. Daher werden Aktionen als atomare (Teil-)Aktivitäten definiert, so dass der Begriff Aktivität in dieser Arbeit sowohl aggregierte als auch atomare Prozessschritte umfasst. Diese Definition ist notwendig da jeder Ansatz unterschiedliche Begriffe verwendet.

Die Microsoft Windows Workflow Foundation (WWF) verwendet ebenfalls den Aktivitätenbegriff für atomare Prozessschritte und bezeichnet vollständige Abläufe als **Workflows**. Eine Aggregation ist in der MS-WWF nicht möglich und führt so zu einem flachen Ablaufmodell, dass direkt an den Code angebunden ist.

Der Begriff der **Metamodellierung** wird sehr unterschiedlich benutzt und weicht vor allem in der Simulation stark von der in dieser Arbeit verwendeten Definition ab, z.B. "*Metamodeling is emerging as a valuable new tool in simulation: complex computer codes can be approximated by surrogate models (analytic, neural network, SVM, etc.) which can easily be evaluated on-the-fly.*" [Hendrickx et al. 2006]

Passend ist die Definition von [VarróPataricza 2002]: „*The term metamodeling usually refers to the definition of language families for various domains of interest. Such a definition must include the specification of its abstract syntax (for instance, by a metamodel), the concrete syntax (preferably by using a graphical language), and its semantics.*“

Ein **Metamodell** stellt eine Formalisierung von Modellzusammenhängen dar, die einer Instanzebene übergeordnet sind (vgl. [Terrasse et al. 2006]), beispielsweise die Formalisierung einer Domänenstruktur. Es umfasst daher die abstrakte Syntax und statische Semantik einer Gruppe von Modellen. [StahlVölter2005]

Die Auswertung der Semantik bereitet jedoch Probleme bei der Definition: Die konkrete Syntax (z.B. Java) ist die Realisierung einer abstrakten Syntax und kann überprüft werden. Die statische Semantik enthält die Regeln und Bedeutungen von Modellen, daher können Fehler bezüglich der statischen Semantik nicht von einem Parser (syntaktische Analyse) sondern erst durch eine statische Analyse erkannt werden. Eine Definition der statischen Semantik kann beispielsweise mit der OCL (Object Constraint Language) erfolgen.

ANHANG B: PlatformPeer-Systemkonzept

Kommunikation im Peer-System

Fehlen für die Vollständigkeit, Interpretation oder Generierung Modellteile, müssen diese bei einem Repository angefordert werden. Hierbei können je nach **Anforderungsmuster** einzelne Komponenten, übergeordnete Teilmodelle (z.B. eine Vererbungshierarchie), ein vollständiges Modell beziehungsweise Diagramm oder die transitive oder invers-transitive⁴⁵ Hülle ab einem bestimmten Punkt – beispielsweise der aktuellen Komponente – angefordert werden. Hierbei muss für jedes Modellelement eine definierte Synchronisationsquelle, das heißt ein Master zur Verfügung stehen, um in einem verteilten, dezentralen System die Modellkonsistenz bei Änderungen sicherzustellen. Beispielsweise wird im Prototyp mit jedem Element neben seiner ID eine Synchronisationsadresse (Referenzquellensignatur) verknüpft werden, von der dieses Element bei Bedarf geladen werden kann. Ein Peer kann dann nur diejenigen Elemente liefern, für die er selbst der Master ist. Andere Elemente können quasi nur als Cache weitergegeben werden und müssen gegebenenfalls vom Master nachgeladen werden, sofern dieser verfügbar ist.

Jeder Peer kann bei einer Quelle ein **Abonnement für Modelländerungen** eintragen und wird so bei Änderungen aktualisiert. Hierdurch entfällt ein aufwändiges Pollingverfahren, bei dem andauernd Änderungen abgefragt werden müssten. Eine mit weniger Aufwand verbundene Methode ist die Nutzung von **Listern mit Broadcasts**, so dass bei einer Änderung an alle Peers ein Broadcast der Änderung versandt wird, der nur von den mit einem entsprechenden Listener versehenen Peers auch tatsächlich angenommen wird.

Um in einem Repository für ein verteiltes, dezentrales System möglichst vollständige und aktuelle Modelle zu erhalten ist neben dem hier beschriebenen, nachrichtenbasierten System mit Abos für Modelländerungen oder Listern mit Broadcasts auch ein Ansatz ähnlich der Suchmaschinen-Robots im Web möglich. Ein solcher **Repository-Agent** könnte dann ausgehend von definierten Modellpunkten das Netzwerk nach neuen oder veränderten Modellkomponenten durchsuchen. Der Repository-Agent aktualisiert so „sein“ Teilmodell ständig und kann es registrierten oder anfragenden Agenten zur Verfügung stellen. Allerdings ist dieses System nur mit einer Ausgestaltung als Polling-System (Abfragetabelle) annähernd so zuverlässig wie das (deterministische) Abo-System und liefert zu keinem Zeitpunkt ein konsistentes Modell: Manche der abgefragten Änderungen sind neuer, manche älter, je nach Weg des Robots.

Cache-Mechanismen

In einem Peer-to-Peer-System für OMICRON sind unterschiedliche Cache-Mechanismen möglich, um Informationen lokal zwischenspeichern:

NoCache: Wird keine Interpretation des Modells benötigt, da beispielsweise nur spezifische, bereits festgelegte Daten ausgelesen werden, und sind daher keine oder wenige Zugriffe (mit guter Anbindung) auf das Gesamtmodell notwendig, kann der Cache entfallen.

ComponentCache: Einmal genutzte Modellinformationen werden zwischengespeichert bis der Puffer voll ist und daher das älteste Element entfernt werden muss. Allerdings wird nur das am längsten nicht mehr benutzte Element entfernt, so dass ein bereits im Cache enthaltenes Element bei erneuter Benutzung eine neue Position erhält und wieder am längsten verbleiben darf.

⁴⁵ invers-transitiv bedeutet hier nicht die Inverse der transitiven Hülle bezüglich eines modell-vollständigen Graphen, sondern die transitive Hülle in der Gegenrichtung, beispielsweise in der Vererbung statt allen (transitiven) Kindern alle (transitiven) Eltern, also die Gegenrichtung zur Vererbungshierarchie.

MetaCache: Auf größeren Teilsystemen oder Peer-Verbänden kann das gesamte Metamodell zur einfacheren Interpretation, beispielsweise in einer Modellierungsumgebung, zwischengespeichert werden. Es müssen dann nur Modelle und Instanzen nachgeladen werden. Neben dem Meta-Cache ist eine Ergänzung durch einen ComponentCache daher meist sinnvoll. Im **ModelCache** wird in Ergänzung zum MetaCache die Modellebene oder ein Ausschnitt dieser repliziert.

CacheAll: Hier wird das gesamte Repository repliziert und bei Veränderungen beispielsweise mit Hilfe von empfangenen Broadcasts oder Abonnements aktualisiert.

Alle Modi müssen bei Modelländerungen benachrichtigt und aktualisiert werden. Dies kann über ein Entfernen aus dem Cache mit oder ohne anschließendem Neuladen geschehen. Mit jeder Komponente müssen auch deren Relationen geladen werden, da für den Peer sonst unklar ist, welche weiteren Komponenten bei Bedarf nachgeladen werden müssen.

Hierzu bestehen zwei Varianten. Die einfachere, im Prototyp verwendete, ist die reine Zwischenspeicherung der Relationen. Sollen diese allerdings im Modell interpretierbar bleiben, kann auch hier als zweite Variante das Konzept der Sentinels eingesetzt werden: Zu einer Relation ohne im Cache verfügbarer zweiter Komponente wird gleichzeitig eine Netzkomponente erzeugt, die bei Bedarf ein Nachladen auslösen kann. Eine solche **Netzkomponente** ermöglicht die Verteilung des Komponentensystems und den Versand von Komponenten (meist Instanzen) als Modellfragmente über das Netzwerk eines verteilten Komponentensystems. Netzkomponenten repräsentieren eine Komponente, die sich nicht mehr auf dem System befindet, referenzieren aber diese eindeutig, so dass beispielsweise über eine Typ-Tabelle (Virtuelle Typ Tabelle, VTT) eine Typ-Prüfung durchgeführt werden kann. Sie stellen damit Randpunkte des in einem Peer oder einer Nachricht enthaltenen Modells dar, beispielsweise die Typkomponente einer Instanz. Wird eine vollständige Information benötigt, muss die Netzkomponente zusammen mit ihren Teilkomponenten (anhand der ID) nachgeladen werden.

ANHANG C: Prozess-Metamodelle

“The focus has been to increase the level of formality of process models in order to make possible their enactment in process-centred software environments” [Rolland et al. 1999].

Ein Prozess-Metamodell wird häufig beschrieben als *“a description at the type level of a process model. A process model is, thus, an instantiation of a process meta-model. [...] A meta-model can be instantiated several times in order to define various process models. A process meta-model is at the meta-type level with respect to a process.”* [Rolland 1998]

Prozess-Metamodelle stellen also die Typebene von modellierten Prozessen (Prozessmodellen) dar. Tatsächlich ausgeführte Prozesseinstanzen befinden sich auf der Instanzebene. Ein Meta-Metamodell kann daher Regeln, Strukturen und Beschreibungsmechanismen für alle Prozess-Metamodelle (Prozessmodelltypen) definieren. Hieraus ergibt sich eine – teilweise auch relativ zu betrachtende – Hierarchie:

1. Meta-Metamodell
2. Prozess-Metamodell
3. Prozess(modell)
4. Prozessinstanz / Projekt

Teilweise werden in der Literatur auch die Prozessinstanzen als Prozesse bezeichnet, die dann Instanzen eines Prozessmodells darstellen: *„Processes of the same nature are classified together into a process model. Thus, a process model is a description of a process at the type level. Since the process model is at the type level, a process is an instantiation of it. The same process model is used repeatedly for the development of many applications and thus, has many instantiations. One possible use of a process model is to prescribe how things must/should/could be done in contrast to the process itself which is really what happens. A process model is more or less a rough anticipation of what the process will look like. What the process shall be will be determined during actual system development”* [Rolland 1998]

Prozesse werden in dieser Arbeit jedoch immer auf der Typebene angesiedelt.

Ein Großteil der vorliegenden Literatur geht davon aus, dass Prozesse/Projekte an sich existieren und nun ein Abstraktionsschritt vorgenommen wird, um diese einer Klasse von Prozessen zuzuordnen, wodurch ein Metamodell entstehen soll. [AtkinsonKühne 2002] zeigen jedoch, dass ein Metamodell auf einer übergeordneten Instanzebene eine typologische Entsprechung benötigt. Ein konsistentes Metamodellierungskonzept ist daher ein ausgeprägtes Defizit existierender Prozess-(Meta-)Modelle.

ANHANG D: Entwicklung der Metamodelle

Geschichte wiederholt sich zu einem gewissen Teil. – Dem kann sich auch die Forschung nicht ganz entziehen. In Form des Transfers bewährter Methoden auf andere Bereiche kann sie sogar davon profitieren.

In der Softwareentwicklung wurde zunächst Maschinencode direkt eingegeben, später mit Hilfe von Assemblern aus einem Quellprogramm übersetzt – die erste Abstraktionsstufe hin zu einer modellbasierten Entwicklung war genommen.

In der Folgezeit wurden neben Makroassemblern Hochsprachen entwickelt, deren Befehle nicht mehr direkte Pendanten zum Maschinencode waren, sondern einer höheren Abstraktionsebene entsprachen. Prozedurale Sprachen ließen zudem den beliebig häufigen, parametrisierten Aufruf von ganzen Prozeduren zu, was zu einer weiteren Abstraktion beitrug. Neben einer noch stärkeren Modularisierung auf höherer Ebene durch sogenannte Sprachen der vierten Generation (4th Generation Programming Languages oder 4GL⁴⁶) hielt mit der Objektorientierung ein Paradigma Einzug, das nicht mehr nur wie bisher neue Abstraktionsebenen sondern eine andere „Weltansicht“ einführte, die sich naturgemäß stärker auf Daten und Statik als dynamische Abläufe konzentriert: Hier tauschen durch ihre Daten charakterisierte Objekte Nachrichten aus.

Bei den Modellierungssprachen im Software Engineering verhielt es sich ähnlich. Ablauforientierte Ansätze wie SA/SD und SADT wurden durch eher objektorientierte Ansätze wie die UML und deren Vorläufer ersetzt. Mit der Entwicklung von UML zum Modellierungsstandard der Softwareentwicklung trat jedoch auch die Tatsache in den Vordergrund, dass UML nur semi-formal festgelegt und teilweise semi-informal⁴⁷ beschrieben ist (vgl. „Semantics“ Abschnitte in [OMG 2007a][OMG 2007b]). Meist finden in der Praxis nur die Diagramme als grafische Notation, weniger jedoch eine integrierende Modell-Semantik und das Metamodell Beachtung. Mit UML 2.0 wird daher der Trend zu Modellintegration bewusst gestärkt:

“Recent initiatives (UML 2.0 RFP) have aimed at evolving UML into a core kernel language, and a family of distinct languages (called profiles) where each profile has its own semantics, which architecture fundamentally requires an appropriate and precise metamodeling technique.” [VarróPataricza 2002]

Generell wird an der UML in allen Versionen häufig die fehlende Formalisierung kritisiert, die eine automatische Auswertung von Modellen und deren Abhängigkeiten durch eine hohe Generalisierbarkeit erschwert: *“The lack of reliable set semantics and model theory for UML prevents the use of automated reasoners on UML models. Such a capability is important to applying Model Driven Architecture to systems integration. A reasoner can automatically determine if two models are compatible, assuming they have a rigorous semantics and axioms are defined to relate concepts in the various systems.”* [Patel-Schneider et al. 2004]

„UML Profiles and the associated MOF effectively make UML completely open-ended in terms of what it is and can describe.“ [Thomas 2003]

Die hier erwähnte Meta Object Facility (MOF) [OMG 2006b] basiert auf der OMG Object Management Architecture (OMA), die eng mit der Common Object Request Broker Architecture

⁴⁶ sogenannte Sprachen der vierten Generation legen meist eine Sprache zugrunde, mit der sich einfach und schnell komplexe Anwendungsfunktionen eines bestimmten Typs erzeugen lassen – in der Praxis häufig datenbankbasierte Systeme. Unter Sprachen der dritten Generation (3GL) werden meist die gängigen „Hochsprachen“ wie C++, Java etc. eingeordnet.

⁴⁷ Siehe [Gomez-Perez et al. 2004]: Highly informal = natürlichsprachlich; semi-informal = eingeschränkte, strukturierte natürliche Sprache; semi-formal = künstliche, formale Sprache; rigorously formal = Semantik, Theorem bewiesen

(CORBA) [OMG 2004b] verbunden ist. Sie wurde als Meta-Metamodell und „oberste Schicht“ eine wichtige Grundlage für UML. [AtkinsonKühne 2002]

Der Fokus von MOF lag daher eher auf der Inspection⁴⁸ als auf der Metamodellierung, was sich auch aus der Verschmelzung von CDIF (vgl. [Chen 1993]) mit MOF erklärt.

Die Verwendung der Object Constraint Language (OCL, [WarmerKleppe 2003]) als Regelsprache soll daher zumindest die Semantik in Form von Bedingungen beschreiben, wie dies in vergleichbaren Ansätzen (vgl. [VarróPataricza 2002]) wie dem formalisierten⁴⁹ MetaModeling Framework (MMF, [Clark et al. 2001]) geschieht, das eine Untermenge der „alten“ UML vor 2.0 darstellt.

Fehlende Beweisbarkeit und Definition der MOF zur Metamodellierung wurden häufig kritisiert und auch erste Ansätze einer Formalisierung, beispielsweise mit der „Constructive Type Theory“ vorgeschlagen (vgl. [Poernomo 2006]).

Die MOF bringt aufgrund ihrer Historie viele Probleme hinsichtlich der Metamodellierung mit sich: *“Although the term MOF has its origins in the OMG’s Object Management Architecture as a repository for dynamically accessible metainformation and for supporting model interchange in the first four versions of the UML, it quickly became associated with (i.e., treated as being the same as) the meta-metamodel at level M3. Unfortunately, these two interpretations are not compatible in the original linear metamodeling framework. If the MOF is identified with the M3 meta-metamodel then it can neither act as a reflection interface nor provide model interchange support for all the levels below (M2 to M0), unless one abandons the concept of a strictly linear modeling framework. [...] In the original one-dimensional, linear modeling framework, this would have required the MOF to manifest itself at every level (except the bottommost), since it needs to provide interfaces (an API for tools to gain metainformation) to elements at all levels.”* [AtkinsonKühne 2002]

Bei den **Prozessmodellen der Softwareentwicklung** ist eine ähnliche Entwicklung zu beobachten: Der Anarchie der Anfangszeit folgten dann gegenläufig starre Modelle wie das Wasserfall- oder Phasenmodell und später auch das V-Modell, die alle einen sequentiellen, rückbezüglichen Ablauf beschrieben und so Planbarkeit sichern sollten. Ernüchtert durch spät im Projekt auftauchende, teure Fehler wurden iterative und inkrementelle Verfahren wie das Spiralmodell ([Boehm 1988]) und Rational Unified Process (RUP) zusammen mit Prototyping-Ansätzen propagiert. Dem immensen Overhead durch hohe Dokumentationsaufwände dieser sogenannten „schwergewichtigen Verfahren“ wurden in der Folge leichte („light-weight“) beziehungsweise agile Verfahren wie Extreme Programming (XP, [Beck 2000]) gegenübergestellt, die sich weniger auf Formalismen und Artefakte (Dokumentation) als auf soziale und projektstrukturelle Zusammenhänge sowie Benutzer oder Kunden konzentrieren. Zudem wurde das V-Modell als V-Modell XT (extreme tailoring) flexibilisiert. Auch hier ist also eine Verlagerung von starren, funktionalen Schritten auf Flexibilisierung und inhaltliche Strukturen zu beobachten.

Software-Prozesse werden immer mehr auch mit den Mitteln der Metamodellierung erfasst, so dass sie teilweise automatisierbar und instanzierbar werden. Jedoch beschränken sich diese Modelle meist auf reine Statik (Typologie) oder verlieren die vollständigen Möglichkeiten der Objektorientierung auf dem Weg zur Dynamik/Ablauforientierung, da Ablaufaspekte nicht mit der Komponentenorientierung vereinigt werden.

⁴⁸ Abfrage von Eigenschaften einer Komponenten beziehungsweise eines Objekts über entsprechende Abfrage- und Beschreibungsmechanismen

⁴⁹ auf Basis der Objekttheorie [AbadiCardelli 1996]

Weitgehend unabhängig vom Vorgehens- oder Prozessmodell, wurde das Problem der Verwendung und Konsistenz von Modellen über die Projektlaufzeit deutlich. Während schwergewichtige Modelle die Modellierung und Konsistenzprüfung für Entwicklerdokumente favorisieren, setzen agile Methoden auf eine Reduktion der Dokumentation, um den Code als einziges oder Haupt-Modell zu nutzen. Da ersteres zu hohem Aufwand und Inkonsistenzen bei Dokumentation und Modellen führte, agile Methoden aber häufig nicht mehr nachvollziehbare Ergebnisse lieferten, sollte eine möglichst enge Verzahnung von Modellen und Code mit Hilfe **generativer Verfahren** ermöglicht werden.

Diese wurden zunächst auf bestimmte Entwicklungsdomänen wie den Benutzungsschnittstellen-Bereich fokussiert, um für diese die zur Generierung notwendige Quellmodell-Vollständigkeit zu erreichen. Die Verwendung proprietärer oder aus der Erweiterung von Standards entstandener Ausgangsmodelltypen soll so die generative Erzeugung von Benutzungsschnittstellen ermöglichen. Problematisch bei allen generativen Verfahren ist jedoch immer die Notwendigkeit, Auswirkungen von Änderungen am generierten Artefakt nachzuvollziehen sowie deren Weiterbestehen im Fall einer neuerlichen Generierung sicherzustellen. Mit so genannten „Roundtrip“-Verfahren soll deshalb eine Rückkopplung möglich werden, die Ausgangs- und Zielmodell in direkte Beziehung setzt und so Änderungen auf beiden Seiten sowie deren Implikationen auf die andere Seite (teil-)automatisiert behandeln kann. Wenngleich vollständige Ansätze nach Kenntnis des Autors noch nicht existieren, werden Teilbereiche beispielsweise im Rahmen von MDA-Werkzeugen bereits abgedeckt.

Für die Prozessmodellierung in der Produktion stellt sich das Problem in gleicher Weise. Allerdings wurde hier bisher kaum spezifische Forschung betrieben. Werden Prozesse dynamisch verändert, müssen auch zugehörige Abläufe und Benutzungsschnittstellen angepasst werden. Zunehmende Flexibilisierung führt also schnell zur Erreichung der Wartbarkeitsgrenze und somit zur Forderung nach einer dynamischen Erzeugung informationstechnischer Komponenten in der Produktion, die sich der veränderten Realität modellgebunden anpassen sollen.

ANHANG E: Kritik und Erweiterung des Vier-Ebenen-Modells

Die OMG führt für die Meta Object Facility (MOF, [OMG 2006b]) beziehungsweise Unified Modeling Language (UML, [OMG 2007b]) eine Restriktion auf vier Abstraktionsebenen ein, die die Metamodellierungskonzepte mit den Grundsätzen aktueller objektorientierter Programmiersprachen verbindet. Hier werden vier allgemeine Ebenen genutzt:

- Meta-Metamodell (M3)
- Metamodell (M2)
- Modell (M1)
- Instanz (M0)

Aufgrund der Ausrichtung auf die Softwareentwicklung zeigen sich viele Konzepte als zu stark auf diese Domäne fokussiert (vgl. Kapitel zu Prozessmodellierungstechnologien). Das Metamodell gibt die eigentlichen Konzepte und Diagramme der UML vor. Nur bedingt werden diese durch ein Meta-Metamodell definiert, für das die MOF vorgesehen ist. Die eigentlichen UML-Modelle werden auf M1 erstellt. Instanzen werden erst zur Laufzeit erzeugt, wodurch Objektdiagramme semantisch eigentlich Ebene M0 angehören müssten. Es existiert jedoch für diese ein Metamodell auf M2, so dass Objektdiagramme ebenfalls auf M1 zu liegen kommen. Das Konzept der Instanzebenen wird also in UML nicht stringent eingehalten. Für die Softwareentwicklung ist meist das Klassendiagramm die Entsprechung zur Typebene des Codes.

Für Prozessmodelle adaptiert kann diese Hierarchie in der OMG Terminologie wie folgt dargestellt werden [OMG 2006e]:

- M3: Meta Object Facility (MOF)
- M2: [OMG 2000], UML [OMG 2005b], Meta-Prozessmodelle [Rolland 1993]
- M1: Prozessmodelle wie RUP [Haumer 2005]
- M0: „Ablaufender Prozess“⁵⁰

Angewandt auf das Szenario der Produktionsprozesse ergibt sich aus dem Vier-Ebenen-Modell der OMG beispielhaft die folgende Aufteilung von Ebenen.

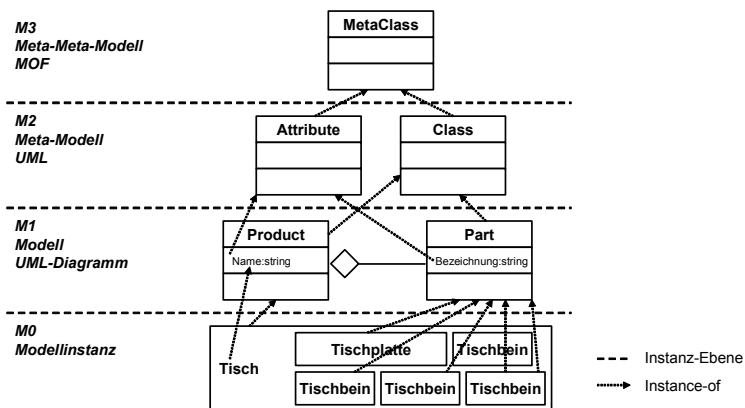


Abbildung 80: Meta-Ebenen des UML-Metamodells, angewandt auf die Produktion

⁵⁰ In der OMICRON Terminologie: Prozessinstanz beziehungsweise Projekt

[Müller 2007] konstatiert: „Die M0-Ebene ist keine Modellebene im eigentlichen Sinne, da sie nur während der Ausführung eines Modells der M1-Ebene existiert und daher vorab nicht modelliert werden kann.“ Jedoch gilt dies bei Laufzeitmodellen nicht und auch dann nicht, wenn mehr als vier Instanzebenen existieren, das heißt M0 ihrerseits Modellebene ist.

In der UML sind alle Klassen Instanzen von UML::Class, die ihrerseits Instanzen von MOF::Classifier sind. MOF definiert also UML. Zur Begrenzung der Instanziierungshierarchie⁵¹ wird jedoch bei MOF die Hierarchie durch Selbstbezüglichkeit von MOF abgebrochen – das Meta-Metamodell ist Instanz seiner selbst und beschreibt sich selbst. Daher werden Beschreibungsmittel der UML zur Definition von MOF eingesetzt. – Der Zirkelschluss ist perfekt.

Die UML definiert die Klasse (S. 5 [OMG 2005a]) als *“A classifier that describes [sic!] of a set of objects that share the same specifications of features, constraints, and semantics.”* Diese Definition greift zu kurz. In einem mehrschichtigen Metamodell ist die Klasse selbst eine Entität, auch weil eine Dualität Typ-Instanz beziehungsweise Klasse-Objekt nur auf einer dedizierten Ebene festzulegen ist (wie M1 und M0 in UML beziehungsweise MOF): Diese Unterscheidung führt nach [AtkinsonKühne 2002] zu

1. einer künstlichen Unterscheidung zwischen Instanzen von Modellelementen auf unterschiedlichen Ebenen und
2. einer Festlegung der Anzahl von Meta-Ebenen nur nach Verfügbarkeit entsprechender physischer Classifier.

Bei mehreren Instanziierungsebenen – wie sie semantisch auch in UML / MOF mit den Ebenen M3 bis M0 vorkommen – ist die Instanz m1 auf M1 einer Klasse m2 auf M2 eigentlich gleichzeitig selbst wieder Klasse beziehungsweise Typ der Instanz m0 auf M0. Nur so ist ein durchgängiges Metamodellierungskonzept realisierbar, das ohne zusätzlich Konstrukte wie Modelltypen auskommt. Die Problematik erklärt sich aus der Herkunft der objektorientierten Modellierung:

„[...] all the current metamodeling approaches have fundamental problems concerning their instantiation mechanism. These problems originate in the fact that all these approaches evolved from object-oriented modeling, which traditionally had only two metalevels, the object level (M0) and the class level (M1) where every element at the M0 level is an instance of exactly one M1, element.“ [VarróPataricza 2002]

[AtkinsonKühne 2002] schlagen daher vor, die Unterscheidung zwischen Objekt, Klasse und Metaklasse aufzuheben und nur die oberste und unterste Instanziierungsebene speziell zu behandeln. DasOMICRON Metamodell folgt diesem Konzept mit der Definition des Meta-Metamodells, kann jedoch prinzipiell ohne eine definierte unterste Ebene auskommen und legt auch für die Meta-Ebene eine besondere Behandlung für Relationstypen fest (vgl. Kapitel zu Relationstypen).

⁵¹ Die UML benutzt „verschiedene Instanziierungen“ ([StahlVölter2005], S. 127) – Instanziierung vom Meta-Modell aus ist nebenläufig zur Instanziierung innerhalb eines Diagramms: Klassen sind Instanzen der UML::Class und deren Objekte sind Instanzen von UML::Object, zugleich aber auch Instanzen der definierten Klassen. Dieses Problem tritt durch eine Explizierung der Vererbungsvorgänge für die Diagramme auf (UML::Object ist Instanz von UML::Class). Derartige Inkonsistenzen sind für ein dynamisches Meta-Modell nicht hinnehmbar, so dass in OMICRON durch die Vererbung dieser Zusammenhang bereits implizit klar ist und Inkonsistenzen vermieden werden.

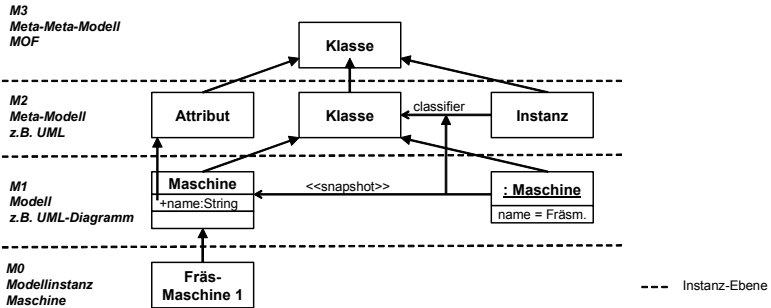


Abbildung 81: Beispiel einer Vier-Ebenen-Hierarchie in Anlehnung an [OMG 2007a]

Dieses Beispiel zeigt ein grundlegendes Problem des Vier-Ebenen-Modells. Instanziierungen einer Klasse in M1 über mehrere Ebenen sind nicht möglich. Der Snapshot im Beispiel sollte eigentlich eine Instanz Maschinentyp des Meta-Konzepts Maschine sein, das Attribute *name* mit einem Wert belegt werden. Diese könnte wieder als konkrete Maschine in der Fabrik instanziiert werden – wenn das Metamodellierungskonzept nicht auf 4 Ebenen beschränkt wäre.

In der Literatur zur Metamodellierung werden drei Ansätze beschrieben, die eine über UML hinausgehende, vielschichtige und konsistente Metamodellierung ermöglichen sollen:

Powertypes [AtkinsonKühne 2002] erzwingen für ihre Instanzen die Ableitung von einer bestimmten Instanz: „*A powertype is a type the instances of which are subtypes of another type (called the partitioned type).*“ [Henderson-SellersGonzalez-Perez 2005b]. Dieses Konzept führt zu einer künstlichen Beschränkung von Instanzen als Subtypen einer bestimmten Instanz und ist für das Komponenten-Relationen-Modell wenig sinnvoll, in Bezug auf die Verteilung untergeordneter Instanzebenen sogar gefährlich.

Das Powertype-Konzept setzt voraus, dass es gewünscht ist, für einen Typ jeder Ebene vorzuschreiben, dass alle Instanzen (der nächst-niedrigeren Ebene) von derselben Komponente erben müssen. Dies führt zu einem inkonsistenten beziehungsweise beschränkten Modell, da dann nur genau eine Instanz Vater aller ererbenden Instanzen ist. In OMICRON können jedoch alle Instanzen gleichwertig verwendet werden und müssten auch nicht von einer bestimmten erben. Vielmehr ist der Normalfall eine Instanz, die nicht von einer anderen Instanz erbt. Sollen wie im obigen Beispiel Attribute definiert werden, so können diese über das Konzept der Konstanten auf der Meta-Ebene erzeugt werden.

Potency und Deep Instantiation (formalisiert in [VarróPataricza 2002]) ermöglichen die Definition von Regeln, Attributen und Zusammenhängen bereits auf weiter übergeordneten Klassen-/Instanzebenen. Ein Potenzwert gibt an, über wie viele Instanziierungen hinweg diese gültig sind und somit konstant bleiben – eine wirkliche Instanziierung von Bestandteilen wird so unterbunden. Hierbei handelt es sich ebenfalls um ein künstliches Konzept, das die willkürliche Definition von Wirkungsreichweiten ermöglicht und so dem eigentlichen Gedanken der Instanziierung und Metamodellierung widerspricht.

Strict Metamodelling (strikte Metamodellierung, vgl. [AtkinsonKühne 2002]) erfordert, dass in einer n-schichtigen Architektur mit den Ebenen $M_0..M_{n-1}$ jedes Element der Ebene M_m eine Instanz genau eines Elements der Ebene M_{m+1} ist (mit $0 \leq m \leq n-1$), wobei M_0 die unterste Ebene darstellt. Eine Relation zwischen zwei Elementen X und Y impliziert dann automatisch $level(X)=level(Y)$.

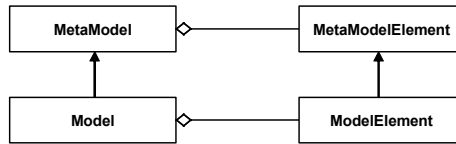


Abbildung 82: Strict Metamodeling in Anlehnung an [AtkinsonKühne 2002]

Ohne Strict Metamodeling kollabiert das Gesamtmodell sonst zu einer oder zwei Ebenen mit unterschiedlichen Konzepten für jede Ebene ähnlich wie in UML [AtkinsonKühne 2001].

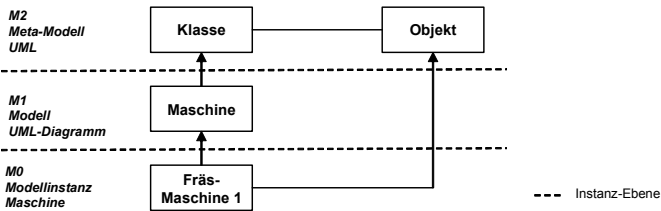


Abbildung 83: Konzept der UML für die Instanziierung nach [AtkinsonKühne 2002]

Es ist also sinnvoll eine strikte Meta-Modellierung zu benutzen, um Modelle nach Ihrer Ebene unterscheiden zu können. OMICRON dehnt dieses Konzept dabei auf die Relationen aus, übernimmt für alle Komponenten aber das Konzept, dass Meta-Ebenen nur von der Instanziierung durchbrochen werden, da sonst keine Zuordnung von Elementen und Regeln mehr möglich wäre.

Allerdings merken [AtkinsonKühne 2002] an: „*This definition deliberately rules out the top level in a hierarchy of levels, since in practice one often wants to terminate the hierarchy of metalevels. A common approach is to model the top level so that its elements can be viewed as instances of themselves. In terms of the model-level instance-of relationship, this is described as a model being an instance of itself.*“

Aus diesem Grund invertiert OMICRON die Richtung der Definition. Von einer untersten Ebene muss nur ausgegangen werden, wenn eine reine Klassenbildung angestrebt wird. In OMICRON muss dagegen immer ein interpretierbares, in Software umzusetzendes Meta-Modell existieren, bevor ein Modell instanziiert werden kann. In einer n-schichtigen Architektur mit den Ebenen $M_0..M_{n-1}$ stellt die Ebene M_{n-1} das Meta-Metamodell dar, von dem ausgehend alle Elemente transitiv instanziiert werden müssen. Jedes Element ist dabei Instanz genau eines übergeordneten. Das oberste Element (*ModelElement*) ist dann Instanz seiner selbst.

Unterhalb der durch das *ModelElement* realisierten **Cut-Off-Grenze** nach oben besteht so nach wie vor die Möglichkeit einer beliebig tiefen Instanzierungshierarchie unter der Voraussetzung, dass die Instanzierungskonzepte – wie für OMICRON beschrieben – für alle Ebenen dieselben sind.

Neue Modelle und das UML-Konstrukt der Stereotypen

Stereotypen werden in der UML genutzt, um in einem UML-Profil andere „Klassen“/Typen von Klassen, das heißt abgeleitete Meta-(Sub-)Klassen zu definieren [Jeckle et al. 2004]. Dieser Ansatz ist für eine strikt objektorientierte Vorgehensweise inkonsistent. Daher wird in

OMICRON auf das Konstrukt der Stereotypen verzichtet und stattdessen auf der Meta-Modellebene eine Vererbung und Typologie eingeführt, die die Stereotypen durch konsistente Unterklassen ersetzt. Dies hat zudem den Vorteil, dass alle verwendeten Klassen durch ihren Meta-Typ über eine eindeutige, modellinherente Semantik verfügen, die nicht erst schlüsselwortabhängig interpretiert werden muss. Auch [AtkinsonKühne 2002] weisen nach, dass Stereotypen bei korrekter Meta-Modellierung unnötig sind.

Stereotypen werden in UML-Profilen meist als Ableitung von *Class* oder *Attribute* gebildet. Quasi handelt es sich damit ebenfalls um eine Vererbung im Metamodell, wobei sich die von den Stereotyp-Metaklassen „instanziierten“ Klassen in einem UML-Diagramm entweder nur hinsichtlich der Stereotypenbezeichnung oder auch durch einschränkende Regeln – meist in OCL [WarmerKleppe 2003] – unterscheiden. Ein anschauliches Beispiel für die Problematik der Metaklassen und Stereotypen bietet das OMG Software Process Engineering Metamodel (SPEM) [OMG 2006e].

ANHANG F: Repräsentation von Teilkomponenten bei komplexer Vererbung

„For the assembly technique to be successful, it is necessary that process models are modular. If the assembly technique is combined with the instantiation technique then the meta-model must itself be modular.“ [Rolland 1998]

Erbt eine Komponente B von einer komplexen Komponente A sind für das Modell unterschiedliche Repräsentationsformen möglich.

Das **Vererbungsdifferenzial** führt zunächst nur zur Erstellung ein Komponente B. Werden Teile von A in B verändert, wird die entsprechende Komponente erstellt und referenziert dann die passende Teilkomponente von A, was sogar mit einer reinen (Teilelement-)Vererbung funktionieren würde. Es werden also nur Komponenten erzeugt, die vorhandene überschreiben, löschen oder ergänzen. Alle anderen Teilkomponenten werden nur bei Bedarf projiziert. Diese Methode ist die speichereffizienteste der beschriebenen Möglichkeiten. Ein Nachteil ist die schwierige Referenzierung der Teilkomponenten der Kind-Komponente beispielsweise durch Instanzen ebenso wie die Erkennung des Änderungshorizonts, das heißt der Auswirkungen von Veränderungen auf andere (beispielsweise erbende) Komponenten. Problematisch ist zudem, dass zunächst auch alle an eine veränderte Relation angrenzenden Komponenten ebenfalls dupliziert werden müssen, da sonst unklar wäre, dass die Änderung in B und nicht in A vorgenommen wurde. Erbt eine Komponente C von B, so müssten für Änderungen in C

- a) entweder auch in B die Komponenten angelegt werden oder
- b) diese direkt mit A verbunden werden.

Werden im Fall b) auch Änderungen in B vorgenommen, können Inkonsistenzen auftreten, da diese sich möglicherweise auf die vorangegangenen Änderungen in C auswirken (sollten). Im Fall a) existiert eine unechte Vererbung oder Referenz in B. Die Veränderung oder Verwendung von Relationen in einer Spezialisierung C kann bei Verbindung mit A weitere Inkonsistenzen begünstigen.

Die **Änderungsabdeckung (Change Layer Coverage)** soll das Problem des übergeordneten Änderungskontexts lösen. Es wird nicht mehr das reine Differenzial, das heißt nur exakt die Änderungen, in der Klasse gespeichert, sondern bezüglich der Änderungen auch die „höher“ gelegenen Aggregationsebenen, durch welche die geänderte Komponente mit ihrer Klasse auf höchster Ebene verbunden ist. Alle Änderungen können so durch eine Tiefen-Traversierung der direkt untergeordneten Komponenten erfasst werden. Da jedoch wiederum nicht die mit der geänderten Komponente verbundenen Komponenten dupliziert werden, fehlt auch hier – abgesehen von der übergeordneten Aggregationshierarchie – der Kontext. Untergeordnete Ein- und Ausgänge der Komponenten sind nicht für Verbindungen verfügbar.

Die **Änderungsebenenabdeckung** soll wiederum dieses Problem lösen. Statt nur die Änderungen und deren Propagationpfad zur übergeordneten Klasse zu speichern, werden alle direkten Kinder der auf diesem Pfad liegenden Komponenten ebenfalls durch Referenzkomponenten abgebildet. So können alle Verbindungen auf einer Ebene genutzt werden. Nicht veränderte Kompositionspfade müssen allerdings weiterhin in den Vater-Komponenten gesucht werden. Schon die Änderung einer beliebigen Teilkomponente löst also eine Duplizierung aller höher gelegenen Ebenen, so dass ein Vorteil nur ohne Änderungen existiert und wie bei der Änderungsabdeckung nicht alle benötigten Teilkomponenten für Relationen zur Verfügung stehen.

Das Anlegen einer **First-Order-Referenzierung** führt grundsätzlich zu Duplizierung aller Komponenten der ersten Aggregationsebene von A in B. Wie bei der objektorientierten Programmierung sind so die primären Elemente als Referenz vorhanden. Extrem nachteilig wird

dieses Vorgehen durch die bei modularen Prozessen hohe Aggregationstiefe mit semantisch äquivalenten Komponentenarten. Alle Ein- und Ausgänge (Daten, Ereignisse etc.) der Teilkomponenten erster Ebene sind ihrerseits Komponenten, die zur Nutzung mit externen Komponenten verbunden werden müssen.

Bei First-Order- und Minimalreferenzierung kommt bei der angestrebten Nutzung in einem verteilten Gesamtmodell noch ein gravierender Nachteil hinzu: Werden Veränderungen an mehreren Orten durchgeführt steht keine Möglichkeit zur Verfügung, mit der die neu veränderten Teilkomponenten eindeutig bezeichnet werden können. Sie besitzen also nur algorithmisch / transitiv erfasste Konstrukte keine Identität. Speziell im Fall einer Instanziierung fehlen dann diese Beziehungen und können möglicherweise bei Änderungen nicht mehr zugeordnet werden. Bei einer Mehrfachvererbung ist die Zuordnung noch weiter erschwert, da gegebenenfalls von einer referenzierten Teilkomponente mehrfach (sub-)geerbt wird.

Einfacher und gegenüber Inkonsistenzen und Änderungen stabiler erweist sich daher die **Komponentenabdeckung**. Hier werden alle Teilkomponenten von A bis zur untersten Ebene durch Referenzkomponenten in B dargestellt. Dieses Vorgehen sichert die vollständige Abbildung der Komponenten in der Vererbungshierarchie und lässt für weitere Vererbungen ein einfaches Vorgehen zu, da alle geerbten Teilkomponenten direkt mit den übergeordneten Komponenten verbunden sind. Auch führen Veränderungen nicht mehr zu einer manuellen Propagierung und weit reichenden Konsequenzen in der Vererbungshierarchie, da keine Relationen überprüft und Komponenten dupliziert oder neu referenziert werden müssen. Auch die Gültigkeitsprüfung wird so vereinfacht. So kann auch sichergestellt werden, dass einmal definierte Komponenten in der Hierarchie nicht verloren gehen. Ein Aufwands-Nachteil kommt jedoch beim Einfügen neuer Teilkomponenten nahe dem Ausgangspunkt der Vererbungshierarchie zum Tragen. Die Referenzkomponenten müssen bis in alle untergeordneten Teile der Vererbungshierarchie nachträglich erzeugt werden. Vorteilhaft ist jedoch die so vereinfachte Prüfung anhand der in untergeordneten Komponenten neu erzeugten Teilkomponenten, die möglicherweise mit der weiter oben neu angelegten einen gemeinsamen Vererbungsstrang bilden könnten: Eine Variable a oben könnte so bis zur Variable a unten (definiert) durchgeerbt werden und könnte dann die unten definierte von oben herab ersetzen beziehungsweise integrieren.

Eine Möglichkeit einer bezüglich der relevanten Komponenten und Relationen vollständigen Referenzierung bietet das Konzept der **Vollreferenzierung** von Komponenten und Relationen. Hierbei werden wie bei der Komponentenabdeckung alle Komponenten als Referenzkomponenten dupliziert, zusätzlich jedoch interne Relationen ebenfalls. Dies wird notwendig, da die Relationen in ablauforientierten Ansätzen eine tragende Rolle spielen und deshalb ebenso für Änderungen relevant sind wie Komponenten. Zum Kontext gehören daher auch die Verbindungen.

Daher hat sich die Vollreferenzierung als einzig sinnvolle und konsistenzsichernde Alternative herausgestellt. Solange keine Änderungen vorgenommen werden, bezeichnen die Referenzkomponenten lediglich die gültigen Teilkomponenten von denen geerbt wird. Bei Änderungen in vererbungstechnisch „höher gelegenen“ Komponenten werden auch weiterhin die richtigen Komponenten (auch transitiv) referenziert. Änderungen an der erbenden Komponente sind ebenfalls problemlos möglich, da alle Komponenten und internen Relationen bereits existieren. Wichtig ist dabei, dass nur interne Relationen dupliziert werden, das heißt Relationen, die Teilkomponenten von A beziehungsweise B verbinden. Relationen mit externen Komponenten würden Seiteneffekte haben und das Konzept der Kapselung von Komponenten verletzen. Eine Vollreferenzierung bringt allerdings eine maximale Redundanz und zusätzlichen Verwaltungsaufwand mit sich.

Durch das Komponentenkonzept wird eine Kapselung eingeführt, die für aggregierte Komponenten erster Ebene nur deren Schnittstellen zur Verfügung stellt. Wird eine höhere

algorithmische Komplexität und zusätzlicher Rechenaufwand in Kauf genommen wäre es auch möglich, eine **Interface-Vollreferenzierung** umzusetzen. Wie bei der Vollreferenzierung könnten alle Komponenten erster Ebene und die Interfaces der untergeordneten dupliziert werden. Da alle Teilkomponenten über eine eigene Typ- und Komponenteneigenschaft verfügen, ist eine Änderung auch mit einer entsprechenden, überprüfaren Modifikation der Teilkomponente verbunden. Die Umsetzung ist jedoch auch hier wesentlich komplexer und anfälliger für Inkonsistenzen als die Vollreferenzierung.

ANHANG G: Segmentierung: Kontext, Namespaces und Projektbereiche

Um sicherzustellen, dass bei der Benennung die Identität auch dann gewährleistet wird, wenn die Semantik eine Mehrfachverwendung von Namen in unterschiedlichen Modellen erfordert (beispielsweise „User“), wird in Programmiersprachen und Modellierungssystemen häufig ein Konzept für Namensräume – sogenannte Namespaces – eingeführt, das die Eindeutigkeit eines Elementnamens innerhalb des Namensraums gewährleistet, gleichzeitig aber den gleichen Namen in anderen Namensräumen zulässt.

OMICRON stellt mit der Identität (*ModelElementKey*) jedem Modellelement (*ModelElement*) eine eindeutige Bezeichnung zur Verfügung. Trotzdem können Partitionen des Modellraums gerade bei großen Modellen hilfreich oder sogar notwendig sein.

In der Softwaretechnik wird die **aspektororientierte Programmierung (AOP)** eingesetzt, um spezielle Aspekte wie die Datenhaltung und das Logging von anderen separieren zu können, so dass für jeden Aspekt neue Konzepte mit wenig Aufwand umgesetzt werden können. In OMICRON ist es möglich mit Hilfe spezialisierter Aggregationstypen Komponenten, der „**Bereichsaggregate**“ nach unterschiedlichen Aspekten zu ordnen. In der einfachen Ausprägung mit einem (Aspekt-)Aggregationstyp ähnelt das Konzept daher der AOP.

Das Konzept der **Segmentierung** wird in OMICRON auch allgemein durch die Aggregation an eine übergeordnete Komponente wie ein Tochterunternehmen realisiert. Auf diese Weise können unterschiedliche Typen von Segmentierungen erzeugt werden. Sowohl der aggregierende Komponenten- als auch der Relationstyp kann dabei zur Differenzierung genutzt werden. Auch die mehrfache Teilnahme desselben Elements in unterschiedlichen Partitionen ist so möglich.

Auch Sichtbarkeitsbereiche können an ModelSpaces geknüpft werden. Neben der Sichtbarkeit in der Modellierung, können zudem Sichtbarkeits-/Zugriffsproblematiken zur Laufzeit, das heißt während der Abarbeitung, auftauchen.

Sichtbarkeiten bei Prozessen definiert die BPMN beispielsweise dreistufig ([Klein et al. 2004]):

1. Private (Internal) Business Processes (interne Geschäftsprozesse)
2. Abstract (Public) Processes (öffentliche Geschäftsprozesse)
3. Collaboration (Global) Processes (globale Prozesse).

In OMICRON können im Metamodell beliebige weitere Bereichstypen definiert und für Restriktionen ergänzend zum Rollen- und Rechtekonzept genutzt werden, beispielsweise Projekt, Abteilung, Betrieb, Konzern, Intranet, Extranet, etc.

Im Rahmen der Arbeiten an OMICRON wurde auch ein in das Metamodell integriertes **Rechtekonzept**⁵² entworfen, das Berechtigungskomponenten an Modell und Einzelkomponenten vergeben kann. So können **Rechtekomponenten** erzeugt werden, die für Rollen zugeordnet werden und deren Zugriffsrechte auf die assoziierten Komponenten definieren.

Durch Vererbung können dabei zunächst **allgemeine Rechte** für eine Rolle vergeben werden, die gegebenenfalls durch Spezialisierungen für besondere Bereiche ergänzt werden. Durch das **objektorientierte Rollenkonzept** übernehmen alle Rollen zunächst die Rechte der Vaterkomponente (Rolle), so dass nur für abweichende Rechte eine Definition erfolgen muss. Wo

⁵² Berechtigungen im Modell können unterschiedliche Operationen wie Ansicht, Ausführung, Änderung, Löschen, Erstellen etc. umfassen. Die Berechtigung, kann sich dabei sowohl auf komplette Modelle als auch auf einzelne Schritte beziehen. Auch kann eine Änderung mit dem Verlauf eintreten, beispielsweise, wenn ein Schritt erst einer Rolle zugewiesen wird.

für eine Spezialisierung besondere Rechte definiert sind, überschreiben diese die allgemeineren Rechte der Vaterkomponente.

Wichtig ist dabei, dass für eine sinnvolle Modellabgrenzung alle für die **Modellverwaltung** wichtigen Komponenten in einem **separaten Bereich** aggregiert werden müssen. Hier können Rollen definiert werden, die mit Rechtekomponenten assoziiert werden. Die Rechtekomponenten ihrerseits können dann an existierende Modelle jeder Ebene gebunden werden. Da sie außerhalb der Modelle liegen, werden sie bei der Vererbung und Instanziierung nicht direkt berücksichtigt, stehen nur transitiv zur Verfügung und sind so von Modellbestandteilen.

Die UML2 [OMG 2005b] sieht ebenso wie EPK-Erweiterungen [Klein et al. 2004] die Definition von **Bereichen** vor, die – übertragen auf die Produktion – meist Rollen (Werker, Meister,...) oder einen Kontext wie Orte (Drehmaschine, Produktionslinie, Lager) vorsehen. Die Notation kann dabei in **Swimlanes** oder als Konotation bei der einzelnen Aktion erfolgen. Für das Gesamtmodell ist diese Unterscheidung jedoch nur eine Frage der Visualisierung, nicht der Modellsemantik.

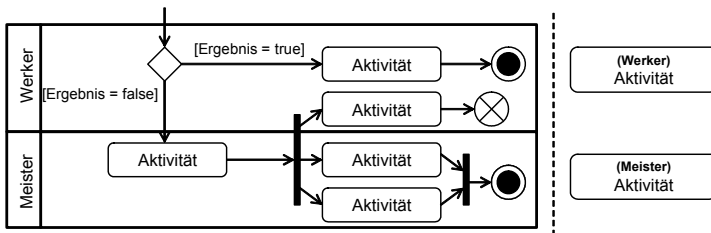


Abbildung 84: Swimlanes in UML

In OMICRON werden alle Arten von Bereichen als sogenannte Bereichsaggregate definiert, eine Spezialisierung der allgemeinen Aggregate. Durch Subklassenbildung kann die Semantik der Bereiche weiter verfeinert werden. Jede Komponente kann mit Hilfe der Aggregationsbeziehung „gehört zu Bereich“ an beliebig viele Bereiche gebunden werden.

Jeder Bereich ist Instanz eines Bereichstyps. OMICRON ermöglicht wie UML2 auch die Definition mehrerer Bereiche im selben Diagramm oder für dieselbe Komponente, beispielsweise die Zuordnung zu Ort und Person. Die Notation wird bei zwei Dimensionen matrixartig, bei drei und mehr müssen die Kombinationen einzeln aufgelöst werden und machen das Diagramm sehr komplex (vgl. [Klein et al. 2004] für EPK, [OMG 2005b] für die UML). Mit dem OMICRON-Modell ist die Bildung mehrerer Sichten auf dasselbe Modell jeweils unter Berücksichtigung ausgewählter Dimensionen möglich, so könnte das Modell mit Hilfe der Swimlane-Darstellung für jeden Aktivitätsbereichstyp speziell untersucht und sogar parallel verglichen werden. Das Gesamtmodell liefert dabei die Informationen für alle Teilsichten. (vgl. auch Model-View-Controller Konzept für generische Editoren [Li 2004]).

Wie im Service Blueprinting [Shostack 1982] können mit diesem Konzept Sichtbarkeitslinien beziehungsweise -ebenen definiert werden, die für bestimmte Rollen den sichtbaren Bereich definieren. Diese können in der Produktion beispielsweise dazu genutzt werden, Benachrichtigungen an alle dem Sichtbarkeitsbereich zugehörigen Rollen zu versenden.

ANHANG H: Beispielhafte Artefaktklassifikation

Dokumenttypen können beispielsweise auf den MIME-Typen [IETF 1996] aufbauen. Diese enthalten unter anderem HyperText Markup Language (text/html) und JPEG image data (image/jpeg). Für die formatunabhängige Klassifikation bieten sich jedoch abstrakter definierte Typen an, die um konkrete Ausprägungen wie die MIME-Typen durch Spezialisierung ergänzt werden können.

```
Content
\Executable
\Text
\  \AnyText
\  \  \BinaryEncodedText
\  \  \  \Word
\  \  \  \  \DOC
\  \  \  \  \RTF
\  \  \Plain Text
\  \StructuredText
\  \HTML
\  \CSV
\Media
\  \Audio
\  \Video
\  \  \AudioVideo (von Audio und Video)
\Picture
\  \Graphics
\  \Photograph
\  \Painting
```

Eine Kombination mit spezifischen Dokumenttypen wie Stückliste durch Mehrfachvererbung ermöglicht dann eine automatisierte Interpretation der Artefakte.

ANHANG I: Interpretation von Basistypen, Zeit, Kosten, metrische Begriffe

Weil die Metamodellierungsmethoden meist für die Softwareentwicklung konzipiert wurden, fehlen unternehmensorientierte Konzepte wie Kosten und Zeit [Dalal et. al. 2004]: „*Moreover, information on cost drivers and process performance measures including time, quality and efficiency, are not readily captured in existing enterprise modelling systems. The challenge for the architect is to create a simple and usable process-modeling technique that also represents enterprise-oriented semantics.*”

Zeit und vor allem Entfernungsbegriffe werden für fahrerlose Transportsysteme (FTS) wie Automatic Guided Vehicles (AGV) oder mobile Roboter in der Produktion (vgl. deren Einbindung im INT-MANUS Projekt) ebenso benötigt wie für die Produktionsplanung und Logistik. Sowohl diese als auch alle größenabhängigen Werte wie Lochdurchmesser sind zwangsläufig Teil des Zustandsraums einer Produktion.

Zeitereignisse werden auch in den UML2-Aktivitätendiagrammen und als generelles UML-Konstrukt Zeit (TimeExpression etc.) [OMG 2007a] unterstützt.

Generell sind Zeit, Kosten etc. im Gegensatz zur Nominalskala (Kategorisierung) beziehungsweise Ordinalskala (Ordnung) der Objektorientierung Bestandteil höherwertiger Skalen, die als Rational- oder Absolutskalen sogar über eine Berechenbarkeit verfügen können. Daher bestehen auch für Ontologien keine integrierten Konzepte für metrische Werte.

Umsetzungen nutzen daher programmtechnische unterstützte Basistypen zur Definition, beispielsweise für Zeitangaben:

- Day : Natural [1..31]
- Month : Natural [1..12] oder enum [Januar, Februar, ...]
- Year: Integer
- DayOfWeek : Natural [1..7] oder enum [Montag, Dienstag, ...]
- Date : { Day, Month, Year }
- Hours : Integer
- Minutes : Integer
- Seconds : Integer
- Milliseconds : Integer
- HourOfDay : Hours [0..23]
- MinuteOfHour : Minutes [0..59]
- SecondOfMinute : Seconds [0..59]
- MillisecondsOfSecond : Milliseconds [0..999]
- DayTime : { HourOfDay, MinuteOfHour, SecondOfMinute }
- TimeStamp : { Date, Daytime }

Prinzipiell ließen sich alle Zeitbestandteile auch als Komposite einer Grundeinheit auffassen – ähnlich wie in der logischen Programmierung durch Rekursion: Eine Stunde ist dann eine Kompositum aus 60 Minuten. Allerdings müssen hierzu entsprechend viele Millisekunden (Grundeinheit) zu einem Tag zusammengefügt werden, so dass eine rein ontologische Beschreibung ohne programmierte Zahlenkonstrukte die Modellgröße sprengen und den Rechen-

und Speicheraufwand proportional zu x^n steigen lassen würde, wobei n die Tiefe der Kompositionshierarchie ausgehend vom höchsten Element, beispielsweise Jahr, darstellt.

Die Ausstattung mit Attributen und eine eventuell auch externe Interpretation kann beispielsweise für Zeit, Entfernungen, Stückzahlen und andere Mengen vorgesehen werden.

Optimierungen hinsichtlich Entfernungen, Laufzeiten etc. können dann zwar mit Daten aus dem Modell, jedoch nur von externen Interpretern und nicht rein anhand der Semantik vorgenommen werden.

ANHANG J: Mengenverarbeitung und Sammlungen

UML2-Mengenverarbeitungsbereiche sollen die Verarbeitung von „**Sammlungen**“ (englisch Collections) in einem abgeschirmten Bereich ermöglichen – informationstechnisch gesehen Listen, Arrays und Vektoren, produktionstechnisch interpretiert Mengen von Teilen, Schüttgüter etc. Während die UML2 ein gesondertes Konstrukt hierfür vorsieht, werden Aggregate (auch über mehrere Stufen) von OMICRON direkt unterstützt.

Beispiel 20: Die Liste `Stückliste1 = {2x M4x30, 8x M6x50, 2x Platte 350x600x10}` kann als Stücklisten-Aggregat direkt verwendet werden. Zwei Schrauben M4x30, acht Schrauben M6x50 und zwei Platten 350x600x10 bilden dabei die Bestandteile des Stücklistenaggregats.

Das Modell kann dann zur Ausführungszeit direkt überprüfen, ob die Teile tatsächlich vollständig in der Aggregatinstanz enthalten sind. Die Konsistenz wird über Kardinalitäten, Ordnung und Typisierung sichergestellt.

Sollen dedizierte Berechnungen oder Aktionen mit allen (egal wie vielen) Teilen des Aggregats durchgeführt werden, müssen „for each“-Konstrukte eingeführt werden, die eine Aktion für alle (for each) oder eine definierte Untermenge von Aggregatbestandteilen durchführen. Aufgrund der Strukturierung und Typisierung von Aggregaten kommt dieser Fall nur bei Feld-Kardinalitäten größer 1 vor. – Typisierte Felder der Kardinalität 1 werden jeweils gezielt angesprochen.

Diesem Umstand sollen auch die neu in UML2 eingeführten **Schleifenknoten** Rechnung tragen. (siehe [Jeckle et al. 2004], S. 254 ff.)

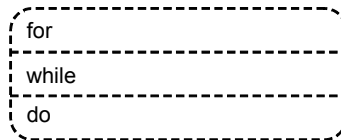


Abbildung 85: UML2 Notation von Schleifen in Aktivitäten

Sie dienen dazu, das programmiersprachliche Konstrukt der Schleifen für iterative Vorgänge nutzbar zu machen. Aus Sicht des OMICRON Modells stellt dies einen Rückschritt dar, da mit typisierten Komponenten eine semantisch korrekte Schleife ohne informale Zuordnungen etc. erzeugt werden kann:

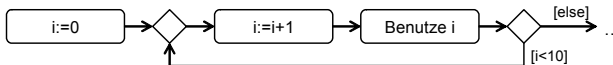


Abbildung 86: Die OMICRON-Schleifensemantik als Schleifenkonstrukt ohne semantisch fragwürdige Muster

Eine Nutzung als Ansicht wäre daher möglich. Die Grundlage muss jedoch ein vollständig verknüpftes Modell mit Daten-/Objektflüssen sein.

ANHANG K: Ausnahme-Klassifikation und Fehlerbehandlung

Für das interaktive Laufzeitsystem ist eine Unterscheidung hinsichtlich der Synchronizität wichtig. Fehler können synchron und asynchron auftreten [Brambilla et al. 2005]:

1. **Synchrone** Ausnahmen entstehen während der Interaktion eines Benutzers mit dem System, so dass eine direkte Bearbeitung im Kontext erfolgen kann.
2. **Asynchrone** Ausnahmen können jederzeit während der Prozessausführung entstehen. Die Zuständigkeit ist damit nicht wie im synchronen Fall per se gegeben. Gerade im Produktionsumfeld mit Teilautomatisierung müssen daher Zuständigkeiten und Eskalationsstufen definiert werden, um sicherzustellen, dass ein schnelles Eingreifen durch eine informierte Person möglich ist. In finalen Prototyp von INT-MANUS sollen erste Eskalationsstufen zudem automatisiert behandelt werden.

Die Art des Fehlers lässt sich weiter klassifizieren (vgl. [Brambilla et al. 2005]):

1. **Verhaltens- (oder Benutzer-) Ausnahmen** entstehen durch Verletzung der im Prozess definierten Aktivitätenfolge, beispielsweise Wiederholung oder Abbruch. Benutzerausnahmen sind nach dieser Definition immer synchron.
2. **Semantische (oder Anwendungs-) Ausnahmen** liefern ein falsches logisches Ergebnis, beispielsweise wegen geänderter Toleranzen für ein Teil (Modelländerung zur Ablaufzeit!). Semantische Fehler können sowohl interaktiv als auch im Hintergrund auftreten und daher sowohl synchron als auch asynchron sein.
3. **System-Ausnahmen** sind auf einen Ausfall oder eine Fehlfunktion von Teilen der Ausführungsschicht (Netzwerk, Betriebssystem, Workflow-Engine) zurückzuführen. Da sie jederzeit auftreten können, spricht man von asynchronen Fehlern.

Können diese Fehlersituationen nicht mehr vom Modell behandelt werden, muss eine programmierte Ausnahmeregelung (Systemeskalation) eingesetzt werden.

Hierzu können bei vorhersehbaren Änderungen reaktive Konzepte von Sprachen wie JIL [SuttonOsterweil 1997] eingesetzt werden. Entweder können für einen Block bereits vorgesehene Abläufe genutzt werden oder es müssen modellexterne Regeln angewandt werden. [SuttonOsterweil 1997]

Für die Behandlung von Ausnahmen im Web-basierten Workflow Systemen beschreibt [Brambilla et al. 2005] folgendes Vorgehen, das exemplarisch für ähnliche System gelten kann:

1. Erkennen von Fehlern/Ausnahmen anhand von Triggern (intern) oder Benachrichtigungen (von extern)
2. Benachrichtig der Nutzer (im Fall von OMICRON auch Peers)
3. Bearbeitung / Wiederherstellung
 - a. Automatisiert:
 - i. Accept – erfolgreiche Durchführung annehmen und auf „complete“ setzen
 - ii. Reject – rückgängig machen und deaktivieren
 - iii. Abort – Änderungen akzeptieren aber weitere Aktion abbrechen
 - iv. Ignore – Benachrichtigung des Benutzers, fortfahren
 - v. Resume – Zurücksetzen der Änderungen und nochmaliges Ausführen
 - b. Benutzerdefinierte (in OMICRON auch modellbasierte) Regeln
 - Sind vordefiniert

- Behandeln kritische Situationen
- Beschreiben Vorgehen zur Wiederherstellung und wie der Prozess fortzusetzen ist

ANHANG L: Komponenteneigenschaften

Komponenten können über beliebige inhärente Eigenschaften verfügen. Diese werden nicht im Modell definiert sondern von der Laufzeitumgebung zugewiesen und verwaltet. Einige Beispiele für mögliche Eigenschaften sind:

- **Zugriffs-/Sichtbarkeitsstufen**

- **Public:** Vollständig zugreifbar
- **Protected:** Nur von Spezialisierungen zugreifbar. Eine Zugreifbarkeit nach oben macht wenig Sinn, da die darüber liegenden zum Zeitpunkt ihrer Definition ihre Spezialisierung nicht kennen konnten.
- **Aggregate:** Alle im selben Aggregat liegenden Komponenten können zugreifen (alle in der Hierarchie aufwärts und von diesen aus abwärts)
- **Private:** Nur für Elemente derselben Prozessinstanz (beispielsweise zur inneren Kommunikation von Ereignissen und Ausnahmen einer Instanz)

- **Vererbung**

- **Virtual** bedeutet, dass eine Komponente bei der Vererbung überschrieben werden kann
- **VirtualExtendOnly:** Nur erweitern möglich, existierende Teile müssen so bleiben
- **VirtualOverwriteOnly:** Nur überschreiben bereits existierender Elemente möglich. Gegenstück zu VirtualExtendOnly.
- **Final:** Kein Überschreiben mehr möglich

- **Instanziierung**

- **Abstract:** Die Komponente darf nicht instanziiert werden, also keine eingehenden instanceOf-Beziehungen haben. Spezialisierungen (Konkretisierungen) erlauben dies möglicherweise jedoch wieder.
- **Static** die Komponente darf niemals instanziiert werden
- **Strict** lässt nur Instanzen desselben Typs zu, kann also dazu dienen, die Polymorphie zu unterdrücken und sollte nur in Ausnahmefällen verwendet werden.

- **InstanceFather** ist ein implizites, berechnetes Feld, das aus der Instanzierungsbeziehung aufwärts abgeleitet wird.

- **ReadOnly:** Nur lesender Zugriff auf ein Datenelement (vergleichbar Konstanten)

ANHANG M: MDA und generative, modellbasierte Softwareentwicklung

Während bisher meist Quellcode als zentrales Element in der Softwareentwicklung – gerade auch für Änderungen – dient, nutzt die Model-Driven Architecture (MDA) [OMG 2001] [KempaMann 2005] Modelle als Basis des Entwicklungsprozesses im generativen Software Engineering. MDA sieht auf MOF [OMG 2006b] basierende Ausgangsmodelle vor, so dass meist UML-basierte Modelle zum Einsatz kommen [StahlVölter 2005]. Im Gegensatz zu gewöhnlichen UML-Modellen [OMG 2005b] (wie OOA-Modellen [Kruschinski 1998][Schlegel 2002]), muss bei MDA-tauglichen Modellen jedoch die Semantik formal definiert sein. Voraussetzung ist dabei allerdings, dass die Software vollständig aus diesen Modellen erzeugt werden kann, so dass Änderungen direkt an diesen durchgeführt werden können. Erster Schritt hierzu ist die Unterteilung beziehungsweise Einordnung der genutzten Modelle in drei Abstraktionsstufen:

Das **Computation Independent Model (CIM)** beschreibt auf fachlicher Ebene die Systemfunktionalität, unabhängig von Struktur und informationstechnischer Umsetzung, beispielsweise in Form eines Lastenhefts. In einigen Beschreibungen von MDA wird diese Ebene allerdings nicht erwähnt [StahlVölter 2005].

Das **Platform Independent Model (PIM)** beschreibt die Funktionalität einer Komponente unabhängig von der Zielplattform, jedoch bereits mit einer definierten Semantik, die eine spätere Umsetzung ermöglicht, dabei aber noch von plattformspezifischen Konzepten abstrahiert.

Das **Platform Specific Model (PSM)** realisiert ein PIM unter Nutzung der durch die Zielplattform zur Verfügung gestellten Schnittstellen und Konstrukte. Es kann dabei mehrere kaskadierende Transformationsebenen geben, die erst als letzte Stufe den ausführbaren Code erzeugen. [StahlVölter 2005]

Die vorgesehene Transformation zwischen PIM und PSM folgt Transformationsregeln, die jedoch nicht zwangsläufig automatisierbar sein müssen. In jedem Fall protokolliert jedoch ein „Record of Transformation“ die angewendeten Regeln, um später eine Synchronisation der Modelle zur ermöglichen. [KempaMann 2005]

Hier zeigt sich ein weiteres Problem, das allen generativen Ansätzen gemein ist: manuelle Änderungen am Ergebnis der Generierung (Generat) sind meist Ausdruck eines nicht zufrieden stellenden Generierungsvorgangs und müssen daher entweder in die spezifischen Transformationsregeln oder das Ausgangsmodell einfließen. Eine Änderung am Generierungsergebnis wird bei der nächsten Generierung wieder überschrieben. Der Begriff „Roundtrip Engineering“ bezeichnet daher die Entwicklung mit Hilfe eines geschlossenen generativen Zyklus.

MDA als abstraktes Konzept erfordert zur Umsetzung immer (domänenspezifische) Modelltypen und Umgebungen (vgl. beispielsweise MEDAL [Guelfi et al. 2003] als Erweiterung zur Rational XDE). Die Konzepte dieser Modelltypen und Umgebungen sind meist sehr spezifisch für die Softwareentwicklung oder einen konkreten Anwendungsbereich und daher anders als das MDA-Konzept kaum auf die Produktionssysteme übertragbar.

MDS-Ansätze sind domänenspezifisch, Beispielsweise für Business Information Systems [Streekmann et al. 2006], während 4GL und CASE-Tools⁵³ meist unabhängig für alle Problemstellungen eine Lösung bieten sollen.

In MDA wird häufig die Verwendung von **Action Semantics** [VarróPataricza 2003] innerhalb des UML-Modells vorgeschlagen, womit die weitgehend statischen UML-Modelle um eine Art

⁵³ Computer Aided Software Engineering (CASE) Tools: computergestützte, (meist) modellbasierte Werkzeuge für die strukturierte Softwareentwicklung, oft auch über mehrere Entwicklungsphasen und -erfordernisse hinweg

Programmcode ergänzt werden können. Dieser kann auf die Modellelemente zugreifen beziehungsweise diese direkt verwenden, so dass eine Ablaufsemantik auf derselben Granularitätsstufe wie ausführbarer Code beschrieben werden kann. Allerdings verwenden unterschiedliche Hersteller auch unterschiedliche Konzepte. Zudem sind Action Semantics nicht über das Meta-Metamodell in das Metamodell eingebunden, sondern stellen gesondert zu interpretierende, modell-externe Text-Beschreibungen dar. In dieser Arbeit wird daher die direkte Einbettung und integrierte Beschreibung der Ablaufsemantik in das statische Modell angestrebt.

Um die Ausdrucksmächtigkeit der Quellsprachen für einen Anwendungsbereich zu erhöhen sieht die MDA **domänenspezifische Sprachen** (Domain Specific Languages, DSL) vor. Diese stellen Konstrukte zur Verfügung, um Modelle für eine bestimmte Domäne erstellen zu können. Neben der Syntax und formalen Semantik müssen diese auch eine Semantik aufweisen, die dem Modellierer entweder durch Kenntnis der Domäne und durch die Entlehnung von Begriffen aus der Sprachwelt des Modellierers intuitiv klar ist oder entsprechend dokumentiert ist, um eine semantisch korrekte Verwendung sicherzustellen. Trotz der Möglichkeit DSLs mit UML zu erstellen, wird der faktische Zwang zur Nutzung von UML als Basissprache in MDA kritisiert: „*We should be celebrating the usefulness of alternative partial specifications not insisting that everything should be done in a single language.*“ [Thomas 2003]

Eine **Domäne** (Domain) wird dabei als begrenztes Interessen- oder Wissensgebiet definiert [StahlVölter 2005], beispielsweise Produktion (fachlich) oder SOA (technisch). Bei der Service-Aggregation in Service-orientierten Architekturen [BurkhardLaures 2003] besteht beispielsweise die Gefahr, dieselbe Funktionalität mehrmals und inkonsistent anzubieten, weil Services für jedes Use-Case⁵⁴ neu erstellt werden. Domänen sollen hier Abhilfe schaffen, indem sie Begriffe für Funktionalitäten in ihrem Geltungsbereich eindeutig definieren helfen.

Meist werden zur Erweiterung des UML-Sprachumfangs in Richtung einer domänenspezifischen Sprache die erwähnten UML-Profile genutzt. Für jedes Metamodell beziehungsweise jedes UML-Profil muss dann ein entsprechendes Framework mit einer Referenzimplementierung oder Plattform existieren, das die generative Umsetzung des speziellen Sprachumfangs gewährleistet und dieses auf eine Programmierschnittstelle (Application Programming Interface, API) als Erweiterung oder Abstraktion des Zielsprachenumfangs abbildet.

In generativen Ansätzen der Softwareentwicklung wird das **Reverse Engineering** häufig ein Kernthema (vgl. [StahlVölter 2005] S.28), da eine transformierende Abbildung meist nicht bijektiv ist, so dass aus dem Ergebnis einer Generierung häufig nicht eindeutig auf das Ausgangsmodell geschlossen werden kann, beispielsweise weil weitere Informationen im Generierungsprozesse hinzugefügt werden. Noch schwieriger gestaltet sich das Reverse Engineering von nicht aus Modellen generiertem Code. Dieses ist nie für den allgemeinen Fall möglich, da Domänenmodelle und selbst verallgemeinerte Modelle nicht jeden Ausgangsfall abdecken können, der zu einer Software führt. Die Menge der durch Programme abbildbaren Fälle ist also größer als die Menge der beschreibbaren Ausgangsmodelle. In einem Laufzeit-Prozesssystem wie es in der Produktion nötig ist, sind die Laufzeit-Instanzen bereits im Gesamtmodell integriert, so dass ein Reverse Engineering dieser Abläufe unmöglich und auch unnötig wird. Eine Veränderung von Funktionalitäten zur Laufzeit kann nur über ein externes Aufrufkonzept erfolgen. Der modellbasiert-generative Ansatz hat also für ein flexibles Produktionssystem Vorteile, jedoch in der Ausprägung nach Art der MDA zu viele Stufen und auf die Entwicklung fokussierte Konzepte, die eine direkte Verwendung zur Laufzeit nur nach Modifikationen des Konzepts möglich erscheinen lassen.

⁵⁴ [BurkhardLaures 2003] nutzen Use-Cases als Top Level Sicht für Geschäftsprozesse, ergänzt um andere Diagrammtypen für ein detaillierte Beschreibung der Kommunikation und der Komponenten

ANHANG N: WRITE: Visualisierungen und Update

Häufig sollen Elemente nicht interaktiv sondern rein informativ sein. Der Benutzer kann nicht direkt mit diesen Elementen interagieren.⁵⁵ Angezeigte Komponenten wie Texte, Werte und Grafiken können jedoch parallel zu dieser Anzeige durch Interaktionselemente oder durch Prozesse in tiefer liegenden Schichten wie Prozesssteuerung und Backend verändert werden.

Daher sind mehrere Fälle der Datenaktualisierung zu unterscheiden:

Typ	Beschreibung
Static	Es wird der bei der Modelldefinition angegebene Wert verwendet, der nicht aktualisiert oder angepasst wird.
Transformed	Es wird immer derselbe Wert verwendet, allerdings gibt es eine Kontexttransformation, die diesen (beispielsweise wegen geänderter Sprache oder Plattform) vor der ersten Anzeige anpasst. Die Funktionalität einer Transformation muss auch für alle anderen Fälle mit Ausnahme von Static verfügbar sein, um regelbasiert Werte anzupassen.
ReadOnce	Der anzuzeigende Wert wird einmal eingelesen. Dabei kann der Zeitpunkt gegebenenfalls festgelegt werden. Eine Transformation ist hier ebenfalls zum Einlesezeitpunkt (oder Anzeigezeitpunkt) verfügbar. ReadOnce ist auch als ReadCyclic mit einer Iteration definierbar.
ReadCyclic	Der anzuzeigende Wert wird zyklisch, das heißt in vorgegebenen Intervallen eingelesen und aktualisiert. Grundsätzlich wäre es auch möglich komplexe Funktionen zur Berechnung der Zyklen zu nutzen. Allerdings ist die Abfrage über einen errechneten Trigger hier besser.
ReadCyclicHistory	Wie ReadCyclic, statt eines Werts werden jedoch eine Auswahl oder alle Werte angezeigt, beispielsweise ein Trend der letzten zehn Werte.
ReadTriggered	Hier löst ein Trigger eine Neubelegung des angezeigten Werts aus. Dabei kann der Trigger direkt den Wert enthalten oder als Ereignis nur das neuerliche Einlesen aus einer Datenquelle oder einem Puffer für empfangene Werte auslösen.
ReadTriggeredHistory	Wie ReadTriggered. Statt eines Werts wird jedoch eine Auswahl oder alle Werte angezeigt, beispielsweise ein Trend der letzten zehn Werte.
RealRealtime	Technologie- bzw. plattformabhängig kann auch eine Echtzeitanzeige zur Verfügung gestellt werden, die erhaltene Werte direkt darstellt. Meist wird hier kaum ein Unterschied zu Triggern oder kurzen Zyklen entstehen.

Neben der Aktualisierungsinformation spielt die Art der Werte beziehungsweise deren Quellbereich eine Rolle für die Darstellung.

⁵⁵ Über verbundene (interaktive) Steuerelemente können jedoch Änderungen herbeigeführt werden.

ANHANG O: Klassifikation möglicher Transformationsansätze

Für die Klassifikation des Ansatzes hinsichtlich der Transformationsunterstützung sollen im Folgenden die einzelnen Aspekte von Transformationen aufgezeigt werden. Die genannten Ansätze und deren Spezifika sind in [CzarneckiHelsen 2003] und [StahlVölter 2005] beschrieben und werden hier nicht gesondert erläutert.

Regelbasierte Transformationen bestehen aus einer linken (Left Hand Side, LHS) und einer rechten Seite (Right Hand Side, RHS), die wiederum folgende Komponenten enthalten.

Variablen enthalten Elemente des Quell- oder Zielmodells, die im Transformationsprozess genutzt und gegebenenfalls verändert werden – auch Meta-Variablen genannt, um sie von den im generierten Code enthaltenen zu unterscheiden. Hierbei können modellinterne Variablen durch modellexterne Variablen ergänzt werden.

Patterns sind Modellfragmente mit Null bis mehreren Variablen. Diese können aus Zeichenketten (in textuellen Templates) oder grafischen Mustern bestehen. Sie liefern einen Rahmen zur Erzeugung des Zielmodells (siehe auch Templates in [Schlegel 2002]). Beispielsweise werden von AndroMDA Velocity und XDoclet als Template Engines benutzt (vgl. [StahlVölter 2005]).

„**Logik**“ beschreibt in [CzarneckiHelsen 2003] Berechnungen und Einschränkungen (Constraints) auf Modellelementen. Diese können ausführbare Elemente oder nicht-ausführbare Beziehungen (statisch/relational) zwischen Modellen sein. Ausführbare Elemente werden dabei in deklarative Konzepte (wie OCL-Abfragen) und imperative Konzepte unterteilt, die Funktionen zur Manipulation des Modells aufrufen. Ein Beispiel für ein imperatives Konzept ist dabei UML Action Semantics [VarróPataricza 2003], die direkt auf eine Programmiersprache umgesetzt werden kann und mittlerweile in die UML2 integriert wurde.

Model-to-Model Transformationen transformieren von Modellen eines Metamodells in Modelle desselben oder anderer Metamodelle. Teilweise werden Zwischenschritte oder -modelle bei großen Abstraktionssprüngen besser genutzt. Es existieren unterschiedliche Verfahren:

- **Direct-Manipulation** benötigt eine reine API (wie JMI als MOF-API) um code-basiert Transformationen durchzuführen
- **Relational** werden mathematische oder logische Beziehungen (deklarativ) genutzt, die gut mit logischer Programmierung umzusetzen sind
- **Graph-Transformation-Based**: Hier werden Graphen speziell für UML typisiert, attribuiert, etc. um eine Transformation des Graphen zu ermöglichen
- **Structure-Driven** ist ein zweiphasiges Konzept, in dem auf den (hierarchischen) Modellaufbau, das Befüllen mit Attributen und Referenzen erfolgt
- **Hybrid**: Mischung aus deklarativem und imperativem Ansatz

Model-to-Code erzeugt Code als Ergebnis. Es sind dabei zwei Vorgehensweisen zu unterscheiden:

- **Visitor-based**: Das Modell wird nach einem bestimmten Algorithmus durchlaufen.
- **Template-based**: Templates werden befüllt beziehungsweise selektieren Informationen aus dem Quellmodell und berechnen Ausgaben in das Zielmodell. Problem ist hier die Korrektheitskontrolle, da reine Texte nicht analysiert werden – es sei den sie wären in OMICRON entsprechend beschrieben oder würden einem Parser zugeführt. Frame processing erweitert das Template-Konzept mit besseren Anpassungs- und Strukturierungsmechanismen.

Die meisten existierenden Modell-Compiler benutzen einen Template-Ansatz, der von UML-Modellen direkt zum finalen Code führt. Jamda (<http://jamda.sourceforge.net/>) fügt dagegen zunächst die Definitionen generierter Klassen in das UML-Modell ein und erzeugt aus diesen dann Code. Es würde so die Rolle des Modell-Compilers in der Object Management Group Model Driven Architecture (OMG MDA) Spezifikation einnehmen.

Transformationsmodelle

Die Typisierung von Variablen und Patterns kann unterschiedlich sein:

- Untypisiert
- Syntaktisch typisiert: Assoziiert mit einem Metamodell-Element, dessen Instanzen die Variable enthalten kann.
- Semantisch typisiert: Während der syntaktische Typ beispielsweise „Expression“ ist, kann der semantische Typ „Expression evaluating to an integer value“ sein. Dies geht funktioniert bisher nur in einigen Meta-Programmiersprachen wie MetaML (MetaML, <http://www.cse.ogi.edu/PacSoft/projects/metaml>) und MetaOCaml (Meta Objective-Caml, <http://www.cs.rice.edu/~taha/MetaOCaml>).

Regelkonzepte können dabei unterschiedliche Aspekte betonen:

- Syntactic Separation: Regeln können Regeln verändern (überschreiben).
- Bidirektionalität: Regeln können in beide Richtungen gelesen beziehungsweise ausgeführt werden.
- Regelparametrisierung: Parameter werden zur Konfiguration der Regeln verwendet.
- Intermediate structures: Es werden Zwischenmodelle genutzt.

In MDS / MDA besteht das Konzept darin, Transformationen am Metamodell festzumachen und darauf basierend am Modell die Transformation durchzuführen. [StahlVölter 2005] Dieser Ansatz ist hinsichtlich der Abstrahierbarkeit zunächst der einzig sinnvolle. In OMICRON besteht zusätzlich die Möglichkeit, die Transformation auf der Modellebene anzupassen. Wird für jedes Transformations-Tupel $T=(M, R)$ aus Modellelement M und verwendeter Transformationsregel R ein Transformationsobjekt O erzeugt, das auf Modellebene (das heißt Metamodell-Instanziierungsebene) für dieses Tupel die Transformation abbildet, so ist dieses zunächst leer und folgt den Definitionen seiner Transformationsregel(klasse) auf der Meta-Ebene. Werden nun individuelle Anpassungen der Transformation für ein Modellelement vorgenommen, können diese im Transformationsobjekt verankert werden. Bei der nächsten Transformation für T können diese Anpassungen dann berücksichtigt werden.

Beim **Rule Application Scoping** wird nur ein Teilmodell in die Transformation einbezogen. Hierbei kann es sich um eine Einschränkung des Quellmodells oder des Ziels (Zielmodell, Zielapplikation) handeln.

Die **Beziehung zwischen Quell- und Zielmodell** kann entweder als Generierung in ein neues Modell (CDI) oder als Generierung im selben Modell (VIATRA, GreAT), das heißt als direktes Update des Ausgangsmodells realisiert werden. Andere Konzepte ermöglichen beides (XDE). Auch destruktives Update und reine Erweiterung (ohne entfernen existierender Bestandteile) sind denkbar. Modelle mit nichtdeterministischer Auswahl und/oder Fixpunktiteration können oft die Terminierung nur über eine Beschränkung auf reine Erweiterung erreichen, um endlose Veränderungen zu vermeiden (beispielsweise VIATRA). Während die Generierung direkt in das Modell gut möglich ist und neue Modelle sehr gut erzeugt werden können, ist eine gleichzeitige Generierung in das Ausgangsmodell und ein neues Zielmodell eher mit Konsistenzproblemen verbunden.

Die Regel-Auswahl kann durch explizite Bedingungen (Jamda), nicht-deterministisch (BOTL) oder interaktiv (XDE) erfolgen. **Anwendungsstrategien für Regeln** können also sein:

- deterministisch: Gesicherte Reihenfolge beispielsweise über Tiefensuche, Breitensuche etc. (Stratego Rewriting Language mit Baum-Traversierung)
- nicht-deterministisch:
 - one-point application: Nicht-deterministisch ausgewähltes Ziel der Regelanwendung
 - concurrent application: Nebenläufige Anwendung einer Regel auf alle passenden Ziele (VIATRA)
- interaktive Auswahl der Regelanwendung (XDE)

Rule Scheduling kann als implizites Scheduling wie BOTL, nicht direkt steuerbar (OptimalJ nur über spezielle Regeln) oder explizites Scheduling erfolgen. Explizites Scheduling hat Konstrukte zur Steuerung der Reihenfolge und kann intern (innerhalb der Transformationsregeln) oder extern (Trennung zwischen Regeln und Scheduling) geschehen.

Konfliktauflösung wird von keinem Ansatz zur Verfügung gestellt, wäre aber möglich.

Regel-Iteration kann beispielsweise mit Rekursion, Schleifen oder Fixpunktiteration erfolgen. Neben einstufigen Verfahren sind auch mehrstufige Prozesse mit mehreren Phasen möglich. Jede Stufe kann dabei unterschiedliche Regeln oder einen anderen Zweck haben (vgl. IML-Transformator, [Schlegel 2002][Schlegel et al. 2004]). Strukturorientierte Ansätze wie OptimalJ oder IOPT bauen beispielsweise erst eine Containment-Hierarchie auf und setzen in einer weiteren Phase Attribute und Referenzen.

Die „**Rule Organization**“ kann je nach Ansatz hinsichtlich Modularität, Wiederverwendung (mit Vererbung, Ableitung, Spezialisierung, Modulvererbung, logische Verknüpfung), Organisationsstruktur etc. stark variieren. Im Rahmen einer besseren Verfolgbarkeit können **Tracability Links**: als Verbindungen zwischen Quell- und Zielelementen eingesetzt werden – zur Auswirkungsanalyse, Synchronisation und allgemeinen Rückverfolgung. Mitunter werden spezielle Verfolgungsmechanismen angeboten. Die Informationen können dabei entweder im Quellmodell, Zielmodell oder separat gespeichert werden.

OMICRON kann sowohl interne Modelle als auch externe generieren und unterstützt so direkte Verweise im Modell, indirekte Verbindungen über IDs im Generat sowie verbindungslose Generierung.

Directionality and Synchronisation: Regeln können unidirektional oder bidirektional sein, wobei eine Mischform durch zwei unidirektionale Regeln bidirektionale simulieren kann. Wichtig ist jedoch, dass nicht nur die Regeln gegebenenfalls Bidirektionalität unterstützen müssen, sondern auch das Modell ein Reverse Engineering beziehungsweise eine Rücktransformation ermöglichen muss. Speziell bei Änderungen außerhalb des Modells können jedoch Inkonsistenzen auftreten. Eine einfache Rücktransformation ist nur bei einer bijektiven Abbildung möglich, die beispielsweise bei der UI-Generierung nicht immer gegeben sein kann, wenn Attribute wie Farbe etc. von mehreren Faktoren abhängen. Einstufig kann dies über Enumerationen berücksichtigt werden. Mehrstufige Abbildungen auf dasselbe Element wachsen mit der Stufenzahl jedoch exponentiell an.

Bidirektionale Regeln sind neben einem geeigneten Gesamtmodell eine Grundvoraussetzung für die Synchronisation von Ausgangsmodell und Zielmodell. Die Synchronisation ist wesentlich einfacher durchzuführen, wenn die Transformationsverbindungen zusammen mit Ausgangs- und Zielmodell im Gesamtmodell gespeichert sind und Veränderungen direkt synchronisiert werden. Eine Offline-Bearbeitung und spätere Synchronisation erfordert andere Algorithmen, wie sie

beispielsweise in Versioning-Systemen wie dem Concurrent Versioning System (CVS) oder dessen Nachfolger Subversion (<http://subversion.tigris.org/>) implementiert sind. Allerdings schränken bidirektionale Regeln die Mehrstufigkeit ein und führen so zu geringerer Kontextbasierung.

Generierungszeitpunkt: Abgesehen von diesen Dimensionen zur Klassifikation von Transformationsansätzen aus MDA oder MDSD sind bei einer Erweiterung des Themenkreises zusätzliche Kategorien zur Klassifikation notwendig.

Während in der modellbasierten Software Entwicklung, wie der Name sagt, Software meist vollständig entwickelt wird, bevor Sie tatsächlich zu Ausführung gelangt, greifen bei Interpreteransätzen auf Code- oder sogar Modellebene sowie bei Laufzeitanätzen wie OMICRON Softwareentwicklung und Ausführung teilweise ineinander. Daher ist zu unterscheiden, ob die Transformation

- im Rahmen eines der Nutzung vorgelagerten Entwicklungsprozesses,
- während der Nutzungszeit aber vor Ausführung oder
- auf Anfrage

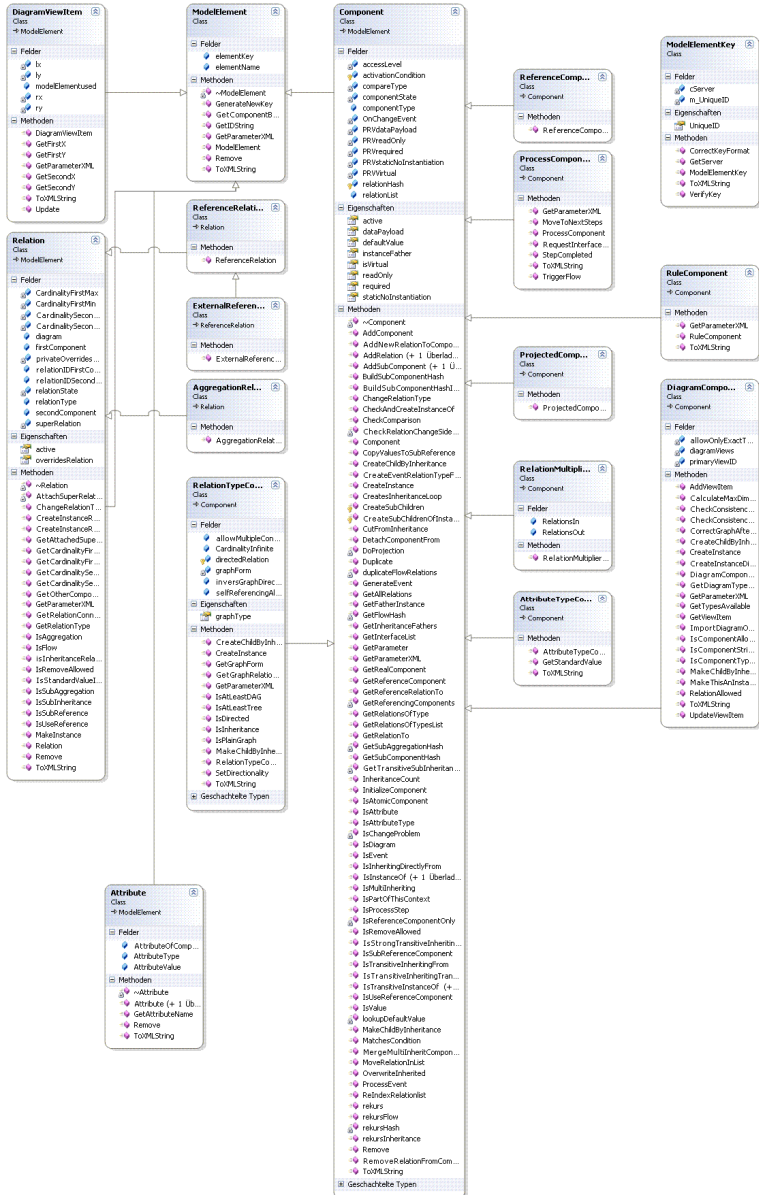
erfolgt.

Zielformat: Die letzte Stufe der Transformation soll das gewünschte Resultat liefern. Dabei kann es sich um Quellcode, Verknüfungsbeschreibungen wie BPEL, Interfacebeschreibungen wie CORBA IDL [OMG 2004b] oder vollständig kompilierte Module handeln. Bei der Kompilierung ist meist keine Teilausführung der Transformation möglich, das heißt ein Modell-System muss vollständig transformiert und dann dem Kompilierungsprozess zugänglich gemacht werden. Für SOA müssen Module fertig erstellt und möglicherweise einem sogenannten Deployment unterzogen werden, bevor das System lauffähig ist.

Hinzu kommt, dass bei der Verwendung von Modellinterpretern diese letzte Stufe entfällt beziehungsweise in den Modellinterpreter verlagert wird. Die Interpretation erfolgt dabei als Sonderform der Transformation zur Laufzeit.

ANHANG P: Modellstruktur im Code: Klassendiagramm

Ausschnitt aus dem Klassendiagramm in Visual Studio 2005 (ComponentServer)



ANHANG Q: Code von BuildSubComponentHash

Für alle semantischen und Teilmengen-Überprüfungen müssen Teilkomponenten von Aggregaten, untergeordnete Komponenten in der Vererbung oder Instanzen identifiziert werden. BuildSubComponentHash ist dabei eine zentrale Funktion imOMICRON-Laufzeitsystem, die einen Hash von Teilkomponenten hinsichtlich einer Gruppe von Relationstypen erzeugt.

```
/// <summary>
/// Creates a hash that contains all subtypes reachable with this
/// SubTypeRelationHash from this Component.
/// reverseDirection means do not follow direction of relations but reverse
direction
/// (like for inheritance - descending)
/// </summary>
public Hashtable BuildSubComponentHash(Hashtable hashTableUsed,
    Hashtable SubTypeRelationHash, bool DoRecursion, bool
reverseDirection)
{
    if (hashTableUsed == null)
        hashTableUsed = new Hashtable();
    foreach (Relation rel in relationList)
        if (SubTypeRelationHash.Contains(rel.GetRelationType().GetIDString()))
            //type of relation is correct (contained in searched types)
            {
                Component subComponent = null;
                if ((RelationTypeComponent) (
                    SubTypeRelationHash[rel.GetRelationType().GetIDString()])
                    .IsDirected())
                {
                    if (reverseDirection ^ rel.relationType.inversGraphDirection)
                        // XOR
                        {
                            if (rel.secondComponent == this)
                                subComponent = rel.firstComponent;
                        }
                    else if (rel.firstComponent == this)
                        subComponent = rel.secondComponent;
                }
                else
                    subComponent = rel.GetOtherComponent(this);
                if (subComponent != null)
                    {
                        if (!hashTableUsed.Contains(subComponent.GetIDString()))
                            hashTableUsed.Add(subComponent.GetIDString(),
                                subComponent);
                        if ((rel.firstComponent != rel.secondComponent) // reflexive?
                            && (DoRecursion))
                            subComponent.BuildSubComponentHash(hashTableUsed,
                                SubTypeRelationHash, DoRecursion, reverseDirection);
                    }
            }
    return hashTableUsed;
}
```

ANHANG R: Aufbau und Ergebnisse der Testläufe

Unterschiedliche Einstellungen im Code sollen in der Evaluation die Performanz des Laufzeitsystems bei unterschiedlichen Konfigurationen des Modells, wie tief, breit, etc., zeigen.

Code für ein breites Modell liefert beispielsweise die Einstellung

```
if (randi.Next(20) > 17)
```

Code für ein tiefes Modell dagegen beispielsweise die Einstellung

```
if (randi.Next(20) > 3)
```

indem die Wahrscheinlichkeit einer zusätzlichen Ebene erhöht wird. Hierzu wird die aktuell erzeugte Komponente mit einer größeren Wahrscheinlichkeit zu derjenigen von der geerbt wird.

Der Code erzeugt basierend auf dem Meta-Modell – ausgehend von der Basiskomponente, die mit Hilfe von bServer (ComponentServerBase) erfragt wird – ein zufälliges Modell aus ungeschachtelten Komponenten, die alle über Vererbungsbeziehungen verfügen und Instanzen der Basiskomponente sind.

```
DateTime dt = DateTime.Now;
Random randi = new Random();
for (int i2 = 1; i2 <= 500; i2++)
{
    Component comp2 = new Component(bServer.GetKeyFromKeyserver(),
        "baumBase" + i2.ToString());
    comp2.instanceFather = bServer.GetBaseComponent();
    for (int i = 1; i <= 100; i++)
    {
        Component comp = new Component(bServer.GetKeyFromKeyserver(),
            "baum" + i.ToString());
        comp.instanceFather = bServer.GetBaseComponent();
        comp.AddNewRelationToComponent(comp2,
            bServer.GetInheritanceRelationType(),
            bServer.GetKeyFromKeyserver(), false, null);
        if (randi.Next(20) > 17)
            comp2 = comp;
    }
}
DateTime dt2 = DateTime.Now;
MessageBox.Show("Zeit: " + (dt2 - dt));
```

Für die Messung der Exportgeschwindigkeit wurde zusätzlicher Code eingefügt, der die Exportroutine aufruft und den Zeitbedarf für den Export misst:

```
DateTime dt = DateTime.Now;
[...]
DateTime dt2 = DateTime.Now;
MessageBox.Show("Zeit: " + (dt2 - dt));

DateTime dt3 = DateTime.Now;
bServer.ExportToXML("F:\exportModel.xml");
DateTime dt4 = DateTime.Now;
MessageBox.Show("Zeit für den Export: " + (dt4 - dt3));
```

Die Dateigröße nach dem Export wurde direkt dem Dateisystem entnommen.

Für die Ermittlung der Effizienz bei der Abfrage von transitiven Instanzbeziehungen und Vererbungsbeziehungen wurde der obige Basiscode für die zufällige Erzeugung des Modells erweitert. Während das Modell wie bisher erzeugt wird, entstehen zwei Listen, denen ein Element mit einer zehnprozentigen Wahrscheinlichkeit zugeordnet wird.

Die fList (from-Liste) enthält alle Komponenten, die als untergeordnete dienen, die tList (to-Liste) alle Komponenten, die als übergeordnete in die Abfrage eingehen.

Alle in der äußeren Schleife erzeugten Komponenten können nur der tList zugeordnet werden, die hierdurch länger als die fList wird.

Daher wird eine Schleife eingesetzt, die alle tList durchläuft und eine „for each“-Schleife enthält, die zweimal vollständig durchlaufen wird. Da das letzte Element von tList mit fast allen Elementen der fList im zweiten Durchlauf verglichen wird, ist das Ergebnis der Abfrage meist false und damit nahe an der oberen Schranke für die Durchlaufzeiten.

Für die Abfrage der transitiven Instanzeigenschaft wird die Funktion *IsTransitiveInstanceOf(...)* verwendet. Bei der Instanzabfrage werden auch Spezialisierungen berücksichtigt, so dass das ganze Modell durchlaufen werden muss.

```
Random randList = new Random();
List<Component> fList = new List<Component>();
List<Component> tList = new List<Component>();
DateTime dt = DateTime.Now;
Random randi = new Random();
for (int i2 = 1; i2 <= 500; i2++)
{
    Component comp2 = new Component(bServer.GetKeyFromKeyserver(),
        "baumBase" + i2.ToString());
    comp2.instanceFather = bServer.GetBaseComponent();
    for (int i = 1; i <= 100; i++)
    {
        Component comp = new Component(bServer.GetKeyFromKeyserver(),
            "baum" + i.ToString());
        comp.instanceFather = bServer.GetBaseComponent();
        comp.AddNewRelationToComponent(comp2,
            bServer.GetInheritanceRelationType(),
            bServer.GetKeyFromKeyserver(), false, null);
        if (randi.Next(20) > 17)
            comp2 = comp;
        //Code for creating lookup lists:
        int which = randList.Next(100);
        if (which > 95)
            fList.Add(comp);
        if (which < 5)
            tList.Add(comp);
    }
    int which2 = randList.Next(100);
    if (which2 > 90)
        tList.Add(comp2);
}
DateTime dt2 = DateTime.Now;
MessageBox.Show("Zeit: " + (dt2 - dt));

//code for walking through the lists:
DateTime dt3 = DateTime.Now;
List<Component>.Enumerator ListEn = tList.GetEnumerator();
bool endReached = false;
do
```

```
{
    foreach (Component c in fList)
    {
        endReached = ListEn.MoveNext();
        Component d = ListEn.Current;
        bool isittrue = c.IsTransitiveInstanceOf(d);
    }
} while (!endReached);
DateTime dt4 = DateTime.Now;
MessageBox.Show("Zeit: " + (dt4 - dt3));
```

Für die Vererbung wird weitgehend derselbe Code ausgeführt. Jedoch ersetzt `IsTransitiveInheritingFrom(...)` das bisherige `IsTransitiveInstanceOf(...)`, um die Vererbungsbeziehung anstelle der Instanzbeziehung abzufragen.

```
Random randList = new Random();
List<Component> fList = new List<Component>();
List<Component> tList = new List<Component>();
DateTime dt = DateTime.Now;
Random randi = new Random();
for (int i2 = 1; i2 <= 500; i2++)
{
    Component comp2 = new Component(bServer.GetKeyFromKeyserver(),
        "baumBase" + i2.ToString());
    comp2.instanceFather = bServer.GetBaseComponent();
    for (int i = 1; i <= 100; i++)
    {
        Component comp = new Component(bServer.GetKeyFromKeyserver(),
            "baum" + i.ToString());
        comp.instanceFather = bServer.GetBaseComponent();
        comp.AddNewRelationToComponent(comp2,
            bServer.GetInheritanceRelationType(),
            bServer.GetKeyFromKeyserver(), false, null);
        if (randi.Next(20) > 17)
            comp2 = comp;
        //Code for creating lookup lists:
        int which = randList.Next(100);
        if (which > 95)
            fList.Add(comp);
        if (which < 5)
            tList.Add(comp);
    }
    int which2 = randList.Next(100);
    if (which2 > 90)
        tList.Add(comp2);
}
DateTime dt2 = DateTime.Now;
MessageBox.Show("Zeit: " + (dt2 - dt));

//code for walking through the lists:
DateTime dt3 = DateTime.Now;
List<Component>.Enumerator ListEn = tList.GetEnumerator();
bool endReached = false;
do
{
    foreach (Component c in fList)
    {
        endReached = ListEn.MoveNext();
        Component d = ListEn.Current;
        bool isittrue = c.IsTransitiveInheritingFrom(d);
    }
} while (!endReached);
```

```
DateTime dt4 = DateTime.Now;
MessageBox.Show("Zeit: " + (dt4 - dt3));
```

Testergebnisse

Die folgenden Tabellen zeigen die Testergebnisse der oben und im Evaluationskapitel beschriebenen Testfälle mit Verzweigungswahrscheinlichkeit r und Schleifenzahlen i und $i2$.

Wert i2	Wert i	Anzahl Elemente	Verzweigungsschwelle	Laufzeit in s
25	500	12525	17/20	2,265625
50	500	25050	17/20	14,515625
100	500	50100	17/20	69,437500
150	500	75150	17/20	159,515625
200	500	100200	17/20	286,734375
250	500	125250	17/20	448,843750
500	500	250500	17/20	1807,406250

Tabelle 22: Laufzeit Modellerzeugung mit $i = 500$, $r = 15\%$

Wert i2	Wert i	Anzahl Elemente	Verzweigungsschwelle	Laufzeit in s
25	100	2525	17/20	0,171875
50	100	5050	17/20	0,437500
100	100	10100	17/20	1,359375
150	100	15150	17/20	3,734375
200	100	20200	17/20	8,234375
250	100	25250	17/20	14,890628
500	100	50500	17/20	70,375000
1000	100	101000	17/20	293,109375
1250	100	126250	17/20	456,421875
2500	100	252500	17/20	1820,750000

Tabelle 23: Laufzeit Modellerzeugung mit $i = 100$, $r = 15\%$

Wert i2	Wert i	Anzahl Elemente	Verzweigungsschwelle	Laufzeit in s
25	500	12525	3/20	2,140625
50	500	25050	3/20	14,593750
100	500	50100	3/20	69,468750
150	500	75150	3/20	161,078125
200	500	100200	3/20	291,671875
250	500	125250	3/20	467,296875
500	500	250500	3/20	1836,468750

Tabelle 24: Laufzeit Modellerzeugung mit $i = 500$, $r = 85\%$

Wert i2	Wert i	Anzahl Elemente	Verzweigungsschwelle	Laufzeit in s
25	100	2525	3/20	0,171875
50	100	5050	3/20	0,437500
100	100	10100	3/20	1,343750
150	100	15150	3/20	3,453125
200	100	20200	3/20	7,828125
250	100	25250	3/20	13,968750
500	100	50500	3/20	66,593750
1000	100	101000	3/20	296,171875
1250	100	126250	3/20	459,618750
2500	100	252500	3/20	1993,250000

Tabelle 25: Laufzeit Modellerzeugung mit $i = 100$, $r = 85\%$

Wert i2	Wert i	Anzahl Elemente	Verzweigungsschwelle	Laufzeit in s	
				Aufbau	Ablauf
10	5000	50010	3/20	68,578125	39,406250
100	500	50100	3/20	69,468750	38,093750
200	250	50200	3/20	69,765625	40,703125
1000	50	51000	3/20	71,281250	37,953125
10000	5	60000	3/20	99,875000	46,406250

Tabelle 26: Laufzeit Modellabfrage auf transitive Instanzen (IsTransitiveInstanceOf)

Wert i2	Wert i	Anzahl Elemente	Verzweigungsschwelle	Laufzeit in s	
				Aufbau	Ablauf
10	5000	50010	3/20	68,282125	1,343750
100	500	50100	3/20	68,812500	0,156250
200	250	50200	3/20	68,765625	0,078125
1000	50	51000	3/20	71,968750	0,031250
10000	5	60000	3/20	99,750000	0,000001

Tabelle 27: Laufzeit Modellabfrage auf transitive Subtypen / Vererbung (IsTransitiveSubtypeOf)

Wert i2	Wert i	Anzahl Elemente	Ablauf Instance A	Ablauf Subtype (Cache) C	Laufzeit Verhältnis A:C
10	5000	50010	39,406250	1,343750	29,32
100	500	50100	38,093750	0,156250	243,80
200	250	50200	40,703125	0,078125	521,00
1000	50	51000	37,953125	0,031250	1214,50
10000	5	60000	46,406250	0,000001	46406250,00

Tabelle 28: Laufzeit Modellabfrage auf Instanzen und Subtypen, Vergleich der Geschwindigkeit

Wert i2	Wert i	Anzahl Komponenten	Exportzeit	Kilobyte
25	100	2525	30,390625	1440
50	100	5050	61,781250	2960
100	100	10100	238,125000	5932
150	100	15150	548,859375	8909
200	100	20200	964,187500	11886
250	100	25250	1554,265625	14863

Tabelle 29: Laufzeit und Modellgröße bei Modellexport

ANHANG S: Externe Daten und Persistenz

Auch ein verteilbares Modellsystem muss Aspekte der Persistenz bei externen Daten beachten. Im Folgenden sind daher Möglichkeiten der Transaktionalität, flache externe Daten und Datenzugriffskomponenten als mögliche Teilkonzepte für ein verteilbares Modellsystem beschrieben.

Transaktionalität bedeutet hier die Umsetzung des ACID-Prinzips für Transaktionen [HärderReuter 1983]. Gerade durch die Vererbung kaskadieren viele Änderungen. Für den praktischen Einsatz muss daher soweit möglich Transaktionssicherheit bei Modelländerungen hergestellt werden. ACID bedeutet dabei zunächst, dass eine Modellveränderung nur vollständig oder gar nicht stattfindet (**A**tomicity) – beispielsweise eine Vererbung mit referenzierenden Teilkomponenten: Es werde alle Teilkomponenten mit erstellt und alle transitiven Änderungen an Spezialisierungen durchgeführt.

Nach Abschluss der Transaktion muss das Modell wieder in einem konsistenten Zustand (**C**onsistency) vorliegen, da die Modellüberprüfung der Transaktion vorgeschaltet ist und das Modell nur über zusätzlich implementierte Korrekturläufe aus einem inkonsistenten Zustand wieder in einen konsistenten Zustand überführt werden könnte. Eine Rückabwicklung ist teilweise trotz bekannter Teilschritte nicht mehr möglich, weil keine inverse Transformation mit den vorhandenen Informationen gebildet werden kann.

Die **I**solation ist problematisch im Modell, jedoch ebenfalls Voraussetzung für eine reibungslose Verteilung und Parallelisierung: Jede Modelländerung wird so durchgeführt, als wäre sie die einzige, obwohl andere parallel laufen. Hierzu müssen beispielsweise Markierungsdurchläufe im Gesamtmodellgraph über einen temporären Index beziehungsweise Hash gelöst werden, da im Modell aus Isolationsgründen keine temporären Markierungen vorgenommen werden sollten. Auch das Locking von Teilbäumen ist eine Konsequenz aus dieser Forderung, da sonst Wechselwirkungen mit anderen Modelltransaktionen entstünden.

Erfolgreiche Transaktionen führen automatisch zu einer persistenten Änderung des Gesamtmodells (**D**urability), sofern dieses über eine persistente Entsprechung verfügt. Durch eine erfolgreiche Datenbanktransaktion kann dies sichergestellt werden. Die Datenbankanbindung zu einer Vorstufe desOMICRON-Modells wird in [Wamatu 2006] beschrieben und umgesetzt. Diese Implementierung unterscheidet Update und Insert, so dass Modelländerungen in Veränderungen existierender Werte und die Erstellung von neuen Elementen differenziert werden können. Eine Threadsynchronisation des Update- und Insert-Thread ermöglicht dabei eine Parallelisierung bei trotzdem möglichem Rollback beziehungsweise Undo.

Um ein Undo auch nach erfolgtem Commit zu ermöglichen, müssen die Werte der zu verändernden Datensätze vor der Transaktion gesichert werden. Alternativ ist ein Undo auf Modellebene möglich, das Veränderungen des Modells speichert, deren Inverse dann in Anweisungen für die Datenhaltung umgesetzt werden kann. Zum Teil ist ein Undo wie oben bereits beschrieben aus technischen oder auch aus fachlichen Gründen nicht mehr möglich oder sinnvoll, wenn beispielsweise ein Teil bereits gefertigt wurde.⁵⁶

Flache externe Daten: Häufig werden datenbank- und dateibasierte Datenzugriffe unterschieden (beispielsweise [Arlow et al. 1997] S. 14). Im Modell selbst kann nur eine Klassifikation, Strukturierung und Komponentendefinition vorgenommen werden. Der Datenzugriff selbst muss mit Hilfe einer Implementierung oder als externer Service realisiert werden. Für das Modell ist

⁵⁶ Auch Prozessfehler oder -ausnahmen können zu Situationen führen, in denen ein Rollback beziehungsweise Recovery notwendig wird. Neben rein transaktionalen / syntaktischen Strategien, die in verteilten Systemen gerade bei Fehlern und parallelisierten / nebenläufigen Prozessen häufig nicht möglich sind, gibt es auch semantische Recovery-Ansätze ohne objektorientierte Konzepte für Prozesse, vgl. [Rinderle et al. 2006].

also die Art des Zugriffs zweitrangig. Nur die Struktur und gegebenenfalls Art der Daten spiegelt sich im Modell wieder.

In OMICRON wird der Zugriff über eine eindeutige Signatur von Datenquellen und -senken realisiert, aus denen die gewünschten Daten extrahiert und als Komponenten ins Modell geschrieben werden.

Für eine Daten(satz)anforderung muss ein eindeutiger Kontext zur Verfügung stehen, der sowohl die Datenquelle an sich, als auch das genaue Datum bezeichnet. Dies kann entweder mit Hilfe einer eindeutigen Objekt-ID erfolgen oder durch einen externen Befehl, der durch die Typisierung vom Empfänger korrekt erkannt wird – beispielsweise als SQL-Abfrage. Im Modell werden diese als flache Daten gespeichert und an die Laufzeitumgebung oder ein externes System zur Verarbeitung übergeben.

Um bereits in der Laufzeitumgebung Persistenz zu bieten, kann eine einfache **Datenbank** angebunden werden, die ein Feld für **Text- oder XML-Fragmente** (OMICRONData) als flache Daten und ein Feld für die zugehörige **Objekt-ID** (ModelElementKey oder OMICRONID) als Schlüssel besitzt. Sie funktioniert damit wie eine Hash-Tabelle für die persistenten Daten. Da jede Komponente im Laufzeitsystem einen String (dataPayload) besitzt, kann dieser zur Speicherung verwendet werden.

Wird eine als persistent markierte Komponente erzeugt oder verändert, wird die Datenbank aktualisiert. Die Vernichtung des Datums muss allerdings explizit erfolgen.

Die Speicherung komplexer Elemente erfolgt als XML-Struktur, die für jedes Teilelement einen Eintrag besitzt. Ein Datenbankzugriff erfolgt dann immer auf die ganze Datenstruktur des komplexen Elements.

Datenzugriffskomponenten

Datenstrukturen entstehen durch die geordnete Aggregation beliebiger atomarer oder ebenfalls strukturierter Datentypen. So können beispielsweise Tabellen oder XML-Strukturen im Modell abgebildet werden.

Neben Datentypen auf der Klassenebene kann es jedoch auch notwendig werden, externe Datenquellen und -senken als Komponenten im System zu beschreiben. So können bereits im Modell Daten auch deren Speicherort zu geordnet werden und auch Abstraktionen wie Abfragen (SQL Set, XML Query) adressiert werden.

Diese **Data Access Components (DAC)** abstrahieren von Datenquellen, die Instanzen enthalten. Beispielsweise können Elemente in einer XML-Datei oder einer Datenbank im Modell abstrakt durch eine DAC verwaltet werden, wobei immer die Werte und Eigenschaften des tatsächlichen Datenobjekts geliefert werden, ohne dass im Gesamtmodell Instanzen erzeugt werden.

DAC können dabei per Aggregation Modellbestandteile desselben Typs an sich binden und wieder selbst gegebenenfalls eine Aggregationshierarchie von DAC bilden. Das Wertefeld einer DAC enthält dabei beispielsweise den SQL-Abfragestring. Durch den Typ kann die Art der Datenquelle definiert werden.

Die Umsetzung eines Zwei-Ebenen-Modells mit grundlegenden, teilweise von OMICRON abweichenden Konzepten [Wamatu 2007] auf eine Datenbank beschreibt [Wamatu 2006].

Erweiterungsmöglichkeiten sind dabei die getrennte Speicherung von Artefakten und temporären Daten sowie eine Verteilung und Replikation der persistenten Modelldaten.

Lebenslauf



Persönliche Daten

Thomas Schlegel
geboren in Ludwigsburg
am 26.09.1976
Staatsangehörigkeit deutsch

Schulbildung, Wehrdienst und Studium

1983 – 1987	Grundschule Affalterbach
1987 – 1996	Friedrich-Schiller-Gymnasium Marbach a.N.
1996	Allgemeine Hochschulreife, Abitur sehr gut
1996 – 1997	Wehrdienst, Stabsdienst
1997 – 2002	Studium Softwaretechnik, Universität Stuttgart
2002	Abschluss Diplom-Informatiker (Dipl.-Inf.), sehr gut

Berufstätigkeit

1990 – 1994	Selbständige Softwareentwicklung und -vertrieb
1994 – 1996	EDV-Systemhaus, Ludwigsburg
1996	Mercedes-Benz AG, Untertürkheim
1996 – 1998	CID Computer GmbH, Oberstenfeld
1998	Daimler-Benz AG, Untertürkheim
1998 – 1999	Communication Consultants GmbH, Stuttgart
1999 – 2000	Hewlett Packard Deutschland GmbH, Böblingen
2000 – 2001	Agilent Technologies Deutschland GmbH, Böblingen
2001 – 2002	ETAS GmbH, Stuttgart
2002 – 2008	Wissenschaftlicher Mitarbeiter Competence Center Human-Computer Interaction Fraunhofer Institut für Arbeitswirtschaft und Organisation
seit 2008	Leiter Team Interaktive Systeme Institut für Visualisierung und Interaktive Systeme Universität Stuttgart

