Jörg Desel, Martin Glinz (Hrsg.)

Modellierung in Lehre und Weiterbildung

Workshop auf der Modellierung 2008
Berlin, 13. März 2008

Technischer Bericht ifi-2008.04
Institut für Informatik, Universität Zürich
http://www.ifi.uzh.ch/techreports/TR_2008.html

Herausgeber

Jörg Desel, Prof. Dr.
Katholische Universität Eichstätt-Ingolstadt
Lehrstuhl für Angewandte Informatik
Ostenstraße 14
85072 Eichstätt
Deutschland
http://www.informatik.ku-eichstaett.de/Desel

Martin Glinz, Prof. Dr. rer. nat.
Institut für Informatik
Binzmühlestrasse 14
8050 Zürich
Schweiz
http://www.ifi.uzh.ch/~glinz

Programmkomitee

Leitung: Jörg Desel, Kath. Universität Eichstätt-Ingolstadt
Martin Glinz, Universität Zürich

Torsten Brinda, Universität Erlangen-Nürnberg
Jürgen Ebert, Universität Koblenz-Landau
Ulrich Frank, Universität Duisburg-Essen
Andreas Harrer, Kath. Universität Eichstätt-Ingolstadt
Wolfgang Hesse, Universität Marburg
Peter Hubwieser, TU München
Johannes Magenheim, Universität Paderborn
Heinrich Mayr, Universität Klagenfurt
Andreas Oberweis, Universität Karlsruhe
Barbara Paech, Universität Heidelberg
Klaus Pohl, Universität Duisburg-Essen
Kurt Schneider, Universität Hannover
Albert Zündorf, Universität Kassel

Inhalt

Vorwort

Modellierung in Lehre und Weiterbildung hat sich als wichtiges Thema im Gebiet der Modellierung etabliert. Nachdem Aspekte der Lehre und der Weiterbildung bereits seit vielen Jahren regelmäßig auf der Tagung „Modellierung“ diskutiert worden sind, gibt es nun zum zweiten Mal nach 2006 einen eigenen Workshop „Modellierung in Lehre und Weiterbildung“ im Rahmen der „Modellierung“. So wie Modellierung innerhalb der Informatik ein Querschnittsthema ist, das viele Bereiche der Informatik tangiert, ist auch der Aspekt der Modellierungslehre bereichsübergreifend. Dies wurde in der Formulierung des Call-for-Papers deutlich, in der Besetzung des Programmkomitees, und auch in den eingereichten Arbeiten.

Das Programmkomitee hat diese Einreichungen begutachtet und sechs Arbeiten für den Workshop ausgewählt. Die Themen reichen von einem virtuellen Weiterbildungsstudiengang Wirtschaftsinformatik, in dem Modellierungsaspekte eine besondere Rolle spielen, über ethische Aspekte der Modellierung und des Software-Entwurfs sowie ihre Berücksichtigung in der Lehre, über konkrete Vorschläge zur Vermittlung von dynamischen Modellen mit Hilfe von Mustern bzw. durch Konzentration auf die Modellabläufe, bis hin zu eher fachlich orientierten Abhandlungen über Metamodelle und Objektmodelle.

Ein Workshop unterscheidet sich von der Haupttagung dadurch, dass Workshop-Beiträge häufig auch vorläufige Arbeiten darstellen, über deren Inhalte auf dem Workshop diskutiert wird. Im Idealfall können die Autoren durch die Diskussion und den Austausch mit anderen Workshop-Teilnehmern in ihren Forschungen bereits ein Stück weiterkommen. In diesem Sinn dokumentieren die Beiträge in diesem Band nicht fertige Arbeiten, sondern zumeist jeweils einen Zwischenstand, über den es lohnt zu reden.

Wir danken allen Autoren und den Mitgliedern des Programmkomitees, die mit ihrem Engagement für das Zustandekommen dieses Workshops gesorgt haben. Wir danken auch allen Personen, die uns bei der Organisation des Workshops und bei der Herstellung dieses Bandes unterstützt haben, insbesonere Wofgang Reisig als Leiter der Gesamttagung, Markus Nüttgens als Verantwortlichem für die Workshops und Dorothea Iglezakis in Eichstätt für die Betreuung des Internetauftritts.

Eichstätt und Zürich, im März 2008

Jörg Desel
Martin Glinz

Modellierung in der Wirtschaftsinformatik-Weiterbildung

Prof. Dr. Elmar J. Sinz

Lehrstuhl für Wirtschaftsinformatik,
insbesondere Systementwicklung und Datenbankanwendung
Otto-Friedrich-Universität Bamberg
96045 Bamberg
elmar.sinz@uni-bamberg.de

Abstract: Seit dem WS 2001/02 wird im Rahmen des virtuellen Weiterbildungsstudiengangs Wirtschaftsinformatik (VAWi) der Kurs *Modellierung von Systemen und Prozessen* angeboten. Der Beitrag stellt die Konzeption des Kurses vor und berichtet über Erfahrungen.

1 Modellierung von Systemen und Prozessen im Weiterbildungsstudiengang VAWi

Der virtuelle Weiterbildungsstudiengang Wirtschaftsinformatik VAWi (www.vawi.de) wird seit dem WS 2001/02 gemeinsam von den Universitäten Bamberg und Duisburg-Essen angeboten. Es handelt sich dabei um einen internetbasierten Studiengang mit kurzen Präsenzphasen zu Beginn und Ende des Semesters sowie multimedial unterstützten Fernlernphasen während des Semesters [AKF+02]. Der durch ASIIN (www.asiin.de) akkreditierte Studiengang ist auf drei Semester Vollzeitstudium ausgelegt; die Mehrzahl der Studierenden absolviert das Programm jedoch berufsbegleitend als Teilzeitstudium. VAWi führt zum akademischen Grad eines Master of Science (M.Sc.). Im WS 2007/08 verzeichnet VAWi ca. 400 eingeschriebene Studierende.

Die Zielgruppe des Studiengangs sind berufstätige Hochschulabsolventen unterschiedlicher Fachrichtungen, die eine fundierte Zusatzqualifikation in Wirtschaftsinformatik anstreben [FeDe06]. Formale Zulassungsvoraussetzungen sind neben einem berufsqualifizierenden Hochschulabschluss im Regelfall eine mindestens zweijährige Berufserfahrung nach Abschluss des Studiums sowie einschlägige Kenntnisse in den Gebieten der Wirtschaftsinformatik, Wirtschaftswissenschaften oder der Informatik aus dem Erststudium oder aus der beruflichen Tätigkeit. Die Mehrzahl der Studierenden besitzt einen ersten Studienabschluss als Betriebswirt, Informatiker, Ingenieur oder Naturwissenschaftler. Konkrete Vorkenntnisse in Wirtschaftsinformatik werden bei der Zulassung nicht vorausgesetzt. Der Umfang des Studienprogramms beträgt 98 ECTS-Punkte: 7 Kurse aus Pflichtmodulen (je 4,5 ECTS), 9 Kurse aus Wahlpflichtmodulen (je 4,5 ECTS), 2 Projektarbeiten (je 4 ECTS) und die Masterarbeit (18 ECTS).

Seit Aufnahme des Studienbetriebs im WS 2001/02 bietet der Verfasser dieses Beitrags den Kurs *Modellierung von Systemen und Prozessen* als Teil des Pflichtmoduls *Systementwicklung (Wirtschaftsinformatik)* an. Der Kurs ist als ein Einstiegskurs in das VAWi-Programm konzipiert und wird von vielen Studierenden bereits im ersten Semester gewählt. Außerdem legt der Kurs methodische Grundlagen für die vom Verfasser im Wahlpflichtbereich angebotenen Kurse *Datenmanagement*, *Data-Warehouse-Systeme* und *Methoden der Systementwicklung* sowie für weitere Kurse des VAWi-Programms.

In den folgenden Abschnitten werden die Auswahl der Inhalte des Kurses *Modellierung von Systemen und Prozessen*, deren Aufbereitung und Präsentation sowie die Lerninfrastruktur und das Betreuungskonzept vorgestellt. Zum Schluss wird über Erfahrungen berichtet.

2 Auswahl der Kursinhalte

Modelle stellen das wichtigste Hilfsmittel zur Analyse und Gestaltung betrieblicher Informationssysteme - des Gegenstandsbereichs der Wirtschaftsinformatik - dar. Fundierte Modellierungskenntnisse sind für Wirtschaftsinformatiker daher unverzichtbar, um betriebliche Sachverhalte mithilfe von Modellen dokumentieren und kommunizieren zu können. Ziel des Kurses ist die Vermittlung methodischer Kenntnisse und Fertigkeiten, die benötigt werden, um

- ein Verständnis für die Modellierung betrieblicher Systeme und Prozesse zu entwickeln,
- die Eignung konkreter Modellierungsmethodiken für vorliegende Problemstellungen beurteilen zu können,
- Modelle betrieblicher Systeme und Prozesse nutzen zu können,
- Modellierungsmethodiken auf konkrete Problemstellungen anwenden zu können und
- Modellierungsmethodiken an konkrete Erfordernisse anpassen und in bestimmtem Umfang weiterentwickeln zu können.

Eine Modellierungsmethodik umfasst dabei neben einer oder mehreren Modellierungssprachen auch die der Systemerfassung zugrunde liegende Metapher, Regeln zur Gestaltung der Modellarchitektur sowie Hinweise für das Vorgehen bei der Modellerstellung. Die methodische Basis des Kurses bildet das Lehrbuch *Grundlagen der Wirtschaftsinformatik* von Ferstl und Sinz [FeSi06].

Da Modelle und Modellierung keinen Selbstzweck, sondern ein wichtiges und methodisch anspruchsvolles Hilfsmittel darstellen, orientiert sich die Auswahl der Kursinhalte an konkreten Fragestellungen der Wirtschaftsinformatik, die unter Zuhilfenahme von Modellen gelöst werden. Hierzu gehören die Analyse und Gestaltung von Geschäftsprozessen, die Entwicklung von Anwendungssystemen, der Entwurf von Datenmanagementsystemen, das taktische Informationsmanagement usw. Der Schwerpunkt der Modellierung und Modellnutzung liegt dabei stärker auf den fachlichen Modellebenen eines betrieblichen Informationssystems, weniger auf den softwaretechnischen Modellebenen. Mehr als softwaretechnische Modelle unterstützen fachliche Modelle die Kommunikation heterogener Nutzergruppen (Anwender, Systementwickler, Berater, Business-Analytiker [ThLo07]).

Für die genannten Fragestellungen erscheinen – auch mit Blick auf die betriebliche Praxis – folgende Klassen von Modellierungsansätzen besonders geeignet: Datenorientierte Modellierung, objektorientierte Modellierung, prozessorientierte Modellierung sowie hybride Ansätze zur objekt- und prozessorientierten Modellierung. Unter Einbeziehung der benötigten methodischen Grundlagen ergibt sich damit die in Abbildung 1 dargestellte Hauptgliederung des Kurses [Sinz07].

1. Einführung und Motivation
2. Methodische Grundlagen der Modellierung
3. Datenorientierte Modellierung
4. Objektorientierte Modellierung
5. Prozessorientierte Modellierung
6. Objekt- und prozessorientierte Modellierung
7. Bewertung von Modellierungsansätzen

Abbildung 1: Hauptgliederung des VAWi-Kurses *Modellierung von Systemen und Prozessen*

Jede der in den Kapiteln 3 bis 6 genannten Klassen wird anhand von einem oder mehreren konkreten Modellierungsansätzen vermittelt: (3) Die datenorientierte Modellierung wird anhand des Entity-Relationship-Modells (ERM) und des Strukturierten Entity-Relationship-Modells (SERM) behandelt. (4) Die objektorientierte Modellierung wird aufbauend auf einer kurzen Darstellung ihrer historischen Entwicklung anhand der UML 2.0 eingeführt. Da die UML primär eine Modellierungssprache und keine Modellierungsmethodik darstellt, wird eine UML-basierte Methodik für die Modellierung der Aufgabenebene betrieblicher Informationssysteme sowie der Anwendungssysteme auf der Aufgabenträgerebene vorgeschlagen und jeweils anhand einer Fallstudie erläutert. (5) Die prozessorientierte Modellierung wird anhand von Petri-Netzen, Ereignisgesteuerten Prozessketten (EPK) und der Workflow-Modellierung gemäß Referenzmodell der Workflow Management Coalition (WfMC) behandelt. (6) Als objekt- und prozessorientierte Modellierungsmethodik wird das Semantische Objektmodell (SOM) vorgestellt. Die Aufgabenebene betrieblicher Informationssysteme wird in Form eines SOM-Geschäftsprozessmodells, die Aufgabenträgerebene in Form einer SOM-Anwendungssystemspezifikation modelliert. Zur Vertiefung werden die gleichen Fallstudien wie in Kapitel 4 verwendet. Die Lösung der Fallstudien mit alternativen Modellierungsansätzen macht deren spezifische Merkmale sichtbar und fördert das Verständnis für die Auswahl der adäquaten Modellierungsmethodik für eine gegebene Problemstellung.

Während allgemeine Grundlagen von Modellen und Modellierung sowie Ziele, Motivation und Anwendungsbereiche bereits in Kapitel 1 vorgestellt werden, behandelt Kapitel 2 spezifische methodische Grundlagen der Modellierung. Hierzu gehören insbesondere systemtheoretische Grundlagen, eine Diskussion des Begriffs Modell und der Eigenschaften von Modellen, die Einführung der Konzepte Metapher, Metamodell und Meta-Metamodell, die Bildung von Teilmodellen und Sichten auf Modelle sowie Ausführungen zur Architektur von Modellen:

- Systemtheoretische Grundlagen und die zentralen Systemmerkmale Struktur und Verhalten bilden den Ausgangspunkt für die Herausbildung eines fundierten Systemverständnisses und für die Abgrenzung und Erfassung von Systemen in Form von Modellen.

- Anhand des abbildungsorientierten Modellbegriffs werden die Konzepte der formalen Konsistenz und Vollständigkeit sowie der Struktur- und Verhaltenstreue von Modellen eingeführt. Dieser Modellbegriff wird anschließend um ein konstruktivistisches Modellverständnis erweitert, welches das modellierende und nutzende Subjekt in die Betrachtung einbezieht und damit das Bewusstsein für die Subjektivität der Erfassung und Rekonstruktion realer Sachverhalte weckt.

- Das Konzept der Metapher dient als Hilfsmittel zur Reduzierung subjektiver Einflüsse bei der Modellbildung und -nutzung. Metamodelle erlauben die präzise Spezifikation der einzelnen Modellierungssprachen. Alle im Kurs verwendeten Modellierungssprachen werden auf der Basis eines einheitlichen Meta-Metamodells beschrieben.

- Obwohl Modelle betrieblicher Informationssysteme stets nur einen Ausschnitt der zugrunde liegenden Realität erfassen (Verkürzungsmerkmal von Modellen; Komplexitätsreduktion durch Modelle) ist die Komplexität der entstehenden Modellsysteme im Allgemeinen erheblich. Diese Komplexität erfordert die Bildung von Teilmodellen und Sichten auf Modelle sowie die Strukturierung von Modellen unter Zugrundelegung einer bestimmten Modellarchitektur (Komplexitätsbewältigung).

3 Aufbereitung und Präsentation der Kursinhalte

Anforderungen an die Auswahl und Aufbereitung der Kursinhalte resultieren einerseits aus der Breite und dem Umfang der Kursinhalte, andererseits aus der Tatsache, dass der Kurs als Einstieg für Studierende mit unterschiedlichen Studienfächern im Erststudium geeignet sein soll. Dabei ist jedoch zu berücksichtigen, dass die meisten Studierenden bereits aus dem Erststudium oder der Berufspraxis über ausgeprägte Modellierungskenntnisse (insbesondere auf Basis der Sprache UML) verfügen. Zielsetzung des Kurses ist also weniger eine erste Hinführung zur Modellierung [BPR06], sondern vielmehr eine Systematisierung, methodische Unterfütterung und Weiterentwicklung von Modellierungskenntnissen für Anwendungen der Wirtschaftsinformatik. Im Kurs *Modellierung von Systemen und Prozessen* werden diese Anforderungen wie folgt umgesetzt:

- Problemorientierung: Die Auswahl der zu vermittelnden Modellierungsmethodiken erfolgt in Bezug auf konkrete Problemfelder der Wirtschaftsinformatik, deren Bearbeitung den Einsatz von Modellen erfordert.

- Methodenorientierung: Die Aufbereitung der Kursinhalte erfolgt methodenorientiert mit dem Ziel, ein vertieftes Verständnis für die problemadäquate Erstellung und Nutzung von Modellen zu vermitteln. Als methodische Grundlage dient dazu eine systemorientierte Sichtweise auf der Basis der Systemmerkmale Struktur und Verhalten, wie sie für ingenieurmäßiges Problemlösen charakteristisch ist (siehe z. B. [GrTa06]).

- Einheitliche Darstellung: Alle behandelten Modellierungssprachen werden anhand einheitlicher, systemorientierter Metamodelle vorgestellt, wobei alle Metamodelle auf einem gemeinsamen Meta-Metamodell beruhen. Das bedeutet z. B. für die UML, dass nicht die begriffsorientierten Metamodelle auf Basis der Meta Object Facility (MOF) verwendet werden, sondern die einzelnen Diagrammarten anhand eigens erstellter systemorientierter Metamodelle eingeführt werden[1].

Die Präsentation der Kursinhalte gegenüber den Studierenden erfolgt in Form eines Online- und eines Offline-Skripts. Das Online-Skript steht auf der VAWi-Lernplattform zur Verfügung und wird interaktiv durchgearbeitet. Es enthält – wo dies sinnvoll erscheint – Animationen und Links auf weiterführende Materialien und Werkzeuge. Das Offline-Skript ist inhaltlich identisch mit dem Online-Skript; es steht in Form von pdf-Dateien zum Download zur Verfügung.

Eine Skriptseite besteht im Allgemeinen aus einer Folie und einem zugehörigen Textteil (siehe Abbildungen 2 und 3). Dem liegt die Metapher eines klassischen Folienvortrags zugrunde: Während eine Folie die jeweiligen Gegenstände in kompakter Form visualisiert, werden diese durch gesprochenes Wort näher erläutert. Letztere Funktion übernimmt der zur jeweiligen Folie gehörige Textteil[2]. Für die Studierenden ergibt sich dadurch eine Wissensvermittlung auf zwei Ebenen. Umgekehrt kann bei der Selbstkontrolle der Lernzielerreichung anhand der Folien überprüft werden, ob die jeweiligen Gegenstände und Zusammenhänge verstanden wurden. Identifizierte Lücken können bei Bedarf anhand der Textteile und der dort angegebenen weiterführenden Literatur geschlossen werden.

---

[1] Die Metamodelle auf Basis der MOF stellen Spezialisierungshierarchien von Begriffen (UML-Konzepten) in den Mittelpunkt. Beziehungen zwischen Begriffen werden auf der jeweils abstraktesten Ebene spezifiziert. Diese Form der Metamodelle erscheint für die Modellierungspraxis eher unhandlich. Systemorientierte Metamodelle stellen das Systemparadigma (System als Menge in Beziehung stehender Komponenten) in den Mittelpunkt. Zum Beispiel bilden die Metaobjekte *Klasse* und *Beziehung* den Kern des Metamodells für das UML-Klassendiagramm. Weiter spezifiziert das Metamodell die Merkmale von Klassen und Beziehungen sowie die relevanten Spezialisierungen von Beziehungen (z. B. *Generalisierung*, *Aggregation*, *Komposition*, *Assoziation*).

[2] Um der Vortragsmetapher möglichst nahe zu kommen wurden die meisten Textteile tatsächlich gesprochen, mithilfe von Spracherkennungssoftware aufgezeichnet und anschließend nachbearbeitet.

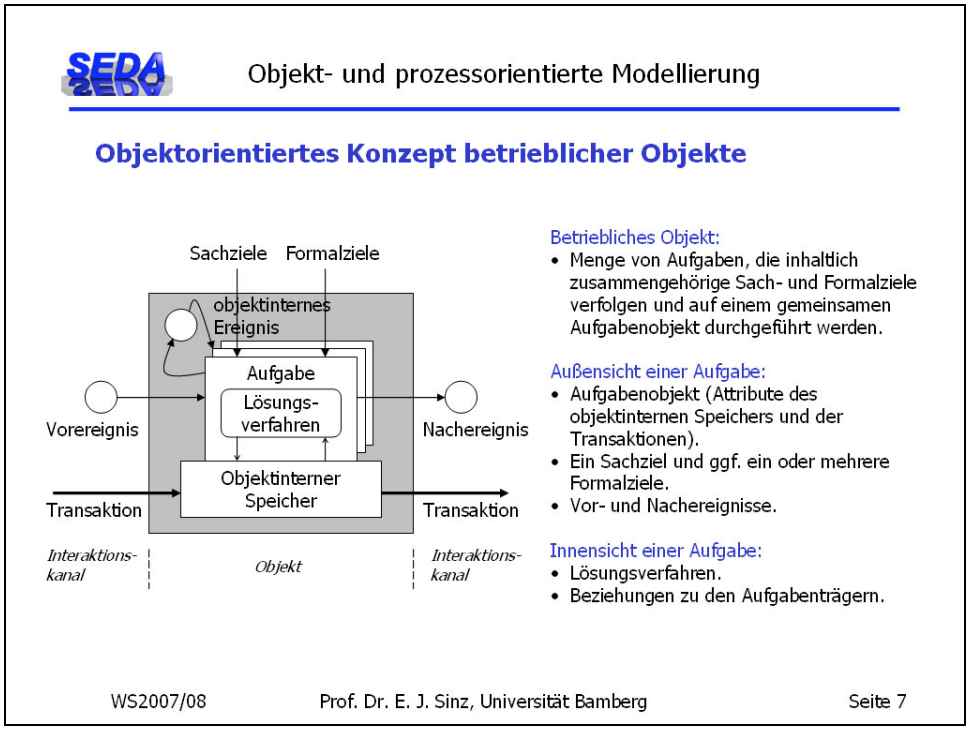


Abbildung 2: Beispiel einer Folie aus Kapitel 6 des Kurses

Die methodische Basis von SOM-Geschäftsprozessmodellen sind das objektorientierte Konzept autonomer, lose gekoppelter betrieblicher Objekte sowie die nachfolgend beschriebene Koordination betrieblicher Objekte in Transaktionen.

Hier soll zunächst das objektorientierte Konzept betrieblicher Objekte diskutiert werden. Ein betriebliches Objekt umfasst danach eine Menge von Aufgaben, die inhaltlich zusammengehörige Sach- und Formalziele verfolgen und auf einem gemeinsamen Aufgabenobjekt durchgeführt werden.

Der Begriff Aufgabe stammt aus der Betriebswirtschaftslehre und wird z.B. von Kosiol (Kosiol, E.: Organisation der Unternehmung, 2. Auflage, Gabler-Verlag, Wiesbaden 1976.) als Zielsetzung für zweckbezogenes menschliches Handeln definiert. Wesentliche Merkmale einer Aufgabe sind ein Verrichtungsvorgang, der an einem Aufgabenobjekt unter Nutzung von Arbeits- und Hilfsmitteln durchgeführt wird, und der sich in Raum und Zeit vollzieht. In der SOM-Methodik wird der Aufgabenbegriff für den Einsatz maschineller Aufgabenträger erweitert. Das bedeutet, dass eine Aufgabe auch eine Zielsetzung für "maschinelles Handeln" spezifizieren kann.

Um Freiheitsgrade bezüglich der Wahl des Automatisierungsgrades und der Automatisierungsform einer Aufgabe erkennen und nutzen zu können, wird zwischen der Außensicht und der Innensicht einer Aufgabe unterschieden. Die Außensicht einer Aufgabe umfasst das Aufgabenobjekt, die Ziele sowie die Ereignisse, welche die Durchführung der Aufgaben auslösen bzw. bei der Durchführung generiert werden. Die Innensicht einer Aufgabe umfasst die Spezifikation des Lösungsverfahrens und stellt die Beziehungen zu den Aufgabenträgern her. Die Spezifikation des Lösungsverfahrens nimmt Bezug auf die Art des eingesetzten Aufgabenträgers. Während bei personellen Aufgabenträgern relativ allgemeine Anweisungen genügen, sind bei maschinellen Aufgabenträgern detaillierte und präzise Arbeitsanweisungen erforderlich, wie sie durch ein Computerprogramm spezifiziert werden.

Ein Beispiel für ein betriebliches Objekt ist *Vertrieb*. Das Objekt fasst unter anderem die Aufgaben *Auftragserfassung*, *Versandsteuerung* und *Fakturierung* zusammen. Sachziel der Aufgabe Auftragserfassung ist, dass nach Durchführung der Aufgabe ein erfasster Auftrag vorliegt. Die zugehörigen Formalziele nehmen auf qualitative Merkmale, wie z.B. Korrektheit, Bezug. Analoge Sach- und Formalziele werden für die weiteren Aufgaben spezifiziert. Verbindende Elemente dieser Aufgaben sind die inhaltlich zusammengehörigen Ziele sowie das gemeinsame Aufgabenobjekt. Dieses besteht aus Attributen des objektinternen Speichers und der Transaktionen. Hierzu gehören Attribute von Aufträgen, Kunden, Artikeln usw.

Abbildung 3: Textteil zur Folie in Abbildung 2

4 Lerninfrastruktur und Betreuungskonzept

Kern der Lerninfrastruktur ist das Learning-Management-System von VAWi, das einem registrierten und angemeldeten Nutzer *Kurs-Räume*, *Service-Räume* und *SIG-Räume* (Special Interest Groups) zur Verfügung stellt. Über die Kurs-Räume erfolgt kursspezifisch der Zugang zu den Skripten und Begleitmaterialien, den Übungsblättern und den Diskussionsforen. Zur Unterstützung der Lernfortschrittskontrolle ist das Skript in 14 Module für je eine VAWi-Semesterwoche aufgeteilt. Die Kursinhalte werden vom Verfasser dieses Beitrags mithilfe von Powerpoint erstellt (Folien und Notizteil); daraus werden HTML-Dateien (Online-Skript) bzw. pdf-Dateien (Offline-Skript) erzeugt.

Während des VAWi-Semesters werden zwei Übungsblätter veröffentlicht, die von den Studierenden zur individuellen Lernzielkontrolle und zum Erwerb von Modellierungskompetenz bearbeitet werden. Die Übungsblätter werden von den Tutoren des Kurses korrigiert und mit individuellen Anmerkungen an die Studierenden zurückgesandt. Je Übungsblatt können maximal 30 Punkte erreicht werden, die in die Kursbewertung eingehen. Der Kurs ist erfolgreich abgelegt, wenn 45 von 90 Punkten in der Abschlussklausur und insgesamt 75 Punkte erreicht wurden. Die Abschlussklausur wird zeitgleich an mehreren Orten in Deutschland durchgeführt.

Das Betreuungskonzept für VAWi-Kurse umfasst neben den kurzen Präsenzphasen zu Beginn und zum Ende des Semesters, in denen jedoch keine Wissensvermittlung stattfindet, eine umfassende Betreuung während des Semesters nach dem Prinzip des One-Level-Support [Ojst07]. Dabei kommunizieren die Lernenden direkt mit den für die jeweilige Frage kompetenten Betreuern (Kurstutor, Kursanbieter). Die Betreuung umfasst:

- Moderation der kursspezifischen Foren,
- Beantwortung von Fragen im Forum,
- Bearbeitung von E-Mail-Anfragen,
- individuelle Rückmeldung zur Bearbeitung der Übungsblätter und
- telefonische Beratung.

5 Erfahrungen

Wie die Erfahrungen zeigen und die Evaluierungsergebnisse bestätigen, sind VAWi-Studierende typischerweise hoch motiviert, sie arbeiten zielorientiert und effizient. Entsprechend hoch sind die Erwartungen an die Qualität von Kursinhalten, Materialien und Betreuung.

Nach jedem Semester wird jeder VAWi-Kurs von den teilnehmenden Studierenden anonym bewertet. Der Kurs *Modellierung von Systemen und Prozessen* erzielt durchwegs sehr gute Bewertungen. Gleichwohl resultieren aus den studentischen Bewertungen wertvolle Anregungen zur inhaltlichen und organisatorischen Weiterentwicklung. Es wird erwartet, dass den Studierenden ein Feedback zu den Bewertungen gegeben wird, das Aussagen darüber enthält, welche Anregungen in welcher Form zukünftig aufgegriffen werden sollen.

Im Vergleich zu den Präsenzstudierenden verfügen VAWi-Studierende naturgemäß über eine größere Studienerfahrung und über Vergleichsmöglichkeiten bezüglich unterschiedlicher Studienangebote. Ihre Rückmeldungen sind daher besonders aussagekräftig und wertvoll. Zum Beispiel zeigte sich im Laufe der Jahre, dass die anfangs angebotenen Chat-Termine nicht als nützlich eingeschätzt wurden. In der Tat wurde bei den Chats nicht die fachliche Tiefe der Diskussion wie in den Foren erreicht. Der Vorteil der Quasi-Gleichzeitigkeit bei Chats wird durch zeitnahe Antworten auf Beiträge im Forum kompensiert. Außerdem wurden von den Studierenden die Bedeutung von Animationen im Online-Skript relativiert und die Anforderungen an die technische Qualität des Offline-Skripts hochgestuft. Der Grund liegt im Lernverhalten vieler VAWi-Studierender, Fahrten mit öffentlichen Verkehrsmitteln von und zur Arbeitsstelle für das Studium der Lehrmaterialien zu nutzen.

Literaturverzeichnis

[AKF+02] Adelsberger, H.H.; Körner, F.; Ferstl, O.K.; Friedrich, S.; Schmitz K.: Der Virtuelle Weiterbildungsstudiengang Wirtschaftsinformatik (VAWi) - Konzepte und Erfahrungen. Tagungsband der Teilkonferenz E-Learning im Rahmen der Multi-Konferenz Wirtschaftsinformatik (MKWI02), Nürnberg, 2002, S. 35 – 52.

[BPR06] Borner, L.; Paech, B.; Rückert J.: Vom Modellverstehen zum Modellerstellen. In: Desel, J.; Glinz, M. (Hrsg.): Modellierung in Lehre und Weiterbildung. Technischer Bericht ifi-2006.03, Institut für Informatik, Universität Zürich, 2006, S. 7 – 15.

[FeDe06] Ferstl, O.K.; Derra, S.: Angebotsformate, Zugangswege und Abschluss im Virtuellen Weiterbildungsstudiengang Wirtschaftsinformatik (VAWi). In Faulstich, P.; Beyersdorf, M.; Vogt, H.; Grieb, I.; Hörr, B.; Strate, U.; Strauß, A. (Hrsg.): Hochschule & Weiterbildung, Zeitschrift der DGWF 2/2006, Verlag DGWF e.V., Hamburg, 2006, S. 114-118.

[FeSi06] Ferstl, O.K.; Sinz, E.J.: Grundlagen der Wirtschaftsinformatik. 5. Auflage, Oldenbourg, München, 2006.

[GrTa06] Gröne, B.; Tabeling P.: Modellierung von Softwaresystemen – Vorlesung und Seminar im Studiengang IT-Systemtechnik. In: Desel, J.; Glinz, M. (Hrsg.): Modellierung in Lehre und Weiterbildung. Technischer Bericht ifi-2006.03, Institut für Informatik, Universität Zürich, 2006, S. 16 – 21.

[Ojst07] Ojstersek, N.: Organisation tutorieller Betreuung beim E-Learning. In: Eibl, Ch.; Magenheim, J.; Schubert, S.; Wessner M. (Hrsg.): DeLFI 2007, Die 5. e-Learning Fachtagung Informatik, 17.- 20. September 2007 an der Universität Siegen, Gesellschaft für Informatik e.V. (GI), Bonn, 2007, S. 67 – 78.

[Sinz07] Sinz, E.J.: Modellierung von Systemen und Prozessen. VAWi-Skript für das WS 2007/08, Bamberg, 2007.

[ThLo07] Theling, Th.; Loos, P.: Die multiperspektivische Verwendung von ERP-Systemen in der Wirtschaftsinformatik-Lehre. In: Breitner, M.H.; Bruns, B.; Lehner, F. (Hrsg.): Neue Trends im E-Learning. Aspekte der Betriebswirtschaftslehre und Informatik, Physika, Heidelberg, 2007, S. 383 – 398.

Software-Modellierung und Ethik

Jochen Koubek

Humboldt-Universität zu Berlin
Institut für Informatik Informatik in Bildung und Gesellschaft
jochen.koubek@hu-berlin.de

Zusammenfassung: Software-, Daten und Unternehmensmodellierung ist eine komplexe Tätigkeit, die an Entwickler vielfältige Anforderungen stellt. Neben mathematisch-technischem Sachverstand sind u.a. ökonomische, ethische, rechtliche, interkulturelle, psychologische, soziale oder politische Kompetenzen gefragt. Im vorliegenden Beitrag wird die ethische Dimension der Software-Modellierung betrachtet und Vorschläge zu ihrer Berücksichtigung in der Lehre werden formuliert.

1 Fallbeispiel

Anna ist in ihrer Firma verantwortlich für die Anforderungsanalyse und Modellierung von Software-Projekten. Eines Tages stellt ihr Kunde eine Anforderung, von der sie weiß, dass in ihrem Entwicklungsteam die nötigen Kompetenzen zur Umsetzung fehlen. Allerdings fehlen zur Zeit auch die Mittel für Neueinstellungen. Es wäre für sie leicht, den Kunden davon zu überzeugen, dass sein Wunsch technisch nicht innerhalb des von ihm vorgesehenen Budgets umsetzbar ist. Wenn sie aber zugibt, dass ihre Firma nicht die nötigen Fachkompetenzen bieten kann, riskiert sie, den Auftrag zu verlieren. Wenn sie die Kundenwünsche nicht umsetzt, liefert sie ein System, das unter den Erwartungen der künftigen Nutzer bleibt und enttäuscht damit das Vertrauen, das in ihre Projektgruppe gesetzt wurde.

Der Konflikt, in dem Anna sich befindet ist ein typisches Beispiel für Probleme, die bei der Modellierung von Software-Systemen auftreten können. Neben den Anforderungen an technische Kompetenzen sind die frühen Phasen der Software-Entwicklung primär geprägt durch den Konflikt zwischen verschiedenen Anspruchshaltungen und sozialen Verpflichtungen. Von derartigen Problemen in Modellierungsprozessen handelt dieser Aufsatz und diskutiert die Frage, wie der einzelne Entwickler mit ihnen umgehen kann und wie derartige Themen bei der Ausbildung berücksichtigt werden können.

Natürlich gibt es effektvollere Fallbeispiele, um ethische Probleme beim Modellieren zu diskutieren, Fälle, in denen ein Modellierer über die Wegrationalisierung von Arbeitsplätzen entscheidet oder ein unter Zeitdruck erstelltes Modell bei seiner Implementierung Leib und Leben der Benutzer gefährdet. Dass hier ein vergleichsweise harmloser Fall gewählt wird, erfolgt in Hinblick auf die Beobachtung, dass ethische Konflikte nicht erst beim Verlust von Arbeitsplätzen oder bei drohenden Gesundheits- und Vermögensschäden auftreten, sondern bereits in kleinen Entscheidungen eine Rolle spielen. Denn diese kleinen Entscheidungen prägen die alltägliche Software-Entwicklung weitaus stärker als die großen Fragen und sollten bereits angemessen diskutiert werden. Dafür aber

gilt es, die wichtigsten Begriffe einzuführen, mit denen ethische Probleme beschrieben werden können.

2 Ethische Grundbegriffe

Ethik ist als Philosophie der Moral die Reflexion menschlichen Handelns im Spannungsfeld zwischen Freiheit und Verantwortung. **Freiheit** bedeutet im positiven Sinn die Ermöglichung von Handlungen, im negativen Sinn die Abwesenheit von Handlungseinschränkungen ([DÜ06] S. 358 ff.). Die positive Freiheit wird begrenzt durch Kompetenzen, ohne die Handlungen nicht möglich sind. Zwar hat ein jeder die Freiheit, Software zu entwickeln, aber nicht jeder hat die erforderlichen Kompetenzen, diese Freiheit konstruktiv zu nutzen. Die negative Freiheit findet ihre Grenzen in sozialen Normen und Gesetzen, die bestimmte Handlungsoptionen unter Androhung negativer Sanktionen ausschließen bzw. deutlich erschweren. Bei der Software-Entwicklung gibt es zahlreiche rechtliche Einschränkungen, z.B. im Urheberrecht (Kopieren von Quelltext) oder im Datenschutzrecht (Sicherung personenbezogener Daten). Nun können normative Vorgaben umgangen werden, was aber regelmäßig unerwünschte Konsequenzen zur Folge hat. Freiheit als Möglichkeit, immer auch anders zu handeln, verweist unmittelbar auf das Konzept der **Verantwortung**, hier (in Anlehnung an [GI04]) in sechs Aspekten verstanden als (i) die Zuständigkeit von Personen, Gruppen oder Institutionen (ii) für das eigene Tun und Lassen (iii) gegenüber einem Adressaten (iv) vor einer gegebenen Instanz (v) im Rahmen eines Kontextes (vi) in Bezug auf Verpflichtungen. Wer anders hätte handeln können, muss auch für seine konkreten Handlungen einstehen.

Die **Verpflichtungen** zeichnen sich dadurch aus, dass sie durch soziale **Sanktionen** geschützt sind, die bei ihrer Überschreitung drohen. Das kann von sozialer Abwertung durch Kollegen bis zu Geld- und Haftstrafen durch das Rechtssystem reichen. Aber auch das **Gewissen**, verstanden als innere Instanz, die in der kindlichen Auseinandersetzung von internalisierten Normen und eigenen Wünschen entsteht, kann durch emotionales Wohl- oder Unwohlsein moralisches Handeln motivieren oder unmoralisches sanktionieren.

Ein **ethisches Dilemma** entsteht, wenn ein Handelnder in einer gegebenen Situation zwei Verpflichtungen einhalten muss, aber nur eine einhalten kann ([HÖ97] S. 206). Jede Entscheidung für eine Verpflichtung bedeutet die Verletzung einer anderen.

3 Zur ethischen Dimension der Software-Modellierung

Die sechs Komponenten des verantwortlichen Handelns sollen nun in aller gebotenen Kürze an einem Software-Entwicklungsprojekt vorgestellt werden.

(i) jemand ist verantwortlich: Jeder Bestandteil des Modells, jedes Diagramm und jede Zeile Programmcode wird direkt oder indirekt von einer natürlichen Person entworfen und geschrieben. Da an einem Software-Entwicklungsprojekt aber mehrere Personen beteiligt sind – Modellierer, Architekten, Programmierer, Korrektoren, Tester etc. – kann

niemand die individuelle Schuld für Fehler übernehmen. Ähnliches gilt für Fehler in CASE-Tools und bei automatisch erzeugtem Code. Wenn hier von Ethik und Software-Modellierung die Rede ist, liegt der Augenmerk natürlich auf den Personen, die für die Erstellung des Modells verantwortlich sind. Dies sind aber nicht nur diejenigen, die UML-Diagramme entwerfen, sondern immer auch der Projektleiter, der sie vertritt und der Auftraggeber, der sie akzeptiert. Die Entscheidung für oder gegen die Umsetzung eines Modells liegt i.d.R. nicht im Aufgabenbereich des Modellierers. Für die Konsequenzen des Modells übernimmt regelmäßig der Arbeitgeber bzw. stellvertretend der Projektleiter die Verantwortung. Insofern liegen evtl. auftretende ethische Konflikte beim Management. Dennoch lässt sich die Verantwortung vom Modellierer nicht einfach nach oben abschieben. Denn die Vorschläge für eine Auftragsumsetzung, die vom Projektleiter abzuwägen und nach außen zu verantworten sind, werden vom Modellierer gemacht und im Modell formuliert. Was dort nicht auftaucht, kann später keine Probleme machen. Insofern wird Modellierung hier verstanden als die Phase der Software-Entwicklung, in der über Ressourcen, ihren Verbrauch und ihre Zuteilung entschieden wird, wobei diese Entscheidung durchaus in mehreren Stufen von verschiedenen Personen erfolgen kann.

für etwas: Verantwortung bezieht sich immer auf eine konkrete Handlung bzw. auf einen Handlungszusammenhang. Jemand kann für eine Software die Verantwortung übernehmen, was aber nichts anderes heißt, als dass er für das erwartungsgemäße Funktionieren der Software verantwortlich ist, was wiederum bedeutet, dass er dafür einsteht, dass sie entsprechend fehlerfrei entwickelt wird, was ein korrektes Modell voraussetzt.

gegenüber einem Adressaten: Jedes Software-Projekt hat eine Fülle von Adressaten: Kollegen, die auf ein Teilmodul warten, Projektleiter und Vorgesetzte, die Qualität erwarten, Kunden, die für die Software bezahlen oder die Benutzer, die sie einsetzen. Regelmäßig ist der Kunde bzw. derjenige, der die Abnahme unterzeichnet und den Scheck ausstellt, der primäre Adressat. Innerhalb des Projekts ist der Modellierer gegenüber dem Projektleiter verantwortlich, ein umsetzbares Modell zu erstellen.

vor einer Instanz: Verantwortliches Handeln muss immer von jemandem gefordert werden. Diese Einforderung kann sich darin erschöpfen, Rechtfertigung zu verlangen, sie kann aber auch in Vermögens- oder Personenhaftung bestehen. Wichtig ist, dass diese Einforderung gegen Widerstand durchgesetzt werden kann. Verantwortung verweist damit immer auf einen Machthaber, der zur Durchsetzung gegebener Normen Sanktionen verhängen und umsetzen kann. In Software-Entwicklungsprojekten können die oben genannten Adressaten die Rolle dieser Instanz einnehmen, bei jeder dieser Personengruppen ist die drohende Sanktion und damit die Art der Verantwortung unterschiedlich. Darüber hinaus gibt es noch die individuellen und der gesellschaftlichen Instanzen, d.h. das eigene Gewissen, das sich bei bewusst nachlässiger Arbeit melden kann oder juristischen Instanzen, die im Falle eines Rechtsstreits die gesellschaftlichen Rechtsnormen durchsetzen. Erfolgt die Modellierung im Rahmen eines Arbeitsvertrags, dann ist der Arbeitgeber die Rechenschaft fordernde Instanz. Im Gegensatz zum Bauwesen, wo Statiker und Architekten regelmäßig für die Fehler ihrer Pläne haften, werden bei der Softwareentwicklung derartige Anspruch fast nie geltend gemacht.

im Rahmen eines bestimmten Kontextes: Die konkrete Verantwortlichkeit für Probleme muss immer innerhalb des Projektablaufs betrachtet werden. Wer war zu welchem Zeitpunkt für welchen Abschnitt verantwortlich?

in Bezug auf Kriterien: Auf welche Werte, Regelungen oder andere gesellschaftliche Verpflichtungen bezieht sich die übernommene Verantwortung? Diese Frage soll im Folgenden weiter entfaltet werden.

4 Verpflichtungen in Software-Projekten

In einem Software-Projekt ergeben sich verschiedene Verpflichtungen, denen sich bereits die für die Modellierung verantwortlichen Personen ausgesetzt sehen. Drei davon sollen im Folgenden kurz angesprochen werden: Verpflichtungen, die sich (1) aus den Interessen des Kunden, die sich (2) aus der ökonomischen Beziehung zwischen Auftraggeber und Auftragnehmer und die sich (3) aus den Interessen des Auftragnehmers ergeben.

1. Unternehmenspolitische Interessen des Kunden

Krüger und Eggebert ([KE03], S. 211 ff.) berichten von einem Systemverantwortlichen, der bei der Neuentwicklung des ihm unterstellten Systems seinen Status als zentraler Ansprechpartner zu verlieren fürchtete und die Entwicklung daher mit allen Mitteln boykottierte. Die Autoren schließen daraus, dass jede Architekturänderung von einem Organisationsmanagement begleitet werden müsse. Doch ein derartiger Prozess ist nicht bei jedem Software-Projekt möglich. Ein Projektleiter, der verhindern möchte, dass Mitarbeiter des Kunden dem Projekt im Weg stehen, sollte die persönlichen Interessen dieser Mitarbeiter berücksichtigen. Software-Architekten und Projektverantwortliche, deren Projekte nicht von organisationsverändernden Maßnahmen flankiert werden, sollten die unternehmenspolitischen Interessen ihrer Kunden als informelle Verpflichtung ernst nehmen und sie sorgfältig gegen die anderen Interessen abwägen.

2. Ökonomische Aushandlungen zwischen Auftraggeber und -nehmer

Die verbindlichsten Verpflichtungen in einem Projekt sind die vertraglich vereinbarten Leistungen zwischen dem Auftraggeber und dem Auftragnehmer. Diese können ein Kunde und eine Softwarefirma sein oder innerhalb einer Firma die Leitung und ein Entwicklungsteam. Die Aushandlung wird in der Regel in Form eines schriftlichen Vertrags festgehalten, dem eine detaillierte Projektplanung folgt. Für jeden Projektschritt werden im Feinplan die benötigten Ressourcen, Zeit, Mitarbeiter und Kosten geplant. Der Auftragnehmer ist dafür verantwortlich, das Projekt innerhalb dieses ausgehandelten Rahmens erfolgreich zum Abschluss zu bringen. Über die vereinbarten Ressourcen kann er im Gegenzug frei entscheiden. Jede Überschreitung der Vereinbarung durch den Auftragnehmer sollte im Interesse des Auftraggebers liegen und mit diesem auch unter Angabe der Gründe abgesprochen werden. Dies sind die Grundregeln des Projektmanagements [PM04].

Nun sind die Überziehungen von Ressourcenvereinbarungen bei Software-Projekten legendär, kein Lehrbuch über Software-Entwicklung verzichtet im ersten Kapitel auf

Zitate aus den Chaos-Reports der Standish-Group [SG] bei der Diskussion möglicher Gründe und Auswirkungen der Software-Krise, von denen überteuerte oder überlange Entwicklungsprojekte eine Ausprägungsform sind. Unter hohem Konkurrenzdruck verpflichten sich Software-Firmen zur Projektabwicklung unter unrealistischen Bedingungen. Werden Aufträge vorrangig an den Anbieter mit dem günstigsten Angebot vergeben, entsteht die Tendenz, in diesem Angebot die eigenen Kräfte zu über- und mögliche Probleme zu unterschätzen. Für Ressourcenüberschreitungen tragen Auftraggeber einen Teil der Verantwortung, wenn sie besonders kurze Liefertermine fordern oder ein sichtlich zu knappes Budget einplanen. Edward Yourdon charakterisiert derartige Projekte als *death march projects:*

> *A death march project is one for which an unbiased, objective risk assessment (which includes an assessment of technical risks, personal risks, legal risks, political risks, etc.) determines that the likelihood of failure is > 50 percent.*
> [YO97], S. 4.

Natürlich muss niemand einen externen Auftrag zu diesen Bedingungen annehmen, zumal das finanzielle Risiko häufig beim Auftragnehmer bleibt. Anders sieht es aus bei Inhouse-Projekten, die von Vorgesetzten verordnet werden oder wenn aus bestimmten Gründen ein Projekt zum Risikofall wird. In derartigen Fällen müssen regelmäßig Zugeständnisse an die Qualität des Produkts gemacht werden ([YO97], S. 77 ff.), die sich nicht selten als ethische Dilemmata präsentieren. Die Verantwortung dafür liegt aber nicht nur beim Auftragnehmer, sondern aufgrund der unrealistischen Forderungen und Randbedingungen auch beim Auftraggeber.
Basieren die Gründe für Überziehungen auf Interessen des Auftragnehmers, die denen des Kunden zuwider laufen, so müssen sie gegen vertragliche Verpflichtungen abgewogen werden, was ebenso in ein ethisches Dilemma münden kann. Denn gerade in der Modellierungsphase wird über die weiteren Ressourcenanforderungen entschieden. Im Eingangsbeispiel musste Anna die Entscheidung abwägen, Anforderungen des Kunden abzulehnen, ohne die wahren Gründe anzugeben und damit die vertraglichen Vereinbarungen zu verletzen, sorgfältig und im Interesse des Kunden zu arbeiten. Ebenso häufig wie versucht wird, Anforderungen abzuwehren, werden sie mutwillig verallgemeinert, ohne den Kunden auf die Mehrbelastung aufmerksam zu machen. Dann werden z.B. einfache Verwaltungsaufgaben mit komplexen Datenbanken modelliert, die für eine viel größere Problemklasse geeignet wären. Brooks [BR75] beschreibt den Second-System-Effect als Versuch des Entwicklerteams, beim zweiten Mal alle Eigenschaften im System unterzubringen, die beim ersten Mal weggelassen werden mussten. Das Wunschsystem der Entwickler muss aber nicht das Produkt sein, das dem Kunden vorschwebt. Auch ist er häufig nicht an der technisch elaboriertesten Lösung interessiert, sondern wäre mit einer kleineren, dafür aber kosten- und ressourcensparenderen Variante zufrieden. Die Versuchung, das eigene Leistungsvermögen über die Interessen des Kunden zu stellen, verweist direkt auf die dritte Quelle für Dilemmata:

3. Interessen des Auftragnehmers

Hohmann betont in [HO03] die Bedeutung des Entwicklerteams für die Software-Architektur. Deren Erfahrungen, Vertrauen, Wünsche, Ambitionen, Vorlieben und Abneigungen beeinflussen den Modellierungsprozess ebenso wie die Kundenvorgaben:

»To check this, ask yourself if you've ever worked on a project where you wanted to work on a particular aspect of the architecture because you knew you could do a better job than anyone else (confidence based on experience); you wanted to learn the underlying technology (desire); you thought that doing a good job might earn you a bonus or promotion (aspiration); or, you were concerned that no one else on the team had the requisite skill or experience to solve the problem "the right way" (fear).« ([HO03], S. 3)

Software-Projekte fallen unter Melvin Conways Gesetz [CO68], eine organisationssoziologische These von hoher Evidenz, die er 1968 formulierte: *organizations which design systems […] are constrained to produce designs which are copies of the communication structures of these organizations.* Diese Aussage beruht auf der Beobachtung, dass jedes System von einer Gruppe von Menschen gestaltet wird, die miteinander kommunizieren. Die Struktur dieser Gruppe führt zur Aufteilung des Systems in Teilsysteme, deren Schnittstellen die Kommunikationswege der Entwickler abbilden und deren Funktionalität mit den Fähigkeiten der Entwickler korreliert.

Ein vorausschauender Software-Architekt wird die Struktur des Entwicklerteams bei der Modellierung berücksichtigen, damit es bei der Umsetzung nicht zu voraussehbaren Komplikationen und Kompetenzrangeleien kommt. Dass solcherlei Umstände die Entwicklung des gewünschten Systems beeinflussen, kann dem Kunden nur selten erklärt werden. Modellierungsentscheidungen, die getroffen werden, um die Struktur des Entwicklerteams zu berücksichtigen, müssen daher anderweitig begründet werden. Eine derartige Unaufrichtigkeit dem Kunden gegenüber bedarf sorgfältiger Abwägung und führt nicht selten zu ethischen Herausforderungen.

Neben den drei genannten gibt es noch eine Reihe weiterer Verpflichtungen, die im Folgenden aus Platzgründen nicht diskutiert werden können, in realen Modellierungsprozessen aber durchaus relevant sein können, darunter rechtliche Aspekte (diskutiert z.B. in *Recht für Software- und Web-Entwickler* [OT06]) oder ökologische Anforderungen und die Fragen nachhaltiger Entwicklung von Informationstechnologien (z.B. Hilty et al. *Das Vorsorgeprinzip in der Informationsgesellschaft* [HI03]).

Ein ethisches Dilemma entsteht, wenn ein Handelnder zwei oder mehr Verpflichtungen einhalten muss, aber nur eine einhalten kann. Annas Dilemma im Eingangsbeispiel lag darin begründet, sowohl den vertraglichen Vereinbarungen als auch der Zusammensetzung des Entwicklerteams verpflichtet zu sein. Anna muss abwägen zwischen dem Ruf ihrer Firma, der unmittelbar mit ökonomischem Erfolg zusammen hängt und zwischen dem in sie gesetzten Vertrauen ihres Kunden. Egal, wie Anna sich entscheidet, sie muss Reputation und Vertrauen gegeneinander abwägen. Soweit die Analyse, die im konkreten Fall natürlich noch die Beschreibung der Vertragsvorgaben und der Teamstruktur beinhalten würde. Wie aber soll Anna sich verhalten? Es liegt im Wesen des Dilemmas, keine eindeutige Lösung zu haben. Eine mögliche Lösung ergibt sich erst im Abwägen verschiedener Werte. Um dieses Abwägen nicht nur der individuellen Verantwortung zu überlassen, wurden gerade für die Arbeit von Informatikern von verschiedenen Berufsverbänden ethische Leitlinien formuliert, in denen zumindest die wichtigsten Werteprioritäten festgehalten werden sollen.

5 Ethische Leitlinien

Die Gesellschaft für Informatik hat in ihren ethischen Leitlinien [GI04] einen Minimalkonsens formuliert, nach dem sich ein GI-Mitglied als Angestellter, als Führungskraft oder in Lehrtätigkeit orientieren sollte. Ähnliche Ansätze gibt es mit dem Code of Ethics der ACM [AC92] oder der IEEE [IE06]. Eine Zusammenstellung ethischer Standards von IT-Berufsverbänden weltweit findet sich bei [BE96].

In den ethischen Leitlinien der GI wird Wert gelegt auf die Ausbildung professioneller Kompetenzen, auf Gestaltung von Arbeitsplätzen, Möglichkeiten der Partizipation und Mitbestimmung. Der Informatiker wird ermutigt, seine gesellschaftliche Verantwortung zu erkennen und wahrzunehmen.

Der Code of Ethics der ACM formuliert einen Katalog von Werten wie »society and human well-being«, »honest and trustworthy«, »privacy« oder »highest quality«. Ähnlich verfährt der Code of Ethics der IEEE.

Bei ethischen Dilemmata wie dem von Anna, in dem das Verhalten einem Kunden gegenüber berührt wird, helfen diese Leitlinien allerdings nicht, weil sie in ihrem Abstraktionsgrad bei subtilen Abwägungen zwischen Werten keine Pauschalvorgaben machen wollen und können. Denn der Grund für Annas Ablehnung der Anforderung liegt ja auch in dem Wunsch, mit den vorhandenen Mitteln ein Höchstmaß an Qualität zu erreichen. Sie würde nur verschweigen, dass sie nicht die Mittel hat, noch besser zu arbeiten. Insofern ergibt sich der Konflikt zwischen den Verpflichtungen bezüglich der Werte *Ehrlichkeit* und *Qualität*. Eine pauschale Über- oder Unterordnung dieser Werte überfordert jede ethische Leitlinie. Deren Nutzen liegt vor allem in der Aufforderung, ethische Probleme und Wertkonflikte als solche erst einmal zu erkennen. Denn vor der Wertentscheidung steht der Wille zur Reflexion über individuelle und korporative Verantwortung und die Bereitschaft, die eigene ethische Urteilskraft zu stärken.

6 Ethik in der Ausbildung

Urteilskraft entzieht sich ihrer operationalisierbaren Vermittelbarkeit, man kann sie nicht lehren. Dies ist nicht etwa eine didaktische Lücke, sondern liegt im Wesen der Urteilskraft selbst begründet: als Vermittlerin zwischen Fall und Regel kann es keine Regeln geben, nach denen sie anzuwenden ist, weil die Anwendung dieser Regeln ihrerseits eine Urteilskraft zweiter Ordnung voraussetzen würde usw. So kann sich ein Personalchef, der sich bei der Auswahl eines Bewerbers nicht mehr nur auf seine Lebenserfahrung berufen möchte, graphologische Gutachten, Intelligenztests oder Assessment-Center als Messinstrument heranziehen. Die Urteilskraft, die bei der Erstellung dieser Tests aufgewandt wurde, kann er allerdings nicht durch weitere Instrumente ersetzen, er kann sie lediglich gegen seine eigene einzutauschen versuchen. Urteilskraft kann man nicht lehren, wenn Lehren bedeutet, die Operationen angeben zu können, die zum gewünschten Verhalten führen. Urteilskraft kann man aber lernen.

Das Mittel der Wahl ist in der Ethikdidaktik seit Kohlberg [KO58] die Ausrichtung an Fallbeispielen für ethische Dilemmata ([FR], S. 315). Die Bearbeitung und Diskussion

ethischer Konfliktfälle (z.B. [SP97], [LA95]) fördert neben der ethischen Urteilskraft auch die explizite Formulierung individueller Wert- und Präferenzordnungen. Die Entscheidung, die es in derartigen Fällen zu treffen gilt, ist immer die Entscheidung zwischen konkurrierenden Werten. Die eigenen Werte und ihre Hierarchie zu kennen, bevor sie in konkreten Entscheidungssituationen gefordert sind, kann den Argumentationsdruck mindern, den eine Entscheidung gegen einen Wert nach sich zieht.

Zur Förderung der ethischen Urteilskraft ist es daher ratsam, Entscheidungssituationen in der Lebenswelt der Lernenden zu verankern. Idealerweise erfolgt dies nicht in einem abgesetzten Unterrichtsblock, sondern integrativ im Kontext der praktischen Ausbildung:

Die **Software-Modellierungsaufgaben** sollten in Hinblick auf ihren potenziellen ethischen Konfliktstoff gewählt werden. Dies gilt für einfache Modelle ebenso wie für komplexe Semesterprojekte, z.B. »Modellieren Sie ein Programm zum Monitoring der Computernutzung von Angestellten.« Bereits bei der Aufgabenstellung kann auf mögliches Diskussionspotenzial hingewiesen werden: »Bedenken Sie, dass Ihr Entwurf dem Betriebsrat zur Prüfung vorgelegt wird.«

Die **ethischen Konflikte** sollten im Rahmen der Ausbildung thematisiert werden, ohne nach einer Lösung zu drängen. Die Entscheidung für oder gegen eine Position ist allerdings auf ihre impliziten Wertkonflikte zu befragen und im Gespräch zu diskutieren oder schriftlich zu analysieren. Diese Fähigkeit zur Dissensanalyse, die Aushandlung der identifizierten Konflikte ist Teil der Kommunikations- und Sozialkompetenzen (vgl. [OT99], S. 115).

Weitere zu fördernden Kompetenzen in diesem Sinne sind nach [BE06]:

Wahrnehmungs- und Deutungskompetenz als die Fähigkeit, Sachverhalte unter ethischer Perspektive wahrzunehmen und zu beschreiben. Das erfordert eine Kenntnis der ethischen Grundbegriffe und ihrer Anwendung.

Argumentations- und Urteilskompetenz als Fähigkeit, ethische Sachverhalte begrifflich zu erschließen, argumentativ zu gewichten und mit Blick auf ihre Voraussetzungen und Folgen zu bewerten.

Empathiekompetenz als die Fähigkeit, sich in die Lage anderer Menschen hineinzuversetzen und ihr gegebenenfalls abweichendes Urteil nachzuvollziehen.

Personale Kompetenz als Fähigkeit, die eigene moralische Überzeugung glaubwürdig zum Ausdruck zu bringen.

Praktische Kompetenz als die Fähigkeit, das zu tun, was man als richtig eingesehen hat. Zwar ist richtiges Urteilen nicht die hinreichende Bedingung für richtiges Handeln, sie ist aber dafür aber notwendig.

Die ethischen Leitlinien der GI fordern darüber hinaus noch **juristische Kompetenz** im Sinne der Kenntnis der einschlägigen rechtlichen Regelungen wie Datenschutzrecht, Urheberrecht, Computerstrafrecht oder Verbraucherschutzrecht [GI04].

Nicht nur Techniker haben Schwierigkeiten mit Problemen, deren Lösung sie nicht zweifelsfrei bestimmten können, zumal wenn dabei auf eine Methode zurück gegriffen wer-

den soll, die ihrem Wesen nach nicht operationalisierbar ist wie die Urteilskraft. Dennoch gibt es auch im Ethikunterricht Fakten in Form von »Positionen, Begründungsmuster und Argumentationsfallen, die gleichermaßen an Stammtischen wie in Expertendiskussionen immer wieder nachzuweisen sind« ([OT99], S. 115). Einige Stichworte mögen das Feld skizzieren:

In der normativen Ethik unterscheidet man zwei große Positionsfamilien mit jeweils eigenen Begründungsmustern für moralisches Handeln: Die **Folgenethiken** (Teleologien) begründen Handlungsentscheidungen aus den möglichen Folgen, z.B. wie der Utilitarismus am ‚größten Glück der größten Zahl.' Verschiedene utilitaristische Positionen richten ihre Begründungen an dem Nutzen individueller Handlungen (Handlungsutilitarismus) oder am Nutzen überindividueller Regeln (Regelutilitarismus). Der Nutzenbegriff kann anhand einer Präferenzordnung weiter objektiviert werden und findet seinen formales Pendant in der mathematischen Handlungs- und Spieltheorie (Präferenzutilitarismus).

Die **Pflichtenethiken** (Deontologien) richten sich nach im Vorfeld definierten zeitlos gültigen Regeln, unabhängig von den Folgen des Handelns. Eine Variante ist die **Diskursethik**, die ihre Grundregel aus der Möglichkeit begründet, über Handlungen einen herrschafts- und vorurteilsfreien Diskurs führen zu können. Wer an einem solchen Diskurs teilnimmt, hat das Moralprinzip bereits akzeptiert. Auf der anderen Seite kann es von niemandem in einer Diskussion sinnvoll angezweifelt werden, ohne in einen Selbstwiderspruch zu geraten. Die universelle Gültigkeit ergibt sich aus der Unmöglichkeit des diskutierbaren Zweifels, wodurch es auch für nicht-diskursive Handlungen relevant wird, die in einem offenen Diskurs von allen Diskurspartnern akzeptiert werden müssen.

Handlungsentscheidungen, die von verschiedenen Parteien unter der impliziten Annahme der Gültigkeit ihrer ethischen Standpunkte diskutiert werden, führen bei verschiedenen Positionen zwangsläufig zu Missverständnissen, solange diese Positionen nicht offen verhandelt werden. Eine derartige meta-ethische Aushandlung setzt aber die Kenntnis ethischer Positionen und Begründungsmuster voraus und sollte im besten Fall bereits im Vorfeld geübt worden sein, um gängigen argumentativen Fallen und Gegenargumenten zu begegnen: darf der Pflichtethiker lügen, wenn er damit das Vermögen eines Unschuldigen rettet?, auf welche Art kann der Utilitarist das Glück messen, das er maximieren möchte?, wie will der Diskursethiker eine Entscheidung treffen, wenn er vorher alle Meinungen diskutieren muss?

Die Geschichte der Ethik ist die Geschichte der Argumente und Konzepte für moralisches Handelns aber auch die Geschichte der Gegenargumente und Gegenkonzepte. Da Wertkonflikte auch in Entscheidungssituationen der Softwareentwicklung unvermeidbar sind, empfiehlt es sich, mit einem Blick in diese Geschichte rechtzeitig die eigene Wahrnehmung zu schärfen, die Urteilsfähigkeit zu üben und die diskursiven Kompetenzen zu stärken.

Literaturverzeichnis

Alle Internet-Quellen wurden im Februar 2008 überprüft.

[AC92] ACM: *ACM Code of Ethics and Professional Conduct*. 1992. http://www.acm.org/about/code-of-ethics

[BE96] Berleur, Jacques; Brunnstein, Klaus: *Ethics of Computing*. London etc.: Chapman & Hall, 1996.

[BE06] SenBJS: *Rahmenplan Ethik*. Berlin, 2006. http://www.berlin.de/imperia/md/content/sen-bildung/schulorganisation/lehrplaene/sek1_ethik.pdf

[BR75] Brooks, Fred: *The Mythical Man Month*. 1975.

[CO68] Conway, Melvin: *How Do Committees Invent?* In: Datamation (14) 4, 1968, S. 28-31.

[DÜ06] Düwell, Marcus et al.: *Handbuch Ethik*. 2. Aufl. Stuttgart: Metzler, 2006.

[FR94] Franzen, Winfried: *Ethikunterricht.* In: Hastedt, H; Martens, E: *Ethik.* Reinbek: Rowohlt, 1994.

[GI04] Gesellschaft für Informatik (Hrsg.): *Ethische Leitlinien*. 2004. http://www.gi-ev.de/fileadmin/redaktion/Download/ethische-leitlinien.pdf

[HI03] Hilty, Lorenz et al.: *Das Vorsorgeprinzip in der Informationsgesellschaft*. Bern: TA-Swiss, 2003.

[HO03] Hohmann, Luke: *Beyond Software Architecture*. Amsterdam: Addison-Wesley, 2003.

[HÖ97] Höffe, Ottfried: *Lexikon der Ethik*. 5. Aufl. München: Beck, 1997.

[IE06] IEEE: *Code of Ethics.* 2006. http://www.ieee.org/web/membership/ethics/code_ethics.html

[KE03] Krüger, Sasche; Eggebert, Jörg S.: *IT-Architektur-Engineering*. Bonn: Galileo, 2003.

[KO58] Kohlberg, Lawrence: The Development of Modes of Thinking and Choices in Years 10 to 16. Ph. D. dissertation, University of Chicago. Nachdruck in Puka, Bill: *Kohlberg's Orginal Study of Moral Development*. Routledge, 1994.

[LA95] Langford, Duncan: *Practical Computer Ethics*. Berkshire: McGraw-Hill, 1995.

[OT99] Ott, Konrad; Busse, Johannes: *Ethik in der Informatik*. Tübinger Studientexte, 1999.

[OT06] Otto, Dirk: *Recht für Software- und Web-Entwickler.* 2. Aufl. Galileo, 2006.

[PM04] Project Management Institute: *Project Management Body of Knowledge.* PMI: 2004.

[SG] Standish Group: *Chaos Reports*. 1994, 2003, 2007. http://www.standishgroup.com/chaos_resources/index.php

[SP97] Spinello, Richard A.: *Case Studies in Information and Computer Ethics.* Prentice-Hall, 1997.

[YO97] Yourdon, Edward: *Death March*. Prentice-Hall, 1997.

Unterstützung des Erlernens von endlichen Automaten mit Hilfe von Mustern

Agathe Merceron

Fachbereich Informatik und Medien
Technische Fachhochschule TFH Berlin
Luxemburgerstrasse 10 53133 Berlin
merceron@tfh-berlin.de

Abstract: Endliche Automaten verbergen sich als Modell hinter Geldautomaten, Zigarettenautomaten, Fahrkartenautomaten usw.. In diesem Beitrag werden Entwurfsmuster vorgeschlagen, um das Lernen dieses Modells und insbesondere das Entwerfen von endlichen Automaten leicht oder leichter für Studierende zu machen. Diese Muster sind bereits in der Lehre im Einsatz. Eine erste Bewertung zeigt, dass sie für Studierende hilfreich sind.

1 Einleitung

Automaten sind uns in den verschiedensten Formen bekannt: als Geldautomaten, Zigarettenautomaten, Fahrkartenautomaten usw.. Dahinter verbergen sich als Modell endliche Automaten, wie sie in der theoretischen Informatik gelehrt werden. Viele Curricula für ein Informatikstudium sehen einen Grundkurs in den formalen Grundlagen der Informatik in den frühen Semestern vor, sehr oft sogar im ersten Semester, ehe Studierende solide Kenntnisse in der Programmierung oder in der Softwareentwicklung haben und ehe sie über Erfahrung mit Abstrahieren und Modellieren verfügen. Dies ist auch der Fall an der TFH Berlin im Studiengang Medieninformatik. Die Veranstaltung Formale Grundlagen der Informatik findet im ersten Semester statt. Im Modulhandbuch heißt es hierzu: „Ziel der Veranstaltung ist das Erlernen der Grundlagen von Automaten und formalen Sprachen als Werkzeug zur Modellierung und Transformation von Systemen und Prozessen.“ [HB08]. In diesem Beitrag werden Entwurfsmuster vorgeschlagen, um das Lernen dieses Modells und insbesondere das Entwerfen von endlichen Automaten leicht oder leichter für Studierende zu machen. Die Muster sind bereits in der Lehre im Einsatz. Eine erste Bewertung zeigt, dass sie für Studierende hilfreich sind.

Der Architekt Christopher Alexander hat den Begriff Muster (englisch *Pattern*) eingeführt: "Jedes Muster beschreibt ein in unserer Umwelt beständig wiederkehrendes Problem und erläutert den Kern der Lösung für dieses Problem, so dass Sie diese Lösung für dieses Problem beliebig oft anwenden können, ohne sie jemals ein zweites Mal

gleich auszuführen" (siehe [Al77] S. x).

Mit dem Buch "Design Patterns" [Ga95] haben Muster eine große Resonanz in der Softwareentwicklung gefunden. Inzwischen haben sich Muster in weitere Gebiete der Informatik verbreitet. Auch im Gebiet "Lehren und Lernen", insbesondere in Verbindung mit Informations- und Kommunikationstechnologien, sind Muster vorgeschlagen worden [E-l03].

Zwei Gründen sprechen dafür, Muster für das Lernen von endlichen Automaten zu entwickeln und in der Lehre einzusetzen. Erstens erweisen sich Muster als gute Werkzeuge, um Erfahrung und Fachwissen in einem Gebiet fest zu halten, wie in [DLM07] argumentiert wird. Zweitens macht ihre Durchsetzung in mehreren Bereichen der Informatik ihre Akzeptanz bei Studierenden einfach.

Dieser Beitrag präsentiert lediglich die zwei wichtigsten Muster für endliche Automaten. Die volle Liste ist in [Me07]. Das Muster *Überblick* schafft einen Überblick sowohl über alle vorhandenen Muster, als auch über das Komponieren von Automaten. Das Muster *3 Schritt-Methode* gibt ein Verfahren, um einen Automaten von Grund auf zu entwickeln. Diese Muster sind durch Bemerkungen von Studierenden und Kollegen verbessert worden. Eine übliche Etappe in der Entstehung von Mustern ist eine gründliche Diskussion im Kollegenkreis. Diese hat für die hier vorgeschlagenen Muster jedoch noch nicht statt gefunden. Ziel dieses Beitrages ist auch diese Etappe zu ergänzen.

Endliche Automaten werden im nächsten Abschnitt eingeführt. Abschnitt 3 ist den Mustern gewidmet. Der letzte Abschnitt präsentiert eine erste Bewertung des Einsatzes dieser Muster in der Lehre und schließt diesen Beitrag ab.

2 Endliche Automaten

Endliche Automaten [HM03, Si05] haben einen begrenzten Speicher in der Form einer endlichen Menge von Zuständen. Sie erlauben, eine bestimmte Kategorie von Problemen zu berechnen oder gleichbedeutend, die regulären Sprachen zu erkennen. Ein deterministischer endlicher Automat A wird wie folgt definiert.

Definition $A=(Q, \Sigma, \delta, q_{0,} F)$ wobei

- Q eine endliche Menge von Zuständen ist,
- Σ eine endliche Menge von Symbolen ist, auch Alphabet genannt,
- $\delta : Q \times \Sigma \rightarrow Q$ eine Transitionsfunktion oder Übergangsfunktion ist,
- q_0 der Startzustand oder Anfangszustand ist,
- F eine Menge von Endzuständen oder akzeptierenden Zuständen ist.

Endliche Automaten werden sehr oft in der Form von Zustandsdiagrammen graphisch dargestellt, wie in Abbildung 3 gezeigt. Zustände werden mit Kreisen dargestellt. Der Startzustand q_0 ist mit einem besonderen Pfeil gekennzeichnet und Endzustände haben einen Doppelkreis. Die Transitionsfunktion wird mittels Pfeilen zwischen Zuständen gegeben. Der zweite Parameter dieser Funktion, das Symbol des Alphabets, beschriftet den Pfeil.

Mit Σ kann man unendlich viele Wörter bilden, indem man Symbolen von Σ mehrfach nacheinander verkettet. Zum Beispiel kann man mit dem Alphabet `{0, 1}` die Wörter `0, 1, 00, 01, 10, 00, 000` usw. bilden. Das leere Wort, das kein Symbol beinhaltet, wird mit ε bezeichnet. `{0, 1}* = {ε, 0, 1, 00, 01, 10, 11, 000 ...}` ist die unendliche Menge von allen Wörtern, die man mit `{0, 1}` bilden kann.

Ein endlicher Automat akzeptiert ein Wort `w`, falls `w` angefangen vom Startzustand mit den Transitionen ganz gelesen wird und am Ende des Lesens ein Endzustand erreicht wird. Das Wort `101` wird vom Automaten im Abbildung 3 akzeptiert, wie die folgende Berechnungsschritte zeigen: $q_0 \xrightarrow{1} q_0 \xrightarrow{0} q_1 \xrightarrow{1} q_1$. Eine Sprache ist eine Menge von Wörtern. Zum Beispiel ist die Mengen `L1` der Wörtern, die man mit dem Alphabet `{0, 1}` bilden kann und die genau einmal das Symbol `0` enthalten, eine Sprache. Der Automat der Abbildung 3 akzeptiert diese Sprache.

Endliche Automaten beinhalten auch die nicht deterministische Automaten. Diese sind besonders wichtig für die Komposition von Automaten. Sie werden hier nicht weiter erläutert, weil sie in den Mustern von Abschnitt 3 nicht vorkommen. Endliche Automaten bilden den Einstieg in die Veranstaltung Formale Grundlage der Informatik. Die Studierenden treffen sie und verwandte Konzepte in weitere Veranstaltungen wie Softwareentwicklung, Algorithmen oder Compilerbau.

3 Muster

Muster werden durch verschiedene Elemente beschrieben. Die Muster für endliche Automaten haben in der Regel 4 Elemente: Zweck, Problem, Lösung und Beispiel. Manche Muster haben auch einen Diskussionsabschnitt. Wie für Muster üblich werden die Studierenden direkt angesprochen.

3.1 Muster Überblick

Zweck: DEA (*Deterministische Endliche Automaten*) entwickeln

Problem: Sie wollen einen endlichen Automaten entwerfen. Die Definition für die Sprache, die der Automat akzeptieren soll, ist beliebig. Sie kann einfach sein. Die Sprache "alle Wörter über dem Alphabet `{0, 1}`, die mit `0` anfangen" ist ein Beispiel für eine einfache Definition. Sie kann auch Wörter wie *oder, und, nicht, von rechts nach links* usw benutzen. Die Sprache "alle Wörter über dem Alphabet `{0, 1}`, die mit

0 anfangen und mit 0 enden" ist ein Beispiel für eine Definition mit *und*. Die Definition kann auch Operatoren benutzen wie Vereinigung, Durchschnitt, Konkatenation usw. Die Sprache {w in {0, 1}* | w beginnt mit 0} ∪ {w in {0, 1}* | w endet mit 0} ist ein Beispiel für eine Definition mit Vereinigung. Welche Schritte sollen Sie unternehmen, um zu einem (richtigen) Entwurf zu gelangen?

Lösung: Erstmal schauen Sie sich fertige Beispiele an. Sie können viele gute Ideen liefern. Falls Ihr gewünschter Automat nicht dabei ist, sollen Sie die Definition der Sprache sorgfältig lesen.

Falls die Definition einfach ist, dann versuchen Sie das Muster **3 Schritt-Methode**.

Falls die Definition *oder* enthält oder den Operator Vereinigung benutzt, trennen Sie die Sprache in zwei Teile. Entwerfen Sie einen DEA für jede Teilsprache, und komponieren Sie die zwei DEA wie im Muster **Oder** erklärt.

Falls die Definition *und* enthält oder den Operator Schnitt benutzt, trennen Sie die Sprache in zwei Teile. Entwerfen Sie einen DEA für jede Teilsprache, und komponieren Sie die zwei DEA wie im Muster **Und** erklärt.

Falls die Definition *nicht* enthält oder den Operator Komplement benutzt, betrachten Sie die positive Sprache. Entwerfen Sie einen DEA für sie. Mit dem Muster **Komplementbildung** erhalten Sie den gewünschten Automaten.

Falls die Definition *gefolgt von* enthält oder den Operator Konkatenation benutzt, trennen Sie die Sprache in zwei Teile. Entwerfen Sie einen DEA für jede Teilsprache, und komponieren Sie die zwei DEA wie im Muster **Konkatenation** erklärt.

Falls die Definition *von rechts nach links* enthält oder den Operator Spiegelung (englisch Reverse) benutzt, betrachten Sie die Sprache, wenn die Eingabe normal (von links nach rechts) gelesen wird. Entwerfen Sie einen DEA für sie. Mit dem Muster **Spiegelung** erhalten Sie den gewünschten Automaten.

3.2 Muster 3 Schritt-Methode

Zweck: DEA (Einen deterministischen endlichen Automaten) 'from Scratch' entwickeln

Problem: Sie wollen einen endlichen Automaten entwerfen. Die Definition der Sprache, die der Automat akzeptieren soll, ist einfach, etwa wie "alle Wörter, die mit 0 anfangen" oder "alle Wörter, die eine gerade Anzahl von 0 enthalten" (Sprachen über dem Alphabet {0, 1}). Welche Schritte sollen Sie unternehmen, um zu einem (richtigen) Entwurf zu gelangen?

Lösung: Erstmal schauen Sie sich fertige Beispiele an. Sie können viele gute Ideen geben. Falls Ihr gewünschter Automat nicht dabei ist, und Sie sich unsicher fühlen, wie Sie das ganze anpacken sollen, kann (hoffentlich) die 3 Schritt-Methode Ihnen weiter helfen.

Erster Schritt: Tests entwerfen. Versichern Sie sich, dass Sie die akzeptierte Sprache verstehen. Dafür sollen Sie Beispiele von Wörtern haben, die zur Sprache gehören und die zur Sprache nicht gehören. Diese Wörter werden Sie später benutzen, um Ihren Automaten-Entwurf zu prüfen. Also der erste Schritt lautet "Tests suchen", am besten systematisch nach der Länge des Wortes. Wir nehmen an, {0, 1} ist das Alphabet.

Wörter der Länge 0: gehört das leere Wort ε zur akzeptierten Sprache? *Wörter der Länge 1:* gehören die Wörter 0 und 1 zur akzeptierten Sprache? *Wörter der Länge 2:* gehören die Wörter 00, 01, 10 und 11 zur akzeptierten Sprache? *Wörter der Länge 3:* gehören die Wörter 000, 001, 010, 011, 100, 101, 110 und 111 zur akzeptierten Sprache? Usw. bis Sie eine klare Idee bekommen, welche Wörter die Sprache bilden.

So erhalten Sie am Ende zwei Listen nach Länge geordnet: die positive Liste, die akzeptierte Wörter enthält, und die negative Liste, die nicht akzeptierte Wörter enthält. Wir wenden diesen Schritt für die Sprache über dem Alphabet {0, 1} L1={ w | w enthält genau eine 0} an.

Gehört ε zur akzeptierten Sprache L1? Nein, weil ε leer ist, also enthält ε nicht 0. Gehört 0 zur akzeptierten Sprache? Ja. Gehört 1 zur akzeptierten Sprache? Nein. Gehört 00 zur akzeptierten Sprache? Nein, weil 00 zwei Nullen enthält usw.. Das Ergebnis dieses Schrittes ist wie folgt.

Akzeptierte Wörter: 0, 01, 10, 011, 101, 110, 0111, 1011, 1101, 1110, 01111 usw.,

Nicht akzeptierte Wörter: ε, 1, 00, 11, 000, 001, 010, 100, 111, 0000, usw..

In diesem Beispiel enthalten die zwei Listen unendlich viele Wörter, was sehr oft der Fall ist.

Zweiter Schritt: Automaten entwerfen. Ein Zustand in einem endlichen Automaten dient dem Zweck, etwas über das bis jetzt gelesene Wort zu speichern. So hat jeder Zustand einen Sinn, den wir durch einen Kommentar erläutern können. Diesen Kommentar nennen wir die Bedeutung oder Interpretation des Zustandes. Seine Funktion ist die gleiche wie Kommentare in Programmen: die Interpretation hilft einzusehen, dass der DEA tut, was er tun soll (und anderen verständlich zu machen, wie der DEA funktioniert). Betrachten Sie die Definition des Automaten gut und fragen Sie sich, was wichtig in der Definition ist und gespeichert sein soll. Dies erlaubt, sowohl Zustände als auch Transitionen zu konstruieren. Hier kann auch ein systematischer Aufbau helfen. Wie vorhin nehmen wir an, dass {0, 1} das Alphabet ist.

Wörter der Länge 0 - Erster Zustand. Ein Automat hat immer mindestens einen Zustand, der notwendigerweise Startzustand ist. Wir nennen ihn `q0`. Dieser Zustand entspricht dem Lesen des leeren Wortes ε. Stellen sie sich die Frage: was speichert `q0`, was ist die Bedeutung von ε für die Sprache? Dies gibt eine erste vorläufige Interpretation dieses Zustands. Falls ε akzeptiert wird, ist `q0` zugleich ein Endzustand.

Wir betrachten weiter die Sprache `L1`. Eine passende Interpretation für `q0` ist "Es ist noch kein Symbol gelesen worden, und somit auch keine `0`". Dieser Zustand ist kein Endzustand.

Wörter der Länge 1 - Zweiter (und dritter) Zustand. Außer den zwei trivialen DEA, die jeweils die leere Sprache und alle Wörter akzeptieren, haben DEA mehr als einen einzigen Zustand. Das Betrachten von immer längeren Wörtern erlaubt, weitere Zustände und Transitionen zu addieren.

Wir fangen mit dem Wort `0` an. Was ist seine Bedeutung für die Sprache? Können die Bedeutungen von ε und von `0` zusammengefasst werden? Wenn ja, brauchen Sie noch nicht einen zweiten Zustand zu addieren. Sie passen eventuell die Interpretation vom ersten Zustand an, und fügen eine Transition von `q0` zu sich selbst für das Eingabesymbol `0` hinzu. Wenn die Bedeutung von `0` ganz anders als die Bedeutung von ε ist, brauchen Sie einen zweiten Zustand. Dies ist besonders der Fall, wenn ε akzeptiert wird und `0` nicht akzeptiert wird, oder umgekehrt. Dieser Zustand wird hier `q1` benannt. Sie geben ihm eine Interpretation und addieren eine Transition von `q0` nach `q1` für das Eingabesymbol `0`. Etwas ähnliches gilt für ε und `1` und für `0` und `1`. Falls ε, `0` und `1` jeweils eine ganz andere Bedeutung haben, bekommen Sie drei Zustände. In allen Fällen haben Sie zwei Transitionen konstruiert, eine für jedes Eingabesymbol. Die vier Möglichkeiten, die sich im Allgemeinen nach dem Betrachten von Wörtern der Länge 1 über dem Alphabet `{0, 1}` ergeben können, sind in Abbildung 1 gezeigt. (Welche Möglichkeiten können sich ergeben nach dem Betrachten von Wörtern der Länge 1 über dem Alphabet `{a, b, c}`?)

Als Illustration betrachten wir weiter die Sprache `L1`. ε und `0` können nicht zusammengefasst werden, aus dem einfachen Grund, weil `0` akzeptiert wird, aber ε nicht. Wir führen einen zweiten Zustand `q1` ein. Eine hier passende Interpretation für `q1` ist "Die erste `0` ist gelesen worden". Dieser Zustand ist ein Endzustand, weil `0` akzeptiert wird.

Die Bedeutung von ε und `1` für `L1` ist gleich: "noch keine `0` gelesen worden". Wir adoptieren diese Interpretation für `q0` (Anpassung der bevorstehenden Interpretation). Wir erhalten somit den Anfang eines Automaten wie in Abbildung 2.

Wörter der Länge 2, 3, usw - Weitere Zustände. Um weitere Zustände und Transitionen sinnvoll einzuführen, müssen Sie sich fragen, welche Bedeutung längere Wörter wie `00`, `01`, `10` und `11`, `000` usw. für die Sprache haben, und inwiefern sie eine andere Bedeutung haben als die Wörter, die Sie schon betrachtet haben.

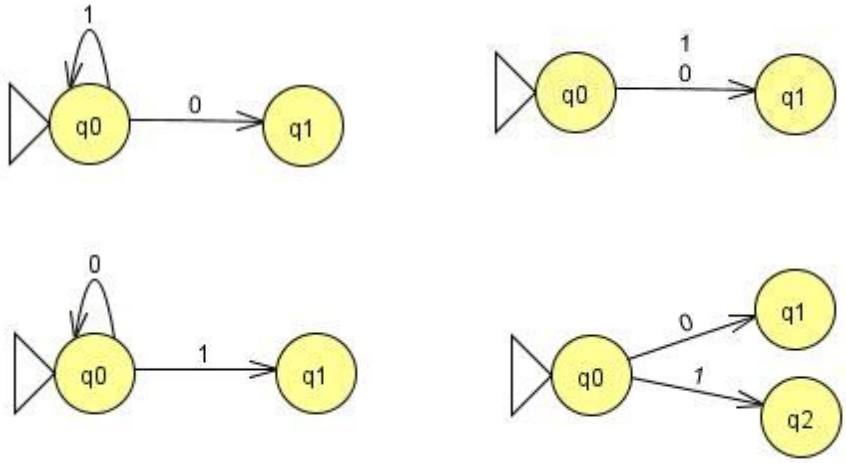


Abbildung 1: Die vier allgemeinen Möglichkeiten nach dem Betrachten von ε, 0 und 1.

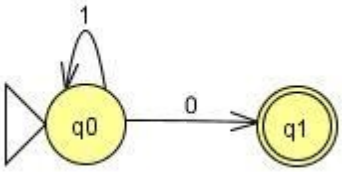


Abbildung 2: Anfang des Automaten für L1.

Wir fangen mit 00 an. Zunächst muss Ihnen klar sein, dass die zweite Null nach der ersten gelesen wird. Deshalb müssen Sie den Zustand betrachten, der nach dem Lesen der ersten Null erreicht wird. Dieser Zustand könnte q0 oder q1 sein (siehe Abbildung 1). Für die Sprache L1, die wir schon betrachtet haben, ist es z.B. q1. Um allgemein zu bleiben, nennen wir diesen Zustand hier p0 (diese Benennung erinnert daran, dass 0 schon gelesen wurde).

Sie führen mit ähnlichen Fragen fort, wie schon gehabt: Was ist die Bedeutung von 00? Können die Bedeutungen von ε und von 00 zusammengefasst werden? Wenn 'ja' passen Sie eventuell die Interpretation vom Startzustand und addieren Sie eine Transition von p0 zum Startzustand für das Eingabesymbol 0, falls diese noch nicht vorhanden ist. Wenn p0 der Startzustand ist, ist diese Transition schon vorhanden. Wenn 'nein' stellen Sie sich die Frage für 0 und 00, eventuell für 1 und 00 und verfahren Sie ähnlich. Falls 00 eine ganz andere Bedeutung als ε, 0 und 1 hat, führen Sie einen neuen Zustand ein, sagen wir p00, geben Sie ihm seine Interpretation und fügen Sie eine Transition von

p0 zu p00 für das Eingabesymbol 0 hinzu.

Verfahren Sie ähnlich mit 01. Vergessen Sie nicht nach ε, 0 und 1 auch die Bedeutung mit 00 (die jetzt schon betrachtet wurde) zu prüfen, ehe Sie einen weiteren Zustand einführen.

Benutzen Sie die gleiche Methode für immer längere Wörter, bis Sie keinen neuen Zustand einführen können.

Betrachten Sie L1 und den Anfang des Automaten weiter. Für die Sprache L1 hat 00 nicht die gleiche Bedeutung wie ε, 0 oder 1, weil Nullen gelesen worden sind, und zwar zwei, nicht genau eine, wie L1 definiert ist. Daher wird ein dritter Zustand addiert, den wir q2 nennen. Als Interpretation schreiben wir "zwei Nullen sind gelesen worden".

Dagegen hat 01 die gleiche Bedeutung wie 0: genau eine Null ist gelesen worden. Wir übernehmen diese Bedeutung als Interpretation für q1 und fügen eine Transition von q1 zu sich selbst für das Eingabesymbol 1 hinzu. 10 hat die gleiche Bedeutung wie 0. Das Wort 10 kann mit dem schon vorhandenen Automaten gelesen werden. Daher werden keine weitere Zustände oder Transitionen addiert. 11 hat die gleiche Bedeutung wie 1. Ebenfalls werden keine weiteren Transitionen oder Zustände hinzugefügt.

Sowohl das Wort 000 als auch das Wort 001 haben die gleiche Bedeutung wie 00, nämlich "es sind mehr als eine einzige 0 gelesen worden". Dies wird als Interpretation für q2 gewählt, und zwei Transitionen von q2 zu sich selbst (für 0 und 1) werden hinzugefügt. Andere Wörter der Länge 3 oder der Länge 4 verändern den Automaten nicht mehr. Die gewählten Interpretationen lassen erkennen, dass alle Fälle bezüglich der Sprache L1 gedeckt sind. Somit ist der Entwurf beendet, siehe Abbildung 3.

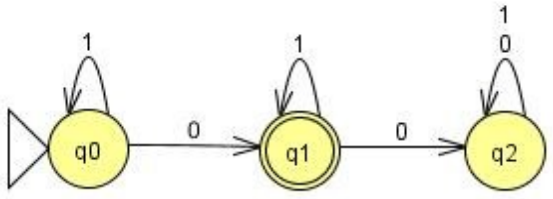


Abbildung 3: DEA für L1.

Interpretation der Zustände: q0 : es ist noch keine 0 gelesen worden. q1 : es ist genau eine 0 gelesen worden. q2 : es ist mehr als eine 0 gelesen worden.

Dritter Schritt: Tests prüfen. Prüfen Sie, ob Ihr Automat mit den Wörtern, die Sie im ersten Schritt gefunden haben, sich richtig verhält. Passen Sie dabei auf, dass Sie alle Zustände und alle Transitionen des Automaten durchlaufen. Falls es nicht der Fall ist, entwerfen Sie weitere Tests, die es ermöglichen.

Wenn Ihr Automat diese Prüfung besteht: Glückwunsch. Höchstwahrscheinlich haben Sie alles richtig gemacht.

Wenn ein Test nicht läuft, wie erwartet, prüfen Sie zwei Dinge: ist die Eingabe in der richtigen Liste? Es könnte sein, dass Sie die Eingabe als akzeptiert eingestuft haben, aber dass sie in Wirklichkeit nicht akzeptiert wird, oder umgekehrt. Dann prüfen Sie und passen Sie den Automaten selbst an, und testen Sie ihn wieder, bis Sie eine stabile Version erhalten und mit dem Ergebnis zufrieden sind.

Diskussion: Diese Methode sieht vor, dass Tests entworfen werden, bevor der Automat selbst entworfen wird. Dies ist ganz im Sinne der Softwareentwicklungsmethode "eXtreme Programming" (XP), die Ende der 90 Jahre vorgeschlagen wurde [Beck 00].

DEA besitzen eine bestimmte endliche Menge von Zuständen. Diese endliche Menge von Zuständen erlaubt eine Sprache zu akzeptieren. Die Sprachen, die DEA akzeptieren können, heißen regulär. Nicht alle Sprachen sind regulär. Mit anderen Wörtern können DEA nicht alle möglichen Sprachen akzeptieren. Es ist nicht immer offensichtlich, welche Art von Sprachen DEA akzeptieren können. Zum Beispiel kann die Sprache {w | w enthält so viele Teilwörter 01 wie Teilwörter 10} von einem DEA akzeptiert werden (wollen Sie versuchen, einen DEA dafür zu entwickeln?). Dagegen gibt es keinen DEA, der die Sprache L01={w | w enthält so viele 0 wie 1} akzeptieren kann. L01 ist ein Beispiel für eine Sprache, die nicht regulär ist, sondern kontextfrei.

Beispiele: Benutzen Sie dieses Muster, um DEA für die folgenden Sprachen über dem Alphabet {0, 1} zu entwickeln. Dann vergleichen Sie Ihre Lösung mit der, die hier vorgeschlagen ist.

L2 = { w | w beginnt mit 0}.

L3 = { w | w endet mit 0}.

L4 = { w | w hat eine gerade Anzahl von 0}.

L5 = { w | w ist ein Vielfaches von 3, wenn es als Binärdarstellung einer ganzen Zahl interpretiert wird}.

4 Erste Quantitative Bewertung und Schlußbemerkungen

Im Winter-Semester 2006-2007 und Sommer-Semester 2007 ist eine erste Version der Entwurfsmuster für endliche Automaten im virtuellen Lernraum als zusätzliches Lernmaterial für die Studierende zu Verfügung gestellt worden. Manche Studierende haben sich spontan geäußert und sie als hilfreich bezeichnet. Im Winter-Semester 2007-2008 erhielten die Entwurfsmuster ihre vorläufig endgültige Form. Ihr Einsatz wird im Unterricht erläutert. Die Studierende werden darauf hingewiesen, dass es die Ressource im virtuellen Lernraum gibt. Es ist Ihnen überlassen, ob sie davon Gebrauch machen oder nicht.

Eine erste Bewertung des Einflusses dieses Materials auf das Lernen der Studierenden findet gerade statt. Sie beruht auf der Beobachtung des Verhaltens der Studierenden im virtuellen Lernraum. Von den 81 angemeldeten Studierenden haben 36 das Material über Muster abgefragt. Von den 81 Studierenden haben 60 die Klausur über endliche Automaten geschrieben. Wie hängen Teilnahme an der Klausur und Abfragen der Entwurfsmuster zusammen? Die Zahlen zeigen eine leichte positive Korrelation:.Von den 60 Studierenden, die die Klausur geschrieben haben, haben 26 die Entwurfsmuster abgefragt. Wie hängen Ergebnisse der Klausur und Abfragen der Entwurfsmuster zusammen? Die Zahlen weisen auf eine positive Korrelation hin. Durchschnitt und Normalabweichung sind 36.627 und 10.855 im Allgemeinen, und 44.077 und 8.827 für diejenigen, die die Entwurfsmuster abgefragt haben. 6 Studierende haben 50 Punkte, das Maximum, erreicht, von denen 4 die Entwurfsmuster abgefragt haben.

Es stehen weitere Extra-Lernmaterialien im virtuellen Lernraum für die Studierende zur Verfügung. So weit zeigen die Zahlen für die anderen Lernmaterialien keinen so großen Einfluss auf die Klausurpunkte wie die Entwurfsmuster. Diese erste Bewertung sollte mit Vorsicht gelesen werden (die Abfrage einer Ressource heißt nicht, dass sie studiert wurde) und wird weiter geführt.

Neue Technologien wie virtuelle Lernräume erlauben, unterschiedliche zusätzliche Lernmaterialien bereit zu stellen. Gemäß ihrer Präferenzen können Studierende sie abfragen und benutzen. Dies ermöglicht eine gewisse Anpassung und Personalisierung des Unterrichts.

Literaturverzeichnis

[Al77] Alexander, C. et all: A pattern language: towns, buildings, construction. New York: Oxford University Press. 1977.

[Be99] Beck, K.: Extreme Programming explained: embrace changes, Addison-Wesley Professional; second edition with Cynthia Andres 2004.

[Gam95] Gamma, Erich et all.: Design Patterns, Addison-Wesley Publishing Company, 1995.

[DLM07] Delozanne, E.; Le Calvez, F.; Merceron, A.; Labat, J-M. : A Structured Set of Design Patterns for Learners' Assessment, Journal of Interactive Learning Research,18(2), 2007; S. 309-333.

[E-l03] The E-LEN patterns site http://www2.tisip.no/E-LEN/patterns_info.php, letzer Zugriff 04.01.2008.

[HB08] Modulhandbuch für den Bachelor-Studiengang Medieninformatik, http://www.tfh-berlin.de/.modulhandbuch/mhb/FB6/MHB_Ba-Medieninformatik_V2.pdf, letzter Zugriff 04.01.2008

[HM03] Hopcroft, J.E.; Motwani R.; Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation.. Addison Wesley, New Jersey, 2003

[Me07] Merceron, A.: Entwurfsmuster für DFAs, Report, TFH-Berlin, 2007. http://www.tfh-berlin.de/~merceron/publi.html letzter Zugriff 04.01.2008

[Si05] Sipser, M.: Introduction to the Theory of Computation.. Course Technology, Boston, 2005

Modellieren lernen über Formalisieren von Ablaufbeschreibungen

Jörg Desel

Angewandte Informatik
Katholische Universität Eichstätt-Ingolstadt
85071 Eichstätt
joerg.desel@ku-eichstaett.de

Zusammenfassung: Es wird eine Projektidee zur Vermittlung von Kompetenz zur Konstruktion dynamischer Modelle bzw. Algorithmen vorgestellt, deren Kern die Vermittlung und das Verständnis von Abläufen als formale Konzepte vor der Generierung des Modells ist.

1 Einleitung

Sowohl im Bereich der Schule als auch in der Lehre an Hochschulen werden Modelle dynamischer Systeme (Automaten, Petrinetze, statecharts, activity diagrams, Prozess-Algebren, …) meist als syntaktische Konstrukte eingeführt, deren Semantik bzw. deren Verhalten durch ihre Abläufe gegeben ist, also durch die konkreten Schritte bei der Ausführung eines Modells auf Instanzebene und ihre Abfolge. Abläufe werden in diesem Sinn als abgeleitete Konstrukte eingeführt. Sie werden stets als Abläufe des Modells verstanden (eine Ausnahme sind formale Sprachen, die von Automaten erkannt oder erzeugt werden; aber dort steht meist eine Grammatik als erzeugendes Modell im Vordergrund).

Diese Reihenfolge – erst das Modell, dann sein Verhalten – wird sowohl bei der Vermittlung von Modellsprachen eingehalten, als auch bei der Vermittlung konkreter Modelle, z.B. für spezielle Referenzgeschäftsprozesse, (verteilte) Algorithmen, usw. Es wird damit den Lernenden nahe gelegt, auch bei der Konstruktion von Modellen in dieser Reihenfolge zu verfahren: Ohne systematische Unterstützung ist ein Modell zu konstruieren, dessen Abläufe dann aber systematisch und evtl. sogar Werkzeug-unterstützt konstruiert werden können und mit dem gemeinten Verhalten verglichen werden.

Bei dieser Vorgehensweise steht das folgende Paradigma Pate: Ein Modell ist Abstraktion eines realen (oder gedachten) Systems; die Abläufe des Modells entsprechen den Abläufen des Systems. Um das System zu verstehen, kann man ein Modell und anschließend Abläufe des Modells erzeugen, analysieren, durch strukturelle Analyse des Modells Eigenschaften der Abläufe des Systems (und damit dynamische Eigenschaften des Systems selbst) ermitteln oder beweisen usw.

Ganz analog werden Programmiersprachen oder, allgemeiner, formale Modelle von Algorithmen, zunächst syntaktisch eingeführt und anschließend ihre Abläufe entweder nur informell betrachtet oder, selten, als abgeleitete formale Konstrukte eingeführt. Das-

selbe gilt für konkrete Programme. Ihre syntaktische Darstellung wird zunächst präsentiert, oftmals durch Hierarchie- oder Modularisierungskonzepte unterstützt. Ihr Verhalten, also ihre Abläufe, werden erst danach betrachtet.

Während bei Systemen bzw. Modellen geringerer Komplexität diese Denkweise zum Systemverständnis ausreicht, ist sie nach meiner Auffassung bei komplexeren Systemen, wie sie in der Informatik praxisrelevant sind, untauglich. So versteht man ein komplexes Systemmodell ohne Betrachtung seiner Abläufe gar nicht und nach Konstruktion von Beispielabläufen nur unzureichend. Noch schwerer wiegt die Kritik, dass diese Denkweise die Konstruktion von Systemen oder von Systemmodellen sehr unzureichend unterstützt. So haben viele Lernende zwar die Fähigkeit, Elemente von Modellierungssprachen wie auch konkrete Modelle zu verstehen, ihnen fehlt aber die Kompetenz, selbst Modelle zu erzeugen, und sie trauen sich dies auch nicht zu. Diese Beobachtung wird nach meiner Erfahrung von vielen Lehrenden geteilt. Eindrucksvoll wird sie in [SK07] beschrieben (allerdings deutlicher im Vortrag als im Beitrag). Auch in [BP06] wird explizit zwischen den Kompetenzen Modellverstehen und Modellerstellen unterschieden, und es wird vorgeschlagen, diese Kompetenzen getrennt zu vermitteln.

Die industrielle Praxis weist einen alternativen Weg bei der Systemkonstruktion: Algorithmische Ideen werden oft mit Hilfe von Beispielszenarien entworfen und kommuniziert. Diese Szenarien stellen die Spezifikation eines Softwaresystems dar. Szenarien sind zwar meist hinreichend präzise formuliert (wenn man genügend implizite Annahmen hinzufügt), aber selten formal. Auch wenn eine formale Syntax verwendet wird, lassen sie Interpretationsspielraum. Oft wird natürliche Sprache verwendet, manchmal eine semi-formale Notation. Als zweiter, kreativer Schritt wird auf Grundlage der Szenarien unmittelbar ein Modell (oder Software) konstruiert, dessen Abläufe den Szenarien in einem vagen Sinn entsprechen. Dies entspricht der Vorgehensweise von Ingenieuren; so überlegt sich ein Maschinenbauer oder ein Verfahrenstechniker zunächst die Funktionsweise der relevanten Prozesse und konstruiert erst dann – über ein Modell – eine die Prozesse unterstützende Anlage. Umgekehrt wird man auch eine komplexe Maschine nie verstehen oder erklären können, wenn man zunächst die Funktionsweise aller Bauteile betrachtet, anschließend ihr Zusammenspiel und schließlich erst die Funktionsweise der gesamten Maschine in Form ihrer Abläufe. Allerdings ist der Schritt von den Prozessen bzw. von den Szenarien zum Modell bei komplexen Informatiksystemen oftmals ungleich schwieriger. Bei einfachen Informatiksystemen kann er systematisch, aber von Hand bewältigt werden, bei komplexen Informatiksystemen müssen automatische oder semi-automatische Syntheseverfahren zum Einsatz kommen. Voraussetzung dafür ist, dass die Abläufe hinreichend formal beschrieben sind.

In diesem Beitrag soll ein Projektvorhaben skizziert werden, das in der Lehre den folgenden Ansatz verfolgt: Bei der Konstruktion eines Modells wird zunächst das Modellverhalten in Form semi-formaler Abläufe erzeugt. Diese Abläufe werden formalisiert, so dass ihre Beschreibung in der Sprache der Modellsemantik vorliegt. Dies erlaubt, Abläufe des späteren Modells, die ja ebenfalls in der Sprache der Modellsemantik formuliert sind, unmittelbar mit den spezifizierten Abläufen zu vergleichen. Daran anschließend erfolgt die Konstruktion des Modells. Diese kann durch Syntheseverfahren aus den formalen Abläufen unterstützt werden.

Das vorgeschlagene Vorgehen hat das Ziel, das Erlernen von Modellierungssprachen

und von konkreten Modellen zu erleichtern, insbesondere aber die Modellierungskompetenz der Lernenden zu steigern. Die Kompetenzen, Modelle zu verstehen und sie zu erzeugen, sollen nicht nacheinander, sondern gemeinsam erlernt werden. Der Ansatz konzentriert sich aber nicht nur auf Lernende, die bislang besondere Schwierigkeiten bei der Modellierung hatten. Ähnlich wie bei der Programmierung gibt es Lernende, die bei der Modellierung im Kleinen sehr erfolgreich sind. Ihnen fehlen aber Konzepte, diese Fähigkeit auf komplexere Systeme zu übertragen. Und insbesondere fehlt ihnen die Einsicht, dass die von ihnen beherrschten ad hoc Verfahren nicht auf komplexe Systeme übertragbar sind. Auch diese Lernergruppe profitiert davon, wenn von Beginn an skalierbare Methoden in der Lehre eingesetzt werden.

Im Bereich des Requirement Engineering gibt es seit vielen Jahren Ansätze, Anforderungsmodelle aus Ablaufbeschreibungen zu erzeugen. Ausgehend von use cases wird in mehreren Schritten ein Modell erzeugt, das die Anforderungen aus Nutzersicht darstellt. Diese Ansätze weisen viele Parallelen zu dem hier vorgeschlagenen Ansatz auf, sie unterscheiden sich aber in einigen wesentlichen Aspekten. So geht es im hier dargestellten Ansatz nicht nur um das Schnittstellenverhalten eines Modells, sondern insbesondere um den algorithmischen Kern, der dieses Verhalten bewirkt. Abläufe sind hier konkrete Instanzen, während use cases oft Alternativen oder sogar Schleifen enthalten und somit auf der Ebene von Algorithmen sind und selbst Abläufe besitzen. Schließlich geht es hier um Abläufe, die formal und in der Sprache der Semantik von Modellen formuliert sind. Typisch für use cases ist dagegen eine Darstellung auf informeller oder semi-formaler Ebene. Allerdings gibt es auch für use cases Ansätze, diese schrittweise zu formalisieren.

Eine weitere Parallele findet sich zu dem Konzept des „Programming by Example“, wie es z.B. in [Lie01] vertreten wird. Dabei geht es zwar auch darum, Programme durch Angabe ihrer Abläufe automatisch zu erstellen oder zu modifizieren. Akteur ist hier aber nicht der Lernende und auch nicht der Modellierer oder der Programmierer beim Software-Entwurf, sondern der Anwender der Software. Die für „Programming by Example“ im Wesentlichen im Bereich der Künstlichen Intelligenz entwickelten Verfahren können aber bei Syntheseverfahren ggfs. wieder verwendet werden.

Im folgenden Abschnitt wird das Konzept genauer beschrieben. Der dritte Abschnitt geht auf Vorarbeiten ein. Zusammenfassung und offene Fragen liefert der vierte Abschnitt.

2 Abläufe zuerst, bei der Systementwicklung

In Abbildung 1 ist der in der Einleitung dargestellte, traditionelle Ansatz der Rolle von Modellen bei der Systementwicklung skizziert, wie er in der Ausbildung wenigstens implizit gelehrt wird, und wie er wohl auch oft in der Praxis zum Einsatz kommt.

Ausgehend von einer Vorstellung der Abläufe des betrachteten (oder zu konstruierenden) Systems wird ein Modell konstruiert. Dieses Modell hat selbst Abläufe in einem formalen Sinn, die per Hand oder automatisch generiert werden können. Diese Abläufe werden mit den (gedachten) Abläufen des Systems verglichen. Ggfs. wird das Modell verändert, bis sein Verhalten dem gewünschten Verhalten entspricht. Schließlich kann das Modell als Grundlage für die Systemkonstruktion dienen (falls es nicht nur abbildenden Charakter hat und z.B. für Analysezwecke konstruiert wurde).

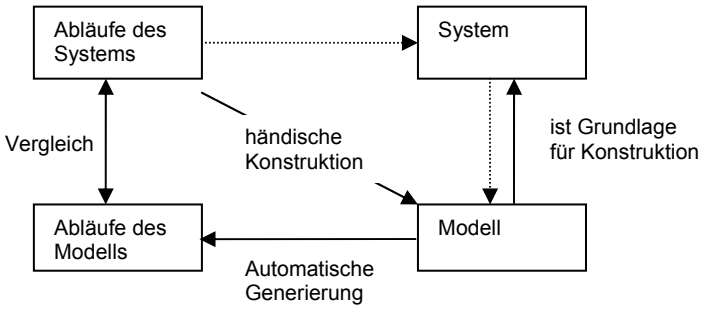


Abbildung 1: Traditioneller Ansatz

Problematisch bei diesem Vorgehen ist der erste Schritt. Faktisch muss der Modellierer zwei Abstraktionen auf einmal durchführen: Er muss aus den Abläufen gedanklich ein System konstruieren, das diese Abläufe unterstützt, und er muss aus dem gedachten System ein Modell abstrahieren (gepunktete Pfeile). Bei einfachen Systemen und bei geübten Modellierern mag dieser Doppelschritt unproblematisch sein, in der Ausbildung scheitern gerade an diesem Schritt viele Schüler und Studenten. Ähnlich ist die Situation beim Programmieren: Anfängern fällt es schwer, einen Algorithmus in einer formalen Sprache zu entwerfen, auch wenn sie die Abläufe dieses Algorithmus verstanden haben und formal beschreiben könnten.

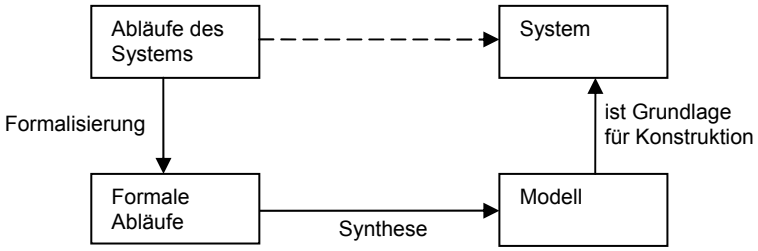


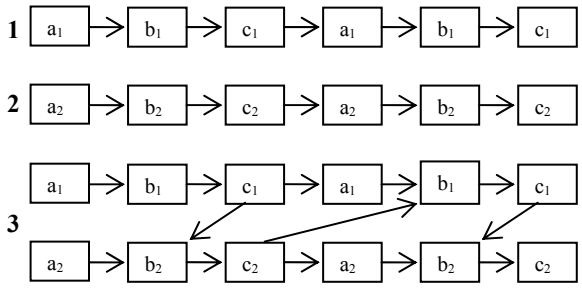
Abbildung 2: Der hier vorgeschlagene Ansatz

Abbildung 2 zeigt ein entsprechendes Bild unseres Ansatzes. Grob gesagt, geht das Spiel nun nicht mehr im Uhrzeigersinn, sondern andersherum. Die (gedachten) Abläufe des Systems werden zunächst formalisiert, so dass sie die formalen Abläufe eines Modells sein können. Es wird also z.B. eine formale Sprache entwickelt. Dies ist nicht so einfach wie es scheinen mag, denn die einzelnen Aktivitäten (das Alphabet der Sprache) müssen identifiziert und voneinander abgegrenzt werden. Elemente dieser Sprachen sind Instanzen von Aktivitäten, wie sie im Modell vorkommen, also z.B. Schaltvorgänge von Transitionen eines Petrinetzes, konkrete Zuweisungen an Variable eines Programms usw. Im traditionellen Ansatz allerdings ist dies im ersten Schritt neben vielen anderen Aspekten ebenfalls zu leisten. Im zweiten Schritt des neuen Ansatzes wird aus den formalen Abläufen das Modell generiert. Dieser kreative Schritt ist schwierig genug, um ihn nicht zusätzlich mit anderen Aufgaben zu belasten. In vielen Formalismen stehen für diesen Schritt Syntheseverfahren zur Verfügung, die wenigstens assistierend eingesetzt werden können. Bei komplexeren Systemen ist es nahe liegend, sowohl händisch als auch auto-

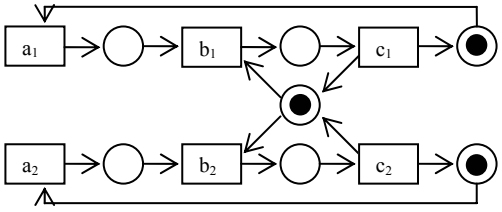
matisch generierte Modelle durch Konstruktion ihrer Abläufe und Vergleich dieser mit den „Formalen Abläufen“ zu validieren. Schließlich wird in einem letzten Schritt das System aus dem Modell (und weiteren Artefakten) generiert. Die drei Schritte realisieren das Gesamtziel, nämlich die Entwicklung eines Systems aus den Vorstellungen der Entwickler, d.h. aus den gedachten Abläufen (gestrichelter Pfeil).

Zur Illustration des Ansatzes sollen zwei Beispiele dienen. Diese sind naturgemäß zu klein, um die erhofften Vorteile des Ansatzes bei komplexen Modellen bzw. Algorithmen deutlich zu machen. Sie sollen nur dem Verständnis des Grundprinzips dienen.

1. Ein Petrinetzmodell zur Realisierung des wechselseitigen Ausschlusses zweier Teilprozesse soll erstellt werden. Diese Teilprozesse durchlaufen zyklisch die Aktionen a_1 ,b_1, c_1 bzw. a_2, b_2, c_2, wobei b_i jeweils den Beginn des kritischen Abschnitts und c_i jeweils dessen Ende bedeutet. Die folgenden halbgeordneten Abläufe beschreiben mögliches Verhalten. Im ersten Ablauf ist nur der erste Teilprozess aktiv, im zweiten nur der zweite, im dritten wird der kritische Abschnitt abwechselnd betreten.



Ein synthetisiertes Petrinetz könnte wie folgt aussehen:



2. Der Euklidische Algorithmus zur Bestimmung des größten gemeinsamen Teilers zweier natürlicher Zahlen wird z.B. durch folgende Abläufe beschrieben:

Eingabe n = 6, m = 9	Eingabe = 5, m = 3
m > n	m < n
m := m – n = 9 – 6 = 3	n := n – m = 5 – 3 = 2
m < n	m > n
n := n – m = 6 – 3 = 3	m := m – n = 3 – 2 = 1
m = n	m < n
Ausgabe n = 3	n := n – m = 2 – 1 = 1
	m = n
	Ausgabe n = 1

> Den dazu gehörenden Algorithmus möge der Leser aus den Abläufen selbst synthetisieren. Wichtig ist die Konstruktion der (hier sequentiellen) Kontrollstruktur erst nach der präzisen Formulierung von Abläufen mit konkreten Daten

Ausgehend vom ersten Beispiel soll nun präzisiert werden, was genau Lernende in dem vorgestellten Ansatz leisten sollen und wie sie durch Werkzeuge unterstützt werden können. Die tatsächlich verwendete Sprache soll nicht ausdrücklich beworben werden, analog kann man auch alternative Sprachen wählen.

Formalisierung von Ablaufbeschreibungen

Ein Ablauf ist hier definiert als (einmalige) Instanz, seine Elemente sind also (einmalige) Aktivitätsinstanzen (die im Petrinetz-Jargon Ereignisse genannt werden) und Ordnungsbeziehungen zwischen ihnen. Um es ganz deutlich zu machen: "Martha Krüger schreibt sich am 12.5.2007 um 14.33 in der Universität Bern ein" ist ein Beispiel einer Aktivitätsinstanz, während "Student(in) schreibt sich an Universität ein" eine Beschreibung derartiger Instanzen auf Typebene ist, also eine Aktivität. Diese beiden Ebenen werden oftmals verwechselt, wenn nämlich mehrere Instanzen derartig ähnlich sind, dass sie ohne eigentlichen Informationsverlust zusammengefasst werden können. Auf Typebene sind aber auch Verzweigungen möglich, unter denen auf Instanzebene ausgewählt wird. Bei einer Spezifikation durch Abläufe wird man sich allerdings nicht tatsächlich auf Instanzebene bewegen, sondern eine Klasse gleichartiger Instanzen betrachten, die sich auch im Detail nicht unterscheiden. Wichtig ist die klare Unterscheidung dieser Konzepte in der Lehre, die allerdings bislang in der Literatur selten sauber vorgenommen wird.

Bei der Formalisierung eines Ablaufs müssen Aktivitätsinstanzen identifiziert werden und eine Ordnung zwischen ihnen festgehalten werden (was muss zuvor geschehen sein?). Jeder Instanz wird eine Aktivität zugeordnet, wobei mehrere Instanzen in einem Ablauf derselben Aktivität zugeordnet werden können (wenn diese Aktivität in dem Ablauf mehrfach ausgeführt wird). Formal erhält man so eine beschriftete Halbordnung (labelled partial order, LPO). Die Elemente dieser LPO sind die Aktivitätsinstanzen, die Ordnung ist die transitive Hülle der Kausalitätsbeziehung und die Beschriftung gibt die Zuordnung zu den Aktivitäten an.

Generierung eines Modells

… aus einer Menge von LPOs. Diese Modelle müssen eine zu den LPOs kompatible Semantik haben, so dass man (leicht) entscheiden kann, ob eine gegebene LPO ein Verhalten des Modells darstellt. In unserem Ansatz sind die Modelle Petrinetze (genauer, Stellen/Transitions-Petrinetze), deren Transitionen entweder stille Transitionen sind, deren Verhalten nicht beobachtet werden kann, oder aber Aktivitäten in dem oben genannten Sinn. Stille Transitionen oder vergleichbare Konstruktionen anderer Sprachen können die Modellierung des Kontrollflusses vereinfachen, müssen aber in der Lehre anfangs nicht thematisiert werden. Das Verhalten eines Petrinetzes lässt sich als Menge von LPOs darstellen. Hierzu gibt es mehrere, äquivalente Ansätze. Zum Beispiel kann man, ausgehend von Schaltfolgen als speziellen voll geordneten LPOs, Ordnungsbeziehungen zwischen Aktivitätsinstanzen wegnehmen, wenn die entsprechenden Transitionen nebenläufig schalten.

Syntheseverfahren erzeugen aus einer Menge von LPOs ein Petrinetz derart, dass einerseits diese LPOs zum Verhalten gehören und andererseits kein Petrinetz mit derselben Eigenschaft weniger Abläufe besitzt. Da diese exakten Syntheseverfahren allerdings oft zu recht komplexen Petrinetzen führen, gibt es andere, effizientere heuristische Syntheseverfahren, deren Ziel ein möglichst einfaches Petrinetz ist, das ebenfalls das geforderte Verhalten aufweist und möglichst wenig zusätzliches Verhalten hat – hier werden also zwei Ziele zugleich verfolgt und ein Trade-off ist notwendig. Oftmals sind diese heuristischen Verfahren den exakten Verfahren überlegen, weil die zusätzlichen Abläufe des generierten Modells durchaus gewollte Abläufe sein können, die nur in der Spezifikation nicht bedacht bzw. nicht aufgeführt wurden.

Iteration

Anschließend ist das Modell schrittweise zu verändern, entweder durch direkte Modifikationen des Modells selbst oder durch Variation der spezifizierten Abläufe, so dass die Synthese ein verändertes Modell ergibt. Auch dieser Schritt wird algorithmisch unterstützt: dem Lernenden werden neben einer Visualisierung des Modells mögliche und nicht mögliche Abläufe dargestellt, so dass er entscheiden kann, ob er dieses Modellverhalten mit seiner Spezifikation gemeint hatte, ob er also die zusätzlichen Abläufe in seine Spezifikation aufnehmen könnte

3 Vorarbeiten und Ausblick

Meines Wissens gibt es keine einschlägigen Forschungsarbeiten zu der Fragestellung, nach welchem didaktischen Konzept dynamische Modelle oder auch Algorithmen in der Lehre vermittelt werden sollten, so dass mit dem Verständnis zugleich auch Modellierungskompetenz erworben wird und so dass die im Kleinen erlernte Vorgehensweise auch für komplexe Modelle anwendbar ist. Insbesondere sind mir keine Ergebnisse zu der Fragestellung bekannt, ob die primäre Betonung der Ablaufbeschreibung noch vor der Modellbetrachtung bzw. der Modellkonstruktion zu besseren Lernerfolgen führt als die klassische Vorgehensweise.

In verschiedenen Arbeiten, z.B. [WB07], wird die Entwicklung von Algorithmen aus didaktischer Perspektive beleuchtet. In der genannten Publikation geht es allerdings darum, mit welchem Anwendungsbezug – hier Roboter – die Algorithmenkonstruktion vermittelt werden kann. Die Autoren schreiben in der Einleitung zwar explizit, dass mit einfachen Abläufen begonnen wird und schließlich die Implementierung eines Programms durchgeführt wird. Jedoch wird unter dem Ablaufbegriff offenbar etwas anderes verstanden als in der vorliegenden Arbeit; der Übergang zwischen einzelnen Abläufen und der Modellierung von Abläufen mit Kontrollstruktur – also von Algorithmen – ist fließend.

Eine erste Vorarbeit aus der Arbeitsgruppe des Autors liefert Christian Neumair (ein Doktorand und Informatiklehrer) in [Neu06]. In dieser Arbeit berichtet er über eine durchgeführte Vergleichsstudie: In einer Schulklasse hat er Algorithmen auf herkömmliche Art vermittelt, in einer weiteren Klasse mit dem in diesem Beitrag beschriebenen Ansatz. Die Evaluation der Ergebnisse zeigt zwar kein eindeutiges Ergebnis. Sie macht aber durchaus deutlich, dass der Ansatz viel versprechende Perspektiven hat.

Weitere Vorarbeiten beziehen sich weniger auf die Lehre, aber grundsätzlich auf Systementwicklungsverfahren, die mit einer formalen auf Abläufen basierenden Anforderungsbeschreibung beginnen. Am konsequentesten wurde dies in [HM03] umgesetzt. Die Grundidee dabei ist, formale Abläufe in (eigens dafür erfundenen) live sequence charts zu formulieren und diese durch eine sog. Play engine zu interpretieren, also gar kein System mehr explizit zu konstruieren. Auch in [SM06] und [GS07] wird betont, dass die Anforderungsanalyse mit Abläufen (dort Szenarien genannt) beginnen sollte und erläutert, wie die Umsetzung in die Modellsprache der state charts realisiert werden kann. Die Konstruktion von Modellen ausgehend von Instanzen wird seit Jahrzehnten auch für Datenmodelle vorgeschlagen. Die Einbeziehung von Verhaltensmodellen findet sich z.B. in [MK02].

Eigene Vorarbeiten beziehen sich auf die Modellkonstruktion unter Verwendung der Sprache von Petrinetzen, insbesondere für halbgeordneten Ablaufbeschreibungen. Zunächst lag der Schwerpunkt auf der Validierung von Modellen, durch Konstruktion von halbgeordneten Ablaufbeschreibungen, sowohl in der Lehre [Des00] als auch werkzeugunterstützt im praktischen Einsatz [Des02], [DJ03a]. Der Einsatz dieses Ansatzes in einem Industrieprojekt [DJ03] machte deutlich, dass die Spezifikation tatsächlich mit Abläufen beginnen muss und dass halbgeordnete Abläufe in der Praxis den Beginn der Anforderungsanalyse darstellen. Das heißt, es geht darum, den früheren Weg (vom Modell zu Ablaufbeschreibungen) umzudrehen (Erzeugung eines Modells aus Ablaufbeschreibungen). Dieser Aspekt wird in [Des08] genauer dargestellt, wobei dort insbesondere auf Aktionsverfeinerungen in Ablaufbeschreibungen und in Systemmodellen Bezug genommen wird.

Wie kommt man nun von Ablaufbeschreibungen zu Systemmodellen? Dieses Thema wurde seit vielen Jahren im Rahmen von Modellsynthese untersucht, für Petrinetze siehe z.B. [DR96]. Die Synthese von Systemmodellen aus halbgeordneten Ablaufbeschreibungen ist mithilfe sog. Faltungen der Ablaufbeschreibungen möglich. Jüngere Arbeiten (z.B. [LB07]) geben an, wie die Synthese von Modellen allgemeiner und systematischer aus halbgeordneten Ablaufbeschreibungen geschehen kann.

Eine ähnliche Fragestellung wird im Zusammenhang mit process mining untersucht. Hier geht es eigentlich darum, aus einer großen Menge (meist sequentieller) Abläufe, die in existierenden Systemen protokolliert werden, ein Modell des Systemverhaltens zu rekonstruieren. Grundsätzlich allerdings bedeutet dies auch die Konstruktion eines Modells aus Verhaltensbeschreibungen. In [BD07] haben wir dargestellt, wie das o.g. Syntheseverfahren in abgewandelter Form auch hier eingesetzt werden kann.

Werkzeuge wie unser VIPtool [DJ03a] sind in der Lehre einsetzbar, wenn es darum geht, Hilfestellungen beim Übergang von formalen Ablaufbeschreibungen zu einem Systemmodell zu geben. Die Synthese liefert hier ein möglichst einfaches Systemmodell, das das eingegebenen Verhalten aufweist, eventuell aber zusätzlich Abläufe besitzt (z.B. wenn kein Modell ausschließlich das vorgegebene Verhalten hat). Durch Erzeugung weiterer Abläufe durch die bereits existierenden Werkzeuge kann sich der Lernende schrittweise zu dem gewünschten Modell vorarbeiten. Dabei lernt er Freiheitsgrade bei der Modellkonstruktion kennen, die ihm vorher nicht bewusst waren und übt eine strenge Formalisierung seiner Verhaltensbeschreibungen. Ohne diese Formalisierung wäre auch im alten Ansatz eine Validierung des Modells nicht tatsächlich möglich (womit

sollte das Verhalten des Modells verglichen werden?), und der Lernende würde Schwächen ihrer Modellierung noch nicht einmal bemerken können.

4 Zusammenfassung und offene Fragen

Im Bereich der Didaktik haben wir Thesen entwickelt, die sich zwar auf Vorarbeiten stützen, aber verifiziert werden müssen. Insbesondere ist zu untersuchen, inwieweit der Ansatz tatsächlich schwächeren Lernenden hilft, die für sie unüberwindliche Hürde zur selbstständigen Konstruktion von Modellen zu überwinden. Dies wird sicherlich auch von der Qualität der Werkzeugunterstützung abhängen. Geeignete Referenzfallstudien sind zu entwickeln. Insbesondere in der Schule, aber auch im Nebenfachunterricht an der Hochschule sind entsprechende Vergleichsstudien durchzuführen.

Eine andere im Beitrag genannte Zielgruppe sind (Hauptfach)-Studierende, denen ein schrittweises und werkzeugunterstütztes Vorgehen beim Modell- und Systementwurf deshalb fremd ist, weil sie meist eine Lösung ohne Unterstützung aufschreiben können. Der Einsatz unseres Ansatzes in der Lehre für diese Zielgruppe ist nur sinnvoll, wenn der Ansatz auch in der Praxis an größeren Projekten eingesetzt worden ist.

Viele Detailfragestellungen in diesem Ansatz sind noch offen. Einige Beispiele:

- Wie können Aktivitäten identifiziert werden? Insbesondere beim Formalisieren von Szenarios durch mehrere Personen ist die Granularität einzelner Aktivitäten uneinheitlich und Aktivitäten können sich überschneiden. Zu diesem Problem sollte zunächst eine präzise Fachsprache festgelegt werden. Bewährte Konzepte aus der Datenmodellierung können hierzu übernommen werden. Im Bereich der Geschäftsprozessmodellierung liefert [BD08] Vorüberlegungen.
- Wie können formalisierte Szenarien entwickelt werden, wenn das Wissen über die Abläufe verteilt vorliegt, also niemad den gesamten Ablauf kennt? Hier helfen spezifische Kompositionalitätseigenschaften der Halbordnungssemantik. Erste Ansätze sind in [Des08] dargestellt.
- Nach welchem Prinzip sollen Beispielabläufe konstruiert und dem Benutzer vorgestellt werden?
- Um im Falle von Verzweigungen im Modell den Grund einer Auswahl mit anzugeben (Condition im IF-THEN-ELSE Befehl) mag es sinnvoll sein, in den Ablaufbeschreibungen bei gewissen Ereignissen anzugeben, warum dieses Ereignis stattfindet (siehe unser zweites Beispiel: Euklidischer Algorithmus). Wie lässt sich diese Zusatzinformation in den Syntheseverfahren berücksichtigen?

Literaturverzeichnis

[BD07] Bergenthum, R.; Desel, J., Lorenz, R., Mauser, S.: Process mining based on regions of languages. Business Process Managenent (BPM 2007), LNCS 4714, S.375-383, Springer (2007)

[BD08] Bergenthum, R.; Desel, J., Mauser, S.: Synthesis of Petri Nets for Business Process Design. Verhaltensmodellierung: Best Practices und neue Erkenntnisse. Workshop auf der Modellierung 2008, Berlin, März 2008

[BP06] Borner, N., Paech. B., Rückert, J.: Vom Modellverstehen zum Modellerstellen. Modellierung in Lehre und Weiterbildung, Workshop auf der Modellierung 2006, Innsbruck, März 2006, ifi 2006-03, Institut für Informatik, Universität Zürich, S. 7-15 (2006)

[Des00] Desel; J: Teaching system modeling, simulation and validation. 2000 Winter Simulation Conference (WSC'00), Orlando, Dezember 2099, S.1669-1675 (2000)

[Des02] Desel; J: Model validation – a theoretical issue? International Conference on Applications and Theory of Petri Nets 2002, LNCS 2360, S. 23-43, Springer, 2002

[Des08] Desel; J: From human knowledge to process models. UNISCON 2008, April 2008, Klagenfurt, LNCS, Springer, 2008

[DJ03] Desel, J.; Juhás, G.; Lorenz, R.; Milijic, V.; Neumair, Chr.; Schieber, R.: Modellierung von Steuerungssystemen mit Signal-Petrinetzen – Eine Fallstudie aus der Automobilindustrie. Entwurf komplexer Automatisierungssysteme (EKA 2003), Universität Braunschweig, S. 273-295, 2003.

[DJ03a] Desel, J..; Juhás, G.; Lorenz, R.; Neumair, Chr.: Modelling and validation with VIPtool. Business Process Managenent (BPM 2003), LNCS 2678, S.380-389, Springer (2003)

[DR96] Desel, J..; Reisig, W.: The synthesis problem of Petri nets. Acta Informatica 33, S. 297-315 (1996)

[GS07] Glinz, M.; Seybold, Chr.; Meier, S.: Simulation-driven creation, validation and evolution of behavioral requirements models. Modellbasierte eingebettete Systeme (MBEES 2007), Informatik-Bericht 2007-1, TU Braunschweig, S. 103-112 (2007)

[HM03] Harel, D., Marelly, R.: Come, Let`s Play – Scenario-Baser Programming Using LSCs and the Play-Engine, Springer 2003

[Lie01] Liebermann, H. (Hrsg.): Your Wish is my Command: Programming by Example, Morgan Kaufmann 2001

[LB07] Lorenz, R.; Bergenthum, R.; Desel; J, Mauser, S.: Synthesis of Petri nets from partial languages. Applications of Concurrency to System Design (ACSD 2007), S. 157-166, IEEE (2007)

[MK02] Mayr, H.C.; Kop., Chr.: A user centric approach to requirements modeling. Modellierung 2002, LNI P-12, S. 75-86, Gesellschaft für Informatik (2002)

[Neu06] Neumair, Chr.: Entwicklung und Bewertung einer Unterrichtssequenz zur ablauforientierten Sichtweise von Algorithmen in der 7. Jahrgangsstufe. Staatsexamensarbeit, 2006

[SM06] Seybold, Chr.; Meier, S.; Glinz, M.: Scenario-Driven modeling and validation of requirements models. Workshop on Scenarios and State Machines: Models, Algorithms ans Tools (SCESM 2006), S. 83-89, ACM (2006)

[SK07] Schulte, C.; Knobelsdorf, M.: Das informatische Weltbild von Studierenden. Didatik der Informatik in Theorie und Praxis (INFOS 2007), LNI P-112, S. 69-80, Gesellschaft für Informatik (2007)

[WB07] Wiesner, B.; Brinda, T.: Erfahrungen bei der Vermittlung algorithmischer Grundstrukturen im Infomatikunterricht der Realschule mit einem Robotersystem. Didatik der Informatik in Theorie und Praxis (INFOS 2007), LNI P-112, S. 133-124, Gesellschaft für Informatik (2007)

Harmonisierung von Metaisierungsprinzipien und Methodenbausteinen

Stephan Schneider

Fachhochschule Lippe und Höxter
An der Wilhelmshöhe 44
D-37671 Höxter
stephan.schneider@fh-luh.de

Abstract: Die konzeptionelle Modellierung befasst sich mit der Konstruktion (Herstellung) von Informationsmodellen. Die methodisch geleitete Tätigkeit der Herstellung von Modellen orientiert sich zur Wahrung der Modellkonsistenz und -vollständigkeit an Metamodellen. In der Literatur sind bislang nur sprach- und prozessbasierte Metamodelle anzutreffen. Sprachbasierte Metamodelle bilden die Sprache, in der das untergeordnete Modell formuliert wird, ab, während prozessbasierte Metamodelle den Prozess der Herstellung eines Modells abstrahiert repräsentieren. Vor einem methodisch fundierten Hintergrund reichen jedoch die genannten Metamodelle nicht aus, um alle relevanten Aspekte innerhalb einer methodisch fundierten Modellkonstruktion zu erfassen. Dieser Artikel beschreibt das vom Autor neu eingeführte strukturbasierte Metamodell, das eine konkrete Struktur eines zu repräsentierenden Systems auf abstraktem Niveau repräsentiert und damit das anzustrebende Methodenziel fokussiert und metaisiert.

1 Einführung

Konzeptionelle Modelle sind zentrale Artefakte einer Angewandten Informatik wie zum Beispiel der Wirtschaftsinformatik. Sie stellen wie Modelle allgemein eine konstruierte Abstraktion eines Systems dar. Da sie Output-Artefakte der frühen Analysephase im Entwicklungszyklus sind und als Grundlage für darauf basierende Entwicklungsphasen (Entwurf, Implementierung) dienen, werden sie mithilfe einer Sprache formuliert, die noch keinen konkreten Bezug zu Hardware- und Softwaregegebenheiten herstellt. Konzeptionelle Modelle sind demnach als systemunabhängige Modelle anzusehen [Yo92, 379], die in semi-formaler Weise die die System prägende Fachbegriffswelt anschaulich und verständlich repräsentiert.

Die Konstruktion von (konzeptionellen) Modellen ist im Wesentlichen durch drei Faktoren geprägt: der *Sprache* zur Modellformulierung, der *Methode* zur Modellkonstruktion und den *Metamodellen* zur Wahrung der Modellkonsistenz und -vollständigkeit. Aus diesem Faktorenkomplex leiten sich zwei Fragestellungen ab, an denen sich dieser Beitrag ausrichtet: (1) Wie sieht eine methodisch fundierte Modellkonstruktion aus und (2) welche Bedeutung hat diese für die Metamodellierung?

2 Modelle als sprachliche Konstruktionsergebnisse

Die *Modellierung* ist ein Aufgabengebiet, das sich einer methodisch geprägten und paradigmatisch geleiteten Konstruktion von Modellen widmet. Ein *Modell* ist als Ergebnis einer Konstruktion die Abstraktion eines Systems (Problemdomäne, Gegenstandsbereich), das Vorhersagen und Rückschlüsse über das System erlaubt [Kü06, 370]. In ein Modell fließen die vorher wahrnehmungsbedingt (re)konstruierten Wissensinhalte ein [Ze99, 44 ff.]. Zur Explikation der Wissensinhalte, oder der Informationen bzw. Vorstellungen über das System, in einem Modell ist eine Sprache notwendig. Die Konstruktion eines Modells lässt sich demnach als sprachliche Handlung zur Herstellung eines explizierten Ergebnisses, eines Modells, interpretieren. Ein Modell ist somit als sprachliches Artefakt das (Repräsentations-)Ergebnis einer (sprachlichen) Herstellung. In diesen Aussagen spiegelt sich die für die Modellkonstruktion fundamentale Bedeutung von Sprache wider.

Einer maßgeblich von de Saussure [Sa01] geprägten Auffassung nach ist eine *Sprache* ein System aus einem geordneten Vorrat an Zeichen, das bestimmten phonologischen, morphologischen, syntaktischen und lexikalisch-semantischen Regeln zu ihrer Verwendung folgt [St96, 17; Wy04, 195]. Eine Sprache besteht grundlegend aus einer (1) abstrakten Syntax, die die Grammatik, d. h. die regelbasierte Zusammensetzung der Zeichen zu Sprachkonstrukten sowie deren Beziehungen untereinander, definiert, (2) einer konkreten Syntax (Notation) und (3) einer Semantik, die die kontextunabhängige Bedeutung der Sprachkonstrukte festlegt.

In sprachphilosophischer Hinsicht lassen sich die Sprachkonstrukte Type und Token unterscheiden [St96, 42 f.; We06; Kü06, 373 ff.]. In Form von Type (Typ, Form, Schema) und Token (Instanz, Exemplar, Ausprägung, Vorkommnis) erfolgt eine metaphysische, auch in der Ontologie gebräuchliche Differenzierung von universalen und singulären Aspekten. Unter einem *Type* ist ein allgemeiner sprachlicher Ausdruck, eine Klasse von Dingen, das abstrakt und eindeutig Universale oder eine Menge von Tokens zu verstehen. Ein Type als eigene Gattung ist sozusagen sui generis. Ein *Token* stellt eine konkrete Realisierung, eine Instanz oder ein Vorkommnis eines Types dar, wobei nicht alle Vorkommnisse eines Types zugleich Tokens sein müssen.

Speziell in der Modellierung werden Sprachen zur Beschreibung von Modellinhalten eingesetzt, die sich gemäß dem sprachkritischen Ansatz in der Wissenschaftstheorie als Orthosprachen [LoSc75, KaLo96] bzw. als auf darauf basierende Sprachansätze (Objekttypenmethode [We91], Normsprache [Or97]) erweisen. Unter *Orthosprache* versteht man eine von Grund auf methodisch aufgebaute Sprache, in der jedes Sprachkonstrukt ausdrücklich und zirkelfrei in seiner Verwendungsweise angegeben ist [LoSc75]. Eine Modellierungssprache als orthosprachliches Derivat schafft ein grundlegendes und abstraktes Begriffssystem aus Types (Prädikatoren, Themenwörtern), auf das sich konkrete Begriffe, die Tokens (Bezeichnungswörter), bei ihrer Ein- bzw. Zuordnung beziehen können.

3 Methodische Grundlegung der Modellkonstruktion

Einem grundlegenden, domänenunspezifischen Verständnis nach versteht man unter einer Methode (von griech. méthodos: »Weg«, »Gang einer Untersuchung«, eigentlich »Weg zu etwas hin«) ein nach Gegenstand und Ziel planmäßiges Verfahren, einschließlich der Kunstfertigkeit des Verfahrens zur Lösung praktischer und theoretischer Aufgaben [Br02]. Eine Methode ist das spezielle Charakteristikum wissenschaftlichen Vorgehens. In der etymologischen Grundbedeutung sind die wesentlichen Merkmale einer Methode bereits angelegt. Eine *Methode* bezeichnet ein zielorientiertes Vorgehen. Die beiden Wesenszüge einer Methode, die Zielorientierung und das (kunstfertige) Vorgehen, sind zwar Bestandteil zahlreicher, allgemein gehaltener und domänen(un)spezifischer Methodendefinitionen [BrHaWo04, 10], jedoch findet das Ziel selbst keine explizite Berücksichtigung in der Explikation und Anwendung konkreter Methodenverständnisse.

3.1 Metamodell einer Methode

Stellvertretend für die Auffassung zahlreicher Methodenverständnisse, die das Ziel nicht explizit als Methodenbaustein berücksichtigen, ist das bei Karagiannis/Kühn vorgestellte Metamodell einer Modellierungsmethode genannt (vgl. Abbildung 1, grau hinterlegter Bereich).

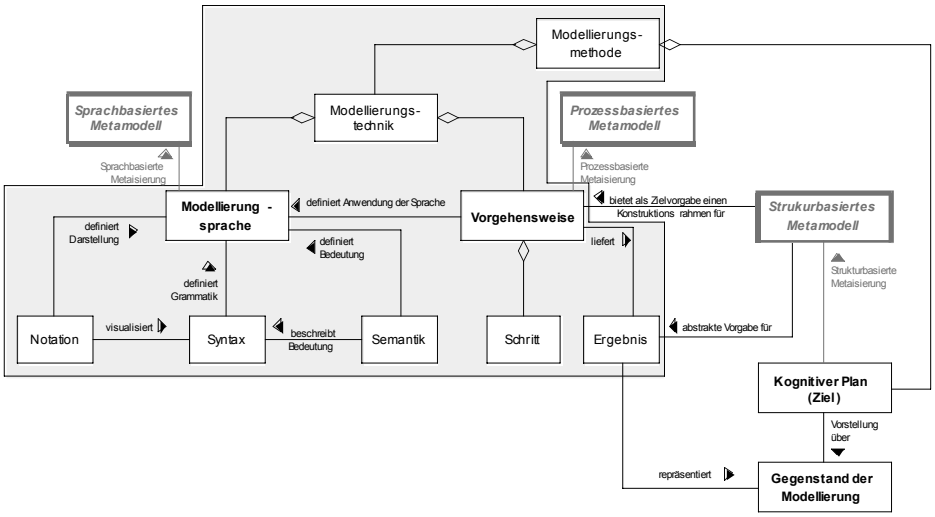


Abbildung 1: Metamodell einer Modellierungsmethode

Der grau hinterlegte Bereich zeigt das Methodenverständnis bei [KaKü02, 184]. Eine Methode besteht demnach aus einer *Modellierungstechnik*, die ihrerseits aus einer Modellierungssprache und einer Vorgehensweise besteht [vgl. ebenso St96, 91]. Eine *Modellierungssprache* wird durch die semiotischen Aspekte Syntax inklusive Notation und Semantik beschrieben. Die *Syntax* definiert die Grammatik, d. h. die regelbasierte Zusammensetzung der Zeichen einer Sprache, die *Semantik* definiert die Bedeutung der (zusammengesetzten) Zeichen und die *Notation* definiert durch die Zuordnung von Visualisierungselementen zu den (zusammengesetzten) Zeichen deren Darstellungsform. Die *Vorgehensweise* komponiert sich aus einer Vielzahl von *Schritten*, die durchgeführt werden, um ein bestimmtes, in der Modellierungssprache formuliertes Ergebnis (z. B. Modell) zu erstellen.

3.2 Das Ziel als Konstituent einer Methode

In der Literatur herrscht weitgehend Einigkeit, dass Methoden stets zielorientiert sind. Das *Ziel* kennzeichnet den Zweck, für den eine Methode konzipiert und eingesetzt wird. Der Zweck ist als Beweggrund für zielgerichtete Handlungen zu sehen. Bereits nach Aristoteles ist der Zweck das in der Vorstellung Gemeinte und als Plan Existierende, das realisiert zu werden verlangt [Br02]. Ist der kognitive Plan realisiert, gilt das Ziel als erreicht.

Für die dokumentierte Realisierung des kognitiven Plans gibt es eine Vielzahl möglicher Ergebnisse (Modelle), die der Vorstellung entsprechen und diese verwirklichen. Eine konkrete Vorgabe des Ziels ist jedoch wegen der zu hohen Anzahl an interpretatorischen Freiheitsgraden bei der Modellkonstruktion nicht möglich. Damit methodische Handlungen ihren Aufgaben gerecht werden, müssen sie sich an einer abstrakten *Zielvorgabe*, d. h. einer Abstraktion der planerischen Vorstellung, orientieren können. Grundlage der inhaltlichen Ausgestaltung einer Zielvorgabe ist die Durchdringung und Konzeptualisierung eines Gegenstandsbereichs. Zur Explikation von Zielvorgaben eignen sich Metamodelle prinzipiell hervorragend, da sie einen Aspekt der Modellbildung abstrahiert repräsentieren und sich aus dem Metamodell eine Vielzahl untergeordneter Modelle ableiten lässt.

Aus dem Konstituenten Ziel leitet sich die Forderung ab, eine weitere Abstrahierung des Ziels vorzunehmen und dieses Ergebnis in Form eines Metamodells mit zu explizieren, um den eingesetzten Handlungen als notwendige und sinnvolle Zielvorgabe zu dienen (vgl. Abbildung 1, rechte Seite).

3.3 Die Technik als Konstituent einer Methode

Neben dem Ziel ist die Technik der zweite Konstituent einer Methode. Der aktive Part einer Modellkonstruktion drückt sich im zielgerichteten Verfahren einer Technik aus. Die wesentlichen Aspekte einer Modellierungstechnik wurden bereits in Abschnitt 3.1 genannt. An dieser Stelle soll die Modellierungstechnik lediglich inhaltlich um die in Abschnitt 2 aufgeführten sprachphilosophischen Charakteristika Themenwort und Bezeichnungswort bzw. Type und Token erweitert werden.

Eine *Technik* ist demzufolge ein systematisches Verfahren zur Zielerreichung, das aus Sprachen und Vorgehensweisen besteht. Die Vorgehensweisen bestehen aus zeitlich-sachlogischen Abfolgen von Schritten, die festlegen, wie das Ziel zu erreichen und das resultierende Ergebnis (Modell) in einer Sprache zu erstellen und zu formulieren ist. Die Kernaufgabe einer Technik ist es, den Entstehungs- und Verwendungszusammenhang der Modellbausteine aufzudecken. Um dieser Aufgabe nachzukommen, muss eine Modellierungstechnik die geeignet erscheinenden Bezeichnungswörter bestimmen und sie den Themenwörtern des orthosprachlichen Begriffssystems einer Modellierungssprache zuordnen. Die Themenwörter entsprechen den Modellbausteintypen, während die Bezeichnungswörter die konkreten, singulären Modellbausteine darstellen.

4 Metamodellierung und Metaisierungsprinzipien

4.1 Grundverständnis eines Metamodells

Die methodisch geleitete Tätigkeit der Herstellung von Modellen orientiert sich zur Wahrung der Modellkonsistenz und -vollständigkeit an Metamodellen. Trotz seiner zentralen Bedeutung und einheitlichen Verwendung gibt es für den Begriff des Metamodells keine eindeutige Definition. Die Interpretationen des Metamodellbegriffs variieren autorenabhängig überaus stark, was zu einem breiten und heterogenen Spektrum an Metamodelldefinitionen führt. In sehr vereinfachter Definition betrachtet man ein Metamodell als ein Modell eines Modells [Kü06, 378; St96, 22; St99, 5]. Innerhalb der Metamodellbildung erweist sich diese Definition als zu ungenau, da die Beziehung zwischen Metamodell und Modell unpräzisiert bleibt und sich dadurch das Spektrum von als Metamodelle zu bezeichnenden Modellen weiterhin als zu breit und heterogen offenbart. Im Allgemeinen beschreibt ein Metamodell zwar immer ein Modell, im Speziellen hingegen bleibt die Frage nach dem konkreten Aspekt, den das übergeordnete Metamodell in Bezug auf das untergeordnete Modell abbildet, unbeantwortet. Allgemein ist ein *Metamodell* ein Modell der zur Modellierung verwendeten Konzepte, das heißt, ein Metamodell bildet einen bestimmten, für die Konstruktion von Modellen relevanten Aspekt abstrahiert ab und schafft dadurch eine Gesetzmäßigkeit, der das zu konstruierende Modell zu genügen hat [Fr94, 172]. Für eine Präzisierung des Metamodellbegriffs ist es zwingend erforderlich, den konkreten Aspekt, den das Metamodell in Bezug auf das zugrunde liegende Modell repräsentiert, fest- und offenzulegen.

4.2 Metaisierungsprinzipien zur Konkretisierung des Metamodellverständnisses

Strahringer verallgemeinert die konkreten Aspekte, die ein übergeordnetes Modell der Abstraktionsstufe n im Hinblick auf ein untergeordnetes Modell der Abstraktionsstufe n-1 beschreibt, im sogenannten Metaisierungsprinzip [St96, 24 ff.]. Das *Metaisierungsprinzip* kennzeichnet somit einen Aspekt, den das Modell der Stufe n in Bezug auf das Modell der Stufe n-1 in abstrakter Form fest- und offenlegt. Die Verallgemeinerung des Aspekts im Metamodell ist ein Hinweis auf die abstrakte Beschreibung, Darstellung und Erklärung des Aspekts, der zur Modellbildung führt. Ein Metaisierungsprinzip stellt mit anderen Worten ein Prinzip bzw. Kriterium dar, das zur Abstraktionsstufen- und Metamodellbildung führt [St99, 6]. Die Abstraktion ist jedoch bereits ein der Modellbildung immanenter Prozess [Kü06] und soll im Rahmen der Metamodellierung nicht zu Missverständnisses führen. Im Zuge der Metamodellbildung (Metaisierung) wird eine weitere Abstrahierung der Systemelemente vorgenommen und in einem Metamodell manifestiert.

Auf dem Verständnis der Abstraktionsstufen aufbauend und dem sprachphilosophischen Ansatz von Type vs. Token folgend, lassen sich Metamodelle als Type-Modelle der Ebene n und Modelle als Token-Modelle der Ebene n-1 interpretieren, wobei ein Token-Modell die Instanz eines Type-Modells darstellt [Kü06, 373].

4.3 Sprach- und Prozessbasierte Metamodellbildung

Strahringer grenzt in ihren Arbeiten [St96; St98; St99] zwei fundamentale Metaisierungsprinzipien voneinander ab, die in der Literatur weithin als etabliert und akzeptiert gelten. Bei den beiden Metaisierungsprinzipien handelt es sich um die Sprache und die Vorgehensweise (vgl. Abbildung 1). Demnach lassen sich zwei grundlegende Metamodellbildungen unterscheiden:

(1) Bei einer *sprachbasierten Metamodellbildung* beschreibt das sprachbasierte Metamodell in abstrakter Art und Weise die Sprache, mit der das untergeordnete Modell formuliert wird.

(2) Bei einer *prozessbasierten Metamodellbildung* beschreibt das prozessbasierte Metamodell in abstrakter Art und Weise die Prozesse in Form einer logischen Prozessstruktur, nach der das untergeordnete Modell erstellt wird.

Vor allem die prozessbasierte Metamodellbildung durchbricht die in der Literatur ebenfalls anzutreffende Anschauung, dass Metamodelle lediglich eine Beschreibung der abstrakten Syntax einer Sprache darstellen. Gerade dieser Aspekt erklärt, weshalb Metamodelle oftmals rein sprachbasiert interpretiert werden.

4.4 Strukturbasierte Metaisierung

Vor einem methodisch fundierten Hintergrund (vgl. Kapitel 3) reichen jedoch die genannten Metaisierungsprinzipien nicht aus, um alle relevanten Aspekte einer methodisch geprägten Modellkonstruktion adäquat zu erfassen. Wie der grau hinterlegte Teil in Abbildung 1 zeigt, ist das Ziel – obgleich Konstituent einer Methode – kein expliziter Bestandteil gängiger Methodenauffassungen. Wie aus dieser Abbildung hervorgeht, wurde das Methodenverständnis durch die Integration der Zielkomponente vervollständigt.

In Anlehnung an Abbildung 1 lässt sich neben der Sprache und der Vorgehensweise das Ziel als drittes fundamentales Metaisierungsprinzip ausmachen und das daraus resultierende Metamodell einem eigenen, strukturbasierten Metamodell zuführen. Das Ziel entspricht allgemein als ein kognitiver Plan einer Vorstellung über das im Modell zu repräsentierende System.

(3) Bei einer *strukturbasierten Metaisierung* beschreibt das strukturbasierte Metamodell in abstrakter Art und Weise die Struktur, die das untergeordnete Modell besitzen wird. Ein strukturbasiertes Metamodell ist demnach ein abstraktes Modell der Struktur eines Systems bzw. ein Modell der Struktur eines Modells.

Der Grund für die Erweiterung liegt in der Feststellung, dass das Ziel einer Methode oder das Ergebnis der Modellkonstruktion, das Modell, selbst nicht als eigenständiger Gegenstand und somit als Aspekt der Metamodellbildung betrachtet und einem separaten Metaisierungsprinzip zugeführt wird. Vielmehr gehen systemstrukturelle Aspekte in einem sprachbasierten Metamodell auf. Daraus ergibt sich aber die Schwierigkeit, keine exakten Aussagen hinsichtlich des metaisierten Aspekts aus einem sprachbasierten Metamodell abzuleiten. Sowohl die Sprache als auch die Systemstruktur können Gegenstand der Metamodellbildung sein. Um Differenzierungen zu ermöglichen, erscheint es sinnvoll, systemstrukturelle Aspekte einem eigenständigen Metaisierungsprinzip zuzuordnen und in einem entsprechenden Metamodellbegriff, dem strukturbasierten Metamodell, auszulagern. Dadurch ist es möglich, rein sprachliche Aspekte einem sprachbasierten und rein systemstrukturelle Aspekte einer strukturbasierten Metamodell zuzuführen.

Eine Modellierungssprache leistet in Form ihres orthosprachlichen Begriffssystems (System von Themenwörtern) eine grundlegende Systemstrukturvorlage. Das Ziel bzw. das Ergebnis einer Modellierungsmethode ist ebenso wie der Prozess der Modellerstellung durch eine Sprache beschreibbar. Die Sprache avanciert dadurch selbst automatisch zum Instrument einer Zielvorgabe, da sie die zur Beschreibung des Ziels relevanten Sprachkonstrukte (Themenwörter) in Form eines orthosprachlichen Begriffssystems umfasst.

Unter sprachlichen Gesichtspunkten dient der Technik einer Methode das in der Modellierungssprache bzw. in der Zielvorgabe (Abbildung 1) definierte orthosprachliche Begriffssystem als Ordnungsrahmen für die zu bewältigenden Aufgaben. Das orthosprachliche Begriffssystem entspricht den Themenwörtern inklusive ihrer Zusammenhänge und somit den Typen an Modellbausteinen und ihren Beziehungen. Die Technik ermittelt die Bezeichnungswörter, die die (Fach-)Begriffe eines betrachteten Sachverhalts und somit potenzielle Bausteine des zu konstruierenden Modells darstellen, und ordnet sie den Themenwörtern des orthosprachlichen Begriffssystems zu. Durch diese Zuordnung wird folglich der Modellbaustein festgelegt, der das Bezeichnungswort bzw. den Fachbegriff konkret im Modell repräsentieren soll. Die Bezeichnungswörter werden aufgrund der Zuordnung zu Themenwörtern zu Bausteinen des Modells, deren Typen das orthosprachliche Begriffssystem in Form von Themenwörtern definiert.

Ist jedoch der Abstraktionsgrad des orthosprachlichen Begriffssystems einem intuitiven Verständnis nach zu hoch, so kann die Modellierungssprache ihren Zweck der Zielvorgabe nicht adäquat erfüllen. Aus diesem Grund erscheint es angebracht, die Systemstruktur vom Begriffssystem der Modellierungssprache zu lösen und sie nicht implizit aus dem orthosprachlichen Begriffssystem abzuleiten. Die eigens kreierte Zielstrukturvorgabe lässt sich zwar als eigenes orthosprachliches Begriffssystem interpretieren, das sich von der verwendeten Modellierungssprache löst, aber dennoch von dieser ableitbar bleibt.

Der Unterschied zwischen sprach- und strukturbasierter Metaisierung ist ontologischer Natur. Eine der weitverbreitetsten Definitionen des Ontologiebegriffs stammt von Gruber. Seiner Meinung nach stellen Konzeptualisierungen den Rahmen für eine geteilte und wiederverwendbare formale Wissensrepräsentation. Unter einer Konzeptualisierung versteht Gruber „an abstract, simplified view of the world that we wish to represent for some purpose“ [Gr93, 199]. Basierend auf diesem Konzeptualisierungsverständnis handelt es sich bei einer Ontologie um eine explizite Spezifikation einer von Individuen geteilten Konzeptualisierung von real- oder vorstellungsweltlichen Phänomenen. Sprachbasierte Metamodelle abstrahieren Begriffssysteme, während strukturbasierte Metamodelle abstrahierte Konzeptualisierungen darstellen. Die Nähe von Begriffen und Konzepten führt dazu, dass die Unterscheidung zwischen sprach- und strukturbasierter Metaisierung im Prinzip trennunscharf ist, jedoch nicht auf einer Abstraktionsebene. Die Durchdringung eines Gegenstandsbereichs und die damit verbundene Klärung von Fragen hinsichtlich geeigneter Konzeptualisierungen zur Repräsentation des Systems führt zur Entwicklung neuer Themenwörter, die sich aus den Themenwörtern eines sprachbasierten Metamodells ableiten. In einer ersten Phase werden diese Themenwörter strukturbasierten Metamodellen zugeführt. Erst die Etablierung der Themenwörter über einen Zeitraum sowie ihre empirische Absicherung führen in einer zweiten Phase zu einer neuen Terminologie, die ihrerseits einer eigenen Sprache zugeführt werden kann.

Durch strukturbasierte Metamodelle wird nach Ansicht des Autors die Lücke in der Metamodellierung geschlossen, Systemstrukturvorgaben explizit zu betrachten und einem eigenen Metamodell zuzuführen. Strukturbasierte Metamodelle entsprechen somit auf Metaebene den sogenannten Leitbildern der Systemgestaltung. Zusammenfassend legt in vereinfachter Form die sprachbasierte Metamodellbildung abstrakt fest, *womit* etwas beschrieben wird, die prozessbasierte Metamodellbildung *wie* etwas zu erreichen ist und die strukturbasierte Metamodellbildung *was* es zu erreichen gilt.

5 Beispiel eines strukturbasierten Metamodells

Das abschließende Kapitel 5 zeigt ein Beispiel eines strukturbasierten Metamodells. Das in Abbildung 2 dargestellte Metamodell ist lediglich ein Ausschnitt des gesamten Metamodells, das Riempp in seiner Arbeit [Ri04, 123] vorstellt. Bei dem gesamten Metamodell handelt es sich um ein Metamodell für eine Wissensmanagementsystem (WMS)-Architektur. Das in der Abbildung in der Notation der semantischen Netze dargestellte Fragment beschreibt in Teilen die Strategie-, Organisations-, Prozess- und Kundenebene. Ausgangspunkt ist eine Marktleistung, die durch den Kundenbedarf bestimmt ist und zu dessen Befriedung beiträgt. Auf Basis strategischer Ziele implementiert das Unternehmen einen oder mehrere Geschäftsprozesse, deren Leistungen die Marktleistung realisiert. Die Geschäftsprozesse bestehen aus Aufgaben, die Mitarbeiter mit bestimmten Rollen erledigen.

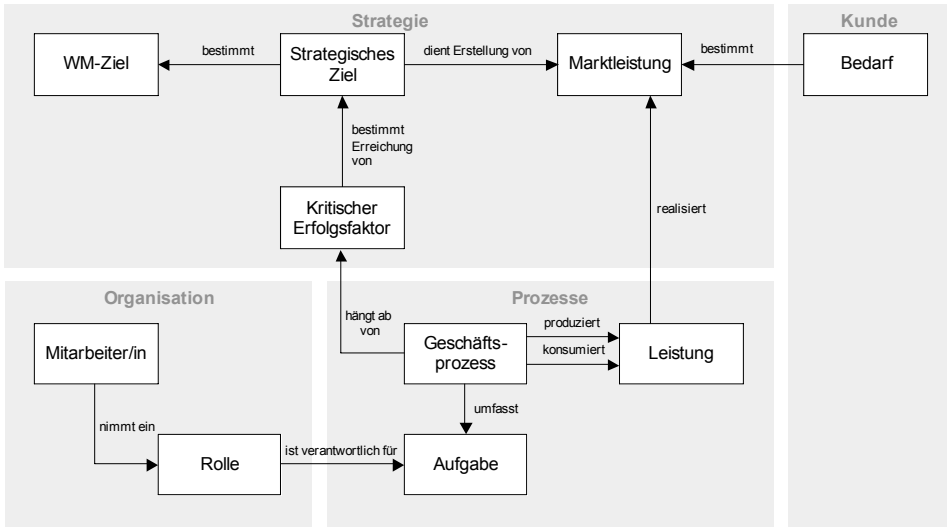


Abbildung 2: Strukturbasiertes Metamodell (Ausschnitt) einer WMS-Architektur nach Riempp

Bei diesem Metamodell handelt es sich um ein strukturbasiertes Metamodell, da es die Struktur eines Systems, bestehend aus den Subsystemen Unternehmen und Kunde, in abstrahiert-klassifizierter Form repräsentiert. Das in diesem Metamodell manifestierte Begriffssystem (Strategisches Ziel, Marktleistung, Geschäftsprozess, Aufgabe usw.) zeichnet sich durch eine Konkretisierung der in sprachbasierten Metamodellen in der Regel höher abstrahierten Begrifflichkeiten (z. B. Klasse, Assoziation) aus.

Die in einem strukturbasierten Metamodell abstrahierte Begrifflichkeit, wie eben Strategisches Ziel, Marktleistung, Geschäftsprozess, Aufgabe usw., ist zwar Bestandteil einer natürlichen oder künstlichen Sprache, jedoch keine Begrifflichkeit, die sich üblicherweise in sprachbasierten Metamodellen vorfindet. Vielmehr beschreibt das *strukturbasierte Metamodell* anhand geeigneter Themenwörtern sowie deren Zusammenhänge die *konkrete Struktur eines Systems auf abstraktem Niveau*. In Anlehnung an den Ontologiegedanken lässt sich ein strukturbasiertes Metamodell auch als *metaisierte Ontologie* interpretieren [Sc07, 496].

Literaturverzeichnis

[Br02] Digitale Fassung der Brockhaus Enzyklopädie, Bibliographisches Institut & F. A. Brockhaus AG, 2002.

[BrHaWo04] Braun, C., Hafner, M., Wortmann, F.: Methodenkonstruktion als wissenschaftlicher Erkenntnisansatz, in: Back, A., Brenner, W., Österle, H., Winter, R. (Hrsg.): Berichte des Instituts für Wirtschaftsinformatik, Bericht Nr. BE HSG/IWI 1, Universität St. Gallen, 2004.

[Fr94] Frank, U.: Multiperspektivische Unternehmensmodellierung: Theoretischer Hintergrund und Entwurf einer objektorientierten Entwicklungsumgebung, München, Oldenbourg Verlag, 1994

[Gr93] Gruber, T. R.: A Translation Approach To Portable Ontology Specifications, in: Knowledge Acquisition 5, Nr. 2, 1993, S. 199-221.

[KaKü02] Karagiannis, D., Kühn, H.: Metamodelling Platforms, in: Bauknecht, K., Min Tjoa, A., Quirchmayer, G. (Hrsg.): Proceedings of the Third International Conference EC-Web 2002 – Dexa 2002, Aix-en-Provence, France, September 2-6, 2002, LNCS 2455, Berlin, Heidelberg, S. 182-194.

[KaLo96] Kamlah, W., Lorenzen, P.: Logische Propädeutik: Vorschule des vernünftigen Redens, 3. Aufl., Stuttgart, Weimar, BI Wissenschaftsverlag, 1996.

[Kü06] Kühne, T.: Matters of (Meta-)Modeling, in: Journal on Software and Systems Modeling, Vol. 5, Nr. 4, December 2006, S. 369-385.

[LoSc75] Lorenzen, P, Schwemmer, O.: Konstruktive Logik, Ethik und Wissenschaftstheorie, 2. Aufl., Mannheim et al., Bibliogr. Inst. + Brockha, 1975.

[Or97] Ortner, E.: Methodenneutraler Fachentwurf, Stuttgart, Leipzig, Teubner Verlag, 1997.

[Ri04] Riempp, G.: Integrierte Wissensmanagementsysteme: Architektur und praktische Anwendung, Berlin et al., Spinger Verlag, 2004.

[Sa01] de Saussure, F.: Grundfragen der allgemeinen Sprachwissenschaft, 3. Aufl., Berlin, New York, Gruyter, 2001.

[Sc07] Schneider, S.: Konstruktion generischer Datenmodelle auf fachkonzeptioneller Ebene im betrieblichen Anwendungskontext: Methode und Studie, Aachen, Shaker Verlag, 2007.

[St96] Strahringer, S.: Metamodellierung als Instrument des Methodenvergleichs: Eine Evaluierung am Beispiel objektorientierter Analysemethoden, Aachen, Shaker Verlag, 1996.

[St98] Strahringer, S.: Ein sprachbasierter Metamodellbegriff und seine Verallgemeinerung durch das Konzept des Metaisierungsprinzips, in: Pohl, K., Schürr, A., Vossen, G. (Hrsg.): CEUR Workshop Proceedings zur Modellierung '98 (GI-Workshop in Münster, 11.-13. März 1998), CEUR-WS/Vol-9.

[St99] Strahringer, S.: Probleme und Gefahren im Umgang mit "Meta"-Begriffen: ein Plädoyer für eine sorgfältige Begriffsbildung, in: Proceedings of the International Knowledge Technology Forum (KnowTechForum)'99, 16.-18. September, Potsdam, 1999, http://wwwfl.ebs.de/Lehrstuehle/Wirtschaftsinformatik/NEW/Publications/Meta-BegriffeKnowtech.pdf, Zugriff: 18.08.2004.

[We91] Wedekind, H.: Datenbanksysteme, Band 1: Eine konstruktive Einführung in die Datenverarbeitung in Wirtschaft und Verwaltung, Mannheim et al., 1991.

[We06] Wetzel, L.: Types and Tokens, in: Stanford Encyclopedia of Philosophy, http://plato.stanford.edu/entries/types-tokens/, Zugriff: 11.02.2007.

[Wy04] Wyssusek, B.: Methodologische Aspekte der Organisationsmodellierung in der Wirtschaftsinformatik – Ein soziopragmatischer-konstruktivistischer Ansatz, Berlin, 2004

[Yo92] Yourdon, E.: Moderne strukturierte Analyse: ein Standardwerk zur modernen Systemanalyse, Attenkirchen, Prentice Hall, 1992.

[Ze99] Zelewski, S.: Grundlagen, in: Corsten, H., Reiß, M. (Hrsg.): Betriebswirtschaftslehre, 3. Aufl., München - Wien 1999, S. 1-125.

Ein Objektmodell für zustandsbasierte Simulationsmodelle

Jörg Höhne
Digitale Medien in der Bildung (dimeb), Informatik, TZI,
Universität Bremen, Bibliothekstr. 1, 28359 Bremen
E–Mail: hoehne@informatik.uni-bremen.de

Abstract: Für die computerbasierte Simulation wird ein Ausgangsmodell durch den Implementierungsprozess in ein Simulationsmodell abgebildet. Die Ausgangsmodelle für individuenbasierte Modellierung liegen z.B. als eine zustandsbasierte Beschreibung, wie in Form eines automatenähnlichen Zustandsdiagramms, vor. Der algorithmische Entwicklungsaufwand für die Abbildung eines Ausgangsmodells in ein Simulationsmodell wird wesentlich durch das Programmiermodell der verwendeten Simulationsumgebung bestimmt.

Ausgehend von der Beobachtung, dass sich individuen- und zustandsbasierte Modelle in Simulationsumgebungen, wie z.B. NETLOGO, nur mit einem zusätzlichem und möglicherweise fehlerträchtigen Aufwand abbilden lassen, ist ein Programmierkonzept entwickelt worden, welches den Implementierungsprozess wesentlich vereinfachen kann. Das Konzept besteht aus einem Objektmodell, welches neben objektorientierten Eigenschaften eine zusätzliche Strukturierungsebene in Form von Zuständen beinhaltet, sowie einem graphischen Editor, der die Möglichkeiten des neuen Objektmodells visualisiert und Modifikationen ermöglicht.

Mit der Vereinfachung des Implementierungsprozesses wird die Erwartung verbunden, dass sich die Simulationstechnik einem Anwenderkreis erschließt, der vormals den zusätzlichen Implementierungsaufwand herkömmlicher Simulationsumgebungen als zu hohe Eingangshürde erfahren oder betrachtet hat.

1 Einleitung

Die Simulationstechnik ist ein wichtiges Hilfsmittel für die Untersuchung von beobachteten Phänomenen, wie z.B. die in (KW04) beschriebene Schwarmbildung. Für die Erstellung einer computergestützten Simulation sind inzwischen eine Vielzahl von Softwareprodukten verfügbar, welche für den Einsatz im Schulunterricht geeignet wären. Die Arbeit mit potentiell geeigneten Simulationsumgebungen zeigte, dass diese die Umsetzung von zustandsbasierten Modellen in Simulationsmodelle nur unzureichend unterstützen, so dass ein neues Konzept für eine einfachere und bessere Umsetzung entwickelt wurde.

Das Konzept besteht aus einer Kombination von Eingabe- und Visualisierungstechnik sowie einem neuen Objektmodell, in dem die Objekte zusätzlich durch Zustände strukturiert sind. Das Konzept kann den Erstellungsprozess eines Simulationsmodells durch eine direktere Abbildung eines vorliegenden Modells und durch eine stärkere Strukturierung des Programmcodes vereinfachen. Das Konzept realisiert im Wesentlichen eine erweiter-

te Modellierungssprache und eine Änderung des Softwareentwicklungsprozesses, woraus sich in der Kombination eine Vereinfachung ergeben soll.

Die Zielgruppe dieses Konzeptes sind primär diejenigen Anwender, welche Simulationsmodelle aus Modellen erstellen, wofür Kenntnisse aus der Informatik und in der Programmierung benötigt werden. Mit dem Konzept wird für den Anwender eine Verringerung der Komplexität hinsichtlich der Implementierung des Simulationsmodells sowie der Modifizierung, wie Änderung oder Erweiterung, erreicht. Mit diesem universellen Ansatz können somit Zielgruppen erreicht werden, welche über weniger umfassende Kenntnisse aus dem Bereich der Informatik verfügen müssen als die etablierten Systeme bisher vorausgesetzt haben. Die Vereinfachung des Implementierungsprozesses bedeutet keine Reduzierung der Leistungs- und Abbildungsfähigkeit, so dass sich dieses Konzept auch für einen Anwenderkreis eignet, welcher sich intensiv mit Simulationstechnik und der Implementierung von Simulationsmodellen beschäftigt, z.B. der universitäre Bereich. Speziell wurde das Konzept unter dem Aspekt des schulischen Einsatzes entwickelt, um Lehrkräften und Schülern gleichermaßen die Entwicklung von Simulationsmodellen zu vereinfachen. Diese Vereinfachung soll den Einsatz von Simulationsmodellen auch außerhalb des Informatikunterrichts in anderen Fächern fördern, so dass ein verstärktes interdisziplinäres Arbeiten möglich ist.

2 Modellierung und Modelle

Die Betrachtung des Konzepts erfolgt im Zusammenhang mit *individual-based modelling*, der Modellierung mit dem Fokus auf das einzelne Individuum (Mikrosimulation (KW04)) anstelle der numerischen Betrachtung von Modellgrößen (Makrosimulation, z.B. STELLA). In einer Makrosimulation wird eine Menge gleichartiger Individuen betrachtet, welche im Modell als eine Größe formuliert und berechnet werden. Ein klassisches Beispiel ist die Entwicklung einer Räuber-Beute-Population nach den Lotka-Volterra-Gleichungen, in denen die Populationsgrößen als Variablen in dem Modell vorliegen. In der Mikrosimulation hingegen wird *jedes einzelne* Individuum algorithmisch formuliert (KW04) und berechnet. Nach (KW04; KCR02) liegt der Vorteil der Mikrosimulation in der leichteren Verständlichkeit für die Schüler, weil algorithmische Beschreibungen für die Schüler leichter verständlich und auch leichter zu erweitern sind.

Weitere Anwendungsmöglichkeiten erschließen sich durch die relativ einfache Möglichkeit zur räumlichen Beschreibung von Prozessen, z.B. die Ausbildung der Spiralmuster der Belousov-Zhabothinsky-Reaktion, vgl. (GS95), so dass eine Übertragung eines beobachteten chemischen Experiments in eine Computersimulation möglich ist. Eine vergleichbare Makrosimulation mittels (partieller) Differentialgleichungen setzt aufgrund der komplexen mathematischen Beschreibung eine erhebliche Hürde, wie das in der Literatur beschriebene einfachere Modell für den eindimensionalen Fall nahelegt.

Modelle liegen in unterschiedlichen Formen vor, z.B. als verbale (textuelle) Beschreibung oder als graphische Darstellung, welche die verschiedenen Zustände abbildet. Die zustandsbasierte Beschreibung ist meines Ermessens sinnvoll, weil in einer graphischen

Darstellung das Verhalten des Individuums anschaulicher, erfassbarer und nachvollziehbarer beschrieben wird (Abbildung 1 und (CFS$^+$01)). Alternative Darstellungsformen, wie eine tabellarische Darstellung und Beschreibung von Zustandsmengen (HMD02), sind zu abstrakt.

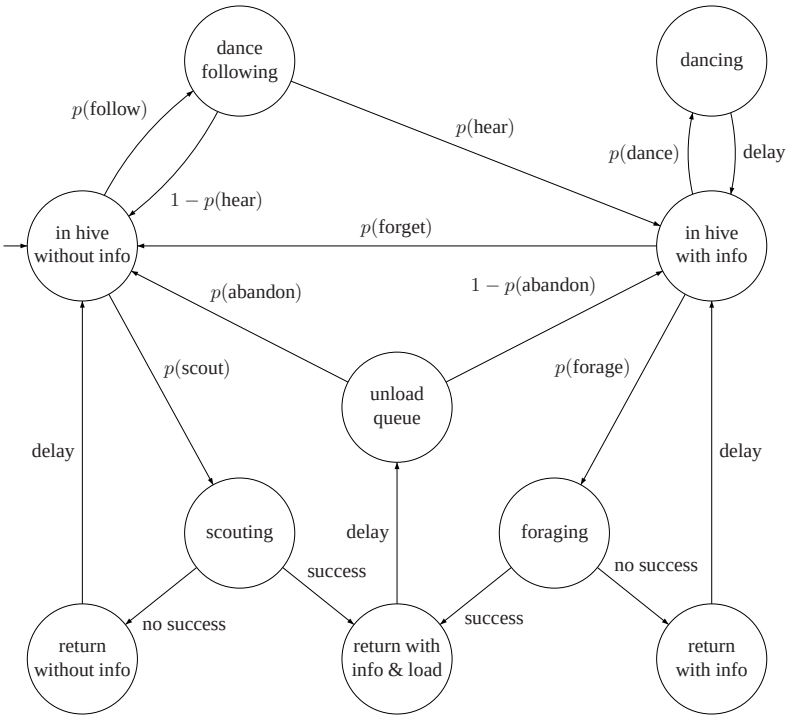


Abbildung 1: Zustandsmodell einer Honigbiene aus (SC04)

Die graphische Notation erlaubt eine Übersichtsdarstellung des Objektverhaltens, wobei zunächst keine aufwendige Analyse wie bei einer verbalen Beschreibung notwendig ist. Neben der vorteilhafteren Darstellung sind Änderungen an dem Modell einfacher zu realisieren: Knoten können entfernt oder eingefügt werden und es sind lediglich die Kanten anzupassen. Bei einer verbalen Beschreibung muss in der Regel der gesamte Text auf formulierte Abhängigkeiten überprüft werden, so dass Modifikationen erheblich aufwendiger sind. Aus genannten Gründen besitzt eine graphische Notation gegenüber einer verbalen Beschreibungsform Vorteile, die für die Implementierung genutzt werden sollen.

Die Abbildung 1 zeigt ein Verhaltensmodell einer Honigbiene aus (SC04). Die Darstellung des Modells orientiert sich stark an einem Endlichen Automaten, d.h. die modellierte Honigbiene nimmt Zustände ein und besitzt Übergangsbedingungen, bei deren Erfüllung der aktuelle Zustand verlassen und ein Folgezustand erreicht wird.

Für eine Implementierung des Modells in ein Simulationsmodell geht diese vorteilhafte graphische Struktur durch die verbale Abbildung in der Regel verloren. Wenn im Gegen-

satz dazu während des Implementierungsvorgangs das graphische Modell (mit geringen) Änderungen übernommen und in einem weiteren Schritt mit Quelltext versehen werden könnte, dann läge eine einfache Art zur Implementierung eines zustandsbasierten Objektverhaltens vor. Diese Art der Programmierung erleichtert die Überführung bestehender Modelle in Simulationsmodelle, weil faktisch eine direkte Übernahme des Modells in eine computerbasierte graphische Notation vorgesehen ist. Ist die Ähnlichkeit zwischen den beiden Abbildungsformen ausreichend groß, dann ist die Übertragung eher einem reinen mechanischen Prozess gleichzusetzen, so dass auf dieser Stufe der Simulationsmodellerstellung verstärkt eine Konzentration auf die Inhalte erfolgen kann.

Sind in einer Simulation mehrere Klassen von Individuen vorhanden, wie z.B. in (DKPT98) beschrieben, dann ist zusätzlich eine Vererbungsstrategie sinnvoll, weil sich einige Klassen gemeinsames Verhalten „teilen“ und innerhalb einer Vererbung die Subklassen das Verhalten entsprechend spezifizieren können.

3 Abbildung von Modellen in Simulationsumgebungen

3.1 Verfügbare Simulationsumgebungen

Unter einer Simulationsumgebung wird eine Software verstanden, welche dem Anwender eine vollständige Arbeitsumgebung zur Programmierung von Objekten (respektive ihrer Klassen), Plazieren (räumlich und zeitlich) von Objekten in einem Simulationsuniversum, Ausführen von Objekten während einer Simulation, Visualisierung des Simulationsuniversums, graphische Darstellung von Simulationszuständen und Interaktionsmöglichkeiten, z.B. mittels Bedienelementen, zur Verfügung stellt. Eine Simulationsumgebung unterscheidet sich von einem *framework* (Programmbibliothek) dahingehend, dass ein framework in der Regel vordefinierte Programmsegmente und Funktionalität zur Verfügung stellt, welche aus einer Hochsprache, wie z.B. Java, heraus aufgerufen werden. Für den Anwender sind eine Hochsprache und eine Entwicklungsumgebung zu erlernen, bevor die Möglichkeiten eines frameworks genutzt werden können. Der Initialaufwand zur Nutzung der Simulationstechnik über ein framework erscheint hoch, daher wird der Fokus auf Simulationsumgebungen gelegt, insbesondere auch deshalb, weil frameworks üblicherweise nicht die erweiterte Funktionalität, wie z.B. spezielle Editoren, zur Verfügung stellen.

Für eine Aussage zur Abdeckung der gewünschten Eigenschaften sind diverse Simulationsumgebungen erfasst (Höh07) und auf Abbildungseignung von Modellen zu Simulationsmodellen untersucht worden. Ausgewählt wurden freie bzw. kostenlose[1] Simulationsumgebungen, die eine gewisse Verbreitung besitzen. In Tabelle 1 sind die wesentlichen Eigenschaften der untersuchten Simulationsumgebungen aufgeführt.

Die Eigenschaft „Zellulärer Automat“ bezeichnet eine besondere Art der Modellierung, in der örtlich fixe Individuen in den Zellen eines regelmäßigen Gitters angeordnet und unter Berücksichtigung ihrer unmittelbaren Umgebung gleichzeitig berechnet werden. Dieses

[1] Die Simulationsumgebung AGENTSHEETS ist nicht kostenlos, wurde dennoch als relativ kostengünstiges kommerzielles Produkt aufgenommen, um einen Vergleich mit nicht kommerzieller Software zu ermöglichen.

Kriterium	AGENTSHEETS	BREVE	NETLOGO	SESAM	MOBIDYC
WINDOWS/OS X/LINUX	+/+/○	+/+/+	+/+/+	+/+/+	+/+/+
Freie Lizenz/quelloffen	–/–	+/+	+/–	+/+	+/+
Kontinuierliches Modell	–	–	+	–	–
Zellulärer Automat	+	–	+	+	+
Individuenbasiert	+	+	+	+	○
Kapselung	+	+	–	–	–
Vererbung	–	+	–	–	+
Zustandsunterstützung	–	–	–	○	–
Object-Inspector	+	+	+	+	–
Programmdebugger	○	–	–	–	–
2D/3D-Visualisierung	+/–	–/+	+/+	+/–	+/–
GUI Konstruktion	+	○	+	+	–

+: Kriterium vorhanden, ○: Kriterium eingeschränkt vorhanden, –: Kriterium nicht vorhanden.

Tabelle 1: Eigenschaften der betrachteten Simulationssysteme.

entspricht im Wesentlichen einem Zellulären Automaten, mit dem sich interessante Simulationsmodelle aus dem Bereich der Chemie oder Biologie erstellen lassen (GS95).

Für die Definition eines Objektes (Individuums) in einer Simulation bietet sich die objektorientierte Programmierung an, d.h. ein Objekt kapselt seine Eigenschaften und auch Methoden vor dem Zugriff von anderen Objekten innerhalb der Simulation. Diese Kapselung ist sinnvoll, weil somit das eigentliche Modell eines *autonomen* Individuums, die wohl zentrale Eigenschaft eines Agenten (Woo05), in der Implementierung realisiert werden kann. Dieses dient dem programmtechnischen Aspekt von der Autonomie eines Individuums.

Die Untersuchung der Simulationsumgebungen in (Höh07) zeigte, dass keine Simulationsumgebung die gewünschten Anforderungen hinsichtlich des Programmiermodells erfüllt. SESAM bietet eine schwache Unterstützung von Zuständen, jedoch fehlt es in dem Programmiermodell an Flexibilität sowie einer Vererbungsstrategie, so dass ein anderes Objektmodell vorgeschlagen werden soll.

3.2 Das vorgeschlagene Objektmodell

Für die Abbildung zustandsbasierter Modelle wird ein spezielles Objektmodell vorgeschlagen, das in der Klassendefinition durch eine zusätzliche Strukturierung in Zustände charakterisiert ist. Ein Zustand ist einer Klassendefinition aus einer objektorientierten Programmiersprache (z.B. JAVA) angenähert und enthält Attribute (Eigenschaften) sowie Methoden. Attribute sind stets gekapselt, ein Zugriff außerhalb der Klasse ist nur über Methoden möglich. Methoden sind ebenfalls nur durch eine explizite öffentliche Deklaration für andere Objekte sichtbar.

Der Zustand BASE definiert die grundsätzlichen Eigenschaften eines Objektes und repräsentiert dieses atomar, d.h. die innere Struktur und der eingenommene Zustand sind nach

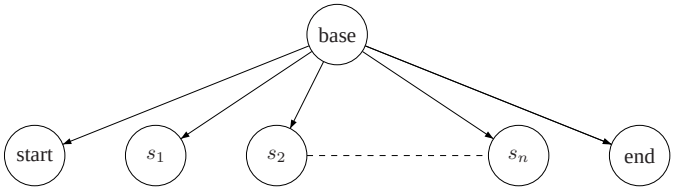


Abbildung 2: Zustandshierarchie

„Außen“ nicht sichtbar. Weitere Zustände einer Klasse leiten sich stets direkt von dem Zustand BASE durch eine einfache Vererbung ab (Abbildung 2). Die abgeleiteten Zustände können Attribute und Methoden des Zustands BASE überschreiben, so dass *zustandsspezifischer* Code implementierbar ist. In einem Objekt sind stets folgende Zustände definiert:

- BASE für die grundlegende Definition der Klasse und ihrer Repräsentation.
- START, welcher bei Erzeugung eines Objektes erreicht wird.
- END, welcher das Objekt terminiert, d.h. bei Erreichen von Zustand END wird das Objekt automatisch aus der Simulation entfernt.

Im Rahmen einer Programmierung können weitere Zustände hinzugefügt werden, wobei jeder Zustand wenigstens die Methoden onEnter() für das Eintreten in den Zustand, onExit() für das Verlassen des Zustands sowie onIterate() definiert. Die Methode onIterate() dient der Iteration eines Objektes und wird während der Simulationsausführung je Simulationszyklus einmal für jedes Objekt aufgerufen. Für das Objektmodell gilt, dass jeweils einer der von BASE abgeleiteten Zustände aktiv ist. Sofern ein Zustand eine Methode von BASE überschreibt wird diese anstelle der ursprünglichen Methode aus BASE aufgerufen. Dieses Ausführungsmodell stellt sicher, dass *zustandsspezifischer* Programmcode anstelle allgemeineren Programmcodes (aus BASE) aufgerufen wird.

3.3 Der technische Rahmen für das neue Objektmodell

Für das neue Objektmodell sind technische Rahmenbedingungen zu schaffen, die den Abbildungsprozess vereinfachen. Es sollte ein graphischer Editor existieren, welcher die Konstruktion des Zustandsgraphen erlaubt. Neben dem „Zeichnen“ des Graphen ist zusätzlich die Definition von Methoden in einem Zustand zu unterstützen. Gleiches gilt für die Definition der Bedingung eines Übergangs zwischen zwei Zuständen. Hierfür sind geeignete Eingabe- und Visualisierungstechniken zu schaffen, welche direkt die Eingabemöglichkeiten am Graphen vorsehen. Diese konzeptionelle Kombination aus Grapheneditor und der Eingabe von Programmcode für die Spezifizierung des Objektverhaltens ist elementarer Bestandteils des Konzepts, weil ansonsten das Objektmodell, wie in anderen Simulationsumgebungen auch, vollständig verbal implementiert wird und die Vorteile einer visuellen und strukturellen Ähnlichkeit zu einem Ausgangsmodell wie Abbildung 1 verloren geht.

Die Ausführung des neuen Objektmodells ist nicht kompatibel zu der herkömmlichen Ausführung von Objekten objektorientierter Programmierung. Nach der Ausführung eines Objektes muss die Ausführungseinheit den Folgezustand eines jeden Objektes berechnen und diesen in dem Objekt vermerken, so dass bei der nächsten Ausführung die zustandsspezifischen Methoden und Attribute zum Einsatz kommen. Diese spezielle Abarbeitungsform ist in dem Simulationssystem entsprechend zu realisieren.

In dem neuen Objektmodell werden die Zustände als zusätzliche Strukturierungshilfe genutzt. Um einen zusätzlichen und fehlerträchtigen Programmieraufwand zur expliziten Implementierung von Zuständen durch den Anwender zu vermeiden, werden die notwendigen Informationen stattdessen in Form eines Graphen mit angefügtem Programmcode eingegeben. Dieses „verdeckt“ die Erstellungs- und Verwaltungskomplexität vor dem Anwender, erfordert jedoch einen erhöhten Implementierungsaufwand in die Simulationsumgebung.

4 Vom Modell zum Simulationsmodell

Die Implementierung eines bereits bestehenden Modells zu einem Simulationsmodell gestaltet sich für ein Objekt relativ einfach. Das Modell liegt, wie bereits beschrieben, in einer automatenähnlichen Beschreibung vor.

Im ersten Schritt wird zunächst die Klasse und ihre Eigenschaften definiert. Dieser Aufwand ist unabhängig von der gewählten Programmiersprache, weil diese Art der Information immer definiert werden muss. Die Definition umfasst die Eigenschaften der Klasse sowie die notwendigen Methoden, welche das Individuenverhalten bestimmen.

Der zweite Schritt besteht in der Übertragung des bereits vorliegenden Modells in das Automatenmodell der Simulationsumgebung. Weil die Modelle bereits in einer automatenähnlichen Beschreibung vorliegen, besteht dieser Schritt in einer einfachen Übertragung. Dieser Schritt kann mit der Übertragung einer handschriftlichen Zeichnung in eine computergestützte Zeichnung gleichen Inhalts verglichen werden. Inhaltlich hat der Anwender wenig beizutragen, der Prozess wird im Wesentlich durch die Möglichkeiten des Editors bestimmt. Am Ende der Übertragung liegt ein Simulationsmodell mit bezeichneten Zuständen und Übergängen vor.

Im dritten Schritt werden in den Zuständen die Methoden definiert, welche für den Zustand den spezifischen Programmcode bereitstellen.

Im vierten Schritt werden die Übergangsbedingungen zwischen den Zuständen mit Inhalt gefüllt. Dieser Schritt kann gleichzeitig mit Schritt 3 erfolgen, jedoch ist es u.U. sinnvoller, erst das Verhalten innerhalb der Zustände zu definieren, bevor das Verlassen eines Zustands (Übergangsbedingung) definiert wird.

4.1 Betrachtung der Arbeitsschritte

Für die vier Arbeitsschritte wird ein graphischer Editor zur Anzeige und das Editieren des Zustandsautomaten benötigt. Dieser Editor bestimmt mit seinem Komfort den zu betreibenden Eingabeaufwand für den Anwender. Mit einem graphischen Editor besteht prinzipiell eine verständliche Visualisierung des Zustands- und somit auch des Verhaltensmodells, so dass ein Anwender bei Betrachtung des Zustandsmodells bereits einen Eindruck über die „Arbeitsweise“ des definierten Individuums erhalten kann. In einer herkömmlichen Programmiersprache ließe sich Vergleichbares nur über eine separate Dokumentation erreichen.

Die Visualisierung ist zur Zeit der wesentliche Vorteil, den dieses Objektmodell gegenüber traditioneller Programmierung besitzt. Es besteht jedoch ein geringfügig höherer Aufwand, weil zunächst in Schritt 1 die allgemeinen Eigenschaften des Objekts definiert werden. Dieser Schritt wäre in traditioneller Programmierung nicht zwingend von den anderen Implementierungsschritten zu trennen, so dass dieses als zusätzlicher Aufwand betrachtet werden kann.

Eine weitere Betrachtung soll diesen möglichen initialen Mehraufwand, den die zusätzliche Strukturierung durch Zustände verursacht, nivellieren.

4.2 Änderung des Modells: Erweiterung um einen Zustand

Eine Zustandserweiterung eines bereits formulierten Modells um einen weiteren Zustand ist relativ einfach, denn es wird in dem graphischen Editor (Schritt 2 und 3) lediglich der neue Zustand eingefügt und seine Methoden definiert. Danach werden die Übergangsbedingungen definiert (Schritt 4) und bestehende bei Bedarf modifiziert.

Bei einer herkömmlichen Programmierung sind alle Prozeduren, welche zustandsspezifischen Programmtext enthalten, zu analysieren und gegebenenfalls zu modifizieren. Dieses umfasst wenigstens alle Prozeduren, welche bisher zustandsspezifischen Programmtext enthalten, wenigstens aber die Prozedur, welche ausgehend vom aktuellen Zustand den Folgezustand berechnet. Dieser Aufwand kann jedoch steigen, weil der gesamte Programmtext auf mögliche Abhängigkeiten untersucht werden muss, die sich durch eine Veränderung des Modells ergeben. Nur bei absoluter Sicherstellung über eine seiteneffektfreie Implementierung kann die Analyse nach Abhängigkeiten ausgelassen werden.

4.3 Änderung des Modells: Löschen eines Zustands

Wenn ein Zustand aus dem Objekt entfernt werden soll, dann ist in dem neuen Objektmodell der Zustand mittels des graphischen Editors zu löschen. Mit dem Löschen wird *sämtlicher zustandsspezifischer Programmtext* entfernt. Es sind in einem zweiten Schritt die Übergangsbedingungen anzupassen.

Bei der herkömmlichen Programmierung sind alle Arbeitsschritte wie bereits in dem vorherigen Fall bei der Zustandserweiterung auszuführen, denn es müssen die Abhängigkeiten innerhalb der Programmlogik gefunden, analysiert und gegebenenfalls umformuliert werden.

5 Realisierung des Objektmodells

Das vorgestellte Objektmodell ist bereits innerhalb eines Prototypen implementiert worden. Im Rahmen der Implementierung konnte die Funktionsfähigkeit gezeigt werden, d.h. während der Ausführung der Simulationen wurde entsprechend des vom Objekt eingenommenen Zustands die für diesen Zustand definierten Methoden ausgeführt und auf die in diesem Zustand definierten Eigenschaften zugegriffen.

Weiterhin ist eine Vererbungsstrategie entwickelt worden, welche den Aufbau einer Klassenhierarchie erlaubt. Innerhalb dieser Klassenhierarchie ist die Vererbung der Zustände und Übergangsbedingungen vorgesehen. Die Implementierung im Prototypen zeigte, dass für eine Vererbungshierarchie mit Vererbung der Zustände ein wesentlicher Aufwand in der Entwicklung des graphischen Zustandseditors investiert werden muss; nur im Rahmen eines Editors ist eine adäquate Unterstützung des Anwenders sowie eine Entlastung bei der Entwicklung des Zustandsmodells möglich.

Die Implementierung des Prototypen verdeutlichte auch den Aufwand, der für die Implementierung des Objektmodells sowie der Unterstützung des Anwenders bei der graphischen Zustandsmodellierung zu betreiben ist. Neben diesen Neuerungen ist zusätzlicher Entwicklungsaufwand hinsichtlich der Simulationsausführung, der Visualisierung sowie der Plazierung von Objekten in dem simulierten Universum notwendig.

Die Entwicklung einer *neuen* Simulationsumgebung für die Realisierung des vorgestellten Objektmodells ist aufgrund des notwendigen Entwicklungsaufwands kritisch zu betrachten, denn eine neue Simulationsumgebung steht zunächst in Konkurrenz zu bereits entwickelten und „am Markt“ etablierten Simulationsumgebungen. Es ist fraglich, ob Anwender zu einer neuen Simulationsumgebung mit einem neuen – wenn auch verbesserten Programmiermodell – wechseln, sofern sie ihre bisherigen Simulationsmodelle nicht „mitnehmen“ können, da ansonsten die Modelle erneut in Simulationsmodelle überführt werden müssen.

Aus genannten Gründen wäre es sinnvoll, eine *bestehende* und *etablierte* Simulationsumgebung zu erweitern. Es bieten sich Simulationsumgebungen an, die auf *open source* basieren oder in einem universitären Umfeld entspringen, so dass die Möglichkeit einer externen Erweiterung bestünde.

Eine für dieses Vorgehen geeignete Simulationsumgebung wäre NETLOGO bzw. eine andere Simulationsumgebung aus dieser Familie. Eine statistische Erfassung in (Höh07) zeigte, dass NETLOGO von allen aus Tabelle 1 aufgeführten Simulationsumgebungen die höchste Akzeptanz, die größte Modellvielfalt besitzt und kontinuierlich weiterentwickelt wird. Eine Erweiterung des bestehenden Programmiermodells von NETLOGO wäre möglich, weil das bisherige Programmiermodell eine Untermenge des neuen Programmmo-

dells darstellte. Bisherige Modelle könnten automatisch in das neue Programmiermodell konvertiert werden, wobei für diese nicht die Vorteile des neuen Modells automatisch verfügbar wären. Der Vorteil läge zunächst in der Weiternutzung bereits entwickelter Modelle. Ausgehend von diesen Modellen wäre eine sukzessive Umsetzung in das neue Programmiermodell möglich, so dass ein direkter Vergleich von herkömmlicher Implementierung und dem hier vorgestellten Objektmodell möglich wäre, um die Vorteile des Konzeptes im praktischen Einsatz verfolgen zu können.

Literatur

[CFS+01] Scott Camazine, Nigel R. Franks, James Sneyd, Eric Bonabeau, Jean-Louis Deneubourg und Guy Theraulaz. *Self-Organization in Biological Systems*. Princeton University Press, 2001.

[DKPT98] A. Dornhaus, F. Klügl, F. Puppe und J. Tautz. Task Selection in Honey Bees - Experiments Using Multi-Agent Simulation. In C. Wilke, S. Altmeyer und T. Martinetz, Hrsg., *Proceedings of the Third German Workshop on Artificial Life, GWAL-98*, Seiten 171–184. Verlag Harri Deutsch AG, 1998.

[GS95] Martin Gerhardt und Heike Schuster. *Das digitale Universum: Zelluläre Automaten als Modelle der Natur*. Vieweg Verlag, 1995.

[HMD02] John E. Hopcroft, Rajeev Motwani und Jeffrey D.Ullman. *Einführung in die Automatentheorie, formale Sprachen und Komplexitätstheorie*. München: Pearson Studium, 2., überarb. aufl. Auflage, 2002.

[Höh07] Jörg Höhne. *Ein zustandsinhärentes Objekt- und Programmiermodell zur Simulation natürlicher emergenter Prozesse: Nutzungsmöglichkeiten von Multiagententechnik für die schulische Bildung*. Dissertation, Universität Bremen, Bremen, Deutschland, 2007.

[KCR02] Eric Klopfer, Vanessa Colella und Mitchel Resnick. New paths on a StarLogo adventure. *Computers and Graphics*, 26(4):615–622, 2002.

[KW04] Horst Koschwitz und Joachim Wedekind. Künstliches Leben im Biologieunterricht: Mikrosimulationen mit Multi-Agenten-Systemen. *LOG IN*, 130:28–34, November 2004.

[SC04] Thomas Schmickl und Karl Crailsheim. Costs of Environmental Fluctuations and Benefits of Dynamic Decentralized Foraging Decisions in Honey Bees. *Adaptive Behavior - Animals, Animats, Software Agents, Robots, Adaptive Systems*, 12(3-4):263–277, 2004.

[Woo05] Michael J. Wooldridge. *An Introduction to Multiagent Systems*. Wiley&Sons Ltd., repr. Auflage, 2005.