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).

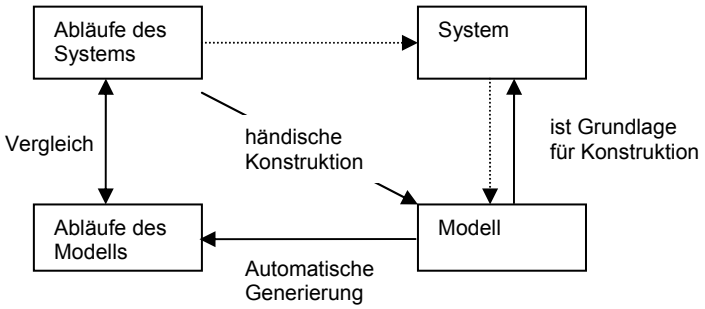


Abbildung 1: Traditioneller Ansatz

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.

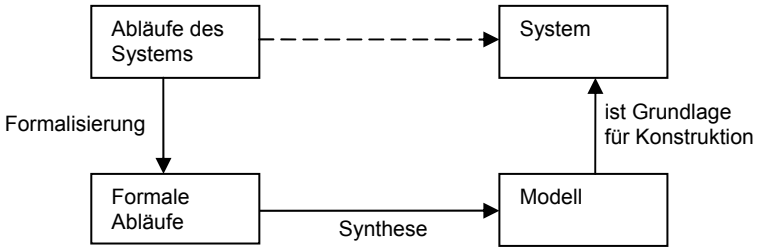


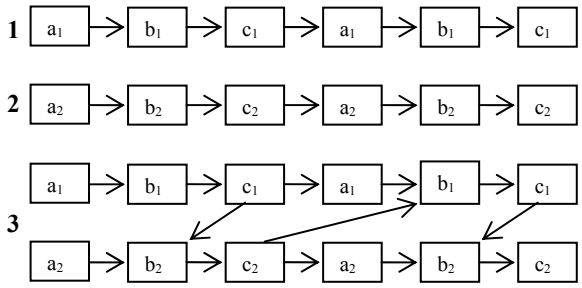
Abbildung 2: Der hier vorgeschlagene Ansatz

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-

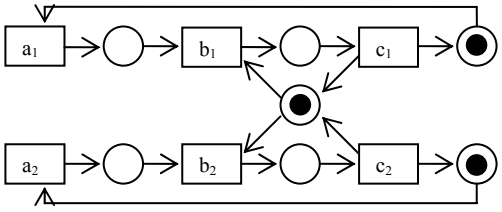
matisch generierte Modelle durch Konstruktion ihrer Abläufe und Vergleich dieser mit den „Formalen Abläufen“ zu validieren. Schließlich wird in einem letzten Schritt das System aus dem Modell (und weiteren Artefakten) generiert. Die drei Schritte realisieren das Gesamtziel, nämlich die Entwicklung eines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